SET Secure Electronic Transaction Specification

Book 2: Programmer's Guide

Version 1.0
May 31, 1997

PC

This Programmer's Guide will continue to be reviewed and updated to ensure clarity and consistency with the Formal Protocol Definition in Book 3.

MasterCard and Visa are creating a process for applications to be certified as compliant with the SET specifications.

Preface

Introduction

The development of electronic commerce is at a critical juncture.

- Consumer demand for secure access to electronic shopping and other services is very high.
- Merchants want simple, cost-effective methods for conducting electronic transactions.
- Financial institutions want a level playing field for software suppliers to ensure quality products at competitive prices.
- Payment card brands must be able to differentiate electronic commerce transactions without significant impact to the existing infrastructure.

The next step toward achieving secure, cost-effective, on-line transactions to satisfy market demand is the development of a single, open industry specification.

Secure Electronic Transaction protocol

Visa and MasterCard have jointly developed the SET Secure Electronic Transaction protocol as a method to secure payment card transactions over open networks. SET is being published as an open specification for the industry. This specification is available to be applied to any payment card service and may be used by software vendors to develop applications.

Advice and assistance in the development of these specifications have been provided by GTE, IBM, Microsoft, Netscape, RSA, SAIC, Terisa, and Verisign.

Payment gateway and certificate authority software

While this specification defines the interface to the payment gateway and the certificate authority, it does not provide all necessary information for a software vendor to create these systems. Specifically, the specification does not address the interface between the payment gateway and the existing financial system and does not address the mechanism for the processing of certificate requests, which depends on payment card brand policy.

Continued on next page

Preface, continued

SET documentation

The SET specification is documented in three volumes:

Book	Title	Contents
1	Business Description	Contains background information and processing flows for SET. Intended as a primer on software that both interfaces with payment systems and uses public-key cryptography.
2	Programmer's Guide	Contains the technical specifications for the SET protocol. Primarily intended for use by software vendors who intend to create cardholder and merchant software.
3	Formal Protocol Definition	Contains the formal protocol definition for SET. Primarily intended for use by: • cryptographers analyzing security, • writers producing programming guides, and • system programmers developing cryptographic and messaging primitives.

Organization

The Programmer's Guide contains the following parts:

System Design Considerations	Part I summarizes the system design considerations to be used in developing SET toolkits and applications. It provides background information and introduces features and notation that will be used throughout the Programmer's Guide.
Certificate Management	Part II defines the certificate request architecture, protocols, and concepts used in SET.
Payment System	Part III presents the description and processing of the payment system portion of the SET protocol, including all messages related to authorization, capture, and management of the payment system.
Appendices	This is a collection of supplementary information related to the SET specifications.

Continued on next page

Preface, continued

Related documentation

The following articles and books contain additional background material. Readers are encouraged to consult these references for more information.

Answers to Frequently Asked Questions about Today's Cryptography, Paul Fahn, RSA Laboratories, 1993. (http://www.rsa.com/rsalabs/faq/)

Applied Cryptography, Second Edition, Bruce Schneier, John Wiley & Sons, Inc., 1996.

"Asymmetric Encryption: Evolution and Enhancements," Don B. Johnson and Stephen M. Matyas, *CryptoBytes*, volume 2, number 1, Spring 1996

BSAFE 2.1™, RSA Data Security, Inc., 1994. (http://www.rsa.com/rsa/prodspec/bsafe/rsa_bsaf.htm)

Data Encryption Standard, Federal Information Processing Standards Publication 46, 1977.

"The HMAC Construction," Mihir Bellare, Ran Canetti, and Hugo Krawczyk, *CryptoBytes*, volume 2, number 1, Spring 1996

HTML Sourcebook, Ian S. Graham, John Wiley & Sons, Inc., 1995.

The Internet for Everyone: A Guide for Users and Providers, Richard W. Wiggins, McGraw-Hill, Inc., 1995.

Optimal Asymmetric Encryption, M. Bellare and P. Rogaway, Eurocrypt 94. (http://www-cse.ucsd.edu/users/mihir/papers/oae.ps.gz)

An Overview of the PKCS Standards, Burton S. Kaliski, Jr., RSA Laboratories, 1993. (http://www.rsa.com/pub/pkcs/doc/ or http://www.rsa.com/pub/pkcs/ps/)

Public-Key Cryptography Standards (PKCS), RSA Data Security, Inc., Version 1.5, revised Nov. 1, 1993.

Extensions and Revisions to PKCS #7, RSA Data Security, Inc., May 13, 1997.

ITU Rec. X.509 (1993) | ISO/IEC 9594-8: 1995, including Draft Amendment 1: Certificate Extensions (Version 3 certificate).

RFC 1750, Randomness Recommendations for Security, D. Eastlake, S. Crocker, J. Schiller, December 1994.

Table of Contents

Table of Figures

Table of Tables

Part I
System Design Considerations

Overview

Introduction

Part I summarizes the system design considerations to be used in developing SET toolkits and applications. It is intended to provide background information as well as to introduce the salient features and notation that will be used in subsequent parts of this Programmer's Guide.

Audience

It is assumed that the reader will be developing software for cardholder and merchant systems. There are instances when requirements specific to the Payment Gateway and Certificate Authority systems are stated. These additional requirements, however, are informative and intended to assist the reader in understanding the processing performed by these systems. In addition to the requirements stated in this document, each brand will have its own specific implementation requirements.

It is also assumed that the reader:

- is familiar with the business requirements defined in Book 1: Business Description of the SET specification, and
- possesses a general understanding of cryptography and networking protocols.

Payment card

To facilitate the use of SET in other card-based payment environments, the term "payment card" is used throughout the remainder of this Programmer's Guide to collectively refer to any of the following: credit card, debit card, charge card, and bank card.

Terminology

Throughout this document, the verb "shall" indicates a requirement that is imposed by this Programmer's Guide and "will" indicates either a goal or an implicit requirement that is imposed external to SET. "Must" indicates a requirement that is imposed external to SET, such as by export requirements.

In processing steps, "shall" is normally implicit. That is, unless otherwise indicated, an instruction such as "verify x" is equivalent to "the application shall verify x."

The sequence of processing steps may be varied so long as the results are the same.

Transport Mechanisms

Transport mechanisms used to implement SET are out of scope of this document, but two classes are recognized: interactive and non-interactive. The World Wide Web is an interactive mechanism, and electronic or postal mail are non-interactive mechanisms.

Continued on next page

Overview, continued

Organization

Part I includes the following chapters:

Chapter 1
Introduction

Overview

Introduction

Chapter 1 provides background information and an overview of the payment processing.

Organization

Chapter 1 includes the following sections:

Section	Title	Contents	Page
1	Background	Provides background information with emphasis on the scope of SET.	4
2	Environment	Describes the environment for processing payment card transactions using SET.	7
3	Capture Process	Provides an overview of the batch capture processing.	10
4	Business Flows	Provides a high-level description of typical business flows relevant to SET.	17

Section 1
Background

Scope

The scope of this document is limited to the payment process and the security services necessary to support the payment aspects of the electronic shopping experience. To provide these services, SET defines not only the electronic payment protocol, but also the certificate management process.

Figure 1 below depicts the SET payment system participants and their interactions.

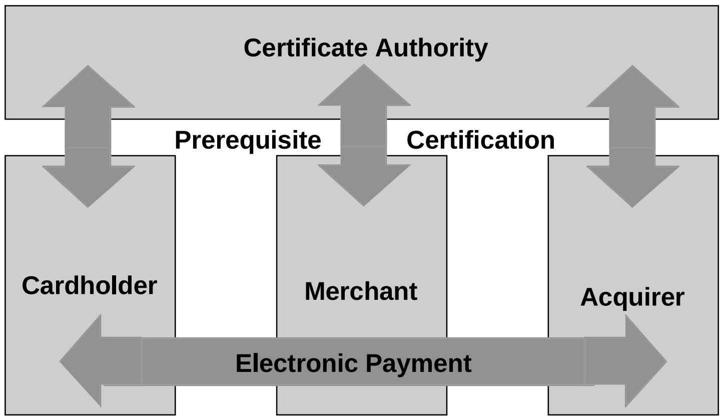


Figure 1: Payment System Participants

In the electronic commerce environment, new opportunities for merchants to conduct business will emerge due to the increased exposure and access by consumers for information about their products and services. Consumers will be able to shop, access information, and pay for goods and services. Unlike a face-to-face or mail order/telephone order transaction, the electronic processing of the payment begins with the cardholder rather than the merchant or the Acquirer. With these new opportunities, there will be new challenges that need to be addressed in order to facilitate payment via electronic shopping in a secure manner.

Continued on next page

Scope, continued

Electronic shopping

Electronic shopping will typically proceed through the following phases. SET supports three of these phases: a) payment authorization and transport, b) confirmation and inquiry, and c) merchant reimbursement.

- Browsing and Shopping
- Merchant and Item Selection
- Negotiation and Ordering
- Payment Selection
- **SET ➧ Payment Authorization and Transport**
- **SET ➧ Confirmation and Inquiry**
- Delivery of Goods
- **SET ➧ Merchant Reimbursement**

Goods and services

The cardholder decides to start shopping for an item. Items shopped for might be tangible goods, electronic media (for example, information, software, etc.), or services.

Assumptions

- The bit stream for the order description and purchase amount at the merchant is identical to the bit stream for the order description and purchase amount at the cardholder.
- The cardholder and merchant software shall agree on a specific protocol (and therefore representation of this data) before SET is invoked.
- Transaction information (for example, capture tokens or certificate responses) will not be stored by SET software indefinitely, particularly when processing credit and reversal transactions. Payment brand operational guidelines will specify minimum time periods for storage of information on merchant and payment gateway systems that must be supported.

Continued on next page

Scope, continued

Processing

SET's relationship with the other phases of the electronic shopping model is described in the following table. SET focuses on phases 5, 6, 7, and 9.

Phase	Description
1	The cardholder browses for items. This may be accomplished in a variety of ways, such as: • using a browser to view an on-line catalog on a merchant's World Wide Web page; • viewing a catalog supplied by the merchant on a CD-ROM; or • looking at a paper catalog.
2	The cardholder selects items to be purchased from a merchant.
3	The cardholder is presented with an order form containing the list of items, their prices, and a total price including shipping, handling, and taxes. This order form may be delivered electronically from the merchant's server or created on the cardholder's computer by electronic shopping software. Some on-line merchants may also support the ability for a cardholder to negotiate for the price of items (such as by presenting frequent shopper identification or information about a competitor's pricing).
4	The cardholder selects the means of payment. SET focuses on the case when a payment card is selected.
5	The cardholder sends the merchant a completed order along with a means of payment. In SET, the order and the payment instructions are digitally signed by cardholders who possess certificates.
6	The merchant requests payment authorization from the cardholder's financial institution via acquirer. If authorization succeeds, the merchant may send confirmation of the order out of band to SET.
7	The merchant ships the goods or performs the services requested from the order.
8	The merchant requests payment from the cardholder's financial institution via acquirer.

Section 2
Environment

Processing Overview

Mail order/ telephone order

The processing of transactions using this Programmer's Guide generally follows that of the Mail Order / Telephone Order (MOTO) environment. In such a transaction, order and payment information is transmitted to the merchant either by mail or by telephone in contrast to a "card present" transaction when a customer is making a purchase at the merchant's store.

Description: The information that follows is a brief description of a simplified MOTO processing model.

Variations: There are many variations on this processing model, but the following represents a typical exchange.

Environment: By using the protocol described in this Programmer's Guide, the interactions between the customer and the merchant can occur in an interactive environment, such as the World Wide Web, or through non-interactive means such as electronic or postal mail exchanges.

Shopping

The flow of activity begins when the shopper (cardholder) contacts a merchant to request a list of the goods and services offered. The customer shops from this list.

Ordering

The cardholder selects items, prepares an order and sends it to the merchant, who receives and processes the order.

Inventory

The merchant checks inventory to determine if the goods and services ordered by the customer need to be back-ordered. The merchant may decide to handle the order as a split shipment.

Authorization request

The merchant sends an authorization request to its financial institution (Acquirer). The Acquirer incorporates the authorization data into a request that is sent via a payment network for processing by the financial institution (Issuer) that issued the payment card to the cardholder.

Continued on next page

Processing Overview, continued

Authorization response

The Issuer responds to the Acquirer via the payment card network with an authorization response. The response includes an indication of whether the authorization request has been approved. The Acquirer responds to the merchant with the outcome of processing.

Shipping

The merchant delivers the goods and services to the customer. The time delay between authorization and shipment (which shall precede capture) can legitimately be several days. Many MOTO merchants are not able to check inventory before authorization. If goods are not available for immediate delivery, the shipment is held up until the order can be fulfilled.

Capture processing

The merchant submits a capture request to the Acquirer in order to obtain payment. This request is sent through the payment card network to the Issuer.

Credit processing

If a credit is to be issued to a customer, such as when the goods are returned or defective, the merchant sends a message to the Acquirer requesting a credit be issued to the cardholder’s account.

Combined authorization and capture

At its option, the Acquirer may allow the merchant to combine the authorization and capture messages into a single message.

Merchant’s processing

At its option, the Acquirer may allow the merchant to initiate the administration and reporting functions related to capture processing.

Continued on next page

Processing Overview, continued

SET and MOTO

Figure 2 illustrates how SET and MOTO work together.

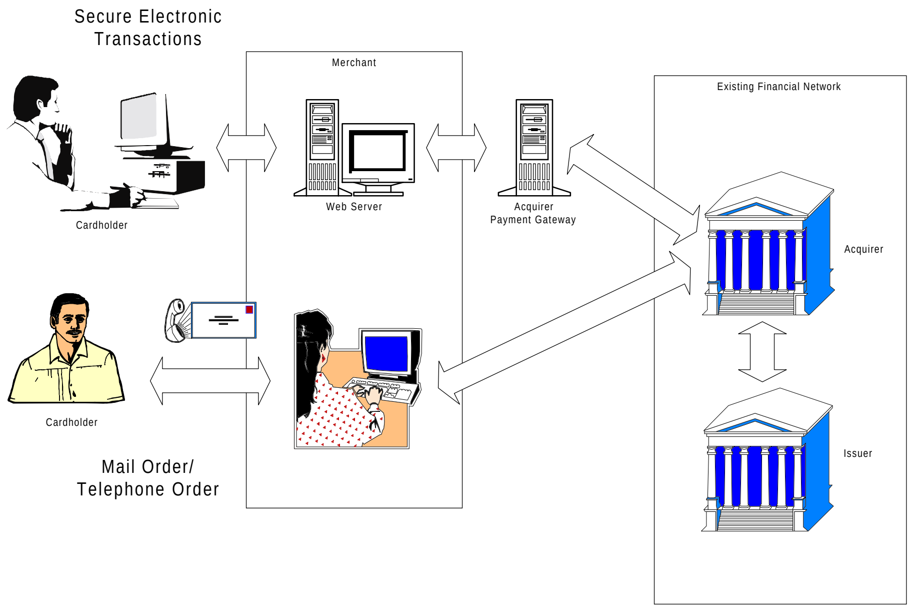

Secure Electronic Transactions
Merchant
Existing Financial Network
Cardholder
Web Server
Acquirer Payment Gateway
Acquirer
Cardholder
Issuer
Mail Order/ Telephone Order

Figure 2: SET / MOTO Comparison

Section 3
Capture Process

Overview

Purpose

This section provides a high-level description of the batch capture processing models relevant to SET.

Types

The types of batch capture processing models include the following:

- Merchant Terminal Data Capture
- Acquirer Host Data Capture

Audience

This information is intended primarily for developers of merchant software.

Batch Capture Processing

Introduction

SET batch capture processing supports the following:

- A payment gateway connected to an Acquirer host.
- A payment gateway connected to an intermediate capture system.
- Processing performed out of band to SET.

Identifying batches

The Merchant, payment gateway, or Acquirer assigns each SET transaction to a specific capture batch and also assigns:

- An integer to identify a batch.
- A unique integer to each item within a batch.

Access to batches

The Merchant, Acquirer, or Payment Gateway is able to open a capture batch. For batches opened by the Acquirer or Payment Gateway, the batch shall only be closed by the Acquirer or Payment Gateway. For batches opened by the Merchant, the batch may be closed by the Merchant, Acquirer or Payment Gateway.

For batches opened by the Merchant, the Merchant can add items to and remove items from an open capture batch, purge all of the items from an open batch, and close the batch.

Merchant Inquiries

The Merchant can inquire of the Acquirer or Payment Gateway to determine the status of:

- A batch and the items within a batch (batch items may be capture or credit requests).
- The transmission of batch information from the gateway to the next upstream system.

The Merchant can send or receive batch totals and item details to (or from) the Acquirer or Payment Gateway.

Out-of-band capture requests

For merchants processing capture requests out of band to SET, the authorization response must include all data necessary to clear the transaction at the best available interchange rate, based on the characteristics of the authorization.

Terminal Data Capture

Overview

Terminal Data Capture (TDC) occurs when a merchant re-presents previously authorized transactions (authorization request/response data) to a Payment Gateway as a request for payment of those transactions. Although transactions are accumulated in a batch by the merchant, each transaction submitted is evaluated individually and may be accepted or rejected for payment processing. The batching of transactions is used for processing efficiency. The gateway (or receiver) needs to ensure that each transaction sent by the merchant was received by the gateway. This is done through the use of transaction identifiers to match up authorization and capture, either **TransIDs.XID**, or an alternate identifier such as **TransIDs.LID-M** if available and agreed to by merchant and payment gateway. By grouping multiple transactions into a batch, the overhead of handling individual request/response pairs is avoided.

Batch Balancing

Batch balancing may be performed by the Merchant after requesting batch details from the Payment Gateway, or by the Payment Gateway on receipt of batch details from the Merchant. Batches are balanced using the Merchant assigned **BatchID**.

Transaction tracking using TransIDs

For TDC, transaction tracking begins when the transaction is first created during the **PInitReq** or **PReq** messages. These are used by the merchant to identify and track each action on the transaction back to the original **PInitReq** or **PReq** and relates the authorization and capture. Within merchant Point Of Sale (POS) software, **PAN**s or transaction identifiers may optionally be used to match authorization responses to authorization requests.

Continued on next page

Terminal Data Capture, continued

Initiate capture

In TDC, the same **TransIDs** are employed from authorization through the completion of clearing. (This facilitates investigation, research, etc., since to re-assign would necessitate a cross-reference.) The **TransIDs** to be associated with each individual transaction is provided by the merchant in the capture request.

Include in capture

It is expected that all authorized transactions will be captured. When this is not to be the case, or if the merchant submits clearing transactions outside SET, the merchant shall not submit those **TransIDs** for capture.

Change amount

If the transaction amount is to change between authorization and capture, or if clearing is to be handled outside the definition of SET, the **TransIDs** shall identify the transaction in which to change the amount.

Continued on next page

Terminal Data Capture, continued

Merchant batching

For TDC, there are two concepts of merchant batching. The first is for the merchant's operational reason. For example, a merchant might want to know how much was collected between noon and 1 p.m. The merchant may create a batch and accumulate all transactions to that batch. All transactions in this batch should use a common merchant batch identifier.

The second batching concept is of special interest to SET, and occurs when a merchant groups authorized transactions for the purpose of draft capture, and assigns a common **BatchID** to these transactions. A batch in this instance consists of those individual transactions (**TransIDs** as defined above) for which the merchant is requesting payment, and has allocated a common **BatchID**.

Batch identification

The merchant shall use the **BatchID** to designate the batch and its contents. In TDC each capture request **CapReq** associates transactions with one or more SET batches referenced by **BatchID**. A SET batch may consist of a single sequence number, representing one authorized transaction, or many sequence numbers representing multiple authorized transactions.

Payment Gateway processing

At the option of the Acquirer / Payment Gateway or through payment card brand mandates, data from the original authorization request/response pair may beaugmented with the **PAN** expiration date and capture token to be included in a capture message, and sent out of band for capture processing by the financial network.

The use of a capture token is optional, based on the requirements of individual payment card brands or acquirers. Merchants using out-of-band clearing or normal transaction processing will most likely bypass the use of capture tokens in favor of **PAN** tokens. Cryptographic treatment differentiates capture tokens from **PAN** tokens. The capture token blinds the merchant to the data, the **PAN** token is encrypted so that the **PAN** data may be recovered by the merchant.

When the Payment Gateway receives a data capture response from the Issuer to its data capture requests from the SET financial network, it generates the SET capture response (**CapRes**) message back to the Merchant.

Refer to the processing descriptions in "Capture Request/Response" on page 374 for additional information.

Host Data Capture

Definition

Host Data Capture (HDC) is a feature that allows merchant terminals to process card transactions and have those transactions submitted to clearing without having to support an out-of-band transaction deposit.

Overview

Essentially, the transaction is logged by the Acquirer host system and assigned to a physical or logical batch depending upon the Acquirer application, the method of accounting of the merchant, and the relationship between the merchant and the Acquirer (that is, the merchant may submit MasterCard and Visa transactions through the Acquirer for 1-day credit, Discover transactions through the Acquirer for 3-day credit, and submit American Express transactions directly to American Express). The Acquirer may log all of these transactions into separate batches by brand, or into a single batch that is submitted at the end-of-day. (Note that multiple brands may be in the same batch, but not in the same capture request.)

Batch balancing

Batch balancing is performed by the merchant by adding the receipts at the point of service and comparing that amount to the batch total(s) at the host. If the amounts are the same, the batch balances. If the amount differs, the merchant may be able to examine each transaction through a look-up based on account number, transaction ID, amount, or various combinations of those elements.

HDC types

HDC merchant transactions can occur as "authorization" transactions followed by "capture" transactions or as single "sale" transactions.

Authorization/ and later capture

The merchant requests transaction authorization (which may be for an approximate amount) followed by a capture amount to be submitted in lieu of a paper draft for merchant payment (deposit of funds to the merchant bank account). This method is often used when the final amount of the transaction is not known (that is, shipping and handling not calculated). The merchant needs authorization to continue the transaction and to verify that the account number is valid and assurance from the Issuer that there is not a problem with the account and amount. When the exact amount of the transaction is known to the merchant and the transaction is "complete" (that is, the order has been shipped) a "capture" transaction may be sent to the Acquirer for merchant payment.

Sale

"Sale" transactions are single transactions that combine the "authorization" and "capture" transactions. Sale transactions are used when the exact amount of the transaction is known at the time of purchase and the transaction is complete (for example, software purchase and download, purchase of network services, etc.).

Continued on next page

Host Data Capture, continued

Batching

Batching of transactions, as indicated above, is handled on the Acquirer host system. Transactions are assigned a transaction ID by the Acquirer host system and placed into a batch as determined by the criteria above. Batch balancing options vary by software vendor and merchant agreement. Merchants usually balance by adding or deleting transactions, or modifying transactions.

End-of-day reporting

At the end-of-day (or end-of-batch) the batch is balanced, closed, and submitted to the Acquirer clearing process. Reconciliation reports are generated at the time of submission, and may also be created to support batch balancing and other reconciliation requirements.

Deposit reporting

Certain systems will also provide deposit reports to the merchant to support the need of the merchant to track deposit information (when deposits are made to the merchant bank accounts and when those amounts will be "collected" and available to the merchant).

Section 4
Business Flows

Overview

Purpose

This section provides a high-level description of typical business flows relevant to SET.

Introduction

The cardholder, using a PC, shops by visiting a site on the World Wide Web or choosing an item from an online catalog. For this description, it is assumed that the cardholder has a certificate, having previously registered for electronic commerce. The shopping experience itself is outside the scope of SET.

When the cardholder decides to make a purchase, the transition from the shopping phase to the SET processing phase begins, which ensures the cardholder of a secure electronic transaction.

The purchase transaction itself may take place in a number of ways, depending on how the cardholder want to arrange for the purchase and on the merchant's business situation. For example:

- The cardholder may want to arrange for payments on an installment plan
- The order may be for tangible goods and the merchant may be out of stock on one or more of the items ordered, but able to ship the rest
- The order may be for non-tangible good, such as a video clip that can be delivered electronically—in which case the merchant can readily process both the authorization and capture request for payment from the Acquirer.

This section illustrated a range of typical business scenarios, which are enabled by SET processing according to the specific circumstances of a purchase. The first scenario described is the most typical—authorize now and capture later. It is followed by variations based on specific business conditions.

Audience

This information is intended primarily for Acquirers, Merchants, and developers of Merchant software.

Typical Business Scenarios

Overview

The scenarios described include the following:

- Authorize now and capture later
- Authorize and capture now: Sale Request
- Authorize now and capture later; partial reversal for a new amount
- Partial reversal with no brand support
- Split shipment
- Installment payments or recurring payments
- A credit for an old transaction

Figure 3 is a state diagram that illustrates SET business flow messages. It shows, at a high level, the transitions from shopping to ordering and processing of the order, with the processed state shown in two variations: "sale processed" in the case of an order that is authorized and captured at the same time, and "captured" for an order that is authorized now and captured later. The figure also shows the processing of a credit from both of these states. The message pairs are implicit in this diagram; for example, **AuthReq** represents both the authorization request and the response message.

In this scenario, there are transitions from one state to another, for example, from the ordered state to the sale processed state. Once the **PReq** message is processed, any transition that follows it can be reversed, with the effect of returning to the previous state. For example, when receiving an order, the merchant submits an authorization request; a subsequent authorization reversal request would take the transaction back to the ordered state. There is one exception, which is when a transaction is authorized and followed by a partial reversal to provide for a new amount.

Not all orders are authorized—for example, the merchant doesn't request authorization when an order is received for an item that is out of stock and will no longer be carried.

Continued on next page

Typical Business Scenarios, continued

Business flows

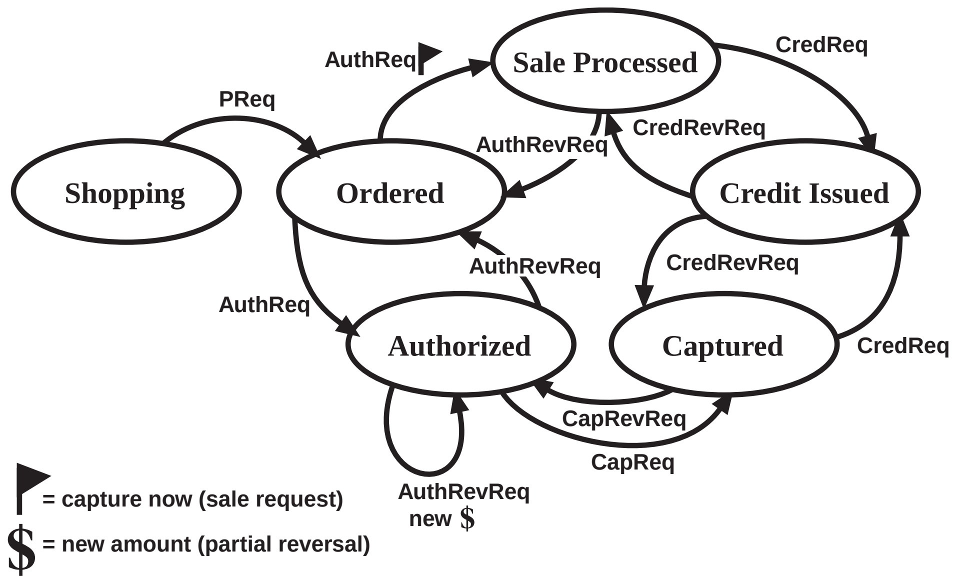


Figure 3: Business Flows

Continued on next page

Typical Business Scenarios, continued

Authorize now and capture later

For the most typical online purchase, the merchant is ready to authorize the transaction now, but wants to submit the capture request later. For example, many merchants prefer to collect their capture requests and submit them in batches at the end of the business day.

After the cardholder creates an order, the cardholder software sends the PReq message as a commitment to place an order (refer to the figure). This message and its response encompass the actual payment between the cardholder and the merchant, and take the cardholder from the shopping state to the ordered state. The **PReq** message includes:

- An Order Instruction (**OI**) from the Cardholder for the Merchant.
- The Payment Instruction (**PI**) from the Cardholder, encrypted and tunneled through the Merchant to the Payment Gateway.

The purchase response (**PRes**) message may be returned to the Cardholder immediately or any time later in the protocol. The information returned will depend upon the stage of processing in the protocol at which the purchase response is returned from the Merchant (for example, order received, transaction authorized, or transaction captured).

The merchant sends an authorization request to the Payment Gateway, but does not set the CaptureNow flag to true, as a capture request will be processed later. The authorization request indicates whether the merchant expects to do another authorization for a split shipment, recurring payment, or installment payment (discussed later in this section). If an authorization reversal is needed, it will return the transaction to the ordered state.

The merchant now has a commitment for payment from the issuer, but will need to process the capture request in order to be paid. The capture request may include multiple capture items, unless the account number is sent. It shall include a capture token if one is provided in the authorization response.

The capture request moves the transaction to the captured state, at which point the order is processed. A capture reversal may be sent; this will return the transaction to the authorized state. A partial authorization reversal may be used to change the amount after the authorization, returning the transaction to the authorized state—see the explanation of a partial authorization reversal later in this section.

Later, the cardholder may request a credit for the order—for example, if the cardholder decides to return the order because it was damaged in shipment. In this case a credit request is processed, moving the transaction from the sale processed state to the credit issued state.

Unlike a reversal, a credit is processed after an order is completed and shipped, and results in a credit on the cardholder's statement.

Continued on next page

Typical Business Scenarios, continued

Authorize and capture now: Sale Request

The Sale Request is used when the Merchant knows the item ordered is in stock and can be shipped right away, once the authorization is received. A Sale Request is also used for purchase of non-tangible goods available electronically—such as video clips, encyclopedia pages, and software programs—for which there is no question of inventory so the order can be fulfilled immediately.

SET allows a merchant to process the transaction as a single message by setting the **CaptureNow** flag in the authorization request to true. This indicates that if the transaction can be authorized, the capture should be done now, as well. In effect, it is a combined authorization and clearing.

When the Payment Gateway processes the request, there is a transition to the sale processed state. From a financial perspective, the sale processed state is equivalent to the captured state mentioned above.

Again, if there is a need to return money to the cardholder, a credit request moves the transaction into the credit issued state.

Authorize now and capture later: partial reversal for a new amount

In this scenario, the merchant sends an authorization request for the amount of the order, but later needs to revise the amount—for example, to factor in the charges for shipping and handling, which were not available when the initial authorization was submitted. The merchant sends an authorization reversal request with the new amount.

Note that some brands do not support partial reversals. In this case, the Payment Gateway may generate an **AuthRevRes** message with the appropriate value from the **AuthRevReq**, without sending a message to the financial network.

Continued on next page

Typical Business Scenarios, continued

Split shipment

When the merchant cannot fulfill the entire order, the items in stock are shipped and the remaining items are backordered.

The merchant may set the subsequent authorization indicator to tell the system that there is a business need for a subsequent authorization.

In this case, the payment gateway returns an **AuthToken** in the authorization response. The **AuthToken** serves the same purpose as the payment instruction, except that it originates with the payment gateway and is a means of allowing one additional authorization.

When the need for a split shipment is known at the time of authorization (for example, when the item is out of stock), a first authorization request is followed by a capture request for the item(s) to be shipped now; this is followed by another authorization request and capture request when the remaining items are available.

When the need for a split shipment is determined after the initial authorization, an authorization reversal is used to change the amount; it includes a capture token to be used for the subsequent authorization and capture.

Continued on next page

Typical Business Scenarios, continued

Installment and recurring payments

The merchant may offer customers the option of paying in installments—for example, three monthly payments. Or, the merchant may offer to process payments on a regular basis—for example, an Internet service provider may offer to bill the cardholder's account for the monthly service charge with no action needed by the cardholder.

The merchant presents the installment or recurring payment option, which is then indicated by the cardholder in the **PReq** message. In general, the payment instruction from the cardholder may only be used for one authorization request, unless the cardholder explicitly says that the merchant will need to do multiple authorizations. In this case, the payment instruction will provide for additional authorizations.

The merchant sets the subsequent authorization indicator to alert the system that there is a business need for subsequent authorization. The payment gateway returns an **AuthToken** in the authorization response, which will allow one additional authorization in place of the next payment instruction. Then, as each authorization request is completed, the payment gateway includes an **AuthToken** for the next authorization—until the authorization for the final installment is processed, when no **AuthToken** will be returned.

Credit for an old transaction

A cardholder may submit a request for credit after all data relating to the original transaction has been purged from the merchant's logs—individual Acquirers will establish with their merchants recommended times for data to be retained. SET supports the processing of a credit when the merchant no longer has the information about the original transaction; in this case, the credit data will need to be key-entered.

SET also supports the processing of a credit to a different account than that used to pay for the order—for example, if a cardholder returns a gift and requests a credit to their account, rather than to the account of the person who purchased the gift. Again, the credit data must be reentered.

Additional Information

Account numbers

SET allows the Payment Gateway to determine whether or not the merchant may receive the cardholder account number as part of the response. If the Acquirer decides not to return the account number, it needs to ensure that the Merchant has an alternative means of matching a chargeback to the original transaction.

The transaction contains a number of fields that can be used:

- **XID**—a 20-byte number that uniquely identifies the transaction, including all authorization and clearing messages for a single order.
- **RRPID**—a 20-byte number that uniquely identifies a request—a single authorization or clearing message.
- **localID-M**—a 1 to 20-byte local identifier assigned to the transaction by the merchant software. Depending on the implementation, this may be a tracking number assigned by staff operating the system or an internal number used solely by the Merchant software.
- **paySysID**—a 1 to 64-byte Transaction Identifier.
- **MerOrderNum**—a 1 to 25-byte merchant order number.

Chapter 2
System Architecture

Overview

Introduction

Chapter 2 provides an overview of the system architecture.

Organization

Chapter 2 includes the following sections:

Section 1
System Overview

Architecture

SET entities

The SET system is composed of a collection of entities involved in electronic commerce. The collection consists of:

- Cardholder, an authorized holder of a payment card supported by an Issuer, and registered to perform electronic commerce;
- Merchant, a merchant providing goods, services, and/or information who accepts payment for them electronically, and may provide selling services and/or electronic delivery of items for sale such as information);
- Issuer, a financial institution that supports issuing payment card products to individuals;
- Acquirer, a financial institution that supports merchants by providing service for processing payment card transactions;
- Payment Gateway, a system that provides electronic commerce services to the merchants in support of the Acquirer, and interfaces to the Acquirer to support the authorization and capture of transactions;
- Brand, a franchiser of payment systems / instruments;
- Certificate Authority (CA), an agent of one or more payment card brands that provides for the creation and distribution of electronic certificates for cardholders, merchants, and payment gateways; and
- Payment card brand's financial network, the existing private network operated by a payment card brand that links Acquirers and Issuers.

Protection of information

SET's architecture is designed to protect the transmission of financial information involved with a payment transaction between a cardholder, merchant, and Acquirer. It does not impose requirements on the transmission of the transaction's order information. Vendors developing shopping and ordering applications and protocols are strongly encouraged to protect this order information.

Continued on next page

Architecture, continued

SET cardholder

The SET cardholder is represented in SET by a workstation. This provides the cardholder with the flexibility to shop and conduct negotiations with merchant systems offering items for sale. The workstation may support all phases of the electronic shopping model described on page 6. In supporting SET, the workstation has the functionality to support the payment process.

Cardholder interfaces

The cardholder's primary interface in SET is to merchant systems. This interface supports the cardholder's portion of the payment protocol, which enables the cardholder to initiate payment, perform inquiries, and receive order acknowledgment and status.

The cardholder also has an indirect interface to the Acquirer through the merchant system. This interface shall support encrypted data fields that are sent via the Merchant to the Acquirer, but can only be decrypted by the Payment Gateway. This enables the Acquirer to mediate interactions between the cardholder and Merchant, and by so doing provide security services to the cardholder. These security services ensure that the cardholder is dealing with a valid, payment card-approved merchant.

Depending upon the policies established by the payment card brand, the cardholder may also interface with a Cardholder CA (CCA) to request and renew public key certificates that support electronic commerce security functions. Performing cryptographic functions in hardware cryptographic modules is recommended, but not required. Secret key generation and storage using tamper resistant hardware cryptographic modules such as smart cards is encouraged.

In addition, the cardholder system shall support security services (integrity, authentication, certificate management as prescribed by SET), and shall support the shopping, payment selection, and communications functions.

Continued on next page

Architecture, continued

Merchant interfaces

The SET merchant computer system provides a convenient interface to the cardholder for the support of electronic payments. In addition, the merchant interfaces with the Acquirer using the payment protocol to receive authorization and capture services for electronic payment transactions. The merchant shall interface with the Merchant CA (MCA) to request and renew public key certificates that support electronic commerce security functions.

The merchant shall support SET protocols for the authorization of electronic commerce transactions initiated by the cardholder. It is expected that the Merchant system will also support captures as well. In addition, the merchant system shall support security services (integrity, authentication, certificate management). Merchant systems shall support the shopping, payment selection, and communications functions. Performing cryptographic functions in hardware cryptographic modules is strongly recommended, but not required. Secret key generation and storage using tamper resistant hardware cryptographic modules such as smart cards is strongly encouraged. Payment card brand requirements for a specific implementation and environment in which the merchant server may operate will dictate requirements for the use of hardware cryptographic support.

Continued on next page

Architecture, continued

Payment gateway

The payment gateway system is operated by the Acquirer. It shall provide electronic commerce services to the merchants in support of the Acquirer, and shall interface with the payment card's financial network to support the authorization and capture of transactions. The payment card's financial network interface is largely unchanged from the interface supporting Acquirers today. The Payment Gateway shall also interface with the Payment Gateway CA (PCA) for requesting and renewing public key certificates to support the electronic commerce security functions. It shall support the distribution of certificate revocation lists (CRLs) on behalf of the brand and financial institution. Cryptographic functions shall be performed in hardware cryptographic modules. In addition, secret key generation and storage shall use tamper resistant hardware cryptographic modules.

Acquirer

An Acquirer is the financial institution (or its agent) that supports the merchant activity through account relationships with merchants. The Acquirer is responsible for gathering the financial data related to the transaction in order to obtain authorization for payment from the cardholder's Issuer.

Issuer

An Issuer is the financial institution that establishes an account for a cardholder and issues the payment card. The Issuer guarantees payment for authorized transactions using the payment card. The processing and interface to the Issuer is out-of-band from SET's perspective.

Third party processor

In some environments, Issuers and Acquirers may choose to assign the processing of payment card transactions to third party processors. This Programmer's Guide does not distinguish between the financial institution and the processor of the transactions.

Continued on next page

Architecture, continued

Certificate management

Certificate Management consists of one or more trusted CA systems that support the issuance and renewal of public key certificates for cardholders, merchants, and Acquirers. In addition, SET's architecture defines a trusted hierarchy of CA systems that begins with a Root CA (RCA), then a Brand-specific CA (BCA) and an optional Brand Geo-political CA (GCA). For example, the CCA systems interface with Issuers to authenticate requests for certificates. Refer to "Certificate Management" (starting on page 113) for details on certificates and certificate formats, certificate issuance, renewal, CRLs, and other certificate management functions. Cryptographic functions shall be performed in hardware cryptographic modules. Secret key generation and storage shall use tamper-resistant hardware cryptographic modules. Certificate management shall be performed in a secure physical environment compliant with payment card brand standards.

Payment card brand's financial network

The payment card brand's financial network is the existing private network through which Acquirers obtain authorization for payment from Issuers. VisaNet and Banknet are examples of these types of networks. These networks are protected by each payment card brand and provide messaging interfaces (such as ISO 8583 formatted messages).

Section 2
Security Services

Overview

Purpose

This section provides a brief summary of fundamental security services and certificates provided in SET's architecture.

Organization

This section includes the following topics:

- Services
- Certificates
- Brand CRL Identifier

Services

Confidentiality

SET provides confidentiality by employing both asymmetric and symmetric data encryption algorithms to protect financial information from eavesdroppers.

As an option, confidential Acquirer-to-cardholder messages are provided. This feature is intended for Issuers to communicate back to cardholders about the reason that a transaction is being declined or to request that the cardholder call the Issuer.

Authentication

SET provides authentication of a message's origin by employing digital signature verification algorithms when signature certificates are available.

Integrity

SET provides integrity by employing one-way cryptographic hashing algorithms and digital signatures to ensure that a message was not modified in transit.

Linkage

SET provides a linkage mechanism for verifying that a message contains a reference to another message by verifying an embedded link using a one-way cryptographic hashing algorithm.

Caveat

SET does not provide non-repudiation. It is the intent to permit non-repudiation via rules and policies of individual payment card brand implementations.

Certificates

Purpose of certificates

A digital signature cryptographically binds the signed data with an unique private key, which is assumed to be under the exclusive control of the cardholder, merchant, financial institution, or CA as appropriate. The private key is mathematically linked to the public key of the key pair. Assuming that the private key has not been compromised, the digital signature has the effect of binding the public key to the data as well. However, anyone can generate a public/private key pair, and so it is essential that some mechanism be established that binds the public key to the entity in a trustworthy manner. This is the fundamental purpose of a certificate – to bind a public key to a uniquely identified entity.

In the case of the cardholder, the signature certificate implicitly binds the public key to the cardholder's primary account number (**PAN**), but the **PAN** is effectively obfuscated by using a blinding technique so that only the CCA, the cardholder, and the Issuer know the account number. The cardholder passes the account number and a secret variable to the Acquirer, so that the Acquirer can then verify the card number against the blinded value contained in the cardholder's certificate. In order to protect the cardholder's confidentiality, the cardholder's name is not included in the certificate. In effect, the blinded account number is a pseudonym of the cardholder.

Since a bogus Certificate Authority could be set up to create certificates that would contain information nearly identical to that contained in a valid certificate, the signature of the Certificate Authority itself shall be certified as authentic by a higher level Certificate Authority. The only exception to this requirement is the industry root Certificate Authority. It is the only directly trusted Certificate Authority.

Continued on next page

Certificates, continued

Cardholder certificates

One function of the Acquirer is to ensure that the private key used to sign a payment is, in fact, associated with the right payment card account. To avoid revealing the cardholder's PAN to third parties, the number is hidden using a keyed-hashing mechanism as a blinding function. The result of this function is what is stored in cardholder's certificates.

The SET architecture allows cardholders without signature certificates to conduct SET transactions. This is an interim option intended only for use in situations where the cardholder's issuing bank does not provide certificate services. Acquirers may choose whether or not to support this option. A flag in the Acquirer's payment gateway certificate indicates support for cardholder transactions, in which the cardholder has no certificate.

Cardholder software and payment gateway software shall use the X.509 Certificate extension, **cardCertRequired**, a boolean flag set to true, to ensure that certificates are included in transactions as required. Brands which support cardholders without certificates may remove such support by reissuing payment gateway certificates and omitting this extension or setting this boolean flag to false. If a Cardholder has certificates available to them, the software should only perform signed transactions.

Support for cardholders with certificates is mandatory: Merchants and payment gateways shall fully support cardholder certificates and transactions based upon them.

Continued on next page

Certificates, continued

Merchant certificates

A merchant shall have at least two key pairs (encryption and signature) to participate in SET transactions. A merchant may have additional sets of encryption and signature key pairs because of physical implementation, security concerns, Acquirer policy, or a variety of other reasons. For example, a merchant that operates multiple servers may elect to have a separate set of encryption and signature key pairs for each server. In addition, new key pairs shall be generated periodically.

The number of certificates needed by a merchant is a function of the number of merchant's encryption and signature key pairs, the number of payment gateways that interface with the merchant, and the number of brands accepted by the merchant. There are a variety of issues that impact how many payment gateways a merchant will interface with. In the simplest case, the merchant shall interface with a single payment gateway to process all brands. However, a merchant may have relationships with multiple Acquirers. For example, a single Acquirer may not process all the brands the merchant accepts, or the merchant may do business in multiple national markets (and currencies) and have corresponding Acquirer relationships. In addition, Acquirers may choose to operate multiple gateways for load balancing.

SET allows the Acquirer to send cardholder payment information back to the merchant, encrypted under the merchant's key. This capability is designated by an indicator in the merchant's certificate. This option is intended to allow merchants to use out-of-band clearing mechanisms.

Continued on next page

Certificates, continued

Payment Gateway certificates

Two key pairs are required at the Payment Gateway:

- A signature pair that is used to sign and verify messages provided to the cardholder and merchant; and
- An encryption key-exchange key pair that is used to protect payment instructions generated by the cardholder and by the merchant.

Certificate chain validation

Certificates shall be validated through a hierarchy of trust. Each certificate is linked to the signature certificate of the certificate issuing entity. Certificates are validated by following the trust hierarchy to the Root CA. The path through which the certificates are validated is called the 'certificate chain'.

The validation of each certificate shall be enforced at all levels of the chain. For example, a cardholder shall validate the merchant, Merchant CA, Brand Geo-political CA, Brand CA, and Root CA certificates. The validation process may stop at a level that has been previously validated. A detailed description of the certificate chain validation processing is provided in "Certificate Chain Validation" on page 78.

Summary of certificate types

Table 1 below lists all certificates needed in SET:

Certificate Types	Digital Signature	KeyEncryption	Certificate & CRL Signing
Cardholder	X		
Merchant	X	X	
Payment Gateway	X	X	
Cardholder CA	X	X	X
Merchant CA	X	X	X
Payment Gateway CA	X	X	X
Brand Geo-political CA	X		X
Brand CA			X
Root CA			X

Table 1: Summary of Certificate Types

Brand CRL Identifier

Purpose

Each brand is responsible for managing CRLs within its own domain. The SET architecture introduces the concept of a **BrandCRLIdentifier** (BCI). A BCI is digitally signed by the brand and used to identify the SET CRLs that the cardholder, merchant, Payment Gateway, and CA systems need to screen against whenever validating certificates as part of signature verifications. Refer to "Certificate Management" (starting on page 113) for additional details about BCI.

Contents

Each instance of a BCI identifies the brand and the brand's CA subject names that have CRLs that need to be processed when validating signatures in SET messages. Each BCI has a sequence number and validity period.

BCI entity types

Table 2 below lists the types of SET CA entities that may exist on a BCI and the motivation for including each entity.

Entity	Reason for BCI
Root CA	Unscheduled replacement or termination of the Root or brand CA certificates
Any brand CA	Unscheduled replacement or termination of a CA certificate issued by the brand CA
Brand Geo-political CA	Unscheduled replacement or termination of CCA, MCA, or PCA entities
Payment Gateway CA	Unscheduled replacement or termination of Payment Gateway certificates

Table 2: BCI Entity Types

Chapter 3
Technical Requirements

Overview

Introduction

Chapter 3 summarizes other design considerations that affect the overall technical requirements for SET.

Organization

Chapter 3 includes the following sections:

Section 1
Security

Overview

Introduction

The intent of SET is to address certain security issues related to three-party payment mechanisms conducted over the Internet.

Public-key signature mechanisms are critically dependent upon the security of the corresponding private keys. SET requires public/private key pairs for the Payment Gateways and merchants, and supports them as a recommended option for cardholders. Developers shall pay particular attention to the methods used to store the private keys of these participants. The private keys shall be protected through encryption or perhaps using tamper-resistant mechanisms. Payment gateways shall use tamper-resistant hardware cryptographic modules to perform cryptographic functions and for generation and storage of secret keys. Merchant servers and cardholder applications should also employ hardware cryptographic modules to perform cryptographic functions and to generate and store secret keys.

SET operates with public keys that are distributed via certificates signed by well-known Certificate Authorities (CAs). Cardholders, merchants, and payment gateways shall authenticate the Certificate Authorities and root keys recognized by their software using mechanisms provided in SET.

SET offers an option that permits the Payment Gateway to provide cardholder account information to the merchant, encrypted under the merchant's public key. When this option is used, care shall be taken to ensure the security of the payment information as it resides on the merchant's systems. The merchant software shall store payment information in encrypted form. Merchants should also store payment information off-line, or behind a firewall or similar mechanism.

Trusted cache

Certificates, CRLs, and BCIs will be accessed frequently when processing SET messages. Thus, the processing of successive SET messages may be optimized by maintaining a local trusted cache of frequently accessed certificates, CRLs and BCIs. Cardholder and merchant systems supporting SET shall enforce a policy to protect its trusted cache containing certificates, CRLs, and BCIs and their corresponding thumbprints from unauthorized access or modification.

Confidentiality

Definition

Data confidentiality is the protection of sensitive and personal information from unintentional and intentional attacks and disclosure. Securing such data requires data encryption and associated key management in uncontrolled environments, such as unsecured networks.

SET uses both asymmetric and symmetric encryption algorithms in conjunction with a digital envelope to provide data confidentiality. Refer to the SET Business Description (Book 1) for an overview of this technique.

Payment data

SET is responsible for the confidentiality of payment data that it needs to manage. Where non-payment data confidentiality is needed, it is provided in the protocol messages by including a reference to the actual data rather than the data itself. For example, SET does not exchange the Order Description, but includes a hash of the Order description in the Purchase Request (**PReq**).

Other data

Although the confidentiality of the non-payment data is outside the scope of SET, system developers are encouraged to protect this data.

Authentication

Definition

Authentication provides assurance that the data received was in fact sent by the party who claims to have sent it. Thus, the receiver can authenticate the sender by verifying the received data. This is accomplished using digital signatures and public key certificates issued by a CA.

Entity authentication

Digital signatures require a trusted third party to vouch for the authenticity of the public key used to verify the signature. The process dictates that a trusted third party, a Certificate Authority (CA), provides an electronic certificate that vouches for the fact that a public key is "owned" by a certain entity. This electronic certificate (itself digitally signed by the CA) is stored by the entity in their computer. The recipient's system uses the certificate to verify the sender's public key. At that point the recipient is sure that:

- The original data was not altered (data integrity);
- The message could only have been signed by the holder of that private key (entity authentication); and
- A trusted third party has vouched for the fact that the signer is in fact the holder of that key pair.

Therefore, the uniqueness of the digital signature and the underlying hash value coupled with the strength of the public key certificate provides an acceptable level of assurance to authenticate the sender and to verify that the sender was the originator of the signed data.

Cardholder authentication

Merchants and Acquirers shall verify that a cardholder is using a valid account number. Mail and telephone order merchants often go to great lengths to verify the identity of cardholders. Also, unauthorized individuals who have stolen valid payment card account numbers and expiration dates may try to initiate electronic commerce transactions. A mechanism that links a user to a specific account number will reduce the incidence of fraud and therefore the overall cost of payment processing.

The cardholder certificate issued by the CCA is evidence that the cardholder's public key has been tied to the account number.

Continued on next page

Authentication, continued

Merchant authentication

A merchant receives verification of an agreement with the Acquirer through the issuance of a certificate. Acquirers shall authenticate the merchant's certificate request and, if appropriate, issue a certificate through its MCA. This certificate provides assurance that the merchant holds a valid agreement with an Acquirer. In essence, this is an "electronic decal," which is equivalent to the brand decal in the merchant's window.

Cardholders and Payment Gateways shall authenticate merchants by verifying the signatures on the merchant's certificate and by validating the certificate's chain.

Payment Gateway authentication

Payment Gateway certificates are issued by a payment card brand's PCA. Payment card brands shall authenticate the acquirer's certificate request before issuing certificates. These certificates provide assurance that a payment gateway has been authorized by the Brand, Acquirer, or Brand Geo-political Certificate Authority.

Merchants shall authenticate payment gateways by verifying the signatures on the payment gateway's certificate and by validating the certificate's chain.

Since the cardholder uses the Payment Gateway's public key for encrypting the symmetric key used to encrypt the payment instruction, the cardholder system needs the ability to authenticate the Payment Gateway. The merchant provides the cardholder with the Payment Gateway's encryption certificate. The cardholder system shall validate this certificate and thereby be assured that the Payment Gateway is legitimate and that the payment instruction is kept confidential.

Integrity

Definition

Data integrity is the assurance that the data received is in fact the data sent. This is accomplished by an integrity value that is generated using the transmitted data. The data and the integrity value are transmitted from the sender to the receiver. The receiver verifies that the data has not been altered during transmission by validating the integrity value on the data.

Hash functions

Data integrity is supported by using a hash function. A hash function is applied to the appropriate data to produce a statistically unique integrity value called the hash value. The hash functions by themselves do not guarantee absolute data integrity. To provide this guarantee, hash functions need to be combined with a secret quantity or key.

Hash functions are different from symmetric ciphers and have the following properties:

- The hash function is a public algorithm.
- The hash function is one-way, that is, given the hash value, it is not possible to recreate the original data.
- The hash value is computed in such a manner that it is not feasible to identify other data that will hash to the same value.

Digital signature

A digital signature is defined as data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, for example, by the recipient.

In SET's architecture, a digital signature is a hash value encrypted using the private key of the sender. The hash value provides integrity of the data within the message; if the payment data is modified, the hash value will be different, and that difference can be detected when the receiver re-computes the hash. The hash is encrypted to ensure that a third party cannot change the hash, since encryption of the new hash value would not be possible without the private encryption key.

Section 2
Adaptability

Variations

Purpose

This section illustrates how SET's architecture has been designed to be adaptable to different business models and operational environments, such as support for cardholders without certificates. Refer to Appendices D, E, and S for more information about this topic.

SET certificates

The design of SET uses X.509 version 3 certificates to support public keys for signature and encryption. These certificates include a public key together with the authentication of that key.

Use of cardholder certificates

The cardholder's signature certificate provide authentication and integrity of information sent to the merchant and to the Payment Gateway. SET supports environments in which cardholder signature certificates are required, and also environments where cardholder certificates are optional. A payment card brand determines if its application of SET requires signature certificates or not.

Certificate-required environments

In environments where certificates are required, all messages that require authentication and integrity from the cardholder shall be signed with a signature authenticated by the cardholder certificate. There are protocol initiation requests that do not include such signatures, since no significant protocol failures would result from their abuse. All other messages are signed, and the recipients of these messages are assured receipt of the corresponding certificates by the protocol.

Non-certificate support

When a cardholder does not have a signature certificate, no digital signature is generated. In place of the digital signature, the cardholder generates the hash of the data and inserts the hash into the digital envelope to ensure the integrity of its contents.

Section 3
Interoperability

General Message Formats

Overview

SET messages shall be formatted using non-proprietary techniques, permitting communication over a variety of real-time and non-real-time mechanisms. Wherever possible, standards are employed to enable the protocol to be easily implemented and to ensure that interoperability among implementations is possible. Cryptographic treatments are constrained to ensure that only as much cryptography is employed as is required by the security needs of the payment card transaction. To promote interoperability and the ability to upgrade, SET uses the Public Key Cryptography Standards (PKCS) for representing the cryptographic parameters and message encapsulation.

SET messages are defined using the ISO/IEC and ITU-T Abstract Syntax Notation (ASN.1) standard and shall be encoded using the Distinguished Encoding Rules (DER). This permits unambiguous encoding through a well-understood and widely-accepted standard.

SET message transport

The SET specification does not define how a SET message is transported between entities. SET messages may be transported using any mechanism that the sender and receiver agree to. It is expected that transport standards will be developed to address the issue of interoperable SET applications.

SET environments

It is envisioned that SET applications will operate in one of two environments:

- Interactive - in this environment, the entities communicate in "real-time" with small time delays between the exchange of messages (such as the World Wide Web); and
- Non-interactive - in this environment, the entities communicate in non "real-time" with large time delays between the exchange of messages (such as E-Mail).

SET Initiation Process

In an interactive environment, it is expected that a "SET Initiation Process" takes place that triggers the SET protocol. This process will allow the Cardholder and Merchant to exchange certain information required for SET. Such information includes (but is not limited to) the brand the cardholder has selected, the order description, and the purchase amount. It is expected that standards will be developed to address how this information is exchanged and how the SET protocol is initiated.

Continued on next page

General Message Formats, continued

ASN.1/DER encoded messages

The ASN.1 provides a clear, unambiguous definition of the content of messages; DER provides an encoding that is both precise and ensures a single format of the encoded data, which is critical to be able to support operations involving hashes and signatures.

The ASN.1 notation includes a collection of intrinsic types that are used to define SET's data fields and messages but depend upon additional restrictions and constraints. These shall be checked by the application software. For example, **IA5String** is used as the ASN.1 intrinsic type to define several data fields that contain character string data (for example, **MerOrderNum**). The allowable character set supported by the **IA5String** type is sometimes referred to as the ASCII character set. In addition, size constraints on the fields are imposed (for example, **MerOrderNum** may not exceeded 25 bytes) and shall be checked by all SET software.

Commercial ASN.1 code generators are available that will enable software developers to generate and receive these messages with only modest programming effort beyond providing the ASN.1 specification itself to the compiler. See Appendix A: External Standards, for information on specific versions of ASN.1 and DER.

Thumbprints

In order to support the security requirements of SET, public key certificates and CRLs shall be carried in the protocol. Since these data structures are large, a thumbprint mechanism is provided to reduce the required traffic associated with certificates, CRLs and BCIs.

A thumbprint is a hash of the data portion of a certificate, CRL, or BCI. More specifically, it is the SHA-1 hash of the data which is signed in one of the above signed entities. If an entity of SET would normally need a certificate or CRL from another entity with which it is communicating, it maysend the remote entity the set of thumbprints corresponding to the certificates, CRLs, and BCIs that it possesses. Software shall only send thumbprints that it expects to be related to the transaction. For example, merchant software shall not send the thumbprints for other cardholders or for other brands. The responding entity should omit from its response message any certificates and CRLs for which it has received thumbprints. Since the thumbprints are very small compared to the certificates and CRLs that they represent, much overhead is avoided.

MessageWrapper

Purpose

The **MessageWrapper** is the top level ASN.1/DER data structure in the SET protocol. It provides the information presented to the receiver of a message at the very start of message processing, without involving any cryptography. It identifies both the type of message and its unique identifiers, sufficient data to base initial decisions to support duplicate detection, etc. (The notation used in the table below is presented on page 59.)

Field Name	Description
Message-Wrapper	**{MessageHeader, Message, [MWExtensions]}**
MessageHeader	**{Version, Revision, Date, [MessageIDs], [RRPID], SWIdent}**
Version	*Version of SET message.*
Revision	*Revision of SET message.*
Date	*Date/time message generated.*
MessageIDs	***{[LID-C], [LID-M], [XID]}***
RRPID	*Request/response pair ID for this cycle.*
SWIdent	*Identification of the software (vendor and version) initiating the request. This is string data.*
Message	**< PInitReq, PInitRes, PReq, PRes, InqReq, InqRes, AuthReq, AuthRes, AuthRevReq, AuthRevRes, CapReq, CapRes, CapRevReq, CapRevRes, CredReq, CredRes, CredRevReq, CredRevRes, PCertReq, PCertRes, BatchAdminReq, BatchAdminRes, CardCInitReq, CardCInitRes, Me-AqCInitReq, Me-AqCInitRes, RegFormReq, RegFormRes, CertReq, CertRes, CertInqReq, CertInqRes, Error>**

Continued on next page

MessageWrapper, continued

Purpose (continued)

Field Name	Description
LID-C	*Local ID; convenience label generated by and for Cardholder system.*
LID-M	*Local ID; convenience label generated by and for Merchant system.*
XID	*Globally unique ID generated by Merchant (or Cardholder, if there is no* **PInitRes***).*
MWExtensions	*SET Message extensions. An extension would be appropriate in the message wrapper under either of two conditions:* • *The data in the extension is general purpose information about SET messages; or* • *The contents of the message are encrypted and the extension contains non-financial data that does not require confidentiality.* *Note: The message wrapper is not encrypted so this extension shall not contain confidential information.*

Notes

All SET-related processing begins with the **MessageWrapper**. Every SET message contains a cleartext **MessageWrapper**, which shall be decoded before message processing. The **TransIDs** and **RRPID** fields have been placed here to permit early duplicate detection; these fields are repeated within the message, so that data can be integrity protected within the body of the message.

At the time the **MessageWrapper** is decoded, the **Message** component may not be processed, but its type can be determined from the DER type field of **Message**. After the **MessageWrapper** processing is performed, the **Message** is decrypted and/or its signature verified as is appropriate, then the contents is decoded to yield the information which is processed individually for each message type.

Backwards Compatibility

Application requirements

In order for SET to be successful, new versions of SET must be able to interoperate with prior versions. In general, software applications shall interoperate with the current revision of SET and the immediate prior version. That is, an application that supports version 2 of SET (when it is published) shall be able to send and receive version 1 messages. A future version of SET may require compatibility with more than one prior version.

Checking the Version

To check the version of the message, the software shall first check **MessageHeader.Version** and **MessageHeader.Revision**.

Old messages

An application that receives a message from a previous version which it can process, shall respond (if appropriate) using messages and formats from the received version.

An application that receives a message with a version number that is lower than it can process (such as a version 1 application receiving a message with a version of 0) shall reject the message by responding with an error message containing an **ErrorCode** of **versionTooOld**.

Software upgrade prompts

An application that receives an **ErrorCode** of **versionTooOld** should display a message with information about how to upgrade to the latest version of the software. (Cardholder software vendors in particular may want to consider including such a feature.)

New messages

An application that receives a message with a version number that is higher than it can process (such as a version 1.0 application receiving a message with a version of 2) shall reject the message with an **ErrorCode** of **versionTooNew**.

An application that receives an Error message with an **ErrorCode** of **versionTooNew** should try sending the message with a lower version message if that option is available.

Extension Mechanism for SET Messages

Explanation

The scope of this version of SET was intentionally limited to the minimum functionality necessary to support cardholders and merchants doing business on the Internet. Consequently, some business functions are not included in the definition of SET payment messages. Furthermore, it is unlikely that SET could ever be robust enough to cover the business practices of every national market and every acquirer. Therefore, it is necessary to provide a mechanism to extend SET payment messages.

An example of a business function that is not supported by the SET messages is Japanese Payment Options. Issuers in Japan have options for payment that are selected by the consumer at the time of the purchase. Since there is no place in the SET message to carry this information, an extension to the protocol is necessary.

Mechanism

The mechanism used to extend SET messages parallels the way that X.509 certificates are extended. Specifically, an extensions field is provided which contains a sequence of extension data. The extension data indicates the type of extension and the criticality of the extension. See Appendix H: Extension Mechanism for SET Messages for details.

PKCS #7 Formats

Purpose

To ensure interoperability and the ability to upgrade, the Public-Key Cryptography Standards (PKCS) #7, Cryptographic Message Syntax Standard, is used as the basis for the cryptographic encapsulation methods used in SET messages. Review the PKCS #7 documentation cited in "Related documentation" on page iii.

Benefits

PKCS #7 formats are used to represent the enveloped data in SET messages. ASN.1 and its encoding rules, a set of international standards, are used throughout the PKCS #7 specification. By using ASN.1 to define the SET messages, one format is used throughout the entire SET specification.

PKCS #7 methods

SET uses the following PKCS #7 data encapsulation methods:

- **SignedData**, for signed data encapsulation,
- **EnvelopedData**, for encrypted data encapsulation,
- **EncryptedData**, for encrypted data encapsulation with symmetric keys,
- **DigestedData**, for hashed (or linked) data encapsulation.

Implicit certificates and CRLs

Signed messages contain all certificates and CRLs necessary for the recipient to verify their signatures; the recipient can indicate certificates that it has previously validated and cached using thumbprints in the corresponding request message.

CRLs and signature certificates are implicit in signed message types. Following PKCS #7, these are contained in the **Certificates** field of the **SignedData** type. Furthermore, SET includes key-exchange certificates in **SignedData** blocks; these are implicit in the protocol. Language in PKCS #7 expressly allows this usage.

In the cases where certificates or CRLs require authentication of origin or integrity protection but are not encapsulated in **EnvelopedData**, they are explicitly defined in the message protocol as specified in Book 3: Formal Protocol Definition. In this case, they are transported in recursively-encapsulated PKCS #7 **SignedData** types.

Continued on next page

PKCS #7 Formats, continued

SignedData

The **SignedData** type from PKCS #7 is shown below to aid in defining the signature process. Multiple occurrences of **SignerInfos** are permitted within **SignedData**; however, a **SignedData** message is signed by no more than two parties in SET.

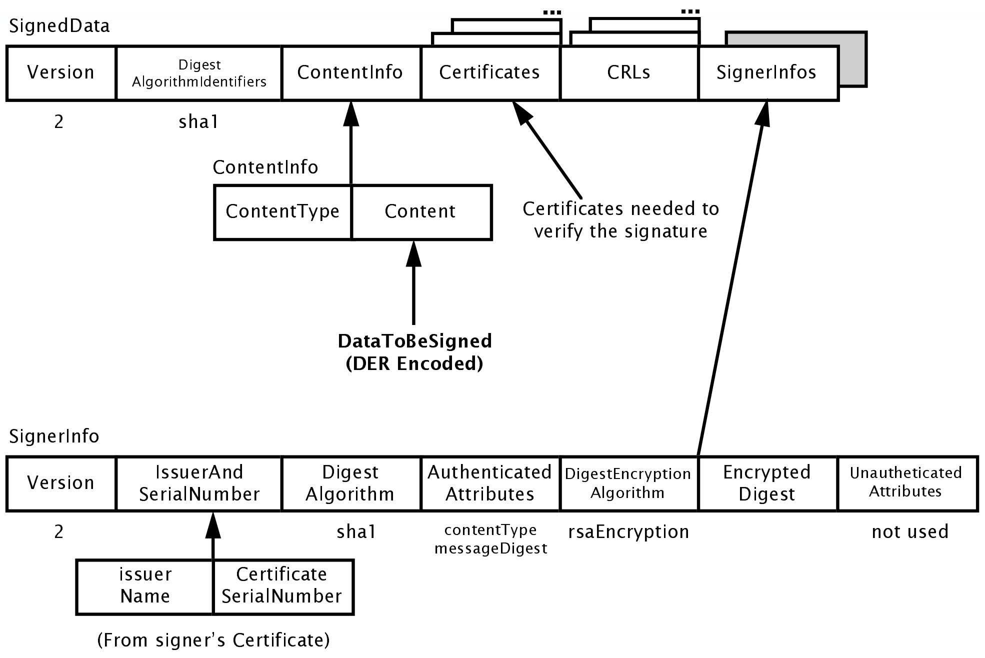


Figure 4: SignedData

For further detail

Appendix M: ContentTypes provides a table of SET messages (or components of messages) with their **content** and **contentType** for **SignedData**.

Continued on next page

PKCS #7 Formats, continued

Authenticated attributes

The type of content being signed shall be protected indirectly in the signature, by including the PKCS #9 **contentType** attribute in the content to be signed. A digest of the data being signed is also included in the PKCS #9 **messageDigest** attribute. SET PKCS #7 **SignedData** always includes these two authenticated attributes: **contentType** and **messageDigest**.

Object identifiers for the **contentType** attribute are defined to uniquely identify each of the SET ASN.1 types that can appear in **SignedData**.

Example

Consider the signature on an ASN.1 type named *Stuff*. The SHA-1 digest of the DER encoding of this type is computed. An authenticated attributes data structure is computed by placing the object identifier *id-set-stuff* into **contentType** attribute and the digest into **messageDigest** attribute as shown in the following table.

contentType	id-set-stuff
messageDigest	SHA-1(*Stuff*)

The SHA-1 digest of this data structure is computed and the result encrypted using the signer's private key; it is this encrypted digest that is placed in the **EncryptedDigest** field of the **SignedData** structure.

The object identifier *id-set-stuff* identifies the content, that is, what piece of data is authenticated.

Continued on next page

PKCS #7 Formats, continued

Enveloped Data

The **EnvelopedData** type from PKCS #7 is shown below to aid in defining the encryption process. Multiple occurrences of **RecipientInfos** are permitted within PKCS #7 **EnvelopedData**; however, only one **RecipientInfo** is used in SET messages.

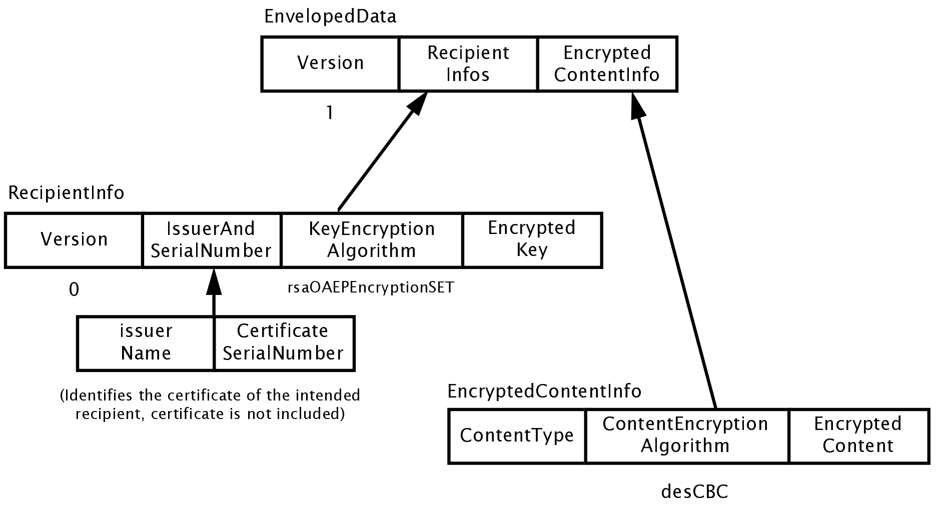


Figure 5: EnvelopedData

For further detail

Appendix M: ContentTypes provides a table of SET messages (or components of messages) with their **content** and **contentType** for **EnvelopedData**.

Continued on next page

PKCS #7 Formats, continued

EncryptedData

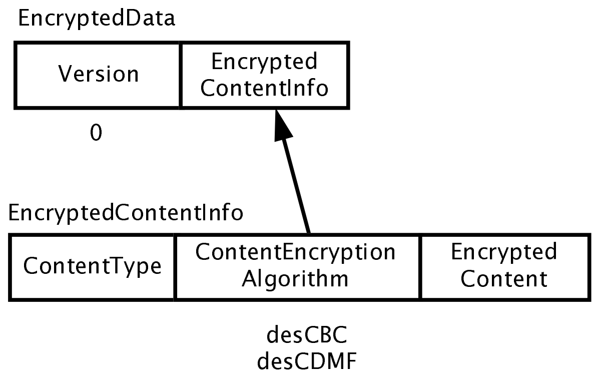


Figure 6: EncryptedData

DigestedData

The **DigestedData** construct from PKCS #7 is shown below to aid in defining the hashing process.

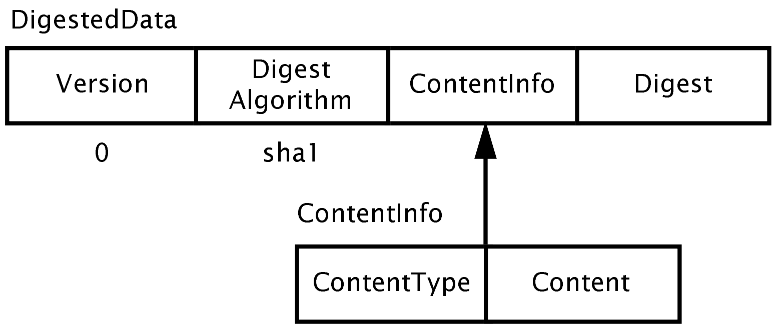


Figure 7: DigestedData

For further detail

Appendix M: ContentTypes provides a table of SET messages (or components of messages) with their **content** and **contentType** for digested data.

Transaction Validation by non-SET systems

Explanation

The evidence that a cardholder has participated in a SET transaction is provided by the cardholder's digital signature and certificate. However, once the payment gateway has processed the transaction and formatted it for use by the legacy payment systems, this evidence is lost. Consequently, there is no means for the issuer to validate the cardholder's participation in the transaction. However, the issuer can verify a hash of data known only to the cardholder and the issuer.

During the certificate registration process, the cardholder sends a secret value to the CA. This secret value, named **cardSecret**, is combined with another secret value generated by the CA to create the **Secret**. By having the cardholder and CA remember the value supplied by the cardholder, a secret exists that can be used by the cardholder to create a hash that only the cardholder can generate and that only the issuer can verify.

The cardholder software shall generate a hash and include it in the payment instructions. The hash is defined as **HMAC(XID, CardSecret)**. Because the hash includes the SET transaction identifier, the value changes for every transaction preventing replay of the hash on future non-SET transactions.

Chapter 4
System Concepts

Overview

Introduction

Chapter 4 summarizes the other important system concepts pertinent to understanding SET's architecture.

Organization

Chapter 4 includes the following sections:

Section	Title	Contents	Page
1	Notation and Definitions	Summarizes the notation and conventions used throughout the remainder of this Programmer's Guide.	58
2	Cryptography	Highlights the specific cryptographic algorithms and features.	64
3	Other Features	Describes other features of SET's design.	69
4	Processing	Describes the step-by-step processing guidelines for cryptographic functions used by the payment and certificate management protocol.	75

Section 1
Notation and Definitions

Overview

Purpose

This section provides a high-level overview of the notation and fundamental cryptographic treatments that are used to describe the payment and certificate processing flows in this Programmer's Guide.

Notation

Purpose

The remainder of this book makes use of the abstract notation described below.

Concept	Notation	Definition	
Tuple	**{A, B, C}**	A grouping of zero or more data elements. These represent *documents* or *messages*, terms occasionally used interchangeably with "tuple." Tuples are denoted by *identifiers:* alphanumeric symbols. This notation means "the tuple containing ***A***, ***B***, and ***C***," which may, themselves, be tuples.	
Component	**T = {A, B, C}**	A tuple may be given a name as shown, in which case ***T.A***, ***T.B***, and ***T.C*** refer to the respective *components* of ***T***.	
Ordered concatenation	**A | B | C**	This notation means that an explicit, *ordered concatenation* of items ***A***, ***B***, and ***C*** is needed.	
Optional	**[A]**	This notation means that item ***A*** is *optional*.	Any other nesting of these brackets is permissible.
Selection	**<A, B, C>**	This notation means that exactly one of ***A***, ***B***, and ***C*** must appear. This is a *selection* notation.	
Optional selection	**[<A, B, C>]**	This notation means that the *selection* is *optional*; that is, that either nothing or exactly one of ***A***, ***B***, and ***C*** may appear.	
Multiple instances	**{A +}**	This notation means a tuple containing *one or more instances* of ***A.***	
	{A *}	This notation means a tuple containing *zero or more instances* of ***A***.	
	{[A] +}	This notation means a tuple containing: • *one or more instances* of **A.** • in an *ordered array*. • where *each instance* of **A** *is optional* (that is, may be null).	
Exclusive-or	⊕	This symbol denotes a bit-wise *exclusive-or* (XOR) operation.	

Table 3: Notation

Cryptographic Treatments

Caveat

The following tables introduce the notations for the hashing, signature, dual signature, encryption, and encapsulation cryptographic treatments on the data elements which are used throughout the remainder of this Programmer's Guide. Review Book 3: Formal Protocol Definition for additional information.

Hashing

The notation corresponding to the hashing and hashed-based operators used by SET is summarized in the following table.

Notation	Operator	Description
H(*t*)	*Hash*	160-bit SHA-1 hash of tuple ***t**.*
HMAC(*t, k*)	*Keyed-hash mechanism*	160-bit keyed-hash of tuple ***t*** using *key* ***k*** based on HMAC-SHA-1 HMAC(t,k) = H((k ⊕ opad) | H((k ⊕ ipad) | t)) where: **ipad** is the byte 0x36 repeated 64 times; **opad** is the byte 0x5C repeated 64 times; and ⊕ is the XOR function.
L(*t1, t2)*	*Linkage*	A reference, pointer, or *link* to ***t2*** is included with ***t1***; equivalent to the tuple ***{t1, H(t2)}***

Continued on next page

Cryptographic Treatments, continued

Signature

The notation corresponding to the signature operators used by SET is summarized in the following table.

Notation	Operator	Description
S(*s*, *t*)	*Signed Message*	The signature of entity ***s*** on tuple ***t***, including the plaintext of ***t***. SET's default signature algorithm is RSA with SHA-1 hash. Corresponds to PKCS #7 **SignedData**.

Signature Only

The notation corresponding to the dual signature operators used by SET is summarized in the following table.

Notation	Operator	Description
SO(s, *t*)	*Signature Only*	The signature of entity ***s*** on tuple ***t***, but not including the plaintext of ***t***. **SO** corresponds to a PKCS #7 *external signature*. SET's default signature algorithm is RSA with SHA-1 digest.

Continued on next page

Cryptographic Treatments, continued

Encryption

The notation corresponding to the encryption operators used by SET is summarized in the following table.

Notation	Operator	Description
E(*r, t*)	*Asymmetric Encryption (Digital Envelope)*	First, encrypt ***t*** with a fresh symmetric key, **k**, then, insert **k** in an PKCS #7 envelope for entity ***r*** under OAEP; that is, encrypt **OAEP(k)** using the public key of entity ***r***, taken from the certificate in tuple ***r***. DES is the default symmetric encryption algorithm. Corresponds to the standard PKCS #7 **EnvelopedData**.
EH(*r, t*)	*Integrity Encryption*	This is like **E** except that the PKCS #7 envelope contains **OAEP({k, H(*t*)})** for a guarantee of integrity when signature is not available. Processing software shall rehash ***t*** and check for match against the **H(*t*)** in the PKCS #7 envelope.
EX(*r, t, p*)	*Extra Encryption*	This is like **E** except that ***t*** and ***p*** are the parts of a two-part message; ***t*** is the tuple to be linked to ***p*** and subjected to ordinary, symmetric encryption, and ***p*** is a *parameter*, or the part to be subject to "extra" processing. ***t*** is linked to ***p***. **OAEP({k, *p*})** is inserted in the PKCS #7 envelope for entity ***r***. The ***t*** slot is called the *ordinary slot* of **EX**, and the ***p*** slot is called the *extra slot* of **EX**.
EXH(*r, t, p*)	*Extra Encryption with Integrity*	This is like **EX** except with **OAEP({k, H(*t*), *p*})** in the PKCS #7 envelope and with the requirement that processing software check **H(*t*)**, as with **EH.**
EK(*k, t*)	*Symmetric Encryption with a provided key, **k***	The symmetric encryption of tuple ***t*** using secret key ***k***. Corresponds to an instance of PKCS #7 **EncryptedData**.

Continued on next page

Cryptographic Treatments, continued

Encapsulation

The notation corresponding to the encapsulation operators used by SET is summarized in the following table. These operators combine signature and encryption operators and are used on most messages, facilitating security analysis of this protocol.

Notation	Operator	Description
Enc(*s*, *r*, *t*)	*Simple Encapsulation with Signature*	Signed, then encrypted message. Corresponds to an instance of PKCS #7 **SignedData** encapsulated in **EnvelopedData.**
EncK(*k*, *s*, *t*)	*Simple Encapsulation with Signature and a Provided Key*	Signed messages encrypted with a known, secret key. Corresponds to an instance of PKCS #7 **EncryptedData**.
EncX(*s*, *r*, *t*, *p*)	*Extra Encapsulation with Signature*	Two-part messages encrypted with the first part of the message in the ordinary (symmetric encryption) slot of **E** and the second part of the message in the extra (OAEP) slot of **E**.
EncB(*s*, *r*, *t*, *b*)	*Simple Encapsulation with Signature and Baggage*	Signed, encrypted messages with external baggage.
EncBX(*s*, *r*, *t*, *b*, *p*)	*Extra Encapsulation with Signature and Baggage*	Signed, **E**-encrypted, two-part messages with baggage.

Section 2
Cryptography

Cryptographic Features

Asymmetric key algorithms

SET uses asymmetric public-key encryption algorithms for digital signatures and digital envelopes.

Asymmetric key sizes

The RSA public-key algorithm is used for all public key operations and certificates, with the following key sizes. The following sizes satisfy the export regulations but are subject to change.

Entity	Message Signature	Key-Exchange	Certificate Signing	CRL Signing
Cardholder	1024			
Merchant	1024	1024		
Payment Gateway	1024	1024		
Cardholder CA	1024	1024	1024	
Merchant CA	1024	1024	1024	
Payment Gateway CA	1024	1024	1024	1024
Brand Geo-political CA			1024	1024
Brand CA			1024	1024
Root CA			2048	2048

Table 4: Asymmetric Key Sizes

Note: These key sizes may change in the future; however, because of import and export restrictions, SET applications shall hard-code these sizes. Application updates will be necessary when these key sizes change.

Symmetric key algorithms

The Data Encryption Standard (DES) is the default symmetric key algorithm used in SET to protect sensitive financial data (for example, payment instructions).

Commercial Data Masking Facility (CDMF) is another symmetric key algorithm used for protecting Acquirer-to-cardholder messages.

Continued on next page

Cryptographic Features, continued

DES

DES is the default symmetric data encryption algorithm used for protecting the financial information. Originally published in 1977 for use by the United States government to protect valuable and sensitive, but unclassified data, this standard was subsequently adopted by the American National Standards Institute (ANSI) as the Data Encryption Algorithm (DEA).

DES specifies a cryptographic algorithm to encrypt and decrypt 64-bit blocks of data under the control of a unique key. The algorithm is defined in Federal Information Processing Standard (FIPS) 46-2, published by the U.S. National Institute of Standards and Technology (NIST). SET uses the Cipher Block Chaining (CBC) mode of DES, as defined in FIPS 81. The key is 8 bytes long, with each byte having a parity bit in position 0 yielding an effective key length of 56 bits. The standard padding rule shall be used with the DES-CBC mode as described below.

SET DES-CBCPadding Rule

The SET padding rule for DES-CBC requires that a padding string always be appended to the final plaintext block being encrypted. This final block may be a complete data block, or a partial data block whose length is not an integral multiple of the block length. A padding string is used in SET regardless of whether the final block is a partial or complete data block

The padding string appended to the final data block makes its length an integral multiple of eight octets. If *BL* represents the length in octets of the final data block, then the padding string consists of 8 - (|| BL || mod 8) octets. Each octet in the padding string has as its value 8 - (|| BL || mod 8).

When the length of the padding string is a single octet, the value of that octet is 01. When the length of the string is two octets, the value of the two octets is 02, and the padding string used is '0202'. When the length is three, the value is 03, and the padding string is '030303', and so on.

Continued on next page

Cryptographic Features, continued

CDMF

CDMF[1] algorithm is the symmetric algorithm for providing data confidentiality intended primarily for tunneling Acquirer-to-cardholder information through the merchant. CDMF is a scrambling technique that relies upon DES as the underlying cryptographic algorithm, but weakens the overall cryptographic operation by defining a key-transformation method that produces the equivalent of a 40-bit DES key instead of the 56-bit key length required for full strength DES. Since the CDMF algorithm is not as resistant to key exhaustion as DES, CDMF provides a form of data masking rather than data encryption.

The CDMF key transmitted in the SET protocol is the key before being transformed for use in a DES encryption/decryption engine. In other words, a CDMF key is treated just like a normal DES key.

Hashing algorithm

Secure Hash Algorithm (SHA-1) shall be used for all hashes in this version of SET, including the hashes used in signatures. All references to hash algorithms shall be interpreted as using the SHA-1 hash algorithm defined in FIPS 180-1. The keyed-hash mechanism (HMAC) shall also use SHA-1 instead ofMD5.

Digital envelope

A digital envelope is a generic cryptographic technique to encrypt data and to send the encryption key along with the data. Generally, a symmetric algorithm is used to encrypt the data, and an asymmetric algorithm is used to encrypt the encryption key.

OAEP

SET uses the Bellare-Rogaway Optimal Asymmetric Encryption Padding (OAEP) method in conjunction with its cryptographic encapsulation operators. In addition, SET uses the hashed data technique developed by Matyas and Johnson as an enhancement to the basic Bellare-Rogaway construction. Although OAEP is not directly related to the digital enveloping process, SET toolkits and applications shall apply OAEP prior to encrypting the DES key and optional data using the public key of the receiver.

[1] Additional information on CDMF is provided in the following paper: "Design of the Commercial Data Masking Facility Data Privacy Algorithm", D. Johnson, S. Matyas, A. Le, J. Wilkins; *Proceedings of the First ACM Conference on Communications and Computer Security*; ACM Press, Fairfax, VA; 1993

Other Cryptographic Implications

Randomness

An area of special consideration for developers of SET toolkits and applications is the implementation of random number generation used for keys and nonces. Although a precise definition of randomness is outside the scope of the SET specification, developers of products need to be cognizant of the importance of this aspect in their implementation. Poor key generation and seeding methods due to using weak random numbers are common downfalls of cryptographic implementations. The reader is encouraged to use the recommendations provided in the reference cited in the preface for sources of randomness and mixing functions (that is, *RFC 1750, Randomness Recommendations for Security, D. Eastlake, S. Crocker, J. Schiller, December 1994*).

For cryptographic purposes, once a strong seed is collected, it shall either be used one time only or it shall be used exclusively in a cryptographically secure random number generator. Also, each instance of random number generation algorithm shall have its own independent key-generation seed.

Statistically unique field values

SET defines several field values as "statistically unique". This means that statistically, the odds are extremely small that any two SET applications will randomly generate the same value.

Nonce

SET defines several fields as "nonces", "salts" or "freshness challenges" to defeat "playback" attacks. The sending entity shall generate a random value and insert this value into the message. The recipient of the message shall copy this value into the corresponding response message.

Algorithm independence

Although this version of the SET specification is explicit about the cryptographic algorithms that shall be supported by cardholder, merchant, and payment gateway systems, the protocol's cryptographic encapsulation operators have been designed to be algorithm independent. All ASN.1 algorithm information object sets are coded with the extension marker (…) to allow additional algorithm objects to be added to future versions of the specification, while remaining backward compatible with this version of SET. The symmetric algorithms for protecting Acquirer-to-cardholder messages is another example of how SET will be moving towards this long-range objective.

Continued on next page

Other Cryptographic Implications, continued

Hardware tokens

Depending on the policies established by the Acquirer and brand, hardware tokens may also be used by systems supporting SET. A hardware token is defined as a hardware cryptographic module which does not allow disclosure of the private key. It is anticipated that hardware tokens may be integrated with systems that depend on a higher level of trust assurance, such as the Payment Gateway or CA.

Performing cryptographic functions in hardware tokens:

- CAs shall use hardware tokens for all private key operations.
- Payment Gateways shall support the use of hardware tokens; their use may be mandated by acquirer or brand policy.
- Merchants should support the use of hardware tokens; their use may be mandated by acquirer or brand policy.
- Cardholders may support the use of hardware tokens.

Section 3
Other Features

Idempotency

Definition

When an operation can be executed any number of times, with no harm done, it is said to be *idempotent*. From SET perspective, idempotency is a property of how a recipient responds to a message.

Any request in SET that does not receive a response shall be resent since it is impossible for the sender to know if the request or response was lost. The re-transmitted message shall be bit-wise identical to the original request message. In general, a duplicate message is not an error condition.

Rationale

The SET protocol is designed to work in environments where message delivery is not guaranteed. If a SET application does not receive a response in a reasonable period of time (as defined by the application or possibly in response to a user query), it re-sends the message. When the receiving SET application determines that it has previously processed the same message, it retrieves the previous response and sends that previous response again.

When the sender of a message does not receive a response, it is impossible to determine if the request was lost or the response was lost. To further exacerbate this condition, it is quite possible that the original request may have been simply delayed somewhere in the network then eventually processed just prior to the re-transmitted request being received. Therefore the protocol allows the sender to repeat the request with a guarantee that the outcome shall be the same regardless of whether the request was lost or the response was lost.

Not all SET messages require idempotency. The purchase request does require idempotency. On the other hand, the inquiry request, for example, has been designed to be sent at any time so it is not necessary for a merchant to store every inquiry request to determine if a duplicate is received; it simply returns the current status of the transaction in the inquiry response. A summary of the per-message idempotency requirements is provided in Appendix C: SET Messages.

Continued on next page

Idempotency, continued

Description

SET products shall guarantee idempotency of the protocol by examining transaction (**XID**) and request/response pair (**RRPID**) identifiers. For example, a payment gateway will reject attempts to replay authorization requests from merchants. It will detect these attempts by examining the **RRPID** of the authorization request and **XID** of the embedded payment instruction, separately signed (or hashed) and encrypted by the cardholder.

Exceptions

If a SET application detects that it is being subjected to a malicious flooding or spamming attack involving one or more idempotent SET messages types, it is not necessary to respond to these messages under this situation.

Special Field Types

XID	**XID** is intended as a statistically unique identifier assigned to a payment transaction so that all messages of the transaction can be related to one another. It is a 20-byte string.
BrandID	**BrandID** is an important field used in both the payment and certificate management protocol messages. It consists of a "brand name" and an optional "product." • The "brand name" corresponds to the brand of the payment card. • The "product" defines the type of product within the specific brand. When "product" is included, it is separated from the "brand name" by a colon (:) as follows: "*brand[:product]*"

Root Public Key Distribution

Significance of root CA certificate

The security of the SET system depends ultimately on the authenticity of the certificates used in the system. These certificates are verified by checking a chain of certificates, with the final certificate in the chain being a single system-wide End Entity. Only through trust in the Root certificate will trust in the SET system be maintained.

Initial distribution of root key

The root public key is initially distributed as a certificate with the SET software. This certificate shall also contain a hash of the next root public key. The initial distribution of the Root certificate shall be self-signed and shall be verified by an out-of-band mechanism. The chaining process for the Root certificates is based on hash values rather than the distinguished name and serial number of the previous Root certificate.

If the next root key/certificate is not the one represented by the hash within the previous certificate, it shall be treated like the initial Root certificate and requires out of band verification.

Root key update

The root key may be updated implicitly using the SET protocol. This is described in detail in "Root Certificate Update" on page 133.

Off-line Certificates

Certificate provision off-line

In the case of catalog orders, such as those envisioned with CD-ROM shopping, abbreviated protocols may be used that omit the initialization phase between the cardholder and merchant. During this phase, the merchant determines which certificates the cardholder already possesses and sends the cardholder any missing certificates. With abbreviated protocols expected in catalog shopping, these certificates shall be delivered off-line (for example, in the CD-ROM catalog).

Cert-PE

Definition

Cert-PE is the certificate generated by the Payment Gateway Certificate Authority (PCA) binding the Payment Gateway to the proposed encryption public key provided in a certificate request (**CertReq**) message. The **certThumbs** will include the thumbprint corresponding to **Cert-PE**. Although the **Cert-PE** does not appear explicitly in any SET message, it is an optional certificate that may be included in the PKCS #7 **SignedData** block of the corresponding certificate response (**CertRes**) message.

Section 4
Processing

Overview

Purpose

This section provides a high-level overview of the step-by-step processing of common cryptographic treatments that are used by the payment and certificate management protocol descriptions in this Programmer's Guide.

Summary

The following treatments and operators are included:

Processing Descriptions	Page
Send Message	76
Receive Message	77
Certificate Chain Validation	78
Thumbprints	79
Simple Encapsulation with Signature (Enc)	81
Simple Encapsulation with Signature and a Provided Key (EncK)	82
Extra Encapsulation with Signature (EncX)	83
Simple Encapsulation with Signature and Baggage (EncB)	84
Extra Encapsulation with Signature and Baggage (EncBX)	85
Asymmetric Encryption (E)	86
Asymmetric Encryption with Integrity (EH)	87
Extra Asymmetric Encryption (EX)	88
Extra Asymmetric Encryption with Integrity (EXH)	90
Symmetric Encryption (EK)	92
Signature (S)	93
Signature Only (SO)	95
Keyed-Hash (HMAC)	96
Hash (H)	97
DigestedData (DD)	98
Linkage (L)	99
Optimal Asymmetric Encryption Padding (OAEP)	100

Send Message

Sending entity

The sending entity shall ensure that the message contents have been properly formatted and encapsulated based on the message type. Additional data such as certificates, CRLs, and BrandCRLIdentifiers shall be included if any portion of the message is being signed by the sending entity.

Compose message wrapper

SET applications shall implement this procedure, or functionally equivalent procedures, for all messages sent. It represents the standard processing required each time a message is sent. Note that the cryptography will be different based on the type of encryption required and whether the message is signed; this is specified for each message. The encryption and signature procedures are described in the "Cryptographic Treatments" (starting on page 60) and in Book 3: Formal Protocol Definition.

Step	Action
1	Generate the SET message as appropriate. *Note: Enveloped messages should contain all certificates and CRLs needed by receiving party in the PKCS 7 envelope.*
2	Insert the current version and revision numbers into **MessageWrapper** (currently 1 and 0 respectively).
3	Insert **Date** (including time). Note: **Date** must be accurate to ensure receiving entity is able to correctly age messages.
4	Populate **MessageIDs** from fields in **TransIDs** in **Message**. If there are no **MessageIDs** in **Message** (for example, certificate messages), then omit this field.
5	Insert **RRPID**. If this is a request, the **RRPID** shall be generated, and saved to compare to the response. If this is a response message, the **RRPID** shall be copied from the request.
6	Insert **SWIdent**. This is a string that identifies the vendor and the version of the vendor software.
7	Insert **Message** (as an ASN.1 open type).
8	DER encode the wrapped message.
9	Pass the message from step 8 to the transport mechanism. Depending on the transport mechanism, the message may be further wrapped (such as, with a MIME or HTTP header).

Receive Message

Receiving entity

The receiving entity shall ensure that the message contents have been properly formatted and encapsulated based on the message type. Additional data such as certificates, CRLs, and BrandCRLIdentifiers shall be extracted from the message to authenticate any digital signatures applied by the sending entity. The receiving entity's system cache should be updated to reflect these new certificates, CRLs, and BrandCRLIdentifiers.

Step	Action
1	If the transport mechanism wraps a SET message before transmitting it, remove the wrapper as required by the transport mechanism.
2	Validate the format and content of the message wrapper fields: version, revision, date/time, and message type. If any failure: A. Return an Error with ErrorCode set to appropriate error. B. Stop processing message.
3	Using RRPID , compare and update the system's log for duplicate message and handle in accordance with brand's operating guidelines.
4	DER decode the message.
5	If the message contains **SignedData**, then perform the following: A. Update the system's cache with any CRLs received. B. For each certificate received, perform the *Certificate Chain Validation* processing. C. Verify the signature of the message.
6	If the message contains encapsulated data, perform inverse encapsulation operation (decryption), according to the type of encapsulation on the contents of the message, including Step 6 above, if the encapsulated data contains **SignedData**.
7	Extract any **BrandCRLIdentifier**s included with the message and update system's cache, check that all CRLs identified on the BCI are in the system's cache; otherwise abort processing the message.
8	Process the message.
9	Update the system log to reflect the state of this transaction.

Certificate Chain Validation

Processing

The validation of the certificate chain requires that each certificate in the path is verified and that each certificate correctly maps to the CA that issued the certificate. The validation procedures shall be enforced for all levels of the chain. For example, a Cardholder application shall validate the Merchant, Merchant CA, Brand CA, and Root CA certificates and related payment card brands. The validation process is comprised of the following components:

- X.509 certificate validation
- SET certificate validation
- Certificate Revocation List (CRL) processing
- BrandCRLIdentifier (BCI) processing

In practice, it is assumed that the validation process will stop at a level that has been previously validated. All SET software shall validate certificate dates as part of the certificate chain validation process. SET software shall provide a warning mechanism for expiring certificates to prevent their attempted use after expiration.

Step	Action
1.	Validate each certificate in the chain according to the rules specified in Section 12.4.3 of X.509 and using the SET chain validation steps specified in Part 2 starting on page 124.
2.	Verify that the certificate extensions KeyUsage, CertificatePolicies, PrivateKeyUsage, and AuthorityKeyIdentifier are being used in accordance with X.509.
3.	If a new BCI was received: a. Validate its signature using the Brand CA CRL signing certificate. b. Verify that the BrandName in the BCI matches that in the certificate chain being validated. c. Verify that the NotAfter date is less than the present date. d. Check the SequenceNum. If it's greater than the SequenceNum in the BCI cache, store the BCI and verify that all CRLs contained on the BCI are held in the CRL cache. Store any CRLs that are on the BCI but are not already in the cache.
4.	For each new CRL that was received, perform CRL validation as described in Part 2 starting on page 248.
5.	Check each certificate against the Signing CA's CRL as specified on page 206.

Thumbprints

Thumb generation

Thumbprints are generated as follows:

Thumbprints are computed by performing the SHA-1 hash of the following DER encoded ASN.1 structures:

- UnsignedCertificate
- UnsignedCertificateRevocationList
- UnsignedBrandCRLIdentifier

The hash is computed over the tag-length-value of the encoded structure. The Thumbprint is the same hash that is used to sign or verify a certificate or CRL or BCI.

Sending entity

Thumbprints are sent by an entity in a SET request message and can always be ignored by the corresponding recipient. The sending entity is not required to send all thumbprints for all certificates, CRLs and BrandCRLIdentifiers currently existing in its cache, but only those that are pertinent to a particular request/response message pair. For example, merchant software will not need to send the thumbprints for other cardholders or for other brands. The thumbprints may be listed in any order.

Step	Action
1	Initialize the buffer for storing thumbprints.
2	Append the thumbprint (hash) corresponding to: • For each certificate that exists in the sending system's cache that is pertinent for processing the response message and for validating the certificate chain, append the thumbprint (hash) corresponding to this certificate. • For each CRL that exists in the sending system's cache which is pertinent for processing the response message and for validating the certificate chain, append the thumbprint (hash) corresponding to this CRL. • For each BrandCRLIdentifier that exists in the sending systems' cache which is pertinent for processing the response message and for validating the certificate chain, append the thumbprint (hash) corresponding to this BrandCRLIdentifier.
3	Return result from step 2.

Continued on next page

Thumbprints, continued

Receiving entity

The recipient shall ensure that the message originator possesses all certificates, CRLs, and the BrandCRLIdentifier needed to complete the processing of the message. The recipient may choose to ignore the thumbprints and send this information to the requester.

Step	Action
1	Initialize the buffer for storing thumbprints.
2	For each: • certificate that is pertinent for processing the response message or for validating the certificate chain, check if the certificate's thumbprint (hash) matches one of the thumbprints received in the request message. If the thumbprint matches, certificate exists in remote system's cache and need not be sent with the response message. If the thumbprint does not match or the list is empty, then include the certificate in the response message. • CRL that is pertinent for processing the response message or for validating the certificate chain, check if the CRL's thumbprint (hash) matches one of the thumbprints received in the request message. If the thumbprint matches, CRL exists in remote system's cache and need not be sent with the response message. If the thumbprint does not match or the list is empty, then include the CRL in the response message. • BrandCRLIdentifier that is pertinent for processing the response message or for validating the certificate chain, check if the CRL's thumbprint (hash) matches one of the thumbprints received in the request message. If the thumbprint matches, CRL exists in remote system's cache and need not be sent with the response message. If the thumbprint does not match or the list is empty, then include the CRL in the response message.
3	Return results from step 2 with list of certificates, CRLs and BrandCRLIdentifiers to be transferred with response message.

Simple Encapsulation with Signature

Enc

The simple encapsulation with signature operator, Enc(s, r, t), models signed then encrypted messages. It corresponds to an instance of PKCS #7 **SignedData** encapsulated in **EnvelopedData**.

Step	Action
1	Using the *Signature* operator, sign the contents of tuple *t* using the private key for entity *s.*
2	Append the results of step 1 to the tuple *t.*
3	Using the *Asymmetric Encryption* operator, encrypt the result from step 2 using asymmetric public key of recipient, *r*.
4	Return result from step 3.

Simple Encapsulation with Signature and Provided Key

EncK

The simple encapsulation with signature and provided key operator, EncK(k, s, t), models signed messages encrypted with a known, shared, secret key provided by the sender of a prior message.

Step	Action
1	Using the *Signature* operator, sign the contents of tuple *t* using the private key for entity *s.*
2	Append the results of step 1 to the tuple *t.*
3	Using the *Symmetric Encryption* operator, encrypt the result from step 2 using symmetric secret key, *k.*
4	Return result from step 3.

Extra Encapsulation with Signature

EncX

The extra encapsulation with signature, EncX(s, r, t, p), models two-part messages encrypted with the first part of the message in the ordinary slot and the second part of the message in the extra slot.

Step	Action
1	Append the contents of parameter *p* to the contents of tuple *t*.
2	Using the *Signature* operator, sign the results of step 1 using the private key for entity *s*.
3	Append the results of step 1 to the tuple *t*.
4	Using the *Extra Asymmetric Encryption* operator, store parameter *p* in the extra slot of OAEP and encrypt the result from step 3 using the asymmetric public key of recipient, *r*.
5	Return result from step 4.

Simple Encapsulation with Signature and Baggage

EncB

The simple encapsulation with signature and baggage, EncB(s, r, t, b), models signed, encrypted messages with external baggage.

Step	Action
1	Compute SHA-1 hash of baggage, *b*.
2	Link tuple *t* with baggage *b* by appending results from step 1 to *t*.
3	Using the *Signature* operator, sign the results of step 2 using the private key for entity *s*.
4	Append the results of step 3 to the results of step 2.
5	Using the *Asymmetric Encryption* operator, encrypt the result from step 4 using asymmetric public key of recipient, *r*.
6	Append the baggage *b* to the results of step 5.
7	Return result from step 6.

Extra Encapsulation with Signature and Baggage

EncBX

The extra encapsulation with signature and baggage, EncBX(s, r, t, b, p), models signed, "extra" encrypted, two-part messages with external baggage.

Step	Action
1	Compute SHA-1 hash of baggage, *b*.
2	Link tuple *t* with baggage *b* by appending results from step 1 to *t*.
3	Append the contents of parameter *p* to the results of step 2.
4	Using the *Signature* operator, sign the results of step 3 using the private key for entity *s*.
5	Append the results of step 4 to the results of step 2.
6	Using the *Extra Asymmetric Encryption* operator, store parameter *p* in the extra slot of OAEP and encrypt the result from step 5 using the asymmetric public key of recipient, *r*.
7	Append the baggage *b* to the results of step 6.
8	Return result from step 7.

Asymmetric Encryption

E

The asymmetric encryption operator, E(r, t), corresponds to PKCS #7 *EnvelopedData* of tuple, t, encrypted for entity r. This operator consists of applying fast, symmetric, bulk encryption to message plaintext and then encrypting the secret key for bulk encryption with the recipient's public key. SET's default symmetric algorithm is DES; RSA is the default asymmetric algorithm; SHA-1 is the default hashing algorithm. OAEP shall be used to obfuscate the contents of the PKCS #7 envelope.

Step	Action
1	Initialize and load data fields depending on the message type.
2	Transform data fields "to be ordinarily encrypted" to their DER equivalent format.
3	Generate a fresh symmetric DES key.
4	Encrypt the result from step 2 using symmetric DES key from step 3; DES-CBC mode shall be used following the standard padding rule as described on page 65.
5	Initialize encrypted content buffer with data content type, DES algorithm identifier, and append result from step 4.
6	Initialize envelope buffer depending upon the recipient (128 bytes).
7	Initialize the "to be extra encrypted" OAEP buffer with symmetric DES key from step 3.
8	Apply OAEP processing to envelope buffer.
9	Encrypt the result from step 8 using asymmetric public key of entity *r*.
10	Initialize recipient information buffer with RSA algorithm identifier and append result from step 9.
11	Initialize PKCS #7 **EnvelopedData** buffer and append result from step 10 and the result from step 5.
12	Return result from step 11.

Asymmetric Encryption with Integrity

EH

The integrity encryption operator, EH(r, t), is similar to E except that the PKCS #7 envelope includes a hash of the tuple t. It consists of applying fast, symmetric, bulk encryption to message plaintext and then encrypting the secret key for bulk encryption and hash with the recipient's public key. Processing software shall re-hash tuple t and check for match against the corresponding hash in the PKCS #7 envelope. SET's default symmetric algorithm is DES; RSA is the default asymmetric algorithm. OAEP shall be used to obfuscate the contents of the RSA envelope.

Step	Action
1	Initialize and load data fields depending on the message type.
2	Transform data fields "<u>to be ordinarily encrypted</u>" to their DER equivalent format.
3	Compute SHA-1 hash of result from step 2.
4	Generate a fresh symmetric DES key.
5	Encrypt the result from step 2 using symmetric DES key from step 4; DES-CBC mode shall be used following the standard padding rule as described on page 65.
6	Initialize encrypted content buffer with data content type, DES algorithm identifier, and append result from step 5.
7	Initialize envelope buffer depending upon the recipient (128 bytes).
8	Initialize the "<u>to be extra encrypted</u>" OAEP buffer with symmetric DES key from step 4 and the hash computed from step 3.
9	Apply OAEP processing to envelope buffer.
10	Encrypt the result from step 9 using asymmetric public key of entity *r*.
11	Initialize recipient information buffer with RSA algorithm identifier and append result from step 10.
12	Initialize PKCS #7 buffer and append result from step 11 and the result from step 6.
13	Return result from step 12.

Extra Asymmetric Encryption

EX

The "extra" asymmetric encryption operator, EX(r, t, p), consists of applying fast, symmetric, bulk encryption to the plaintext in tuple t, and a separate "extra" process to the other plaintext in parameter p. In SET's implementation of this operator, p is put inside the PKCS #7 envelope and the tuple t is linked to p prior to bulk encrypting its contents. A fresh 20-byte nonce (EXNonce) is also included to foil dictionary attacks on p. The secret key for bulk encryption and parameter p is encrypted with the recipient's public key. SET's default symmetric algorithm is DES; RSA is the default asymmetric algorithm. OAEP shall be used to obfuscate the contents of the RSA envelope.

Step	Action
1	Initialize and load data fields depending on the message type.
2	Transform data fields "<u>to be ordinarily encrypted</u>" to their DER equivalent format.
3	Initialize buffer and copy parameter *p* for "extra processing".
4	Generate a fresh nonce and append to result from step 3.
5	Compute SHA-1 hash of result from step 4.
6	Link tuple *t* with parameter *p* by appending results from step 5 to results of step 2.
7	Generate a fresh symmetric DES key.
8	Encrypt the result from step 6 using symmetric DES key from step 7; DES-CBC mode shall be used following the standard padding rule as described on page 65.
9	Initialize encrypted content buffer with data content type, DES algorithm identifier, and append result from step 8.
10	Initialize envelope buffer depending upon the recipient (128 bytes).
11	Initialize the "<u>to be extra encrypted</u>" OAEP buffer with symmetric DES key from step 7 and the buffer with parameter *p* and nonce initialized in step 4.
12	Apply OAEP processing to envelope buffer.

Continued on next page

Extra Asymmetric Encryption, continued

EX (continued)

Step	**Action**
13	Encrypt the result from step 12 using asymmetric public key of entity *r*.
14	Initialize recipient information buffer with RSA algorithm identifier and append result from step 13.
15	Initialize PKCS #7 buffer and append result from step 14 and the result from step 9.
16	Return result from step 15.

Extra Asymmetric Encryption with Integrity

EXH

The "extra" asymmetric encryption with integrity operator, EXH(r, t, p), is similar to EX except that the PKCS #7 envelope also includes a hash of the tuple t. It consists of applying fast, symmetric, bulk encryption to the plaintext in tuple t, and a separate "extra" process to the other plaintext in parameter p. In SET's implementation of this operator, p is put inside the PKCS #7 envelope and the tuple t is linked to p prior to bulk encrypting its contents. Similar to EX, a fresh 20-byte nonce (EXNonce) is also included to foil dictionary attacks on p. The secret key for bulk encryption, hash of tuple t and parameter p is encrypted with the recipient's public key. Processing software shall re-hash tuple t and check for match against the corresponding hash in the PKCS #7 envelope. SET's default symmetric algorithm is DES; RSA is the default asymmetric algorithm. OAEP shall be used to obfuscate the contents of the PKCS #7 envelope.

Step	Action
1	Initialize and load data fields depending on the message type.
2	Transform data fields "<u>to be ordinarily encrypted</u>" to their DER equivalent format.
3	Compute SHA-1 hash of result from step 2.
4	Initialize buffer and copy parameter *p* for "extra processing".
5	Generate a fresh nonce and append to result from step 4.
6	Compute SHA-1 hash of result from step 5.
7	Link tuple *t* with parameter *p* by appending results from step 6 to results of step 2.
8	Generate a fresh symmetric DES key.
9	Encrypt the result from step 7 using symmetric DES key from step 8; DES-CBC mode shall be used following the standard padding rule as described on page 65.
10	Initialize encrypted content buffer with data content type, DES algorithm identifier, and append result from step 9.
11	Initialize envelope buffer depending upon the recipient (128 bytes).
12	Initialize the "<u>to be extra encrypted</u>" OAEP buffer with symmetric DES key from step 8, the hash computed from step 3, and the buffer with parameter *p* and nonce initialized in step 5.

Continued on next page

Extra Asymmetric Encryption with Integrity, continued

EXH (continued)

Step	Action
13	Apply OAEP processing to envelope buffer.
14	Encrypt the result from step 13 using asymmetric public key of entity *r*.
15	Initialize recipient information buffer with RSA algorithm identifier and append result from step 14.
16	Initialize PKCS #7 buffer and append result from step 15 and the result from step 10.
17	Return result from step 16.

Symmetric Encryption

EK

The symmetric encryption operator, EK(k, t), encrypts the plaintext in tuple t with a provided key, k. Either the DES or CDMF algorithm may be used.

Step	Action
1	Initialize and load data fields depending on the message type.
2	Transform data fields "<u>to be ordinarily encrypted</u>" to their DER equivalent format.
3	Encrypt the result from step 2 using symmetric key, *k*, using either DES or CDMF depending on the algorithms supported by the message recipient. For DES, the DES-CBC mode shall be used following the standard padding rule as described on page 65.
4	Initialize encrypted content buffer with data content type, DES (or CDMF) algorithm identifier, and append result from step 3.
5	Return result from step 4.

Signature

S

The signature operator, S(s, t), corresponds to PKCS #7 **SignedData** of tuple, t, signed by entity s. SET's default signature algorithm is RSA with SHA-1 hash. All SET toolkits and applications shall adhere to the "sign before encrypt" policy.

SET PKCS #7 **SignedData** digital signature operations are always performed on values that are the DER representations of ASN.1 types. **SignedData** signature operations are never performed on arbitrary octet strings, such as ASCII text files or random strings with no internal structure, so the **data** content type is never used. In such situations, PKCS #7 requires that at least two authenticated attributes be included in the content to be signed. The parameterized types, **S{}** and **SO{}**, both represent **SignedData** in SET, and both require authenticated attributes.

Two authenticated attributes, **contentType** and **messageDigest**, are always included in the content to be signed in SET. For **SignedData**, a message digest results from the application of the PKCS #7 message-digesting process to some SET ASN.1 type, the content to be signed. For SET **SignedData**, the content to be signed is always the complete DER representation, including the tag and length octets, of two authenticated attributes tightly coupled with the **content** component of **ContentInfo**.

The initial input to the message-digesting process is the DER representation of the **content** component of the **ContentInfo** sequence. **ContentInfo** binds a **contentType** component object identifier to the type in its **content** component. In SET, each **SignedData** content type is uniquely named by an object identifier. Since this value is not protected directly against a substitution attack, it is also included in the **authenticateAttributes**. The **contentType** attribute shall specify an object identifier that matches the value in the **contentType** component of the **ContentInfo** sequence. The **messageDigest** attribute contains the value of the digested **content** component of **ContentInfo**.

The definition of the **SignerInfos** sequence in PKCS #7 allows any number of signers to be included in the collection, providing one **SignerInfo** per signer. In a degenerate case, PKCS #7 allows **SignedData** to be used with no signers. SET PKCS #7 **SignedData** requires one signer for all messages except **CertReq** and **CertInqReq**, which requires two signers when used for certificate renewal. It is constrained to permit only one or two signers, so that the general processing requirements of PKCS #7 are simplified in SET.

In the **SignerInfo** component of **SignerInfos**, both the **authenticateAttributes** and the **unauthenticateAttributes** components are specified as optional. Under DER, the ASN.1 Distinguished Encoding Rules used by SET, when optional sequence components are absent they do not appear in an encoding of a value of that type. In SET, the **unauthenticateAttributes** component of the **SignerInfo** sequence is always absent, and never appears in an encoding of **SignedData**. The **authenticateAttributes** component is always present, and shall be included in the message-digesting process.

Continued on next page

Signature, continued

Composing SignedData

Step	Action
1	Initialize a **SignedData** type with the version, algorithm identifier and content type to be signed.
2	Encode the type <u>to be signed</u> to obtain its DER equivalent format.
3	Use the result of step 2 to initialize the **content** component of **ContentInfo**.
4	Initialize a **SignerInfo** type with the version, digest algorithm and digest encryption algorithm.
5	Compute the message digest, using SHA-1, of the result of step 3.
6	Initialize an **authenticatedAttributes** structure and populate the structure with two attributes: **contentType** and **messageDigest**. Set the **type** components of these attributes with the identifiers of the two attributes.
7	Initialize the **values** component of the first attribute with the content type to be signed, and the second attribute with the message digest computed in step 5.
8	Encode the authenticated attributes, and encrypt this result using the sender's private key, placing the signature in **EncryptedDigest**.
9	Select the pertinent X.509 certificates and CRLs needed to verify the signature and included them in the **SignedData**.
10	If the message type requires two signatures, repeat steps 4 through 9.

Signature Only

SO

The dual signature only operator, **SO**(s, t), is an optimization of separately signing tuple t, signed by entity s. SET's default signature algorithm is RSA with SHA-1 hash. All SET toolkits and applications shall adhere to the "sign before encrypt" policy.

The processing steps for the **SO** operator differ slightly from those of the **S** operator. With **SO**, the **content** component of **ContentInfo** is absent, and does not appear in an encoding of a value of that type. Perform all of the steps of **S** except step 3, which is skipped.

Keyed-Hash

HMAC

The keyed-hash operator, HMAC(t, k), corresponds to the 160-bit HMAC-SHA-1 hash of the tuple, t, using the secret, k. This function shall be used as the blinding function to protect the account number in the cardholder's certificate. The shared secret is known only by the cardholder and their certificate Issuer. HMAC is also used to form transaction stains.

Step	Action
1	Set *ipad* equal to buffer containing 64 bytes with 0x36 repeated 64 times.
2	Set *opad* equal to buffer containing 64 bytes with 0x5c repeated 64 times.
3	Append zeros to the end of *k* to create a 64 byte buffer (for example, if *k* is of length 20 bytes, it shall be appended with 44 zero bytes 0x00).
4	Compute bit-wise exclusive-or result of step 3 and *ipad.*
5	Append the data stream in tuple *t* to the 64 byte buffer computed from step 4.
6	Compute the SHA-1 hash of result of step 5 via *Hash* operator.
7	Compute bit-wise exclusive-or result of step 3 and *opad.*
8	Append the SHA-1 result from step 6 to the 64 byte buffer resulting from step 7.
9	Compute the SHA-1 hash of result of step 8 via *Hash* operator.
10	Return result from step 9.

Hash

H

The hash operator, H(t), corresponds to the 160-bit SHA-1 hash of the tuple, t. This operator corresponds to the SET ASN.1 parameterized type H{}. Though this type is never directly referenced in any SET message, it is used in OAEP processing and included here for completeness. To achieve algorithm independence, SET hash values are packaged in PKCS #7 **DigestedData**, along with an identifier that names the algorithm used to create the hash.

Step	Action
1	Set *B* equal to the address of tuple *t* to be hashed.
2	Set *L* equal to length of tuple *t* to be hashed.
3	Initialize a 160-bit buffer for holding SHA-1 hash value.
4	Compute SHA-1 hash of buffer using *B* and *L*.
5	Return result from step 4.

Digested Data

DD

The **DigestedData** operator DD(T) corresponds to a 160-bit SHA-1 hash of the tuple embedded in a PKCS **DigestedData** sequence. SET uses the **DD{}** parameterized type to specify "*detached digests*", **DigestedData** in which the content that is digested is not included in the **content** component of **ContentInfo**. Each type of content digested in SET has a name, a unique object identifier. The **contentType** component of **ContentInfo** is set to this object identifier value.

The **digest** component of **DigestedData** is the result of the message-digesting process. It contains a message digest of the SET type identified by the **contentType** component of **ContentInfo**. The message digest is computed using one of the algorithm objects in the **DigestAlgorithms** information object set. It is computed on the complete DER representation, including the tag and length octets, of the SET ASN.1 type to be digested.

The **digestAlgorithm** component of **DigestedData** is set to the values of the two fields of the selected algorithm object. This specifies an object identifier which uniquely names the **algorithm** used to compute the message digest, and any associated **parameters** required by the cryptographic algorithm. The **digestAlgorithm** is used by the recipient to verify the message digest.

In a DER encoding of **DD{}**, the **content** component of **ContentInfo** will not be present. The recipient must obtain the message content from elsewhere to verify the message digest. The verification is accomplished by independently computing a message digest, and comparing it to the value of the **digest** component of **DigestedData**.

Step	Action
1	Set *B* equal to the address of tuple *t* to be hashed.
2	Set *L* equal to length of tuple *t* to be hashed.
3	Initialize a 160-bit buffer for holding SHA-1 hash value.
4	Compute SHA-1 hash of buffer using *B* and *L*.
5	Insert result of step 4 in **digest** field of **DigestedData**.
6	Insert the SHA-1 hash algorithm identifier into **digestAlgorithm** .
7	Set the version to zero.

Linkage

L

The linkage operator, L(t1, t2), corresponds to a sequence of the tuple t1 and a PKCS #7 **DigestedData**. The **DigestedData** linkage component contains a message digest of tuple t2, and can be represented by DD(t2). The linkage operator is not symmetric. It does not link t2 to t1.

Step	Action
1	Build a structure with two fields.
2	Populate the first field with tuple t1.
3	Using tuple t2, populate the second field with the results of DD(t2).

Optimal Asymmetric Encryption Padding

Purpose

The purpose of OAEP is to ensure that individual pieces of a message cannot be extracted from an PKCS #7 block. There are cryptoanalytic techniques that make some bits of a message easier to determine than others.

OAEP randomly distributes the bits of an PKCS #7 block making each bit as difficult to extract as all other bits.

Algorithm description

The **E**, **EH**, **EX**, and **EXH** encryption primitives combine RSA encryption and Bellare-Rogaway Optimal Asymmetric Encryption Padding (OAEP). SET uses OAEP to provide "extra" protection of the account information associated with the cardholder, merchant or Acquirer in the digital envelope.

This section provides a brief description on how to implement SET's use of OAEP to support its "extra encryption" and "extra decryption" operators. The reader is encouraged to supplement this description with the OAEP information provided in Book 3: Formal Protocol Definition.

Continued on next page

Optimal Asymmetric Encryption Padding, continued

Extra encryption

SET "extra encryption" involves the following processing steps:

Step	Action
1	Prepare any "extra" data as described in the message formats.
2	If **EH** or **EXH** encryption is used, compute the SHA-1 hash of the data to be DES-encrypted prior to its encryption.
3	Generate a fresh, random, key for DES-encrypting the "regular" part of the data.
4	Concatenate the DES key, the SHA-1 hash of the data (*HD*) prior to being DES encrypted (if used), and any "extra" data to form the Actual Data Block, *ADB*.
5	Prepend a single byte containing 0x03 (the Block Type byte, *BT*) and seven bytes of zeros (the Verification bytes, *V*), and Block Content byte (*BC*) to *ADB* to form the Data Block, *DB*. $DB = BT \mid BC \mid V \mid ADB$
6	Generate a random 16-byte string *E-Salt*, and compute *H1(E-Salt)*. *H1* returns the leading bytes of the SHA-1 hash.
7	Compute $A = DB \oplus H1(E\text{-}Salt)$.
8	Let $B = E\text{-}Salt \oplus H2(A)$. Concatenate *A* to *B* to form *PDB*. *H2* returns the trailing bytes of the SHA-1 hash.
9	Set *I* to a random value not equal to either 0x00 or 0x80.
10	Encrypt the final block, *R*, with RSA, by raising it to the power of the public encryption exponent, modulo the RSA modulus.

Continued on next page

Optimal Asymmetric Encryption Padding, continued

Extra decryption

SET "extra decryption" requires the following steps:

Step	Action
1	Decrypt the received block with RSA, by raising it to the power of the private encryption exponent, modulo the RSA modulus.
2	Verify that the first byte (*I*) of the resulting block is neither 0x00 nor 0x80.
3	Compute *E-Salt* = *B* ⊕ *H2*(*A*).
4	Compute *DB* = *A* ⊕ *H1(E-Salt)*.
5	Verify *BT*, *BC*, and *V*, and obtain *ADB*.
6	Parse *ADB* for a DES key, possible hash, and possible extra-encrypted data, depending upon *BC*.
7	Decrypt the DES data using the retrieved DES key.
8	If BC indicates that a SHA-1 hash (*HD*) is present, verify the hash against the decrypted DES data.

Continued on next page

Optimal Asymmetric Encryption Padding, continued

Processing

Figure 8 below illustrates the processing flow for SET's application of OAEP.

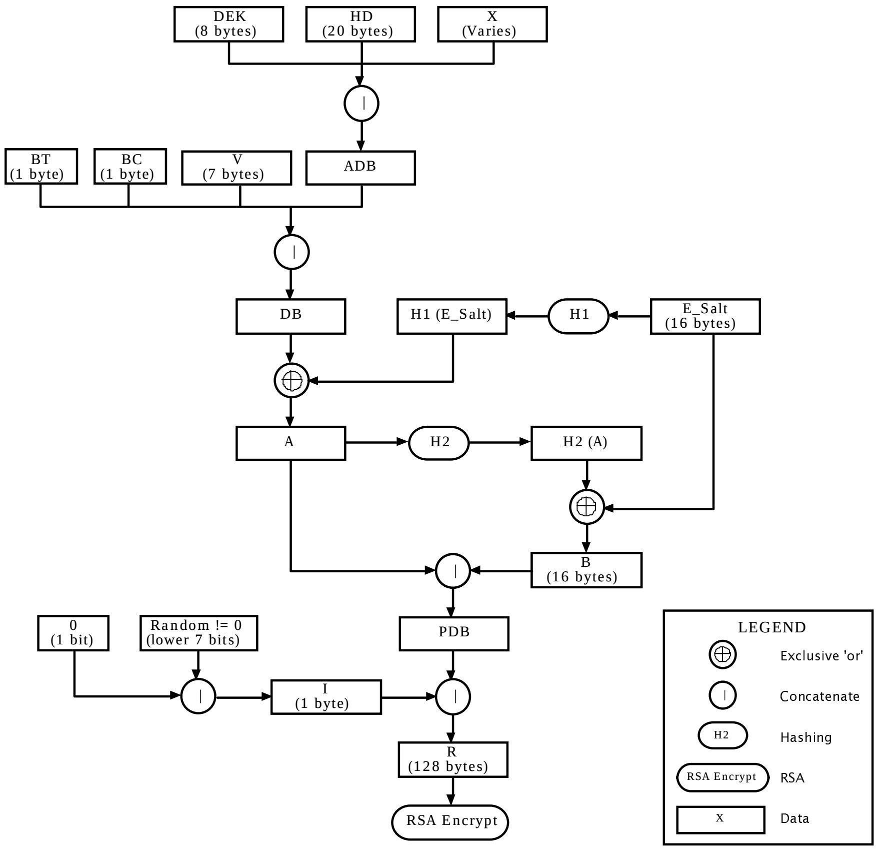


Figure 8: OAEP Processing Flow

Continued on next page

Optimal Asymmetric Encryption Padding, continued

Encoding of DB

Only atomic (in the sense of ASN.1) data elements are present in **DB**. Each element is encoded within **DB** in the canonical form used by DER encoding, but without tag or length octets. When transferring data from DER-encoded format to **DB**, add pad characters (0x00) to the end of the data; when transferring from **DB** to DER-encoded format, strip all pad characters from the end of the data.

To understand the format of a **DB** field, examine the ASN.1 used to define the field for signature purposes. Determine the matching DER-encoded wire format, and store the field in **DB** accordingly.

SET Error Processing

Introduction

From the perspective of a SET participant, SET flows always occur in pairs. Each message transmitted by a requesting participant is answered by a responding participant. The Error flow (unlike all the other flows) is defined with respect to requesters and responders because it is used when the responder cannot reliably identify an incoming message.

Error indicates that a responder rejects a message because it fails format or content verification tests. The responder sends Error (rather than a negative response code) when the responder cannot trust any of the fields of an incoming message. In general, Error shall be used only to respond to the direct sender of the message, and when it is not possible to clearly isolate the error to an incorrect value of a field. Error is intended to respond to messages which may be interpreted as corrupted or unintelligible.

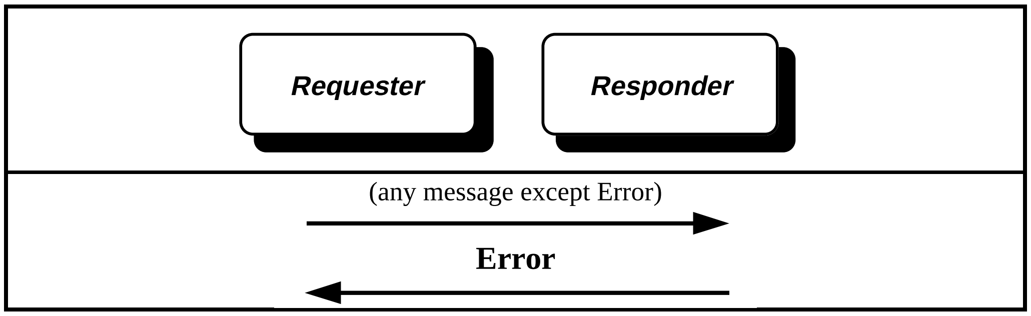


Figure 9: Error Message

Not for business results

The Error message is not used to indicate normal business results such as a declined authorization. Business results are indicated by explicit codes in standard response messages.

Error categories

Errors in the SET system can result from a number of different sources. SET messages can be parseable but malformed by not adhering to the requirements of the protocol, values can be illegal, or messages can be corrupted, usually as a result of transmission errors.

Continued on next page

SET Error Processing, continued

Duplicate message

Enough information appears in cleartext in the message wrapper that one can detect whether a message is a retransmit or not. The recipient's reaction to a duplicate message is dependent upon the idempotency property of the message type, the number of duplicated messages, the source of the duplicate message, and observing the frequency between successive duplicate messages. If a system suspects that it is being subjected to flooding or spamming attacks, duplicate messages may be ignored.

Corrupted messages

A corrupted message is one that cannot be parsed. Normally, a potential corrupted message should not be received, because communication transport mechanisms will cause corrupted data to be resent. However, if a corrupted message is received, an appropriate error message shall be returned indicating that a corrupted message was received and providing the request/response pair identifier of the message, if available. It is acceptable to ignore the message completely if not enough data is extracted to be able to respond.

Malformed messages

If a message can be parsed, but is otherwise illegal due to values that are out of range or options that are inconsistent, an appropriate error message shall be returned to the originator.

Failed cryptography

If a message is received in which authentication tests fail, an appropriate error message shall be returned. A delayed, generic error message shall be used to avoid disclosing any details about the failure. The software shall log a message when any failed cryptography error is detected.

Errors to Error messages

An application shall never generate an Error message in response to another Error message.

When to send *Error*

Merchant, payment gateway and CA software should send an *Error* message when encountering a low-level processing error on a SET request message. Any messages that do not appear to be SET messages should be ignored.

Cardholder and merchant software should send an **Error** message when encountering a low-level processing error on a SET response message. The **Error** message should be sent to a *diagnostic log* port if one has been defined for the system that sent the response. Applications should avoid sending **Error** messages on the same port as request messages; however, if no *diagnostic log* port is available, the application may send one **Error** message per day on the request port.

The software may limit the number of error messages that are sent to mitigate the effects of denial-of-service attacks. For example, the software may elect to only send one error message per day to a given requester.

Continued on next page

SET Error Processing, continued

When NOT to send *Error*

An **Error** message shall never be sent in response to anything that appears to be an **Error** message. A valid SET message will begin with a tag of [30] and a length for the entire message (**MessageWrapper** plus message body). The **MessageWrapper** will contain:

- a tag of [30] followed by
  - the length of the **MessageHeader**
  - the content of the **MessageHeader**
- a tag for the type of message followed by the length and content of the **Message**.

If the tag for the type of message is 999 (indicating an **Error** message), a SET application shall never send a response even if the message appears to be malformed. This is to prevent loops where one **Error** message triggers another.

External errors

Cardholder and merchants systems shall also anticipate errors originating from systems external to SET at any time. Examples include too many message transmission retries, exceptions occurring in the payment system, underlying network transport failures, and incorrect handling of the order information by the shopping software.

Continued on next page

SET Error Processing, continued

Creating an Error

When an application encounters a SET error, it shall create an Error message as follows:

Step	Action
1	Construct ErrorTBS as follows: A. Set ErrorCode to a value specified in ErrorCode (see page 110). B. Generate a fresh ErrorNonce. C. If the error occurred because the application did not know how to process a critical extension (certificate or message), populate ErrorOID with the object identifier of the extension. D. If the error occurred because of a problem with a certificate, populate ErrorThumb with the **Thumbprint** of the certificate. E. If the error resulted from a signature verification failure, populate ErrorThumb with the hash of the certificate. F. Construct ErrorMsg as follows: populate either the **MessageHeader** or the entire message (up to the size restriction of 20,000 bytes). The choice of whether to copy only the header or the entire message is left to each implementation.
2	Sign the **Error** message if a signature certificate is available.
3	Invoke "Compose Message Wrapper" to send message. (See page 76.)

Continued on next page

SET Error Processing, continued

Error message

The following fields are defined for **Error**:

Field Name	Description
Error	**< Signed Error, UnsignedError >**
SignedError	**S(EE, ErrorTBS)**
UnsignedError	**ErrorTBS** *The unsigned version of* **Error** *shall only be used if the entity does not have a signature certificate.*
ErrorTBS	**{ErrorCode, ErrorNonce, [ErrorOID], [ErrorThumb], ErrorMsg}**
ErrorCode	*Enumerated code defining the error condition. See page 110.*
ErrorNonce	*A nonce to ensure the signature is generated over unpredictable data.*
ErrorOID	*The object identifier an unsupported critical extension that caused the error.*
ErrorThumb	*The thumbprint of the certificate that caused the error or the hash of the certificate if signature verification failure occurred.*
ErrorMsg	**<MessageHeader, BadWrapper>**
MessageHeader	*The message header of the message that produced the error.*
BadWrapper	*The message wrapper of the message that produced the error (up to 20,000 bytes).*

Continued on next page

SET Error Processing, continued

ErrorCode

The following values are defined for **ErrorCode**.

unspecifiedFailure	*The reason for the failure does not appear elsewhere in this list.*
messageNotSupported	*This valid message type is not supported by the recipient*
decodingFailure	*An error was encountered during the DER decoding process on the message.*
invalidCertificate	*A certificate necessary to process this message was not valid (for a reason not specified elsewhere in this table). The* **ErrorThumb** *field identifies the invalid certificate.*
expiredCertificate	*A certificate necessary to process this message has expired. The* **ErrorThumb** *field identifies the invalid certificate.*
revokedCertificate	*A certificate necessary to process this message has been revoked. The* **ErrorThumb** *field identifies the invalid certificate.*
missingCertificate	*A certificate necessary to process this message is not available in the recipient's certificate cache and was not included in the message.*
signatureFailure	*The digital signature of the message could not be verified.*
badMessageHeader	*The message header cannot be processed.*
wrapperMsgMismatch	*The contents of the message wrapper are inconsistent with the internal content of the message, e.g., the* **RRPID** *does not match.*
versionTooOld	*The version number of the message is too old for the recipient to process.*
versionTooNew	*The version number of the message is too new for the recipient to process.*
unrecognizedExtension	*The message or a certificate contains a critical extension that the recipient cannot process. The* **ErrorOID** *field identifies the extension. If the extension appears in a certificate, the* **ErrorThumb** *field identifies the certificate.*
messageTooBig	*The message is too big for the recipient to process.*
signatureRequired	*The unsigned version of this message is not valid*

Continued on next page

SET Error Processing, continued

ErrorCode (continued)

messageTooOld	*The date of the message is too new for the recipient to process.*
messageTooNew	*The date of the message is too new for the recipient to process.*
thumbsMismatch	*Thumbs sent in an unsigned request did not match those returned to the requester checking for substitution attack.*
unknownRRPID	*An unknown* **RRPID** *was received.*
unknownXID	*An unknown* **XID** *was received.*
unknownXID	*An unknown local identifier was received.*
challengeMismatch	*A challenge sent in a request did not match challenge in response.*

Sending error message

The SET protocol is based on request/response pairs throughout the protocol. The Error message does not conform to this paradigm, since it may be a response to either a request or a response. The former case poses no difficulty. However, in the latter case, difficulties may arise if the underlying transport is based on a request/response paradigm, as in a World Wide Web browser. In this case, the error message may be sent as a request message, and the protocol will not require a response message (the underlying protocol may time out). It is recognized that it may require user permission for an error message to be sent as a result of the operational constraints of a World Wide Web browser. This is acceptable, but not encouraged.

Part II
Certificate Management

Overview

Introduction

Part II defines the Certificate Management Architecture, protocols, and concepts used in SET.

Organization

Part II includes the following chapters:

Chapter	Title	Page
1	Certificate Management Architecture	114
2	Certificate Request Protocols	134
3	Certificate Revocation	202
4	Certificate Format	207
5	Certificate Revocation List and BrandCRLIdentifier	242
6	CA to CA Messages	252

Chapter 1 Certificate Management Architecture

Overview

Introduction

Chapter 1 provides an overview of the Certificate Management Architecture and describes the Certificate Management functions.

Organization

Chapter 1 includes the following sections:

Section	Title	Contents	Page
1	Architecture Overview	Introduces the Certificate Management Architecture and defines each of the architectural components.	115
2	Functional Overview	Describes the certificate issuance, renewal, and revocation functions.	118
3	Certificate Chain Validation	Describes the certificate validation process.	124
4	Root Certificate Distribution	Describes the issuance and management of the Root certificates.	131

Section 1
Architecture Overview

General Overview

Architecture diagram

The Certificate Management Architecture consists of the nine components identified in Figure 10. The Architecture is based on the hierarchy of trust defined for the management and verification of SET certificates by Certificate Authorities (CAs).

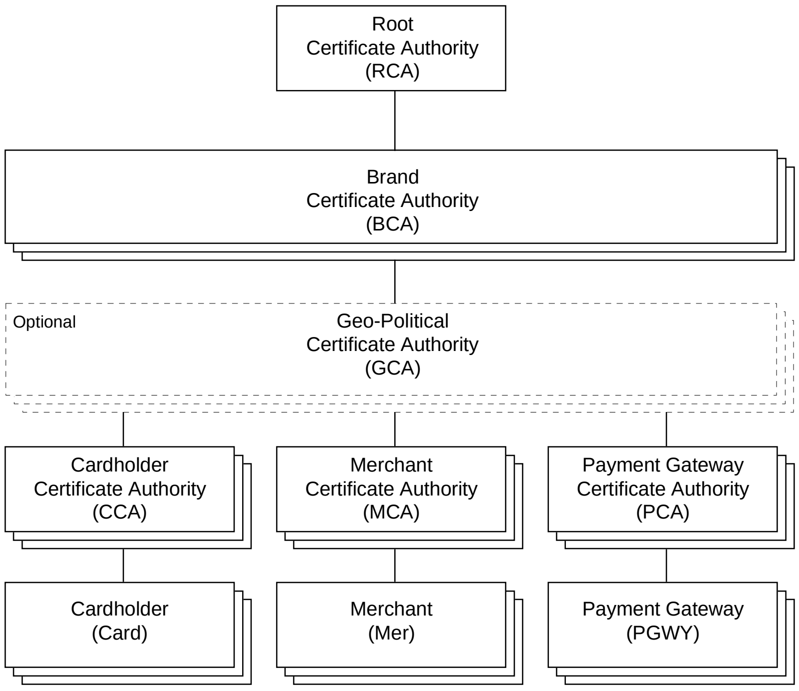

Root Certificate Authority (RCA)
Brand Certificate Authority (BCA)
Optional
Geo-Political Certificate Authority (GCA)
Cardholder Certificate Authority (CCA)
Merchant Certificate Authority (MCA)
Payment Gateway Certificate Authority (PCA)
Cardholder (Card)
Merchant (Mer)
Payment Gateway (PGWY)

Figure 10: Certificate Management Architecture

Architecture Overview

Root Certificate Authority

The Root CA (RCA) is kept off-line under extremely tight physical controls. The RCA will be accessed very infrequently to issue new Brand CA certificates and a new Root certificate. If the unlikely compromise of a Brand CA private key does occur, the Root CA will generate and distribute a Certificate Revocation List (CRL) identifying the Brand CA certificate.

Brand CA

The Brand CA (BCA) allows for some degree of autonomy in each Brand's certificate management. Like the Root CA, these CAs are operated under tight physical controls and will be used to issue Geopolitical and/or Cardholder, Merchant, or Payment Gateway CA certificates to the entities below them in the hierarchy. The Brand CA will generate, maintain, and distribute CRLs for compromised certificates that it generated and signed. It will also generate, maintain, and distribute BCIs containing all of the CRLs in the brand hierarchy.

Geo-Political CA

The Geo-Political CA (GCA) allows the Brand to distribute responsibility for managing types of certificates to geographic or political regions. This level in the architecture allows Brand policies to vary from one region to another as deemed necessary by the Brands. The Geo-Political CA will generate, maintain, and distribute CRLs for compromised certificates that it generated and signed.

Cardholder CA

The Cardholder CA (CCA) is responsible for generating and distributing Cardholder certificates to Cardholders. At the CCA's option, it may accept certificate requests from Cardholders via Web and or e-mail. The CCA maintains relationships with card issuers to allow for the verification of Cardholder accounts. While the CCA will not generate and maintain a CRL, it will be responsible for distributing Root, Brand, Geopolitical, and Payment Gateway CA's CRLs.

This CA may be operated by a payment brand, an Issuer, or another party according to payment brand rules.

Continued on next page

Architecture Overview, continued

Merchant CA	The Merchant CA (MCA) is responsible for distributing certificates to Merchants. Acquirers shall verify and approve their Merchant certificate requests prior to issuance by the MCA. This CA may be operated by a payment brand, an Acquirer, or another party according to payment brand rules. While the MCA will not generate and maintain a CRL, it will be responsible for distributing for distributing Root, Brand, Geopolitical, and Payment Gateway CA's CRLs.
Payment Gateway CA	The Payment Gateway CA (PCA) manages the issuance of certificates for SET Payment Gateways. This CA may be operated by a payment brand, an Acquirer, or another party according to payment brand rules. The PCA is responsible for generating, maintaining, and distributing CRLs for compromised Payment Gateway certificates.
Cardholder	Cardholders request and receive certificates from a CCA.
Merchant	Merchants request and receive certificates from a MCA.
Payment Gateway	The Payment Gateways request and receive certificates from a PCA.

Section 2
Functional Overview

Overview

Introduction

The CA provides three basic services to the entities below them in the Certificate Management Hierarchy: Certificate Issuance, Certificate Renewal, and Certificate Revocation. Each service is briefly described here, with a more detailed description in following chapters.

Organization

Section 2 includes the following topics:

- Certificate Issuance
- Certificate Renewal
- Certificate Revocation

Certificate Issuance

Overview

Certificates are issued to subscribers by a variety of methods depending on the SET entity. End entities may be issued signature and or encryption certificates, depending on the entity. Cardholders are only issued signature certificates; Merchants and Payment Gateways may be issued both signature and encryption certificates.

Cardholder

Cardholder certificates are issued by CCAs. The issuance of Cardholder certificates involves the following communications between the Cardholder and CCA:

- The Cardholder initiates a request for a certificate.
- The CCA responds with an encryption certificate for the Cardholder to use to protect the transmission of its payment card number to the CCA.
- The Cardholder encrypts its payment card number using the CCA's certificate and sends it to the CCA.
- The CCA responds with the appropriate payment card-specific certificate registration form.
- The Cardholder completes the registration form, which includes the Cardholder public key, and sends it to the CCA for certification.
- The CCA verifies the Cardholder registration information with the Issuer, generates the certificate, signs it, and then sends it to the Cardholder.

Merchant

Merchant certificates are issued by MCAs. Before a Merchant certificate is issued, the request shall be verified by the Merchant's Acquirer or payment brand authority. The certificates are obtained from the MCA using the following protocol:

- The Merchant initiates a request for a certificate.
- The MCA responds with a registration form.
- The Merchant completes and sends the registration form and public keys to the MCA for processing.
- The Acquirer or payment brand authority verifies the Merchant request and the MCA generates, signs, and sends the certificate to the Merchant.

Payment Gateway

Payment Gateway certificates are issued by the Payment Gateway CA (PCA). The Issuance of certificates parallels the communications described in the Merchant scenario. The Acquirer verifies the Payment Gateway certificate request and authorizes the certificate generation by the PCA.

Continued on next page

Certificate Issuance, continued

CA certificates

CA certificates are issued by the superior CA in the SET hierarchy. Brand CA certificates are issued by the Root CA, Geopolitical CA certificates are issued by the Brand CA, and Cardholder, Merchant, and Payment Gateway CA certificates are issued by either the Geopolitical CA or the Brand CA, depending on if a Geopolitical CA exists for that CA's area. The security required for the issuance of CA certificates may dictate the use of a combination of hardware tokens and electronic media for certificate issuance and is outside the scope of SET. SET defines a protocol for CA Certificate Requests and Responses in Chapter 6.

Certificate Renewal

Overview

Certificates for Cardholders, Merchants, and Payment Gateways will be renewed on a periodic basis following the same issuance procedure as for initial certificate generation, which is described in Chapter 2.

Registration form

The registration form for renewals may request different or minimal information as deemed necessary by the Issuer or Acquirer. Depending on the brand's policy, identification and authentication based on the use of the previous certificate may aid in the authentication of the individual.

CA certificates

The protocol for CA certificate renewal is identical to that used for initial issuance, which is described in Chapter 6.

Certificate Revocation

Overview

Certificates are used by Cardholders, Merchants, and acquirers as a means of authenticating each other prior to transacting business. A certificate may need to be revoked for a number of reasons: for example, due to a real or suspected compromise of the private key, a change in the identification information contained in the certificate, or termination of use.

Cardholders

Payment Gateways - Cardholders need to be assured that they do not send account numbers to an unauthorized Payment Gateway. This is enforced using the following mechanisms:

- PCA CRLs - Revoked Payment Gateway certificates are included in CRLs distributed to Cardholders.
- CA CRLs - Revoked CA certificates are included on a CRL that is distributed to Cardholders. The Root, Brand, Geopolitical, and Payment Gateway CAs each maintain a CRL. Cardholder applications will identify unauthorized Payment Gateway certificates created using this CA certificate.
- Immediate re-distribution of the Payment Gateway certificate to all Merchants will purge the older Payment Gateway certificate from the Merchant certificate cache.
- Assigning a short cryptoperiod (for example, a one month validity period) to Payment Gateway certificates results in the certificate expiring soon after revocation. The cryptoperiod will be determined according to the brand's policy.

Merchants - Cardholders do not need to identify revoked Merchant certificates because Cardholders do not send any sensitive payment information to Merchants.

- CA CRL - The Cardholder shall verify that all CAs in the Merchant's certificate path are valid (not in a CA CRL).

Continued on next page

Certificate Revocation, continued

Merchants

Payment Gateways - Merchants need to identify revoked Payment Gateway certificates. This is enforced using the following techniques:

- PCA CRLs - Revoked Payment Gateway certificates are included in CRLs distributed to Merchants.
- CA CRLs - Revoked CA certificates are included on a CRL that is distributed to Merchants. Merchants will identify unauthorized Payment Gateway certificates created using this CA certificate.
- Immediate re-distribution of the Payment Gateway certificate to all Merchants will purge the older Payment Gateway certificate from the Merchant certificate database.
- Assigning a short cryptoperiod (for example, a one month validity period) to Payment Gateway certificates results in the certificate expiring soon after revocation. The cryptoperiod will be determined according to the brand's policy.

Cardholders - Merchants do not need to verify the validity of Cardholder certificates to protect payment information. The Merchant may perform the following validation of the Cardholder certificate:

- CA CRLs - Use of the CA CRL verifies that no CA certificate in the Cardholder certificate path has been revoked.

Payment gateway

Cardholders - The Payment Gateway:

- shall verify that the Cardholder certificate path does not include a CA that is in a CRL, and
- shall validate the information in the Authorization Request with the Issuer.

Merchants - The Payment Gateway:

- shall verify that the Merchant certificate path does not include a CA in the CRL, and
- shall verify that the Merchant maintains a valid relationship with the Acquirer.

Section 3
Certificate Chain Validation

Overview

Introduction

Certificates are verified through a hierarchy of trust illustrated in Figure 11 below. Each certificate is linked to the signature certificate of the certificate issuing entity. Certificates are validated by following the trust hierarchy to the Root CA. The path through which the certificates are validated is called the "signature chain."

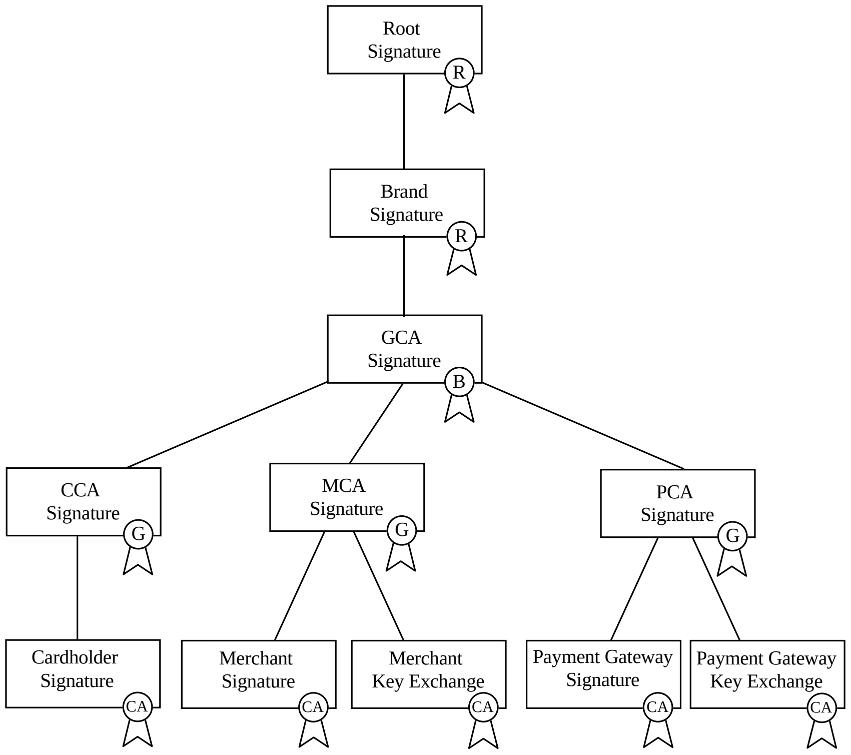


Figure 11: Hierarchy of Trust

Continued on next page

Overview, continued

Organization

Section 3 includes the following topics:

- Validation of Certificate Chain
- Dates in Certificates
- Thumbprints

Validation of Certificate Chain

Overview

The validation of the signature chain requires that:

- Each certificate in the path – from the End Entity (EE) certificate through the Root certificate – is validated, and
- Each certificate correctly maps to the CA that issued the certificate.

SET certificate chain validation is performed according to the processing requirements specified in Section 12.4.3 of Amendment 1 to X.509 and according to the SET requirements specified below. SET requirements for chain validation are in addition to those specified in X.509.

Certificate Chain Definition

The SET certificate chain is comprised of the set of certificates from the EE to the Root certificate, plus all of the Root's predecessors back to the initial Root certificate.

EE to CA Certificate Chainvalidation

In addition to the processing steps of X.509, the following constraints on the certificate chain shall be met:

1. The certIssuer field in the AuthorityKeyIdentifier extension of the EE certificate shallmatch the Issuer Name of the signing CA certificate.
2. The certSerialNumber field in the AuthorityKeyIdentifier extension of the EE certificate shall match the SerialNumber of the signing CA's certificate.
3. The Validity dates (in the certificate and in the PrivateKeyUsagePeriod extension) in the EE certificate shall be within the certificate Validity dates of the CA certificate.
4. The notBefore Validity date in the EE certificate shall be within the Validity dates in the PrivateKeyUsagePeriod extension of the CA certificate.
5. For each certificate below a Root certificate, the Brand Names within the organization Name of the subject Distinguished Name of each certificate shall match. If present, the product type within the organization Name of each certificate shall also match.
6. When checking a certificate's revocation status, the verifier shall ensure that it is holding an up to date BrandCRLIdentifier(BCI) and that it holds all of the CRLs on the BCI.

These constraints are illustrated graphically in Figure 12 on page 128.

Continued on next page

Validation of Certificate Chain, continued

EE certificate validation

The following shall be validated in EE certificates, in addition to the certificate chain processing requirements of X.509:

1. The KeyUsage field of the KeyUsage extension is valid for the intended purpose.
2. The subjectType field of the BasicConstraints extension indicates End Entity.
3. The CertificateType private extension corresponds with the context in which the certificate is being used.
4. The signature verifies.

CA certificate validation

The following shall be validated in each CA certificate, in addition to the certificate chain processing requirements of X.509:

1. The KeyUsage field of the KeyUsage extension is valid for the intended purpose.
2. The CertificateType private extension corresponds with the context in which the certificate is being used.

Chain validation

The validation procedures described above shall be enforced for all levels of the chain. For example, a Cardholder shall validate the Merchant, Merchant CA, Brand CA, Geo-Political CA, and Root CA certificates as described above. In practice, it is assumed that the validation process will stop at a level that has been previously validated.

Continued on next page

Validation of Certificate Chain, continued

Detailed diagram

Figure 12 below provides a logical view of the certificate data elements, with an emphasis on the data elements used for signature chain validation. The **bold** arrows indicate which fields are validated and the thin arrows show which fields should contain the same value.

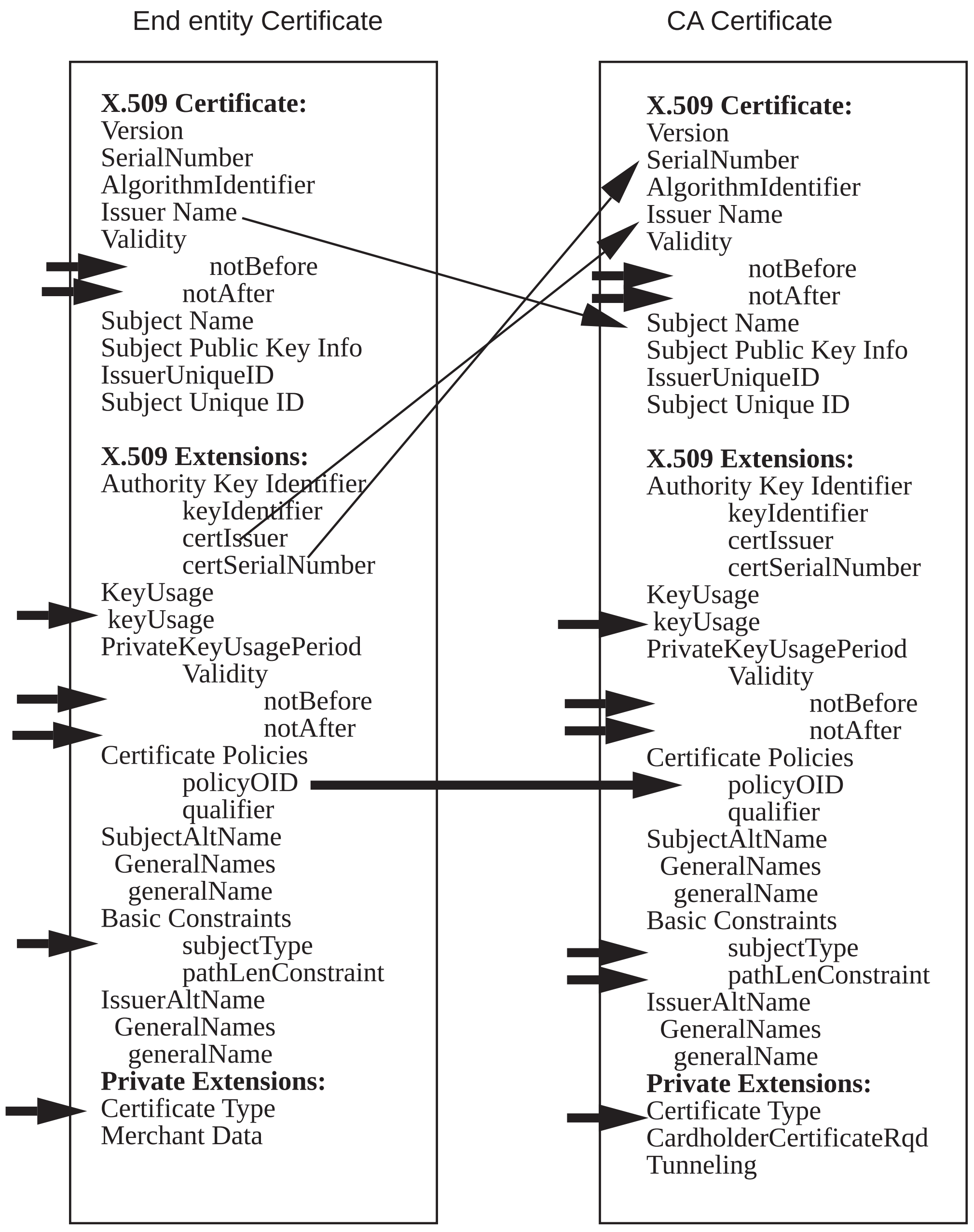


Figure 12: Certificate Comparison

Dates in Certificates

Overview

The validation of certificate expiration dates is an integral part of the signature chain validation process. The validation of an EE certificate requires that all of the trust chain is valid and that no certificate in the chain has expired.

Processing requirements

To ensure valid certificate dates for all certificates in a signature chain:

- All SET software shall validate certificate dates as part of the signature chain validation process.
- SET software shall provide a mechanism to prevent attempts at using expired certificates.
- The validity period of a certificate shall be contained within the validity date range and the private key usage period of the issuing CA's signature certificate.

The private key associated with the certificate expires before the certificate expires, allowing the public key in the certificate to be used to verify signatures after the private key has expired. Private key and certificate validity periods will be set according to brand policy. Appendix T: Private Key and Certificate Duration (on page 566) illustrates an example of the relationship between the private key and certificate validity periods for each CA and End Entity.

Thumbprints

Overview

Thumbprints are hashes of certificates, CRLs, or the BrandCRLIdentifier. An End Entity includes Thumbprints in a message as a compact way to identify the certificates, CRLs, etc., that it is holding.

The recipient of a message containing Thumbprints optionally checks the Thumbprints, and includes, in the downstream message, any certificates, CRLs, or the BrandCRLIdentifier that the sender does not have but will need for the transaction. If the recipient opts to ignore the Thumbprints, it shall send all of the certificates, CRLs, and the BrandCRLIdentifier that the sender will need. The recipient is required to ensure that the requester possesses all certificates, CRLs, and the BrandCRLIdentifier needed to complete the processing of the message.

Format

```
Thumbs ::= SEQUENCE {
     digestAlgorithm   AlgorithmIdentifier  {{DigestAlgorithms}},
     certThumbs        [0] EXPLICIT Digests  OPTIONAL,
     crlThumbs         [1] EXPLICIT Digests  OPTIONAL,
     brandCRLIdThumbs  [2] EXPLICIT Digests  OPTIONAL
  }

-- SHA-1 is used for SET and is indicated in the digestAlgorithm
```

Thumbprint Generation

Thumbprints are computed by performing the SHA-1 hash of the following DER encoded ASN.1 structures:

- UnsignedCertificate
- UnsignedCertificateRevocationList
- UnsignedBrandCRLIdentifier

The hash is computed over the tag-length-value of the encoded structure. The Thumbprint is the same hash that is used to sign or verify a certificate or CRL or BCI.

Section 4
Root Certificate Distribution

Section Overview

Introduction

Validating a certificate chain depends on the possession of an authentic Root public key. The SET Root certificate is self-signed and linked to the next Root public key. The initial SET Root public key may be distributed with the SET software.

Organization

This section includes the following topics:

- Initial Root Certificate Verification and Distribution
- Root Certificate Update

Root certificate format

The SET Root certificate is a version 3 X.509 certificate containing the extensions described in “Required CA Certificate Extensions” on page 241. The same Root certificate is used for both CA certificate signing and CRL signing.

Initial Root Certificate Verification and Distribution

Certificate generation

Before the system is deployed, the following are generated:

- R1 = Root key pair #1
- C1 = certificate for Root key #1 (contains H2)
- R2 = Root key pair #2
- H2 = Thumbprint (hash) of the public component of R2

H2 is contained within the SET private extension, Hashed Root Key, in the Root certificate, C1. C1 is self-signed. C1 is distributed when the system is deployed. The HashedRootKey private extension is described on page 229.

Root key Distribution and Authentication

The SET Root certificate and its successors are delivered to the SET application via the Certificate Request protocol and the Payment protocol. The initial Root certificate, it's public key, or the hash of the public key may also be delivered with the SET application software. Whenever a new Root certificate is received by a SET application, the application shall verify that the Root certificate chains back (via the HashedRootKey extension) to a previously authenticated Root certificate. If the Root certificate does not chain back to an authenticated Root, it shall be verified for authenticity by checking one of the following.

- Verify that the Root certificate received matches one delivered with the SET application.
- Verify that the SubjectPublicKeyInfo of the received Root certificate matches the SubjectPublicKeyInfo from a Root certificate delivered with the SET application.
- Verify that the SHA-1 hash of the DER encoded SubjectPublicKeyInfo from the received Root certificate matches the same value obtained from the Root certificate delivered with the SET application.
- Verify that the SHA-1 hash (in hex) of the DER encoded SubjectPublicKeyInfo from the received Root certificate matches the value obtained from a trusted source (for example, a national newspaper, a software help desk, etc.) and entered by the End Entity.

Root Certificate Update

Root certificate update

When the time comes to replace the first Root certificate R1, the following are generated:

- R3 = public component of Root key #3
- H3 = thumbprint of R3
- C2 = certificate for Root key #2 (contains H3 in the SET private extension)

The new Root certificate is distributed electronically via SET messages and may also be distributed via other transport methods (HTTP, FTP, SMTP).

Validation of new Root certificate

The SET application:

- validates the signature applied using R2 and
- computes the hash of R2 and compares it to H2 (obtained from an extension in C1).

This is an iterative process with R4, C3, and H4 being generated and C3 (including H4) being distributed when it's time to replace C2.

Unscheduled Root Certificate Duplication

There are circumstances under which a Root certificate in the chain has to be duplicated with a different HashedRootKey extension. This will result in two Root certificates with the same subjectPublicKey and different HashedRootKey extensions each having a common predecessor Root certificate. The process of Certificate Chain Validation shall allow for the Root certificate chain to contain more than one successor of a single Root certificate and shall not assume that each Root certificate has a single successor.

Chapter 2
Certificate Request Protocols

Chapter Overview

Introduction

This chapter defines the Certificate Request protocols for Cardholders, Merchants, and Payment Gateways that allow them to obtain their original certificate and to renew certificates.

Organization

This chapter includes the following sections:

Section	Title	Contents	Page
1	Main Protocol	Describes the overall protocol for obtaining and renewing certificates.	135
2	Protocol Variations	Identifies variation of the protocol used when certain error conditions are encountered or when e-mail is used to obtain a certificate.	142
3	Cardholder Certificate Initiation Request/Response Processing	Defines how the certificate request process is started for a Cardholder.	146
4	Cardholder Registration Form Request/Response Processing	Defines how the Cardholder requests and obtains a registration form.	152
5	Merchant/Payment Gateway Certificate Initiation Processing	Defines how the certificate request process is started for a Merchant or an Payment Gateway.	162
6	Certificate Request and Generation Processing	Defines the processing associated with the CertReq message, the generation of the certificate, and the generation of the CertRes message.	173
7	Certificate Inquiry and Status Processing	Defines how the EE queries the CA to obtain the status of the certificate request.	197

Section 1
Main Protocol

Overview

Preface

This section defines the protocol and message processing for a Cardholder, Merchant, or Payment Gateway (that is, an End Entity) to request and obtain signature and/or data encryption X.509 certificates from a Certificate Authority (CA). The protocol defined herein shall be used whether the EE is requesting its first certificate or renewing a certificate.

Required Cardholder data

The Cardholder shall possess the following prior to requesting a certificate:

- An established valid Brand account.
- The ability to generate public/private key pairs and to securely store the private key.
- Knowledge of certain information to be used for purposes of identifying and authenticating the Cardholder as required by the payment card Issuer (Issuers will have different requirements for this information).
- The Universal Resource Locator (URL) or Internet mail address for the CCA.
- SET-compliant browser or bolt-on application.

Required Merchant data

The Merchant shall possess the following prior to requesting a certificate:

- An established valid Merchant ID with an Acquirer.
- The ability to generate public/private key pairs and to securely store the private key.
- Knowledge of information from the agreement between the Merchant and the Acquirer (Acquirers will have different requirements).
- The Universal Resource Locator (URL) or Internet mail address for the MCA.
- SET-compliant browser or bolt-on application.

Continued on next page

Overview, continued

Required Payment Gateway data

The Payment Gateway shall possess the following prior to requesting a certificate:

- The ability to generate public/private key pairs and to securely store the private key.
- Its Bank ID Number (BIN).
- Knowledge of certain information to be used for purposes of identifying and authenticating the Payment Gateway as required by the Acquirer (Acquirers will have different requirements).
- The Universal Resource Locator (URL) or Internet mail address for the PCA.
- SET-certified browser or bolt-on application.

Certificate protocol initiation

The certificate protocol is started differently depending on the underlying communications mechanism.

- On the World Wide Web, the SET application will receive an initiation message (not defined in this specification).
- The user of an e-mail application shall initiate the SET application locally.

Subsequent processing

After the SET application has been started by the user or kicked off by a Web certificate initiation message, exchanges described in the following sections take place between:

- the Cardholder and the CCA,
- the Merchant and the MCA, and
- the Payment Gateway and the PCA,

to generate or renew a signature and/or encryption certificate.

Continued on next page

Overview, continued

Cardholder / CCA processing

1. The Cardholder application sends a CardCInitReq to the CA, using the stored Brand ID or one obtained from the certificate initiation message.
2. The CCA sends a CardCInitRes to the SET application.
3. The Cardholder application sends a RegFormReq to the CCA.
4. The CCA sends a RegFormRes containing the registration template and the policy statement.
5. The Cardholder application displays the registration template and policy statement. The user enters the requested information and agrees to the policy.
6. The Cardholder application includes the filled-in registration form, the new public key, and the certificate being renewed , if applicable, in a CertReq, and sends it to the CCA.
7. The CCA generates the certificate.
8. The CCA includes the certificate in a CertRes and sends it to the Cardholder.

These exchanges are illustrated graphically in Figure 13 on page 139.

Merchant or Payment Gateway processing

1. The SET application sends a Me-AqCInitReq to the CA, using the BIN and Merchant ID obtained from the Merchant or Payment Gateway system administrator.
2. The CA sends an Me-AqCinitRes to the SET application, containing the registration form and policy statement.
3. The SET application displays the registration template and policy. The user enters the requested information and agrees to the policy statement.
4. The SET application includes the filled in registration information, the new public key(s), and the certificate(s) being renewed, if applicable, in a CertReq, and sends it to the CA.
5. The CA generates the certificate(s).
6. The CA includes the certificate(s) in a CertRes and sends it to the Merchant or Payment Gateway.

These exchanges are illustrated graphically in Figure 14 on page 140.

Detailed description of main protocol

The protocol

The figures on the following pages show the basic information exchanges between the Cardholder and the CCA, and the Merchant or Payment Gateway and the associated CA (MCA or PCA).

Figure 13 and Figure 14 illustrate the messages that shall be exchanged to renew or obtain a new certificate. The messages exchanged to obtain and submit a certificate registration form are different for the Cardholder than for the Merchant or Payment Gateway. The certificate request and response are the same for all End Entities.

Figure 15 shows the exchanges involved in a certificate inquiry. These are the same for all End Entities.

Continued on next page

Detailed description of main protocol, continued

Cardholder exchanges

Figure 13 below shows the exchanges for the Cardholder to register and obtain a new certificate or to renew a certificate.

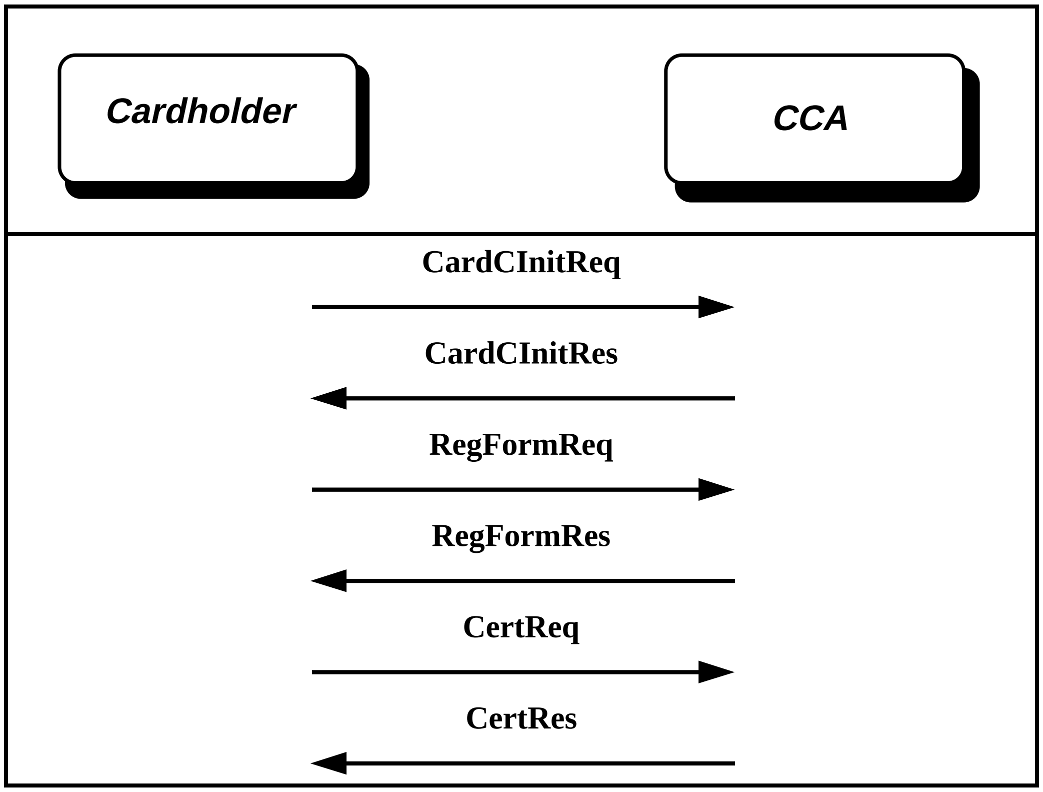


Figure 13: Cardholder Certificate Request Exchanges

Continued on next page

Detailed description of main protocol, continued

Merchant or Payment Gateway exchanges

Figure 14 below shows the exchanges for the Merchant or Payment Gateway to register and obtain a new certificate or to renew a certificate.

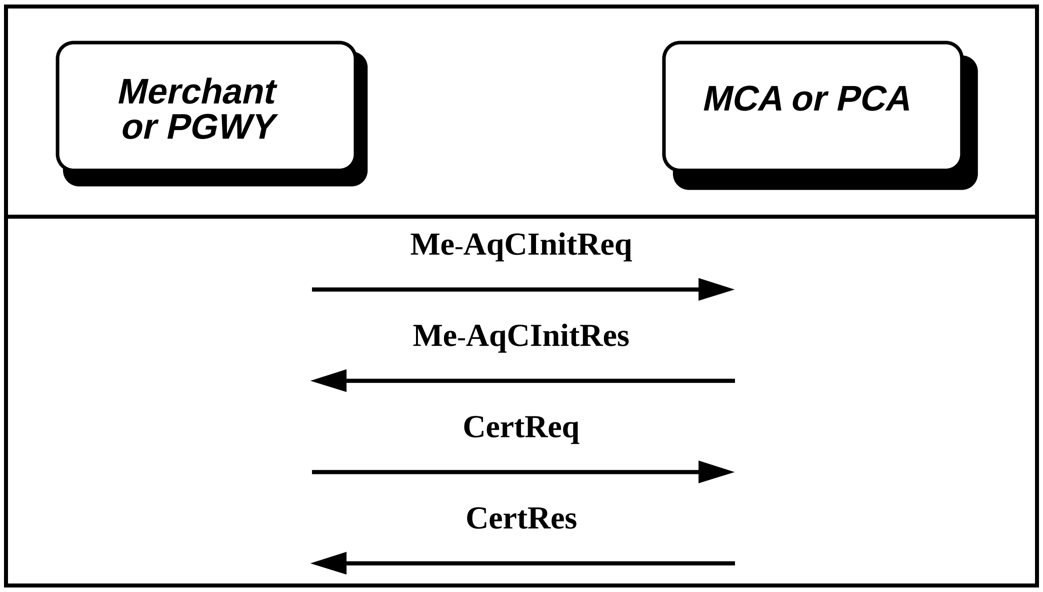


Figure 14: Merchant/Payment Gateway Certificate Request Exchanges

Continued on next page

Detailed description of main protocol, continued

Certificate inquiry exchange

If the CertRes indicates that the certificate is not ready, the EE may send a CertInqReq message to obtain the status of the request. The CertInqRes returns the certificate if it's ready, provides status if there was a problem with the certificate request, or indicates when the certificate will be ready for pickup.

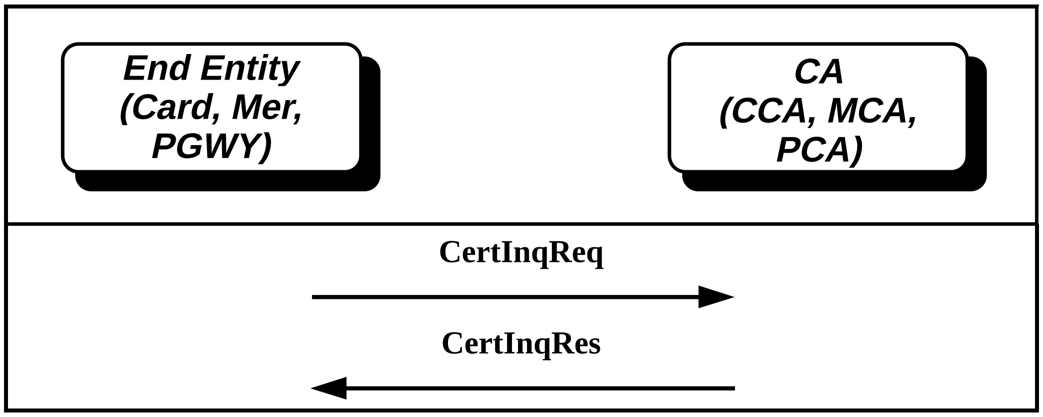


Figure 15: Certificate Inquiry Message Exchange

Section 2
Protocol Variations

Variations

Cardholder CertReq approved via e-mail

Figure 16 shows the protocol when a non-interactive communications mechanism like e-mail (SMTP) is used and the certificate request is approved. For the CardCInitReq/Res and RegFormReq/Res messages to be omitted from the protocol, the EE shall already be holding a registration form as well as the applicable CA certificates required to encrypt the CertReq.

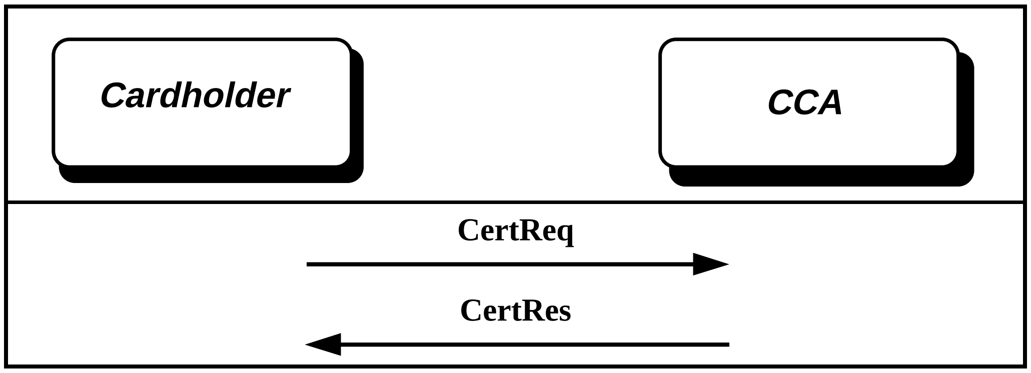


Figure 16: Cardholder CertReq via E-Mail

Continued on next page

Variations, continued

Cardholder CertReq approved via the World Wide Web

Figure 17 shows the protocol for a Cardholder when an interactive communications mechanism like the WWW is used and the certificate request is approved.

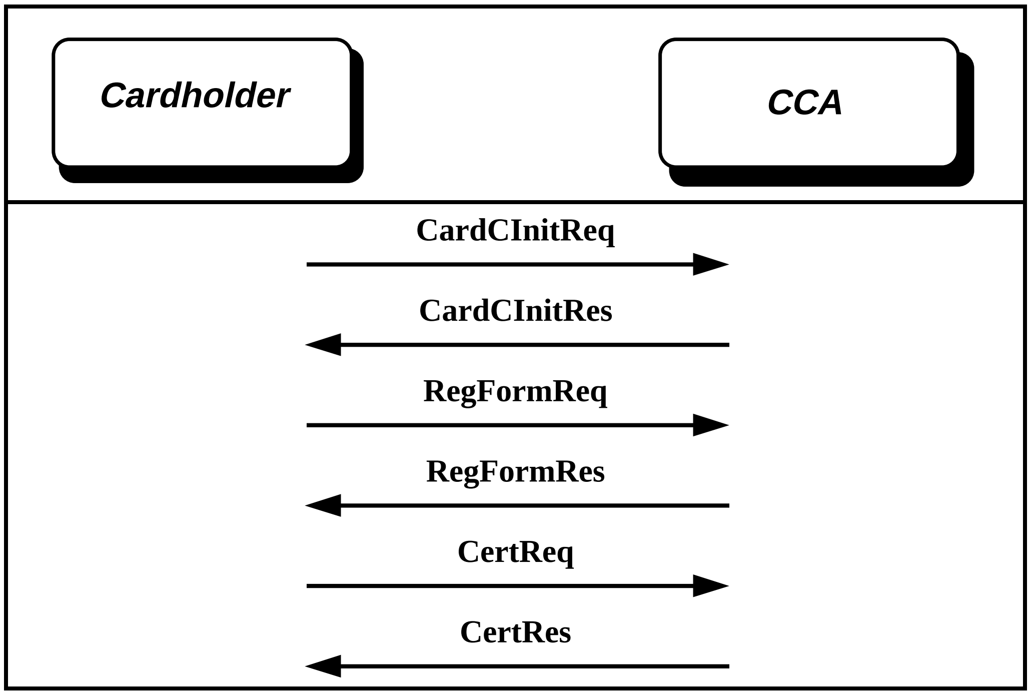


Figure 17: Cardholder CertReq via World Wide Web

Continued on next page

Variations, continued

Merchant or Payment Gateway CertReq approved

Figure 18 shows the protocol for a Merchant or an Payment Gateway when an interactive communications mechanism like WWW (HTTP) is used and the certificate request is approved.

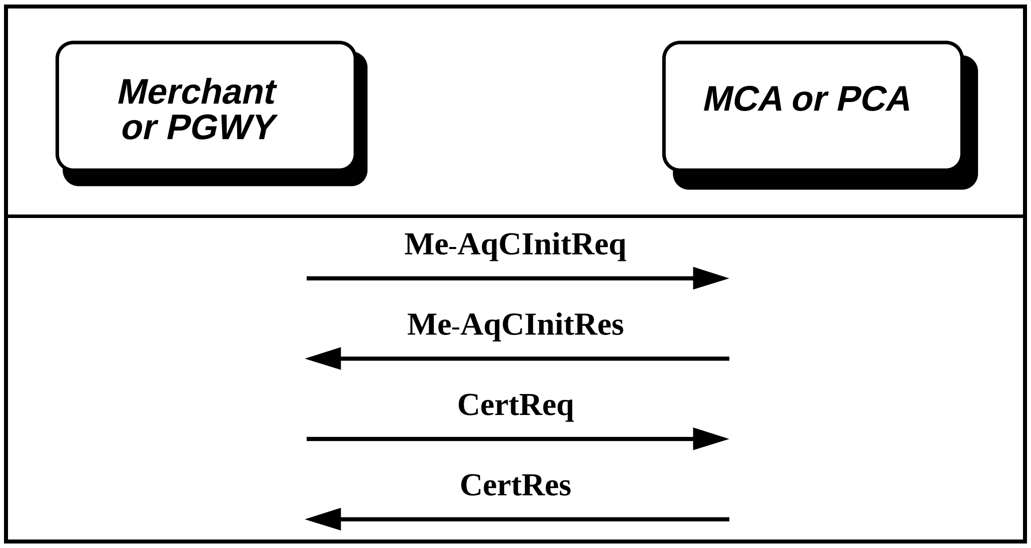


Figure 18: Merchant or Payment Gateway CertReq via e-mail

Continued on next page

Variations, continued

CertInqReq with CertInqRes

Figure 19 shows the certificate inquiry protocol when a certificate has been generated. This protocol is used when the original CertRes is pending and the SET application needs to obtain the certificate at a later time.

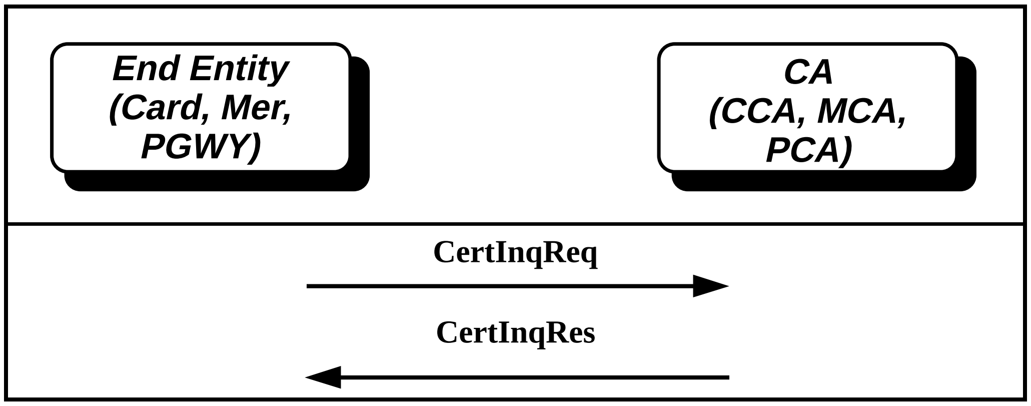


Figure 19: CertInqReq with CertInqRes

Section 3
Cardholder Certificate Initiation Request/Response Processing

Overview

Introduction

This section describes the certificate initiation process for the Cardholder. After the SET application has been started, the Cardholder sends a CardCInitReq to the CCA, indicating via Thumbprints the certificates, CRLs, and the BrandCRLIdentifier that are contained in its certificate cache. The CCA responds with a CardCInitRes containing any certificates, CRLs, and the BrandCRLIdentifier that the Cardholder will need for signature verification, as well as an encryption certificate to use for subsequent messages.

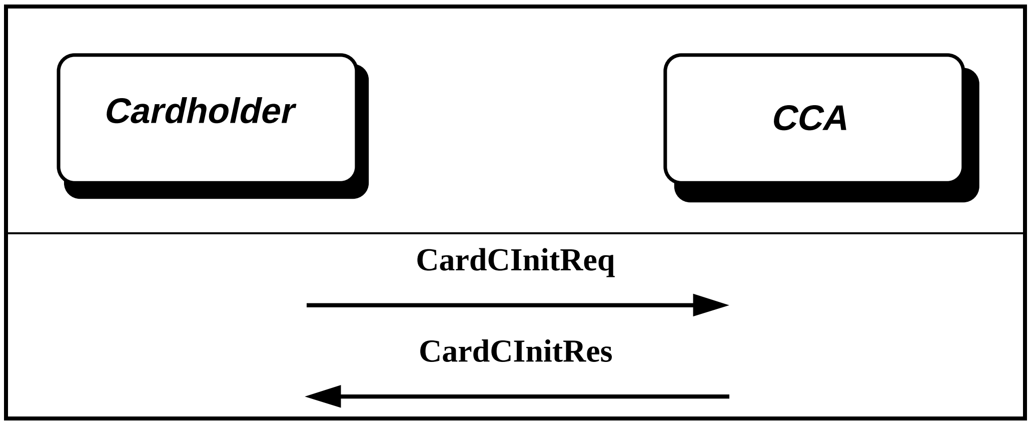


Figure 20: Cardholder Certificate Initiation Process

E-mail initiation

The certificate request protocol is initiated either directly by the user launching or by another application launching the SET application. No SET initiation message is necessary.

World Wide Web initiation

The certificate request protocol is initiated by the user performing a specific action (such as clicking a button on a Web page) that results in the Web server (the CCA in this case) creating and sending the SET initiation message to the EE. This SET message, containing the appropriate MIME type initiates the SET application.

Cardholder Generates CardCInitReq

Create CardCInitReq

The SET application shall perform the steps below to create a CardCInitReq message.

Step	Action
1	Generate an RRPID.
2	Generate LID-EE.
3	Generate a fresh random Chall-EE.
4	Copy the BrandID that's stored or was received in the initiation message.
5	Optionally populate Thumbs, which holds the thumbprints for each CRL, SET certificate, BrandCRLIdentifier, and Root certificate resident in the Cardholder's trusted cache, if any exist.
6	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CardCInitReq to the CCA.

CardCInitReq

CardCInitReq	**{RRPID, LID-EE, Chall-EE, BrandID, [Thumbs]}**
RRPID	*Request/response pair ID.*
LID-EE	*Local ID; generated by and for the Cardholder system.*
Chall-EE	*Cardholder's challenge to CCA's signature freshness.*
BrandID	*BrandID of certificate requested.*
Thumbs	*Lists of Certificate (including Root), CRL, and BrandCRLIdentifier thumbprints currently held by Cardholder.*

Table 5: CardCInitReq

CCA Processes CardCInitReq

CCA processing

When the CCA receives the CardCInitReq it shall process it as follows:

Step	Action
1	Receive the CardCInitReq from *Receive Message* (described in Part I on page 77).
2	Verify that the RRPID in the CardCInitReq matches the RRPID in the Message Wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
3	Store the Thumbs, LID-EE , Chall-EE, and RRPID to be used in the CardCInitRes.

CCA Generates CardCInitRes

CCA processing

After the CCA processes the CardCInitReq it shall perform the following steps to generate the signed CardCInitRes. As with any SignedData, the certificates and CRLs needed to verify the signature are included in the CardCInitRes outside of the "To Be Signed" data.

Step	Action
1	Build "CardCInitResTBS" data as follows: a. Copy the RRPID, LID-EE, and Chall-EE, from the values received in the CardCInitReq. b. Optionally generate LID-CA. d. Populate the CAEThumb with the thumbprint of the CCA's data encryption certificate. e. If the BrandCRLIdentifier is not specified in the Thumbs received in CardCInitReq, populate the BrandCRLIdentifier. f. Copy the Thumbs from the CardCInitReq.
2	Sign the DER encoded CardCInitResTBS, as described in the "Signature" processing steps on page 93. Set the content type of SignedData to be id-set-content-CardCInitResTBS. **Note:** Include in the *SignedData* any certificates, CRLs, or the BrandCRLIdentifier that is not indicated by the Thumbs and that the Cardholder may need to verify the CCA's signature or to encrypt the RegFormReq and the CertReq.
3	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CardCInitRes to the Cardholder.

Continued on next page

CCA Generates CardCInitRes, continued

CardCInitRes

CardCInitRes	**S(CA, CardCInitResTBS)**.
CardCInitResTBS	**{RRPID, LID-EE, Chall-EE, [LID-CA], CAEThumb, [BrandCRLIdentifier], [Thumbs]}**
RRPID	*Request/response pair ID.*
LID-EE	*Copied from* **CardCInitReq**.
Chall-EE	*Copied from* **CardCInitReq**.
LID-CA	*Local ID; Generated by and for the CCA system.*
CAEThumb	*Thumbprint of CCA key-exchange certificate that Cardholder should use to encrypt* **RegFormReq**.
BrandCRLIdentifier	*See page 249.*
Thumbs	*Copied from* **CardCInitReq**.

Table 6: CardCInitRes

Cardholder Processes CardCInitRes

Processing

The Cardholder application shall process the CardCInitRes as follows.

Step	Action
1	Receive the CardCInitRes from *Receive Message* . **Note:** The processing performed on the received certificates, CRLs, and BrandCRLIdentifier is described in "Receive Message" (Part I, page 77).
2	Verify that the RRPID matches the one sent in the CardCInitReq and the one received in the CardCInitRes message wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
3	Verify, as specified in the "Thumbprint" processing steps (described in Part I on page 79), that the Thumbs received match those sent in the CardCInitReq message. If it does not verify, return an Error Message with ErrorCode set to thumbsMismatch.
4	Verify that the Chall-EE received is equal to the one sent in the CardCInitReq. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
5	If it was included, store the received LID-CA to return in the RegFormReq. Verify that the Chall-EE received is equal to the one sent in the CardCInitReq.
6	Verify that the Cardholder application supports one of the algorithms indicated in the Tunneling extension in the CA's encryption certificate. If the Cardholder application does not support a common encryption algorithm with the CA, notify the user and abort further CA message processing.

Section 4
Cardholder Registration Form Request/Response Processing

Introduction

Following the receipt of the appropriate certificates, CRLs, and the BrandCRLIdentifer, the Cardholder can then securely request a certificate registration form via the RegFormReq. If the CCA successfully validates the registration form request, it returns the form in the RegFormRes. If the CCA does not have a registration form for the Cardholder's request and/or has additional information concerning the service request denial to convey to the Cardholder, it is also indicated in the RegFormRes.

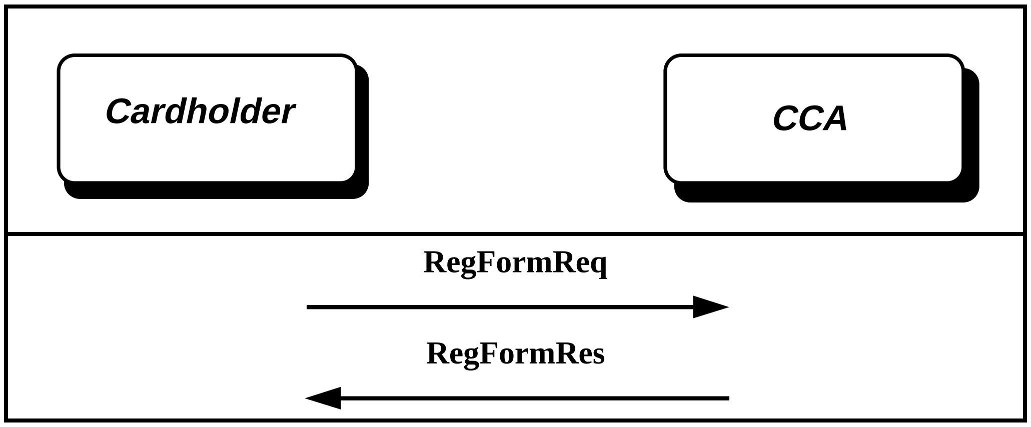


Figure 21: Cardholder Registration Form Processing

Cardholder Generates RegFormReq

Cardholder processing

Following successful processing of the CardCInitRes, the Cardholder application shall generate the RegFormReq using the steps below. The RegFormReq is encrypted by the Cardholder application using the certificate received from the CCA in the CardCInitRes.

Step	Action
1	Build "RegFormReqData" as follows: a. Generate a new RRPID. b. Copy the LID-EE sent in the CardCInitReq. c. Generate a fresh Chall-EE2. d. If one was included in the CardCInitRes, copy LID-CA if one was included in the CardCInitRes. e. Populate the RequestType, according to Table 8: Cardholder Registration Form RequestType Values on page 155. f. Populate the Language. g. Optionally include Thumbs, which holds the thumbprints for each CRL, SET certificate, BrandCRLIdentifier, and Root certificate resident in the Cardholder's trusted cache (Thumbs), if any exist.
2	Build "RegFormReqTBE" as follows: a. Insert RegFormReqData. b. Populate PANOnly using the PAN and ExNonce. The PAN is not padded. c. Generate the SHA-1 hash of the DER encoded PANOnly. Set the content type of digestedData to id-set-content-PANOnly.
3	Build the "To Be Extra Encrypted" Data as follows: a. Populate the PAN. If the PAN is less than nineteen bytes, pad out to nineteen bytes. b. Generate a newly generated nonce, EXNonce, to mask the PAN.
4	Encrypt the data using the *EXH* processing (described in Part I on page 90) with a. RegFormReqTBE as "To Be Ordinarily Encrypted" data and the contentType of EnvelopedData to id-set-content-RegFormReqTBE and b. result of step 3 as "To Be Extra Encrypted" data. **Note:** Extra encrypt the data using the CCA's key encryption certificate identified in the CardCInitRes by CAEThumb.
5	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the RegFormReq to the CA.

Continued on next page

Cardholder Generates RegFormReq, continued

RegFormReq

RegFormReq	**EXH(CA, RegFormReqData, PANOnly)**
RegFormReqData	**{RRPID, LID-EE, Chall-EE2, [LID-CA], RequestType, Language, [Thumbs]}**
PANOnly	*See below.*
RRPID	*Request/response pair ID.*
LID-EE	*Copied from* **CardCInitRes**.
Chall-EE2	*EE's challenge to CA's signature freshness.*
LID-CA	*Copied from* **CardCInitRes**.
RequestType	*See page 155.*
Language	*Desired natural language for the rest of this flow.*
Thumbs	*Lists of Certificate (including Root), CRL, and BrandCRLIdentifier currently held by Cardholder.*

Table 7: RegFormReq

PANOnly data

The PANOnly is comprised of the following fields:

Field Name	Description
PAN	*Cardholder's Payment Card Number.*
EXNonce	*random number used to mask the PAN.*

Continued on next page

Cardholder Generates RegFormReq, continued

RequestType

RequestType can have any one of the values shown in Table 8 below.

Request Type	Signature Cert only	Encryption Cert only	Both Certs
Cardholder Initial	**1**	**2***	**3***
Cardholder Renewal	**10**	**11***	**12***

Table 8: Cardholder Registration Form RequestType Values

Note: The * indicates options that are reserved for future versions of SET

Additional restrictions

The following additional restrictions apply to the request types above:

Request Type	Restrictions
2*	Shall have a valid Signature certificate and shall use the corresponding private key to sign the request for an Encryption certificate.
10, 12	Both the private key corresponding to the certificate being renewed and the private key of the new signature certificate shall be used to sign the renewal request.
11*	Renewal of Encryption certificates: the certificate Subject distinguished names of the Signature certificate (used to sign the request) and Encryption certificate shall match.

CCA Processes RegFormReq

CCA processing

When the CCA receives the RegFormReq it shall perform the following steps to validate the message and determine if a registration form will be returned.

Step	Action
1	Receive the RegFormReq message from *Receive Message* (described in Part I on page 77).
2	Store the PAN from the "To Be Extra Encrypted" data, after removing any extra padding.
3	From the "RegFormReqData" data, store the RequestType, RRPID, LID-EE, Chall-EE2, LID-CA, Thumbs, and Language.
4	Verify that the RRPID received in the message wrapper matches the one in the RegFormReqTBE If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.

CCA Generates RegFormRes

CCA processing

Following validation of the RegFormReq, the CCA shall generate the RegFormRes as follows. If validation of the request was successful, the registration form, policy statement, and URLs for brand and card logos are returned. If validation was unsuccessful, a reason and optionally a URL or e-mail address for more information will be returned.

Step	Action
1	Generate "RegFormTBS" as follows: 1. Copy the RRPID, RequestType, LID-EE, Thumbs, and Chall-EE2 from the RegFormReqData. 2. If the LID-CA is provided in the CardCInitRes, copy the LID-CA, otherwise, optionally generate a LID-CA for this service request. 3. Generate a fresh Chall-CA. 4. If the BrandCRLIdentifier is not specified in the Thumbs received in RegFormReq, populate the BrandCRLIdentifier. 5. If a Cardholder registration form is available for the PAN, Language and RequestType, build RegFormData as follows: a) populate the RegTemplate and PolicyText corresponding to the RequestType, PAN, and Language, i) include the RegFormID and RegFieldSeq. The RegFieldSeq may be omitted in the case of a renewal. ii) optionally include URLs for displaying the Brand and/or Card Logos. b) the CertReq is to be encrypted with a different key than was used to encrypt the RegFormReq,populate CAEThumb with a different thumbprint than was sent in the CardCInitRes. 6. If an appropriate Cardholder registration form is not available, populate ReferralData as follows: a) populate the Reason with the service denial information that will be displayed to the Cardholder, and b) optionally populate the ReferralLoc with an e-Mail address and/or URLs where the user can obtain more information concerning the service denial.

Continued on next page

CCA Generates RegFormRes, continued

CCA processing (continued)

Step	Action
2	Sign the result of step 1 (that is, the "RegFormTBS" data) according to *Signature Processing* on page 93. Set the contentType of SignedData to id-set-content-RegFormTBS.
3	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the RegFormRes message to the Cardholder.

RegFormRes

RegFormRes	**S(CA, RegFormResTBS)**
RegFormResTBS	**{RRPID, LID-EE, Chall-EE2, [LID-CA], Chall-CA, [CAEThumb], RequestType, RegFormOrReferral, [BrandCRLIdentifier], [Thumbs]}**
RRPID	*Request/response pair ID.*
LID-EE	*Copied from* **RegFormReq**.
Chall-EE2	*Copied from* **RegFormReq**.
LID-CA	*Local ID; generated by and for CA system (new value may be specified).*
Chall-CA	*CA's challenge to requester's signature freshness.*
CAEThumb	*Thumbprint of CA key-exchange certificate that should be used to encrypt* **CertReq***; if this field is not present, the certificate identified in* **CardCInitRes** *is used.*
RequestType	*See page 155*
RegFormOrReferral	*See page 159.*
BrandCRLIdentifier	*See page 249.*
Thumbs	*Copied from* **RegFormReq**.

Table 9: RegFormRes

Continued on next page

CCA Generates RegFormRes, continued

RegFormOrReferral

RegFormOrReferral	**< RegFormData, ReferralData >**
RegFormData	**{[RegTemplate], PolicyText}**
ReferralData	**{[Reason], [ReferralURLSeq]}**
RegTemplate	**{RegFormID, [BrandLogoURL], [CardLogoURL], RegFieldSeq}**
PolicyText	*Statement to be displayed along with* **RegTemplate** *on requester's system.*
Reason	*Statement concerning request to be displayed on requester's system.*
ReferralURLSeq	**{ReferralURL +}** *Optional URLs pointing to referral information, listed in the order of relevance.*
RegFormID	*CA-assigned identifier.*
BrandLogoURL	*The URL for the payment card brand logo.*
CardLogoURL	*The URL for the financial institution logo.*
RegFieldSeq	**{RegField +}**
ReferralURL	*Uniform Resource Locator of alternate CA for processing of certificate requests for this entity.*
RegField	**{[FieldId], FieldName, [FieldDesc], [FieldLen], FieldRequired, FieldInvisible}**
FieldID	*See Appendix L: Object Identifiers for Registration Form* Fields *in SET Book 2: Programmer's Guide.*
FieldName	*One or more field names to be displayed as labels for a fill-in form on requester's system; text is in the language specified in* **RegFormReq** *or* **Me-AqCInitReq**.
FieldDesc	*Description of contents of field in the language specified in* **RegFormReq** *or* **Me-AqCInitReq**; *contains additional information for use when the cardholder requests help filling out the form.*
FieldLen	*Maximum length of field.*
FieldRequired	*Boolean indicating whether data is required (either entered by the Cardholder or, if the field is invisible, populated by the application).*
FieldInvisible	*Boolean indicating that the field should not be displayed to the user; the application should either fill in the* **FieldValue** *based on* **FieldID** *or leave it empty.*

Table 10: RegFormOrReferral

Cardholder Processes RegFormRes

Cardholder processing

The Cardholder application shall process the RegFormRes as follows:

Step	Action
1	Receive the RegFormRes message from *Receive Message* (described in Part I on page 77).
2	Verify the signature. If it does not verify, return an Error Message with ErrorCode set to signatureFailure.
3	Obtain the RRPID, RequestType, LID-EE, Chall-EE2, CAEThumb from "RegFormTBS".
4	Verify that the RRPID is the same as the one received in the message wrapper and sent in the RegFormReq. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
5	Verify that the RequestType, LID-EE , and Chall-EE2 are the same as those sent in the RegFormReq. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
6	If a CAEThumb was included, store the corresponding Encryption certificate to be used for encrypting the CertReq.
7	Verify, as specified in the "Thumbprint" processing steps (described in Part I on page 79), that the Thumbs received match those sent in the CardCInitReq message. If it does not verify, return an Error Message with ErrorCode set to thumbsMismatch.
8	If the RegFormData is included in the "RegFormTBS" data: a) Store the LID-CA . b) Display the policy text and require user acknowledgment before the SET application generates a CertReq. c) Display the visible fields in the registration form and prompt the user to fill in the fields. d) If the RegFormRes contains URL(s), display the Brand and/or Card Logos. e) Populate any invisible fields in the registration form. If a field is required and invisible and the application cannot populate the field, the field shall be left empty and the remainder of the registration form shall be populated and transmitted in the CertReq as specified. f) After the user has completed the registration form generate a CertReq.

Continued on next page

Cardholder Processes RegFormRes, continued

Cardholder processing (continued)

Step	Action
9	If the ReferralData is included in the "RegFormResTBS" data: a. Display the Reason. b. If the ReferralLoc is included, display the URLs or e-mail address from ReferralLoc. c. Do not generate a CertReq. The protocol shall start over at the CardCInitReq.

Section 5
Merchant/Payment Gateway Certificate Initiation Processing

Overview

Introduction

The Me-AqCInitReq/Res message pair is used by the Merchant or Payment Gateway to obtain a certificate registration form. The Merchant or Payment Gateway starts the certificate protocol by sending the Me-AqCInitReq. The Me-AqCInitReq contains the bank information for the Merchant or Payment Gateway, the type of certificate being requested, and the certificates, CRLs, and the BrandCRLIdentifier that are in the trusted certificate cache. If the MCA or PCA has a registration form in the correct language for the indicated bank, it is returned in the Me-AqCInitRes along with any certificates, CRLs, and the BrandCRLIdentifier that the Merchant or Payment Gateway will need for signature verification. If the MCA or PCA does not have a registration form and/or has additional information concerning the service request denial to convey to the Merchant or Payment Gateway, it is also indicated in the Me-AqCInitRes. The certificate protocol is started by the Merchant or Payment Gateway as shown in Figure 22 below. Following receipt of the Me-AqCInitRes containing a registration form, the EE may send a CertReq containing the completed form.

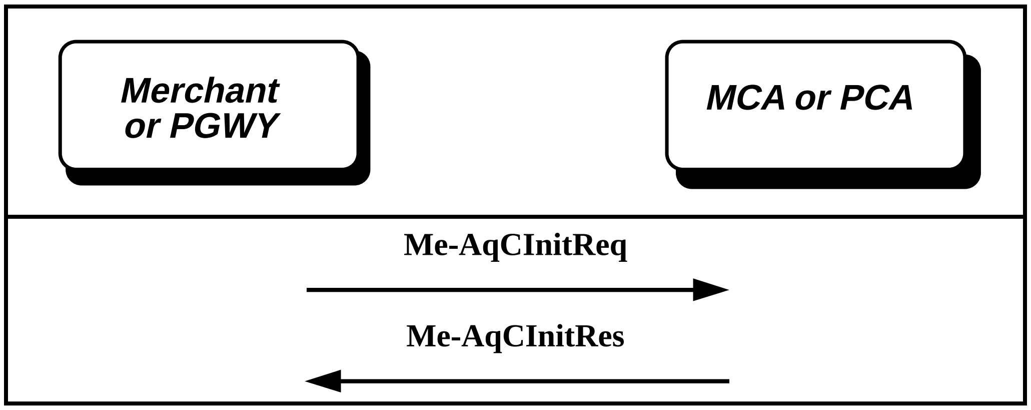


Figure 22: Merchant/Payment Gateway Certificate Initiation Processing

Merchant/Payment Gateway creates Me-AqCInitReq

Create Me-AqCInitReq

The SET application shall generate the Me-AqCInitReq as follows:

Step	Action
1	Generate a new RRPID.
2	Generate a fresh LID-EE.
3	Generate a fresh random Chall-EE.
4	Populate the BrandID that's stored or was received in the SET initialization message.
5	Populate the RequestType.
6	Populate the Language.
7	Optionally create the thumbprints for each crl, set certificate, BrandCRLIdentifier, and root certificate resident in its trusted cache, if any exist.
8	If the EE is a Merchant, populate the Merchant's BIN and ID. Otherwise, populate the Acquirer's BIN and optionally populate the Acquirer's business ID.
9	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the Me-AqCInitReq to the CA.

Continued on next page

Merchant/Payment Gateway creates Me-AqCInitReq, continued

Me-AqCInitReq

Me-AqCInitReq	**{RRPID, LID-EE, Chall-EE, RequestType, IDData, BrandID, Language, [Thumbs]}**
RRPID	*Request/response pair ID.*
LID-EE	*Local ID; generated by and for EE system.*
Chall-EE	*EE's challenge to CA's signature freshness.*
RequestType	*See next page.*
IDData	*See below.*
BrandID	*BrandID of certificate requested.*
Language	*Desired natural language for the rest of this flow.*
Thumbs	*Lists of Certificate (including Root), CRL, and BrandCRLIdentifier currently held by EE.*

Table 11: Me-AqCInitReq

IDData

IDData	**< MerchantAcquirerID, AcquirerID >** *Only for Merchants and Acquirers*
MerchantAcquirerID	**{MerchantBIN, MerchantID}**
AcquirerID	**{AcquirerBIN, [AcquirerBusinessID]}**
MerchantBIN	*Bank Identification Number for the processing of Merchant's transactions at the Acquirer*
MerchantID	*Merchant ID assigned by Acquirer*
AcquirerBIN	*The Bank Identification Number of this Acquirer*
AcquirerBusinessID	*The Business Identification Number of this Acquirer*

Table 12: IDData

Continued on next page

Merchant/Payment Gateway creates Me-AqCInitReq, continued

RequestType

The RequestType for the Merchant or Payment Gateway can have any one of the following values.

Request Type	Signature Cert only	Encryption Cert only	Both Certs
Merchant Initial	**4**	**5**	**6**
Payment Gateway Initial	**7**	**8**	**9**
Merchant Renewal	**13**	**14**	**15**
Payment Gateway Renewal	**16**	**17**	**18**

Table 13: Merchant/Acquirer Certificate RequestType Values

Additional restrictions

The Merchant or Payment Gateway shall either have a Signature certificate or be requesting one so that it can sign the CertReq. The following additional restrictions apply to the request types above:

Request Type	Restrictions
5, 8	Shall have a valid Signature certificate to sign the request for an encryption certificate.
14, 15, 17, 18	Renewal of Encryption certificates: the certificate Subject distinguished names of the Signature certificate (used to sign the request) and Encryption certificate shall match.
13, 15, 16, 18	Both the private key corresponding to the signature certificate being renewed and the private key of the new signature certificate shall be used to sign the renewal request.

CA Processes Me-AqCInitReq

CA processing

The CA receives the Me-AqCInitReq and shall process it as follows:

Step	Action
1	Receive the Me-AqCInitReq message from *Receive Message* (described in Part I on page 77).
2	Verify that the RRPID received in the message wrapper matches the one received in the Me-AqCInitReq. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
3	Store the RRPID, LID-EE, Chall-EE, BrandID, Language, Thumbs, and IDData.

CA Generates Me-AqCInitRes

CA processing

Following validation of the Me-AqCInitReq, the CA shall generate the Me-AqCInitRes according to the following steps. If the request is successful, the registration form, policy statement , and URLs for brand and card logos are returned. If the request is unsuccessful, a reason and optionally a URL or e-mail address for more information will be returned.

Continued on next page

CA Generates Me-AqCInitRes, continued

CA processing (continued)

Step	Action
1	Build "Me-AqCInitResTBS" as follows: 1. Copy the RRPID, LID-EE and Chall-EE from the Me-AqCInitReq. 2. Optionally generate a LID-CA for this service request. 3. Generate a fresh Chall-CA. 4. If a Merchant or Payment Gateway registration form is available for the BIN, RequestType and Language: a) populate the RegFormData as follows: retrieve the RegTemplate and PolicyText corresponding to the RequestType, BIN, and Language, i) optionally include URLs for displaying the Brand and/or Card Logos, ii) include the RegFormID and RegFieldSeq. The RegFieldSeq may be omitted in the case of a renewal. b) If the CA authenticates the Merchant or Payment Gateway via the AcctData, populate the AcctDataField indicating the name of the data to be entered, a description, a length, and whether the field shall be entered by the EE. 5. If an appropriate Merchant or Payment Gateway registration form is not available: populate ReferralData as follows: a) include the Reason for the service denial that will be displayed by the Merchant or Payment Gateway, and b) optionally include, in the ReferralLoc, an e-Mail address and/or URLs where the user can obtain more information concerning the service denial. 6. Include the Thumbprint of the CA's key encryption certificate, CAEThumb. 7. If the BrandCRLIdentifier is not specified in the Thumbs received in Me-AqCInitReq, populate the BrandCRLIdentifier. 8. Copy the Thumbs from the Me-AqCInitReq. 9. Copy the RequestType received in the Me-AqCInitReq. 14.

Continued on next page

CA Generates Me-AqCInitRes, continued

CA processing (continued)

2	Sign Me-AqCInitResTBS according to *Signature Processing* on page 93. Set the content type of SignedData to id-set-content-Me-AqCInitResTBS.
3	Invoke *Compose Message Wrapper* (described on page 76) to send the Me-AqCInitRes to the Merchant or Acquirer.

Continued on next page

CA Generates Me-AqCInitRes, continued

Registration Form Template

The MCA or PCA uses the same registration form template specified for the CCA:

Me-AqCInitRes

Me-AqCInitRes	**S(CA, Me-AqCInitResTBS)**
Me-AqCInitResTBS	**{RRPID, LID-EE, Chall-EE, [LID-CA], Chall-CA, RequestType, RegFormOrReferral, [AcctDataField], CAEThumb, [BrandCRLIdentifier], [Thumbs]}**
RRPID	*Request/response pair ID.*
LID-EE	*Copied from* **Me-AqCInitReq**.
Chall-EE	*Copied from* **Me-AqCInitReq**.
LID-CA	*Local ID; generated by and for CA system.*
Chall-CA	*CA's challenge to EE's signature freshness.*
RequestType	*See page 165.*
RegFormOrReferral	*See page 159.*
AcctDataField	**RegField** *(see* "RegFormOrReferral" *on page 159); an additional registration field to be displayed to collect the value for* **AcctData** *in* **CertReq**.
CAEThumb	*Thumbprint of CA key-exchange certificate that should be used to encrypt* **CertReq**.
BrandCRLIdentifier	*See page 252.*
Thumbs	*Copied from* **Me-AqCInitReq**.

Table 14: Me-AqCInitRes

Merchant/Acquirer Processes Me-AqCInitRes

Me-AqCInitRes processing

The SET application shall process the Me-AqCInitRes as follows.

Step	Action
1	Receive the Me-AqCInitRes message from *Receive Message* (described in Part I on page 77). **Note:** The processing performed on the received certificates, CRLs, and BrandCRLIdentifier is described in the *Receive Message* processing steps (in Part I on page 77).
2	Verify the signature. If it does not verify, return an Error Message with ErrorCode set to signatureFailure.
3	From the "Me-AqCInitResTBS" data, store the RRPID, LID-EE, Chall-EE, CAEThumb, BrandCRLIdentifier, Thumbs and RequestType.
4	Verify that the RRPID matches the one in the message wrapper and the one sent in the Me-AqCInitReq. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
5	Verify that the Chall-EE received matches the one sent in the Me-AqCInitReq. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
6	Verify that the Thumbs received match those sent in the Me-AqCInitReq message. If it does not verify, return an Error Message with ErrorCode set to thumbsMismatch.

Continued on next page

Merchant/Acquirer Processes Me-AqCInitRes, continued

Me-AqCInitRes processing (continued)

Step	Action
7	If the RegFormData is included in Me-AqCInitResTBS: a) Store the LID-CA and the Chall-CA. b) Display the policy text and require the user to acknowledge before the SET application will generate a CertReq. c) Display the visible fields in the registration form and prompt the user to fill in the fields. d) If the Me-AqCInitResTBS contains URL(s), display the Brand and/or Card Logos. e) If the AcctDataField is present, display the Name and Description and prompt the user to fill in the field. f) Populate any invisible fields in the registration form. If a field is required and invisible and the application cannot populate the field, the field shall be left empty and the remainder of the registration form shall be populated and transmitted in the CertReq as specified.
8	After the Merchant or Payment Gateway has completed the registration form and entered the AcctData, if applicable, generate a CertReq.
9	If ReferralData is included in Me-AqCInitResTBS data: a) Display the Reason. b) If the ReferralLoc is included, display the URLs or e-mail address from ReferralLoc. c) Do not generate a CertReq. The protocol shall start over at the Me-AqCInitReq.

Section 6
Certificate Request and Generation Processing

Overview

Introduction

The Cardholder, Merchant System Administrator, or Payment Gateway System Administrator enters the information needed by the RegForm and the SET application sends the CertReq message to the CA. Following successful validation of the CertReq, the generated certificate(s) are returned to the EE in a CertRes. If there are any errors in the registration form, the CA indicates this in the CertRes. The SET application can re-submit the corrected registration form in a new CertReq.

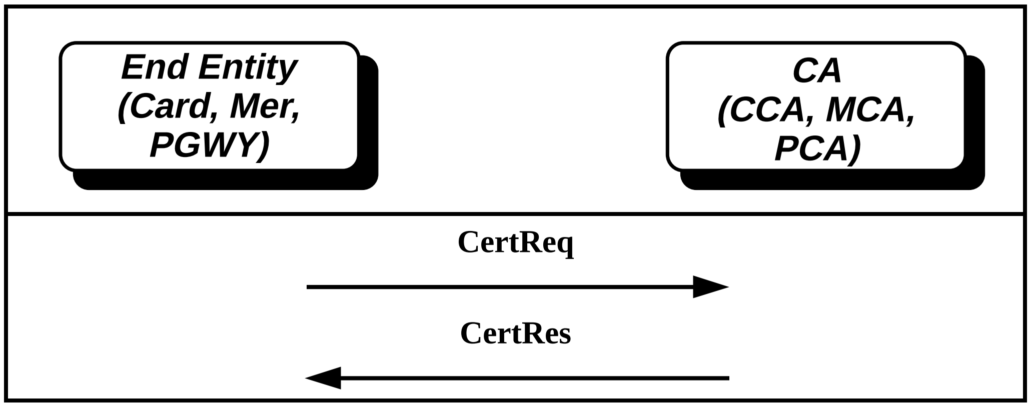


Figure 23: Certificate Request and Generation Processing

End Entity Generates CertReq

Cardholder input

The Cardholder enters their payment card number, expiration date, and other information requested by the CCA (contained in the registration form).

Merchant input

The Merchant System Administrator enters the Merchant authentication data (if any) and the other information requested by the MCA (contained in the registration form).

Payment Gateway input

The Payment Gateway System Administrator enters the Payment Gateway authentication data (if any) and the other information requested by the PCA (contained in the registration form).

CertReq

The Certificate Request (CertReq) contains:

- the new public keys,
- the certificates being renewed, if applicable,
- the filled-in registration form,
- EE account information,
- secret keys to be used by the CA to encrypt the Certificate Response (CertRes),
- other reference numbers and challenges.

The payload of the message and optionally a hash of the EE account information is signed using the private key corresponding to the signature certificate being renewed, if it exists, and the new signature private key. The signed data and the signatures are then encrypted using a symmetric algorithm. The symmetric key used for this encryption is OAEP'd along with the EE account information, if present, and the result is encrypted using a public key algorithm.

CertReq re-submission

If the CA finds errors in the submitted registration form, they are indicated in the CertRes and a corrected registration form may be re-submitted in a new CertReq.

Continued on next page

End Entity Generates CertReq, continued

CertReq

The EE application shall generate the CertReq as specified below. The CertReq is generated using EncX or Enc processing depending on the presence of AcctInfo. If the EE is a Cardholder, the AcctInfo always contains the PAN and EncX is always used. If the EE is a Merchant or a Payment Gateway, AcctInfo contains authentication data that may or may not be required by the CA. The Me-AqCInitRes indicates if AcctInfo is required in the AcctInfoField. EncX is only used if AcctInfo is present.

If the CertReq is being re-submitted with the corrected registration form, the value for Chall-EE3 and RRPID shall be re-generated for the re-submitted CertReq.

Continued on next page

End Entity Generates CertReq, continued

CertReq (EncX) Generation

If the EE application is for a Cardholder, or a Merchant or a Payment Gateway with AcctInfo to send, the CertReq shall be generated using EncX processing as follows:

Step	Action
1	If the RequestType is for a new or renewed signature certificate, generate a private/public key pair for the signature certificate.
2	If the requesting entity is not a Cardholder and if the RequestType is for a new or renewed encryption certificate, generate a private/public key pair for the encryption certificate.
3	If the EE is a Cardholder, generate a 160-bit random number, CardSecret.
4	Generate a 160-bit random number, EXNonce.
5	Build the CertReqTBS as follows: a) Generate a new RRPID. b) If the EE received a RegFormRes or a Me-AqCInitRes, copy the RequestType from that message; otherwise populate the RequestType. c) Populate the RequestDate as the current date. d) Copy LID-EE from a previous message. If one doesn't exist, generate a new one. e) Generate a fresh Chall-EE3. f) Copy LID-CA, if included, and Chall-CA from a previous message, if one exists. g) If the EE is a Merchant or Payment Gateway: • populate the IDData, and • if the AcctDataField was included in the Me-AqCInitRes and was a required field, include the AcctData entered by the EE. h) If the EE is a Cardholder, populate the PAN, CardExpiry, and CardSecret. i) Generate EXNonce. j) Copy the RegForm ID that was sent in the RegFormRes or Me-AqCInitRes. k) If the RegFieldSeq was present in the Me-AqCInitRes or RegFormRes, include the new or corrected RegForm.

Continued on next page

End Entity Generates CertReq, continued

CertReq (EncX) Generation, (continued)

Step	Action
6	a) If a Cardholder application, select, from the "Tunneling" private extension in the CA key encryption certificate, a common preferred encryption algorithm for the CA to use to encrypt the CertRes. Populate the algorithm ID and a key in CaBackKeyData. If a common algorithm is not found, abort processing and notify the user. b) Populate the newly generated public keys, PublicKeyS and/or PublicKeyE, for the CA to certify. c) If the EE is a Merchant or Payment Gateway and the request type is for the renewal of an Encryption certificate, populate EEThumb with the thumbprint of the certificate being renewed. If the request type is for the renewal of a Signature certificate, a thumbprint of the Signature certificate being renewed is not required because the CertReq is signed with it. d) Optionally include Thumbs, which holds the thumbprints for each CRL, SET certificate, BrandCRLIdentifier, and Root certificate resident in the Cardholder's trusted cache (Thumbs), if any exist.
7	Next, format the "To Be Extra Encrypted" data: If the EE is a Cardholder, populate the PAN, CardExpiry, CardSecretCardNonce, and EXNonce in PANData0. If the EE is a Merchant or Payment Gateway, optionally populate AcctData if it is required and EXNonce.

Continued on next page

End Entity Generates CertReq, continued

CertReq (EncX) Generation, (continued)

Step	Action	
8.	Envelope the data using *EncX* encapsulation (as described in Part I on page 83):	
	Include:	**Processing**
	a. CertReqTBS as "To Be Signed" data, and	How the data is signed depends on the RequestType. There is a minimum of one and possibly two signatures, i.e. SignerInfos, on a CertReq. If the Request Type is for a new Signature certificate, sign the data using the private key corresponding to the public key contained in PublicKeyS. If the Request Type is for a renewed Signature certificate, sign the data using the private key corresponding to the public key contained in PublicKeyS, and using the private key corresponding to the certificate being renewed. If the request type is for an Encryption certificate, sign the data using the private key corresponding to an existing signature certificate. If the data is signed with a private key that does not yet correspond to a certificate, set the SignerInfo.SerialNumber to zero and the Signer Info.IssuerName to the "Null-DN", i.e., the RDNSequence is an encoded NULL. Also, set the content type of SignedData to id-set-content-CertReqTBS.
	b. Result of step 6 as "To Be Extra Encrypted" data.	"Extra" encrypt using the CA certificate indicated by CAEThumb in the CardCInitRes or RegFormRes, if one was included, or Me-AqCInitRes.
	c. CertReqTBEX as "To Be Ordinarily Encrypted" data.	Encrypt CertReqTBEX and set the content type of EnvelopedData to id-set-content-CertReqTBEX
9.	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertReq to the CA.	

Continued on next page

End Entity Generates CertReq, continued

CertReq (Enc) Generation

If the EE application is for a Merchant or a Payment Gateway that does not have AcctData (in AcctInfo) to send, the CertReq shall be generated using Enc processing as follows:

Step	Action
1	If the RequestType is for a new or renewed signature certificate, generate a private/public key pair for the signature certificate.
2	If the RequestType is for a new or renewed encryption certificate, generate a private/public key pair for the encryption certificate.
3	Generate a 160-bit random number, EXNonce.
4	Build the "CertReqData" data as follows: a) Generate a new RRPID. b) If the Merchant or Payment Gateway received a Me-AqCInitRes, copy the RequestType from that message; otherwise populate the RequestType. c) Populate the RequestDate from the current date. d) Copy LID-EE from a previous message. If one doesn't exist, generate a new one. e) Generate a fresh Chall-EE3. f) Copy LID-CA, if included, and Chall-CA from a previous message, if one exists. g) Populate the IDData. h) Populate the RegFormID received in the Me-AqCInitRes. i) Populate the new or corrected RegForm. j) Populate the newly generated public keys, PublicKeyS and/or PublicKeyE, for the CA to certify. k) If the RequestType is for the renewal of an Encryption certificate, populate the EEThumb with the thumbprint of the "to be renewed" certificate. l) Optionally include Thumbs, which holds the thumbprints for each CRL, SET certificate, BrandCRLIdentifier, and Root certificate resident in the Cardholder's trusted cache (Thumbs), if any exist.

Continued on next page

End Entity Generates CertReq, continued

CertReq (Enc) Generation (continued)

Step	Action	
5	Envelope the data using *Enc* encapsulation (as described in Part I on page 83):	
	Include:	**Processing**
	• CertReqData as "To Be Signed" data, and	How the data is signed depends on the RequestType. There is a minimum of one and possibly two signatures, i.e. SignerInfos, on a CertReq. If the Request Type is for a new Signature certificate, sign the data using the private key corresponding to the public key contained in PublicKeyS. If the Request Type is for a renewed Signature certificate, sign the data using the private key corresponding to the public key contained in PublicKeyS, and using the private key corresponding to the certificate being renewed. If the request type is for an Encryption certificate, sign the data using the private key corresponding to an existing signature certificate. If the data is signed with a private key that does not yet correspond to a certificate, set the SignerInfo.SerialNumber to zero and the Signer Info.IssuerName to the "Null-DN", i.e., the RDNSequence is an encoded NULL.Also, set the content type of SignedData to id-set-content-CertReqData. DER encode the resulting SignedData to obtain CertReqTBE.
	• DES Key as "To Be Extra Encrypted" data.	For Enc processing, the only "extra" encrypted data is the symmetric key used for the "ordinarily" encrypted data. Encrypt the key using the certificate indicated by CAEThumb in the Me-AqCInitRes.
	• CertReqTBE as "To be Ordinarily Encrypted Data"	Encrypt CertReqTBE and set the ContentType equal to id-set-content-CertReqTBE.
6	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertReq to the CA.	

Continued on next page

End Entity Generates CertReq, continued

CertReq

CertReq	**< EncX(EE, CA, CertReqData, AcctInfo), Enc(EE, CA, CertReqData) >** *Up to two signatures are implicit in the encapsulation.* **CertReqTBE** *and* **AcctInfo** *may be signed by any or all of the private keys corresponding to the following end entity certificates:* • *the private key for which a new Signature certificate,* • *an existing Signature certificate, for an Encryption certificate request, or* • *an existing Signature certificate, for a renewal request.* *These "signatures" without a corresponding signature certificate are pro forma only; they prove only that EE holds the private key.*
CertReqData	**{RRPID, LID-EE, Chall-EE3, [LID-CA], [Chall-CA], RequestType, RequestDate, [IDData], RegFormID, [RegForm], [CABackKeyData], PublicKeySorE, [EEThumb], [Thumbs]}**
AcctInfo	**< PANData0, AcctData >** *If the requester is a Cardholder,* **PANData0** *is included.* *If the requester is a Merchant or an Acquirer,* **AcctData** *is optional.*
RRPID	*Request/response pair ID*
LID-EE	*Copied from* **RegFormRes** *or* **Me-AqCInitRes**
Chall-EE3	*EE's challenge to CA's signature freshness*
LID-CA	*Copied from* **RegFormRes** *or* **Me-AqCInitRes**
Chall-CA	*Copied from* **RegFormRes** *or* **Me-AqCInitRes**
RequestType	*See page 157.*
RequestDate	*Date of certificate request.*
IDData	*See page 164 Omit if EE is Cardholder.*

Table 15: CertReq

Continued on next page

End Entity Generates CertReq, continued

CertReq (continued)

RegFormID	*CA-assigned identifier*
RegForm	**{RegFormItems +}** *The field names copied from* **RegFormRes** *or* **Me-AqCInitRes***, now accompanied by values filled in by EE's implementation.*
CABackKeyData	**{CAAlgId, CAKey}**
PublicKeySorE	**{[PublicKeyS], [PublicKeyE]}** *The entity's public key(s). At least one key shall be specified. A user may request a signature certificate, an encryption certificate, or both.*
EEThumb	*Thumbprint of entity key-encryption certificate that is being renewed.*
Thumbs	*Lists of Certificate (including Root), CRL, and BrandCRLIdentifier currently held by EE.*
PANData0	*See next page.*
AcctData	*See next page.*
RegFormItems	**{FieldName, FieldValue}**
CAAlgId	*Symmetric key algorithm identifier.*
CAKey	*Secret key corresponding to the algorithm identifier.*
PublicKeyS	*Proposed public signature key to certify.*
PublicKeyE	*Proposed public encryption key to certify.*
FieldName	*One or more field names to be displayed as a fill-in form on the requester's system, as a text field in the language specified in* **RegFormReq** *or* **Me-AqCInitReq**.
FieldValue	*Values entered by EE.*

Table 15: CertReq, continued

Continued on next page

End Entity Generates CertReq, continued

PANData0

PANData0	**{PAN, CardExpiry, CardSecret, EXNonce}**
PAN	*Primary Account Number; typically, the account number on the card.*
CardExpiry	*Expiration date on the card.*
CardSecret	*Cardholder's proposed half of the shared secret,* **PANSecret**. *Note: this value is saved for use in generating* **TransStain** *(see "PIHead" on page 273).*
EXNonce	*A fresh nonce to foil dictionary attacks on* **PANData0**.

Table 16: PANData0

AcctData

AcctData	**{AcctIdentification, EXNonce}**
AcctIdentification	*For a Merchant, this field is unique to the Merchant as defined by the payment card brand and Acquirer.* *For an Acquirer, this field is unique to the Acquirer as defined by the payment card brand.*
EXNonce	*A fresh nonce to foil dictionary attacks on* **AcctIdentification**

Table 17: AcctData

CA Validates CertReq (EncX)

Validation EncX

The CA shall validate the CertReq (EncX) as follows:

Step	Action
1	Receive the CertReq message from *Receive Message* (as described in Part I on page 77). **Note:** Decrypt and verify the signature of the CertReq message, in the inverse of the steps listed for *EncX* processing (described in Part I on page 83), with the following qualifications. Depending on the RequestType that was sent in the CertReq, there will be one or two signatures, i.e. SignerInfos, on the CertRes. 1. If the RequestType indicated a new Signature certificate or both new Signature and Encryption certificates, there will be one signature on the CertReq. Verify it using the new signature public key contained in PublicKeyS. If it does not verify, return a CertRes with CertStatusCode set to sigValidationFail. 2. If the RequestType indicated the renewal of a Signature certificate or the renewal of both a Signature and an Encryption certificate, there will be two signatures (SignerInfos) on the CertReq. a) For the SignerInfo with a Null Issuer DN, verify the signature using the new signature public key contained in PublicKeyS. If it does not verify, return a CertRes with CertStatusCode set to sigValidationFail. b) For the SignerInfo with the Issuer DN and Serial Number equal to the values in the renewed Signature certificate, verify the signature using the public key in that certificate. If it does not verify, return an Error Message with ErrorCode set to signatureFailure. 3. If the RequestType indicated a new or the renewal of an Encryption certificate, there will be one signature on the CertReq. Verify it using the public key from the EE's Signature certificate. If it does not verify, return an Error Message with ErrorCode set to signatureFailure.

Continued on next page

CA Validates CertReq (EncX), continued

Validation EncX (continued)

Step	Action
2	From the "CertReqTBS" data, store the RRPID, LID-EE, Chall-EE3, RequestType, LID-CA, Chall-CA, IDData, RegForm, CaBackKeyData, Thumbs, and the new Signature and/or Encryption certificates.
3	Verify that the RRPID and RequestDate match those received in the message wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
4	Verify that the Chall-CA received matches the one sent in the Me-AqCInitRes or RegFormRes. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
5	Verify the PAN, if it's included, according to the Brand's policy; otherwise verify the AcctData. If it does not verify, return a CertRes with CertStatusCode set to rejectedByIssuer.
6	If the RequestType indicates a renewal, verify that the certificates being renewed have not been renewed before (that is, guarantee that a specific certificate is not renewed multiple times). If it does not verify, return a CertRes with CertStatusCode set to rejectedByCA.
7	Verify that the RegFormID is valid for the language, RequestType and BIN or PAN. If it does not verify, return a CertRes with CertStatusCode set to rejectedByCA.
8	If the sender of the CertReq was a Cardholder, store the algorithm and key included in CABackKeyData to use to encrypt the CertRes to be returned to the Cardholder.
9	Verify the invisible registration form items. If any invisible fields are required and are not populated correctly, return a CertRes with CertStatusCode set to rejectedByIssuer.
10	If the above checks are successful, verify the registration form items. For each item in the registration form, verify that the length, format and character type are correct. Verify that the required fields are present. If any errors are found, return the item number(s) and text messages indicating the error(s) in the FailedItems sequence in CertRes with CertStatusCode set to regFormAnswerMalformed.

CA Validates CertReq (Enc)

Validation Enc

The CA shall validate the CertReq (Enc) as follows.

Step	Action
1	Receive the CertReq message from *Receive Message* (as described in Part I on page 77). **Note:** Decrypt and verify the signature of the CertReq message, in the inverse of the steps listed for *Enc* processing (described in Part I on page 83). Verify the signature as specified for the CertReq(EncX) processing. If it does not verify, respond as specified for CertReq(EncX).
2	From the "CertReqData", store the RRPID, LID-EE, Chall-EE3, RequestType, LID-CA, Chall-CA, IDData, RegForm, Thumbs, and the new Signature and/or Encryption certificates.
3	Verify that the Chall-CA received matches the one sent in the Me-AqCInitRes or RegFormRes. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
4	Verify that the RRPID and RequestDate match the ones received in the message wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
5	If the RequestType indicates a renewal, verify that the certificates being renewed have not been renewed before (that is, guarantee that a specific certificate is not renewed multiple times). If it does not verify, return a CertRes with CertStatusCode set to rejectedByCA.
6	Verify that the RegFormID is valid for the language, RequestType, and BIN. If it does not verify, return a CertRes with CertStatusCode set to rejectedByCA
7	Verify the invisible registration form items. If any invisible fields are required and are not populated correctly, return a CertRes with CertStatusCode set to rejectedByIssuer.
8	If the above checks pass, verify the visible registration form items. For each item in the registration form, verify that the length, format and character type are correct. Verify that the required fields are present. If any errors are found, return the item number(s) and text messages indicating the error(s) in the CertRes with CertStatusCode set to regFormAnswerMalformed.

Continued on next page

CA Validates CertReq (Enc), continued

Failure

If validation fails, the CA shall prepare and send a CertRes message with the appropriate status. If the validation failure is due to errors in the CertReq, the CA shall indicate the errors in the CertRes and the EE application can re-send the CertReq with the corrected registration form.

Success

If the validation of all fields is successful, processing continues with financial institution authentication.

Financial Institution Authentication

Overview

The financial institution verifies the data in the CertReq prior to the generation of a certificate. The specific method used depends on the brand of certificate being issued and is outside the scope of SET.

Status return

Using a process negotiated and implemented between the financial institution and the CA, the CertReq may or may not be accepted. If not, status is returned to the CA for use in composing CertRes CertStatus Codes.

CA Generates CertRes

CertRes overview

The CertRes contains either the requested certificates or the status of the certificate request. The CertRes will be signed and optionally encrypted, depending on the data that's to be included in the message. If the CertRes is successful and is intended for the Cardholder, the message is encrypted using a common symmetric algorithm supported by both the CA and the Cardholder application. If an encryption algorithm cannot be negotiated between the CA and the Cardholder application, the request shall be rejected and the appropriate status returned. If the CertRes is intended for a Merchant or Payment Gateway or is returning status to a Cardholder, the message is signed but not encrypted.

Generate certificate

If the CertReq is successful the CA generates the certificate. See Chapter 4 for additional information about how the fields are populated.

Continued on next page

CA Generates CertRes, continued

Generate CertRes: Signed Data within Enveloped Data to Cardholder

If the CertReq is authentic, valid, and the CA has generated a certificate using the submitted key, a CertRes with completed status shall be returned. If the CertRes is intended for a Cardholder and included a key (in CaBackKeyData) to encrypt the CertRes, the CA shall generate the CertRes as Signed Data within Enveloped Data by performing the steps listed below. (Otherwise, see page 191.)

Step	Action
1	Build "CertResData" as follows: a) Copy RRPID, LID-EE, Thumbs, and Chall-EE3 from the CertReq. b) Generate LID-CA, or copy from CertReq, if present. c) Optionally populate the CardLogo URL, BrandLogo URL, CardCurrency, and/or the CardholderMessage (CaMsg). d) Set the CertStatusCode to "Request Complete". e) Generate Nonce-CCA. f) Compute and populate the thumbprints of the EE certificates, CertThumbs. g) If the BrandCRLIdentifier is not specified in the Thumbs received in the CertReq, populate the BrandCRLIdentifier.
2	Sign and envelope the data using *EncK* encapsulation (described in Part I on page 82) using CertResData, as the "To Be Signed" data. a) Sign the data with the CA Signature certificate. b) Set the content type of SignedData to id-set-content-CertResData. c) Include the new, certified EE Signature and/or Encryption certificates in the certificates portion of Signed Data. d) Encrypt the signed data, using a CA generated initialization vector and the algorithm and key indicated by CaBackKeyData in the CertReq. e) Set the content type of EncryptedData to id-set-content-CertResTBE.
3	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertRes to the EE.

Continued on next page

CA Generates CertRes, continued

Generate Signed CertRes

The CertRes shall be signed but not encrypted if the EE is a Merchant or Payment Gateway.

If the CA is returning status in the CertRes, the EEMessage is included to convey information to the Cardholder, Merchant, or Payment Gateway. The following steps shall be used to generate the signed CertReq.

Step	Action
1	If the CA has generated a certificate that will be included in the CertRes, perform create CertResTBS, specified in step 1 on page 190.
2	If the CA has not generated a certificate, i.e. has status other than "Request Complete", build CertResData as follows: • Copy LID-EE and Chall-EE3 from the CertReq. • Optionally populate the EEMessage. • Populate the CertStatusCode from Table 19 on page 194. • If the CertStatusCode is set to regFormAnswerMalformed, populate the ItemNumbers and ItemReasons for each FailedItem in the registration form.
3	Sign the data using the *Signature Processing* on page 93, using CertResData, as the "To Be Signed" data. Set the content type of SignedData to id-set-content-CertResData. **Note:** Sign the data with the CA digital signature certificate.
4	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertRes to the EE.

Continued on next page

CA Generates CertRes, continued

CertRes

CertRes	**< S(CA, CertResData), EncK(CABackKeyData, CA, CertResData) >** *The* **EncK** *version of this message is only needed if the optional* **CAMsg** *component is included in the* **CertRes** *and it is only used if* **CaBackKeyData** is included in the **CertReq**.
CertResData	**{RRPID, LID-EE, Chall-EE3, LID-CA, CertStatus, [CertThumbs], [BrandCRLIdentifier], [Thumbs]}**
CABackKeyData	*Copied from* **CertReq**.
RRPID	*Request/response pair ID.*
LID-EE	*Copied from prior* **CertReq**.
Chall-EE3	*Copied from* **CertReq**. *Requester checks for match with remembered value.*
LID-CA	*Copied from* **CertReq**. *If not present in the* **CertReq**, *new values are assigned.*
CertStatus	**{CertStatusCode, [Nonce-CCA], [EEMessage], [CaMsg], [FailedItemSeq]}**
CertThumbs	*If request is complete, the thumbprints of the enclosed signature and or encryption certificates.*
BrandCRLIdentifier	*See page 252.*
Thumbs	*Copied from* **CertReq**.
CertStatusCode	*Enumerated code indicating the status of the certificate request.*
Nonce-CCA	*If request is complete and from a cardholder, the other half of the ultimate shared secret between Cardholder and CCA. See* **PANData0** *under "CertReq". Present only if EE is Cardholder.*

Table 18: CertRes

Continued on next page

CA Generates CertRes, continued

CertRes (continued)

EEMessage	*Message in natural language to be displayed on the EE system.*
CAMsg	**{[CardLogoURL], [BrandLogoURL], [CardCurrency], [CardholderMsg] }** *If request is complete and from a cardholder.*
FailedItemSeq	**{FailedItem+}**
CardLogoURL	*URL pointing to graphic of card logo (issuer-specific).*
BrandLogoURL	*URL pointing to graphic of payment card brand logo.*
CardCurrency	*Cardholder billing currency.*
CardholderMsg	*A message in the Cardholder's natural language to be displayed by the software.*
FailedItem	**{ItemNumber, ItemReason}**
ItemNumber	*Indicates the position of the failed item in the list of registration fields. A value of 0 indicates the* **AcctData** *field.*
ItemReason	*The reason for the failure, as a text field in the language specified.*

Table 18: CertRes, continued

Continued on next page

CA Generates CertRes, continued

CertStatus codes

The following table lists valid status codes for certificate requests.

Code	Meaning	Source
requestComplete	Certificate request approved	CA
invalidLanguage	Invalid language in initiation request	CA
invalidBIN	Certificate request rejected because of invalid BIN	Issuer or Acquirer
sigValidationFail	Certificate request rejected because of signature validation failure	CA
decryptionError	Certificate request rejected because of decryption error	CA
requestInProgress	Certificate request in progress	CA, Issuer, or Acquirer
rejectedByIssuer	Certificate request rejected by Issuer	Issuer
requestPended	Certificate request pending	CA, Issuer, or Acquirer
rejectedByAquirer	Certificate request rejected by Acquirer	Acquirer
regFormAnswerMalformed	Certificate request rejected because of malformed registration form item(s)	CA
rejectedByCA	Certificate request rejected by Certificate Authority	CA
unableToEncryptCertRes	Certificate authority didn't receive key, so is unable to encrypt response to cardholder	CA

Table 19: Certificate Request Status Codes

End Entity Processes CertRes

Validate CertRes

The EE validates the new certificate(s) by performing the following:

Step	Action
1	Receive the CertRes message from *Receive Message* (described in Part I on page 77). **Note:** If the CertRes contains signed data within enveloped data, decrypt and verify the signature of the CertRes, in the inverse of the steps listed for *EncK* processing (described in Part I on page 82), performing symmetric decryption using the algorithm and key that was sent to the CA in CaBackKeyData, in the CertReq. The processing performed on the received certificates, CRLs, and BrandCRLIdentifier is described in "Receive Message " (Part I, page 77).
2	Verify, as specified in the "Thumbprint" processing steps (described in Part I on page 79), that the Thumbs received match those sent in the CardCInitReq message. If it does not verify, return an Error Message with ErrorCode set to thumbsMismatch.
3	Verify the LID-EE and Chall-EE match those sent in the CertReq. If it does not verify, return an Error Message with ErrorCode set to challengeMismatch.
4	If the CertStatusCode indicates "Certificate request complete": 1. Retrieve the new certificates from the certificates portion of SignedData and validate the signatures. 2. Verify that the CertThumbs received match those sent in the CertReq. If it does not verify, return an Error Message with ErrorCode set to thumbsMismatch. 3. If a Cardholder and if they exist, retrieve the CaMsg, display the logos and CardholderMsg from the CaMsg, and store the CardCurrency. 4. Verify that the public keys in the certificate(s) correspond to the private keys. If it does not verify, return an Error Message with ErrorCode set to invalidCertificate. 5. If a Cardholder, perform the following additional steps: a) Compute (Nonce-CCA ⊕ CardSecret) to obtain PANSecret. b) Compute the Unique Cardholder ID, HMAC-SHA-1{{PAN, cardExpiry}, PANSecret}, that is, the Salted Hash, as described on page 213, and verify that the result matches the value in the certificate.

Continued on next page

End Entity Processes CertRes, continued

Validate CertRes (continued)

Step	Action
5	If the CertStatusCode indicates "Malformed Registration Form Items", some of the registration form items were in error. For each item that was in error, the EE application shall display the item number and the corresponding error message that was returned in the CertRes. The EE shall be allowed to re-enter fields and the EE application shall re-submit the CertReq as a new certificate request.
6	If the CertStatusCode is set to requestinProgress or requestPended, the certificate may be picked up via submission of a CertInqReq at a later time.
7	If CertStatusCode indicates any other status, display EEMessage.

Validation failure

If the validation fails, the EE sends an error message to the CA indicating the failure.

Section 7
Certificate Inquiry and Status Processing

Certificate Inquiry Protocol

Introduction

If the certificate is not returned immediately in the CertRes, the EE can request the status of its certificate request by sending a CertInqReq to the CA. The CertInqRes will return the certificate if it's ready or will provide information as to when the certificate will be ready.

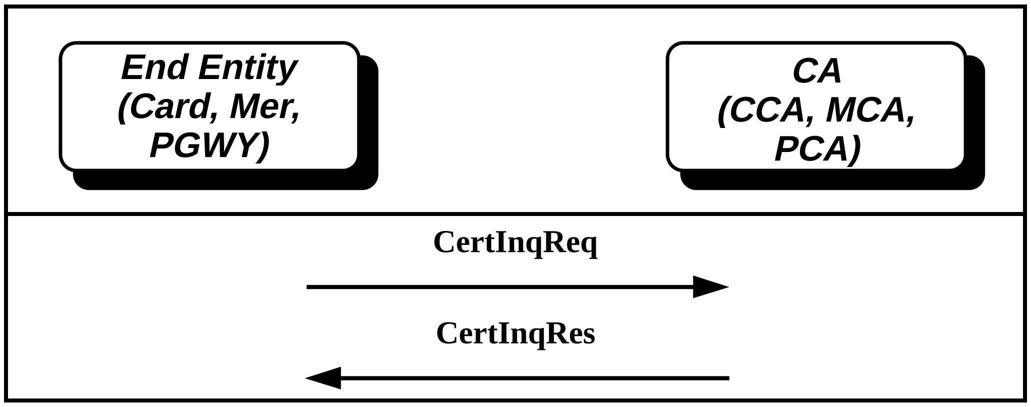


Figure 24: Certificate Inquiry Protocol

End Entity Generates CertInqReq

Generate CertInqReq

If the CertStatusCode of the CertRes indicated "Certificate Request in Process" or "Certificate Request Pending", the EE generates the CertInqReq to retrieve the certificate as described below.

Step	Action
1	Copy LID-CA3 from the CertRes into "CertInqReqTBS" data.
2	Generate a new RRPID.
3	Generate a new LID-EE.
4	Generate a new Chall-EE3.
5	Copy the LID-CA from the preceding CertRes.
6	Sign CertInqReqTBS using the *Signature Processing* on page 93. Set the content type of SignedData to id-set-content-CertInqReqTBS. The CertInqReq is signed the same way as the CertReq: i.e., there is a minimum of one and possibly two signatures, i.e. SignerInfos, on a CertInqReq.
7	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertInqReq to the CA.

CertInqReq

CertInqReq	**S(EE, CertInqReqTBS)**
CertInqReqTBS	**{RRPID, LID-EE, Chall-EE3, LID-CA}**
RRPID	*Request/response pair identifier.*
LID-EE	*Copied from* **CertRes**.
Chall-EE3	*EE's challenge to CA's signature freshness.*
LID-CA	*Copied from* **CertRes**.

Table 20: CertInqReq

CA Processes CertInqReq

Processing of CertInqReq

The CA will process a CertInqReq as follows:

Step	Action
1	Receive the CertInqReq message from *Receive Message* (described in Part I on page 77). The signature of the CertInqReqTBS will be validated in the same way as the CertReq is validated. If it does not verify, respond as specified for CertReq(EncX).
2	Verify that the RRPID matches the one sent in the message wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
3	Using LID-CA as an index, determine the status of the CertReq.
4	If a certificate has been generated, send it to the EE in the CertInqRes, otherwise send an updated CertStatusCode in the CertInqRes.

CA additional processing

The CA holds onto the CertRes or, if it contains newly issued certificates, the CertInqRes for a policy-definable period of time (possibly a week) to support re-transmission to the EE if needed.

CA Generates CertInqRes

Processing

After processing a **CertInqReq**, the CA will generate a **CertInqRes**. The **CertInqRes** has the same message content and format as the **CertRes**. It is generated using the same processing steps described for the **CertRes**.

Step	Action
1	As appropriate, perform either: • the steps to generate *CertRes Signed Data within Enveloped Data* (see page 190), or • the steps to generate *CertRes Signed Data* (see page 191). Exceptions: Copy the RRPID, LID-EE and Chall-EE3 received in the CertInqReq rather than that from the original CertReq.
2	Invoke *Compose Message Wrapper* (described in Part I on page 76) to send the CertInqRes to the EE.

CertInqRes data

The CertInqRes contains the same data as the CertRes message.

CertInqRes	*Identical to a* **CertRes***; see page* 192.

End Entity Processes CertInqRes

CertInqRes processing

The SET application will process a CertInqRes using the same steps as to process a CertRes.

Step	Action
1	Receive the CertInqRes message from *Receive Message* (described in Part I on page 77). **Note:** If the CertInqRes contains signed data within enveloped data, decrypt and verify the signature of the CertInqRes, in the inverse of the steps listed for *EncK* processing (described in Part I on page 82), performing symmetric decryption using the algorithm and key that was sent to the CA in CaKey, in the CertReq.
2	Perform steps 2 through 5 of the CertRes processing described on page 195.

Chapter 3
Certificate Revocation

Chapter Overview

Introduction

This chapter describes the revocation or cancellation process for SET certificates. A compromised certificate is "revoked" if it is placed on a CRL. A compromised certificate is "canceled" if a mechanism other than a CRL is used to prevent the certificate from being used.

Organization

Chapter 3 includes the following topics:

- Cardholder Certificate Cancellation
- Merchant Certificate Cancellation
- Payment Gateway Certificate Revocation
- CA Compromise Recovery

Assumptions

In defining the certification revocation concepts, the following assumptions were made:

- It is a requirement to minimize change to the Issuers' existing payment card system to support certificate revocation and to maximize reuse of the existing payment card infrastructure where applicable.
- A Cardholder certificate is bound to the payment card account. When a certificate is canceled, the associated payment card number will be canceled. When a payment card is lost/stolen or the account is terminated, the certificate is also no longer usable.
- When a Merchant's certificate is canceled, only the Acquirer needs to know since all payments are authorized via the Acquirer. If a Cardholder attempts to purchase from a Merchant whose certificate has been canceled, the Acquirer will reject the purchase. Furthermore, a person in possession of a compromised private key from a Merchant cannot extract payment card numbers directly from Cardholder Purchase Requests since the account numbers are encrypted with the Payment Gateway's public key.

Section 1
Cardholder Certificate Cancellation

Purpose

A Cardholder's certificate and associated private key are used to provide and authenticate the payment card information in the electronic payment protocol. If the private key corresponding to the public key in a certificate is compromised, the associated certificate shall be canceled.

Issuer-based cancellation approach

Since payment requests shall go to the Cardholder's Issuer for payment authorization, the Issuer can maintain the Cardholder certificate canceled status in the context of the payment card exception files (hotlists) maintained today. When a payment authorization request is received from the Acquirer for an account with a canceled certificate, the Issuer will reject the request because the account number has been canceled.

Section 2
Merchant Certificate Cancellation

Purpose

A Merchant's certificates and associated private keys are used to provide and authenticate the Merchant's identity in the electronic payment protocol. If the Merchant's private key in a certificate is compromised, the associated certificates must be canceled to avoid an adversary impersonating the Merchant.

Acquirer-based cancellation approach

As Merchants terminate their association with a specific Acquirer, Acquirer's have the capability to reject all payment requests from that Merchant. The Payment Gateway will either use the existing system to support Merchant authentication, or it will maintain a local list of Merchant certificates that are not to be accepted.

Section 3
Payment Gateway Certificate Revocation

Background

A Payment Gateway has two certificates:

- Key Encryption Certificate, used for encrypting Payment Instructions
- Signature Certificate

The storage of the private keys associated with these certificates is determined by the brand's policy. However, the preferred method of storage is on hardware cryptographic modules with restricted physical access.

Compromise recovery

To protect against a cryptanalytic attack and to support compromise recovery, relatively frequent re-keying of the encryption key is desired. Any damage caused by a compromise will be limited to that key's usage period, and the adoption of this policy by a brand may deter would-be adversaries.

Revoking certificate

In the event that one or more of the Payment Gateway's private keys are compromised (or suspected of compromise), the Acquirer shall immediately remove the private keys from the Payment Gateway.

The certificates corresponding to compromised Payment Gateway private keys will be placed on Certificate Revocation Lists (CRLs). These CRLs will be generated and distributed by the Payment Gateway CA.

Distribute new certificate

Once the new certificates are distributed to the Payment Gateway, the Merchants will receive the new certificate using the same method as Payment Gateway certificate renewals. As Merchants receive certificates with more recent validity dates, the older certificates are purged from the system (that is, the suspect certificate will be effectively removed from the system as soon as a newer Payment Gateway certificate is received by the Merchants).

Section 4
CA Compromise Recovery

Overview

The likelihood of a successful attack against a CA is very low. However, if a successful attack were to occur, a new CA certificate shall be distributed and the old certificate shall be revoked.

Use of CA CRL

The identity of the compromised CA certificate is included in a CA CRL and distributed to all entities in the system. A list of all up-to-date CRLs in effect is contained within the BrandCRLIdentifier.

Distribution of CA CRL

The CA CRL is distributed in existing messages.

- The CA distributes the CRL to Payment Gateways in the Me-AqCInitRes message with each renewal.
- The CA distributes CRLs to the Cardholder in all downstream response messages.
- The Payment Gateway distributes the CRL to Merchants in the AuthRes message when the ThumbPrint in the AuthReq does not include one or more of the latest CRLs.
- The Merchant distributes the CA CRL to the Cardholder in the PInitRes or PRes when the PInitReq or PReq Thumbprints do not include one or more of the latest CRLs.

BrandCRL Identifier

The BrandCRLIdentifier contains a list of all CA CRLs that are current. An entity uses the BrandCRLIdentifier to check that it holds all of the up-to-date CRLs. The BrandCRLIdentifier is included in all downstream response messages.

Chapter 4
Certificate Format

Chapter Overview

Introduction

This section contains a description of the X.509 Version 3 certificate format and certificate extensions used in SET. The certificate format includes the use of public and private extensions to support all SET certificate requirements.

Organization

Chapter 4 includes the following sections:

Section	Title	Contents	Page
1	X.509 Certificate Definition	Describes all of the fields in the basic X.509 certificate.	208
2	X.509 Extensions	Describes all of the fields in the standard X.509 extensions used in SET.	214
3	SET Private Extensions	Describes all of the fields in the SET specific extensions.	228
4	Certificate Profiles	Describes all certificate and extension fields and identifies how/when they are used for each of the different types of certificates used in SET.	239

Section 1
X.509 Certificate Definition

X.509 Certificate Data Definitions

Format and value restrictions

The table below defines the format and value restrictions for each field in the X.509 certificate.

Name	Format and Value Restrictions	Description
Version	Integer	Indicates the certificate version. Always set to 3 indicating Version 3.
SerialNumber	Integer	Unique serial number assigned by the CA that issued the certificate.
Signature .AlgorithmIdentifier	OID and type	Defines the algorithm used to sign the certificate.
Issuer	Name	Contains the Distinguished Name (DN) of the CA that issued the certificate.
Validity .notBefore	UTC Time	Specifies when the certificate becomes active.
Validity .notAfter	UTC Time	Specifies when the certificate expires. If a Cardholder's certificate, the Validity period shall not extend beyond the card's expiration date.
Subject	Name	Contains the Distinguished Name of the entity owning the key.
SubjectPublicKeyInfo .algorithm .AlgorithmIdentifier	OID and type	Specifies which algorithms can be used with this key.
SubjectPublicKeyInfo .subjectPublicKey	Bit string	Contains the public key provided in the certificate request.

Table 21: X.509 Certificate Data Definitions

Continued on next page

X.509 Certificate Data Definitions, continued

Format and value restrictions (continued)

Name	Format and Value Restrictions	Description
IssuerUniqueID		Not used in SET.
SubjectUniqueID		Not used in SET.
Extensions .extnId	OID format	Contains the extension's OID as defined by X.509 or SET.
Extensions .critical	Boolean; 0 false (Default) , 1 true	Each extension description states how this field will be set.
Extensions .extnValue		The extension data.

Table 21: X.509 Certificate Data Definitions**, continued**

Certificate Subject Name Format

Name fields

The following Object Identifiers (OIDs) shown in brackets are needed for defining items with a format of Name in SET Certificates:

- countryName [2 5 4 6]
- organizationName [2 5 4 10]
- organizationalUnitName [2 5 4 11]
- commonName [2 5 4 3]

The following paragraphs describe the attributes which comprise the Subject Distinguished Name for each SET entity indicated in the CertificateType extension.

Name OIDs ASN.1

```
id-at-countryName            OBJECT IDENTIFIER ::= { id-at 6 }
id-at-organizationName       OBJECT IDENTIFIER ::= { id-at 10 }
id-at-organizationalUnitName OBJECT IDENTIFIER ::= { id-at 11 }
id-at-commonName             OBJECT IDENTIFIER ::= { id-at 3 }
```

Cardholder

countryName=<Country of Issuing Financial Institution>
organizationName=<BrandID>
organizationalUnitName=<Name of Issuing Financial Institution>
organizationalUnitName=<Optional - Promotional Card Name>
commonName=<Unique Cardholder ID>

Note: To distinguish between the two organizationalUnitNames, the organizationalUnitName representing the "Name of Issuing Financial Institution" shall appear first in the generated Cardholder certificate.

Merchant

countryName=<Country of Acquiring Financial Institution>
organizationName=<BrandID>
organizationalUnitName=<Name of Acquiring Financial Institution>
commonName=<Name of Merchant as printed on Cardholder statement>

Payment gateway

countryName=<Country of Acquiring Financial Institution>
organizationName=<BrandID>
organizationalUnitName=<Name of Acquiring Financial Institution>
commonName=<Unique Payment Gateway ID>

Continued on next page

Certificate Subject Name Format, continued

Cardholder Certificate Authority	countryName=<Country of Issuing Financial Institution>
	organizationName=< BrandID>
	organizationalUnitName=<Descriptive Name>
	commonName=<Optional - Unique ID>

Merchant Certificate Authority	countryName=<Country of Acquiring Financial Institution >
	organizationName=< BrandID >
	organizationalUnitName=<Descriptive Name>
	commonName=<Optional - Unique ID>

Payment Gateway Certificate Authority	countryName=<Country of Acquiring Financial Institution >
	organizationName=< BrandID >
	organizationalUnitName=<Descriptive Name>
	commonName=<Optional - Unique ID>

Geo-Political Certificate Authority	countryName=<Country of Geo-political organization>
	organizationName=< BrandID >
	organizationalUnitName=<Descriptive Name>
	commonName=<Optional - Unique ID>

Brand Certificate Authority	countryName=<Country of the Brand>
	organizationName=< BrandID>
	organizationalUnitName=<Descriptive Name>
	commonName=<Optional - Unique ID>

Root Certificate Authority	countryName=<Country where CA is located>
	organizationName=<SET Root>
	commonName=<Optional - Unique ID>

Name Fields

Definitions

The Name fields in the Certificate Subject Name are defined as follows.

Country	The 2 character ISO 3166 country code.
BrandID	<Brand Name>:<Product >, where the Product name is optional.
Brand Name	The brand of payment card. To be defined by the payment card brand.
Product Type	This optional field defines the type of product within the specific brand.
Descriptive Name	This is a descriptive name of the entity responsible for issuing the certificates under this CA. Examples include: • Name of financial institution • Name of the organization operating the CA • Name of the brand • Name of the entity responsible for approving the certificates Brand and financial institution policies may restrict the choices available for Descriptive Name.
Promotional Card Name	This optional field contains the promotional name of the card. Examples include Frequent Flyer Program, Affinity Program etc.
Name of Financial Institution	The name of the issuing financial institution.
Unique Cardholder ID	The unique Cardholder ID in the Cardholder's certificate is the keyed-hashed account number. See detailed description on the next page.
Unique Payment Gateway ID	This field contains the BIN followed by an Acquirer or brand assigned serial number. This field is formatted as <BIN:Serial Number>. The serial number allows each Payment Gateway associated with the same Acquirer to be uniquely identified. Multiple certificates may exist for a BIN within a Brand.

Continued on next page

Name Fields, continued

Unique Cardholder ID

The unique Cardholder ID in the Cardholder's certificate is the keyed-hashed account number. The PAN is masked using the shared secret value, PANSecret, that's comprised of a Cardholder secret value (CardSecret) and a CA secret value (Nonce-CCA). The keyed-hashed value is computed using the HMAC-SHA1 algorithm as specified in RFC 2105. If there are any differences between the following specification and that specified in RFC 2105, the RFC shall take precedence. The HMAC-SHA1 function is defined in terms of a key, K, and the "Text" that is masked by the keyed-hash using the following function:

$$\text{hash}(K \oplus \text{opad} \mid \text{hash}((K \oplus \text{ipad}) \mid \text{text}))$$

where the "⊕" operator designates XOR, and the '|' operator denotes concatenation.

K, Text, ipad, and opad are defined as follows for SET:

K	Equal to PANSecret. PANSecret is a 20 byte value computed by Exclusive OR'ing the DER decoded values of CardSecret (the Cardholder nonce) and Nonce-CCA (the CA nonce).

Text	Equal to the DER encoded value of `Text`, shown below, and comprised of the PAN and the CardExpiry.
	`Text ::= SEQUENCE {`
	`        pan            PAN,`
	`        cardExpiry     CardExpiry`
	`                           }`
	`  PAN ::= NumericString (SIZE(1..19))`
	
	`  CardExpiry ::= NumericString (SIZE(6))  -- YYYYMM`
	`                            expiration date on card`

ipad	64 repetitions of the byte x'36' = b'00110110', (6)
opad	64 repetitions of the byte x'5C' = b'01011100', (\)

K is padded with zeros out to a 64 byte string.

Following the HMAC computation, the resulting digest shall be base64 encoded prior to being placed in the certificate commonName field.

Section 2
X.509 Extensions

Section Overview

Purpose

This section describes the use of the following X.509 extensions for use in SET:

- AuthorityKeyIdentifier
- KeyUsage
- PrivateKeyUsagePeriod
- CertificatePolicies
- SubjectAltName
- BasicConstraints
- IssuerAltName

AuthorityKeyIdentifier Extension

Overview

A CA may have more than one certificate, either for functionally different purposes, or as key updating occurs. This extension is used to identify which CA certificate shall be used to verify the certificate's signature. This extension contains the following fields:

1. Key Identifier
2. Certificate Issuer
3. Certificate Serial Number

In SET, the Certificate Issuer and the Certificate Serial Number are always set and the Key Identifier is not used. The Issuer Distinguished Name and Serial Number are inherited from the signing CA's certificate and are used to populate the Certificate Issuer and Serial Number of this extension.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
authorityKeyIdentifier .AuthorityKeyId .keyIdentifier		Not used in SET.
authorityKeyIdentifier .AuthorityKeyId .certIssuer	Name	Contains the Issuer DN of the issuing certificate authority's certificate.
authorityKeyIdentifier .AuthorityKeyId .certSerialNumber	Positive Integer	Contains the serial number of the issuing certificate authority's certificate.

Continued on next page

AuthorityKeyIdentifier Extension, continued

ASN.1

```
authorityKeyIdentifier EXTENSION ::= {
    SYNTAX         AuthorityKeyIdentifier
    IDENTIFIED BY id-ce-authorityKeyIdentifier
 }

 AuthorityKeyIdentifier ::= SEQUENCE {
    keyIdentifier              [0] KeyIdentifier  OPTIONAL,
    authorityCertIssuer        [1] GeneralNames  OPTIONAL,
    authorityCertSerialNumber  [2] CertificateSerialNumber  OPTIONAL
 } ( WITH COMPONENTS {  keyIdentifier ABSENT,
       authorityCertIssuer PRESENT, authorityCertSerialNumber PRESENT } )
```

KeyUsage Extension

Overview

The KeyUsage extension is an X.509 extension that indicates how the public key in the certificate may be used.

SET's key usages

A SET certificate may have the following key usage(s) assigned:

- digital signature,
- certificate signature,
- CRL signature,
- data encryption and key encryption, or
- certificate signature and CRL signature.

Criticality

This extension is critical.

Restrictions

Name	Format and Value Restrictions	Description
keyUsage .keyUsage	0, 5, 6, or {2, 3} or {5, 6} only	Indicates whether the public key contained in the certificate may be used for signature verification, encryption, etc.

Continued on next page

KeyUsage Extension, continued

KeyUsage and BasicConstraints

The values set in the KeyUsage and BasicConstraints extensions shall not conflict. The following table indicates the KeyUsage and the BasicConstraints CA values for each EE or CA certificate.

SET Entity Type	KeyUsage Value	BasicConstraints CA Value	Description
EE	0	EE	Public key may be used to verify message signatures.
EE	2 & 3	EE	Public key may be used to encrypt keys and data in the OAEP envelope.
CA	0	EE	Public key may be used to verify message signatures.
CA	2&3	EE	Public key may be used to encrypt keys and data in the OAEP envelope.
CA	5	CA	Public key may be used to verify certificate signatures.
CA	6	EE	Public key may be used to verify CRL signatures.
CA	5&6	CA	Public key may be used to verify certificate and CRL signatures.

ASN.1

```
keyUsage EXTENSION ::= {
   SYNTAX         KeyUsage
   CRITICAL       TRUE
   IDENTIFIED BY id-ce-keyUsage
}

KeyUsage ::= BIT STRING {
   digitalSignature  (0),
   nonRepudiation    (1),
   keyEncipherment   (2),
   dataEncipherment  (3),
   keyAgreement      (4),
   keyCertSign       (5),    -- For use in CA-certificates only
   cRLSign           (6)     -- For use in CA-certificates only
}
```

PrivateKeyUsagePeriod Extension

Overview

The PrivateKeyUsagePeriod extension is an X.509 extension used to delimit the period of time that the private key corresponding to the certificate is valid. This extension is only used in signature certificates - it is not applicable to encryption certificates.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
privateKeyUsagePeriod . PrivateKeyUsagePeriod .notBefore	Date and Generalized Time	The start date and time of the private key's validity period.
privateKeyUsagePeriod . PrivateKeyUsagePeriod .notAfter	Date and Generalized Time	The end date and time of the private key's validity period.

ASN.1

```
privateKeyUsagePeriod EXTENSION ::= {
   SYNTAX         PrivateKeyUsagePeriod
   IDENTIFIED BY id-ce-privateKeyUsagePeriod
}

PrivateKeyUsagePeriod ::= SEQUENCE {
   notBefore  [0] GeneralizedTime  OPTIONAL,
   notAfter   [1] GeneralizedTime  OPTIONAL
} ( WITH COMPONENTS { ..., notBefore PRESENT } |
    WITH COMPONENTS { ..., notAfter  PRESENT } )
```

CertificatePolicies Extension

Overview

This extension specifies the policies governing the use of the certificate. A certificate policy is a set of rules defining the use of the certificate in the SET application. The policy is identified in the certificate by an Object Identifier.

Policy

CertificatePolicies is an X.509 extension containing a list of one or more certificate policies. Each certificate policy is denoted by a globally unique Object Identifier and may optionally contain corresponding qualifiers. SET certificates shall contain at least one policy Object Identifier (OID), that of the SET Root policy. The SET Root certificate shall contain this policy OID and this policy shall be inherited by all subordinate certificates. SET certificates shall only be used according to the rules specified in the policy.

Qualifier

Qualifiers to the policy may be included in this extension. SET uses qualifiers to provide pointers to the actual policy statement and to add qualifying policies to the Root policy. SET defines the following qualifiers:

- a root policy qualifier
- additional policies and qualifiers

Root Policy Qualifier

The root policy qualifier contains information related to the location and content of the SET root policy:

- policy URL
- policy Email
- policy Digest
- terse Statement

Each of the above qualifiers is optional. The policy URL and policy Email contain a URL and an e-mail address where a copy of the root policy statement can be obtained. A hash of the policy may be included in policy Digest and the value may be compared with the hash of the policy obtained from the URL.

Continued on next page

CertificatePolicies Extension, continued

Additional Policy Qualifiers

In addition to the root policy qualifier, each CA (Brand, Geopolitical, or MCA, CCA, PCA) may add one qualifying statement to the root policy in a subordinate certificate. The additional qualifier is a policy statement for that CA. Like the Root policy, it is indicated by an Object Identifier and qualified using the same indicators specified above. The signing CA also indicates its certificate type as a qualifier, so that a subordinate certificate holder may determine which policy statement corresponds to a given CA.

There may be a maximum of four policy Object Identifiers in a SET EE certificate: the Root CA's, the Brand CA's, the Geopolitical CA's, and the Cardholder, Merchant, or Payment Gateway CA's.

Certificate Generation

A generated certificate shall inherit all of the policy information of the SET CA signing certificate. Further, the subordinate certificate may contain an additional policy that is inserted by the signing CA.

Criticality

This extension is critical.

Restrictions

Name	Format and Value Restrictions	Description
policy	Object ID	The OID which points to the Root policy statement. The policy may be obtained from the URL or e-mail address provided in the qualifiers.
policyQualifierId	Object ID	Set to id-set-setQualifier.
qualifier	SETQualifier	• Contains optional qualifiers to the Root policy. • Contains additional optional qualifying policies and their qualifiers.
SETQualifier .policyURL.	IA5String	URL where a copy of the policy statement may be found.

Continued on next page

CertificatePolicies Extension, continued

Restrictions (continued)

SETQualifier .policyEmail	IA5String	E-mail address where a copy of the policy statement may be found.
SETQualifier .policyDigest	Octet String	The hash of the policy statement, computed using the indicated digestAlgorithm.
SETQualifier .terseStatement	DirectoryString	A statement declaring any disclaimers associated with the issuing of the certificate.
SETQualifier .additionalPolicies .policyOID	Object ID	The OID which points to the CA's policy statement. The policy may be obtained from the URL or e-mail address provided in the associated qualifiers.
SETQualifier .policyAddedBy	Certificate Type	Indicates the CA that the policy corresponds to and that added the policy to the generated certificate.

Continued on next page

CertificatePolicies Extension, continued

ASN.1

```
certificatePolicies EXTENSION ::= {
   SYNTAX         CertificatePoliciesSyntax
   CRITICAL       TRUE
   IDENTIFIED BY id-ce-certificatePolicies
}

CertificatePoliciesSyntax ::= SEQUENCE SIZE(1..MAX) OF
PolicyInformation

PolicyInformation ::= SEQUENCE {
   policyIdentifier  CertPolicyId,
   policyQualifiers  SEQUENCE SIZE(1..MAX) OF
                                          PolicyQualifierInfo
OPTIONAL
}

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
   policyQualifierId  CERT-POLICY-QUALIFIER.&id

({SupportedPolicyQualifiers}),
   qualifier          CERT-POLICY-QUALIFIER.&Qualifier

({SupportedPolicyQualifiers}{@policyQualifierId})

OPTIONAL
}

SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= {
   setPolicyQualifier,
   ...
  }

CERT-POLICY-QUALIFIER ::= CLASS {
   &id         OBJECT IDENTIFIER UNIQUE,
   &Qualifier  OPTIONAL
}
WITH SYNTAX {
   POLICY-QUALIFIER-ID  &id
   [ QUALIFIER-TYPE     &Qualifier ]
}
```

Continued on next page

CertificatePolicies Extension, continued

ASN.1 continued

```
setPolicyQualifier CERT-POLICY-QUALIFIER ::= {
      POLICY-QUALIFIER-ID  id-set-setQualifier
      QUALIFIER-TYPE       SetPolicyQualifier
   }

SetPolicyQualifier ::= SEQUENCE {
     rootQualifier      SETQualifier,
     additionalPolicies AdditionalPolicies  OPTIONAL
   }

AdditionalPolicies ::= SEQUENCE SIZE(1..3) OF AdditionalPolicy

AdditionalPolicy ::= SEQUENCE {
     policyOID          CertPolicyId  OPTIONAL,
     policyQualifier    SETQualifier  OPTIONAL,
     policyAddedBy      CertificateTypeSyntax
   }

SETQualifier ::= SEQUENCE {
       policyDigest     DetachedDigest  OPTIONAL,
       terseStatement   SETString {ub-terseStatement}  OPTIONAL,
       policyURL        [0] URL  OPTIONAL,
       policyEmail      [1] URL  OPTIONAL
   }
```

SubjectAltName Extension

Overview

This extension contains one or more alternate subject names, using any of a variety of name forms. This field is optional, and is only included if the requesting entity specifies an alternate name in the request.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
subjectAltName .GeneralNames	Name	One or more alternate names for the distinguished name in the certificate; the alternate name may be an e-mail address, a URL, etc.

ASN.1

```
subjectAltName EXTENSION ::= {
   SYNTAX         GeneralNames
   IDENTIFIED BY id-ce-subjectAltName
}

GeneralNames ::= SEQUENCE SIZE(1..MAX) OF GeneralName

GeneralName ::= CHOICE {
   directoryName              [4] EXPLICIT Name,
   uniformResourceIdentifier  [6] IA5String,
   registeredID               [8] OBJECT IDENTIFIER
   -- Other choices defined in X.509 not used by SET
}
```

BasicConstraints Extension

Overview

This extension indicates whether the certified subject may act as a CA or an EE. If the certified subject may act as a CA, the extension indicates by path length the number of levels of sub-CAs that the CA may authenticate. This extension shall be used in validating certificates used to sign other certificates

Criticality

This extension is critical.

Restrictions

Name	Format and Value Restrictions	Description
basicConstraints .basicConstraints .cA	Boolean	True for all CAs and subordinate CAs; false for End Entities.
basicConstraints .basicConstraints .pathLenConstraint	Integer; only set if the subjectType is a CA	Indicates the number of levels of CAs that this certificate may sign certificates for. For example, a zero in this field means that the CA certificate may only be used to sign EE certificates.

Usage

The BasicConstraints cA shall be set to CA only if the KeyUsage extension is set to either KeyCertSign or the combination KeyCertSign plus CRLSign. Otherwise (and including all EE certificates), cA shall always be set to false. Note that a CA may own certificates in which the basicConstraints.cA is false and may use the keys associated with such certificates in the manner specified in the KeyUsage.

ASN.1

```
basicConstraints EXTENSION ::= {
   SYNTAX         BasicConstraintsSyntax
   CRITICAL       TRUE
   IDENTIFIED BY id-ce-basicConstraints
}

BasicConstraintsSyntax ::= SEQUENCE {
   cA                 BOOLEAN  DEFAULT FALSE,
   pathLenConstraint  INTEGER (0..MAX)  OPTIONAL
}
```

IssuerAltName Extension

Overview

This extension contains one or more alternate names for the Issuer certificate. This field is optional, and is only included if the issuing CA chooses to set this extension.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
issuerAltName .GeneralNames .GeneralName	Name	One or more alternate names for the distinguished name in the certificate; the alternate name may be an e-mail address, a URL, etc.

ASN.1

```
issuerAltName EXTENSION ::= {
   SYNTAX         GeneralNames
   IDENTIFIED BY id-ce-issuerAltName
}
GeneralName ::= CHOICE {
directoryName              [4] EXPLICIT Name,
uniformResourceIdentifier  [6] IA5String,
registeredID               [8] OBJECT IDENTIFIER
-- Other choices defined in X.509 not used by SET
 }
```

Section 3
SET Private Extensions

Section Overview

Purpose

This section defines the following private extensions for SET:

- HashedRootKey
- CertificateType
- MerchantData
- CardCertRequired
- Tunneling
- SETExtensions

HashedRootKey Private Extension

Overview

This extension is used only in Root certificates and contains the thumbprint (hash) of the next root key. The hash is computed using SHA-1 over the DER encoded subjectPublicKeyInfo structure as follows:

HashedRoot := DD[subjectPublicKeyInfo]

The subjectPublicKeyInfo contains the public key algorithm identifier and the public key for the next Root and is used to authenticate the next Root certificate.

Criticality

This extension is critical.

Restrictions

Name	Format and Value Restrictions	Description
hashedRootKey .DigestedData .digestAlgorithm .algorithm	OID	The OID of the hashing algorithm used over the root key. In SET, SHA-1 is used.
HashedRootKey .DigestedData .digestAlgorithm .parameters		Set to NULL.
hashedRootKey .DigestedData .contentInfo .contentType	OID	Set to id-set-rootKeyThumb.
hashedRootKey .DigestedData .contentInfo .content		Omitted.
hashedRootKey .DigestedData .digest	Octet String	The hash of the DER encoded subjectPublicKeyInfo.

HashedRootKey Private Extension, continued

ASN.1

```
hashedRootKey EXTENSION ::= {               -- Only in root
certificates
   SYNTAX         HashedRootKeySyntax
   CRITICAL       TRUE
   IDENTIFIED BY id-set-hashedRootKey
}

HashedRootKeySyntax ::= RootKeyThumb

RootKeyThumb ::= SEQUENCE {
   rootKeyThumbprint  DD { SubjectPublicKeyInfo{{SupportedAlgorithms}}
}
}
```

CertificateType Private Extension

Overview

The certificate type is used to distinguish between the different entities. For the following EE or CA types the certificate can have only one type:

- Cardholder (CARD)
- Merchant (MER)
- Payment Gateway (PGWY)
- Geo-political Certificate Authority (GCA)
- Brand Certificate Authority (BCA)
- Root Certificate Authority (RCA)

For the following CA types, multiple certificate types are possible. For example, the certificate type can be both a Cardholder Certificate Authority and a Merchant Certificate Authority.

- Cardholder Certificate Authority (CCA)
- Merchant Certificate Authority (MCA)
- Payment Certificate Authority (PCA)

This extension is used to identify the entity with respect to the SET CA hierarchy. It is independent of the CA indicator in the Basic Constraints extension that indicates whether the certificate may be used to verify certificate signatures.

This extension is included in every SET certificate.

Criticality

This extension is critical.

Restrictions

Name	Format and Value Restrictions	Description
certificateType .certificateTypeSyntax	One value of 0 - 9 or any grouping of values 3, 4, and 5	Specifies what type of entity will be using the certificate. This field is based on the type of certificate request received.

Continued on next page

CertificateType Private Extension, continued

ASN.1

```
certificateType EXTENSION ::= {
   SYNTAX         CertificateTypeSyntax
   CRITICAL       TRUE
   IDENTIFIED BY id-set-certificateType
}

CertificateTypeSyntax ::= BIT STRING {
   card  (0),
   mer   (1),
   pgwy  (2),
   cca   (3),
   mca   (4),
   pca   (5),
   gca   (6),
   bca   (7),
   rca   (8),
   acq   (9)
}
```

MerchantData Private Extension

Overview

In the payment protocol, an Acquirer needs certain information about Merchants. This extension contains all of the data needed by the Payment Gateway. This data is obtained from the Merchant in the certificate request processing (in the registration form). The Merchant's name and address in this extension can be represented multiple times in different languages. The information shall be listed in the order of language preference.

Merchant Data in Multiple Languages

The Merchant's name and address may be repeated in multiple languages in this extension. If multiple names are included, they shall be placed in the order of the certificate holder's language preferences. The following set of fields may be included in multiple languages:

- Merchant name
- City
- State/province
- Postal code
- Country

Criticality

This extension is non-critical.

Continued on next page

MerchantData Private Extension, continued

Restrictions

Name	Format and Value Restrictions	Description
MerID	Character String; Required	The merchant identification assigned by the Acquirer
MerAcquirerBIN	Numeric String; Required	The BIN used for settlement of the merchant's transactions with the Acquirer
MerCountry	INTEGER	The ISO-3166 numeric country code for the location of the merchant
MerAuthFlag	BOOLEAN: (FALSE) not authorized to receive Cardholder information (TRUE) authorized to receive Cardholder information	Some Acquirers allow certain Merchants to receive additional Cardholder payment information in order to accommodate non-SET business processing of transactions.
The following items may appear more than once to carry information about the merchant in multiple character sets or translated into multiple languages:		
Language	Character String; Optional	RFC 1766 definition of language
Name	Character String; Required	The name by which the merchant is known to its customers
City	Character String; Required	The name of the city where the merchant is located
StateProvince	Character String; Optional	The state or province where the merchant is located
PostalCode	Character String; Optional	The postal code for the merchant's location
CountryName	Character String; Required	The name of the country (corresponds to MerCountry)

Continued on next page

MerchantData Private Extension, continued

ASN.1

```
merchantData EXTENSION ::= {
   SYNTAX         MerchantDataSyntax
   IDENTIFIED BY id-set-merchantData
}

MerchantDataSyntax ::= SEQUENCE {
   merID              MerchantID,
   merAcquirerBIN     BIN,
   merNameSeq         MerNameSeq,
   merCountry         CountryCode,
   merAuthFlag        BOOLEAN DEFAULT TRUE
}

 MerNameSeq ::= SEQUENCE SIZE(1..32) OF MerNames

 MerNames::= SEQUENCE {
   language       [0] Language OPTIONAL,
   name           [1] EXPLICIT SETString { ub-merName },
   city           [2] EXPLICIT SETString { ub-cityName },
   stateProvince  [3] EXPLICIT SETString { ub-stateProvince }
OPTIONAL,
   postalCode     [4] EXPLICIT SETString { ub-postalCode }  OPTIONAL,
   countryName    [5] EXPLICIT SETString { ub-countryName }
 }
```

CardCertRequired Private Extension

Overview

The CardCertRequired private extension indicates whether the Payment Gateway supports exchanges with Cardholders that don't have a certificate.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
cardCertRequired	Boolean	Indicates whether a Cardholder's certificate is required by the brand

ASN.1

```
cardCertRequired EXTENSION ::= {
    SYNTAX         BOOLEAN
    IDENTIFIED BY  { id-set-cardCertRequired }
 }
```

Tunneling Private Extension

Overview

The Tunneling private extension indicates whether the CA or the Payment Gateway supports the "tunneling" of encrypted messages to the Cardholder. If "tunneling" is supported, the extension indicates a list of symmetric encryption algorithms that the Payment Gateway or the CA supports. The list is in order of the CA's algorithm preference.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
Tunneling .tunneling	Boolean	Indicates whether "tunneling" is supported by the CA or Payment Gateway.
Tunneling .tunnelAlgIDs	Object Identifier	Contains a list (ordered by preference) of symmetric encryption algorithm identifiers that the CA or Payment Gateway supports.

ASN.1

```
tunneling EXTENSION ::= {
   SYNTAX         TunnelingSyntax
   IDENTIFIED BY id-set-tunneling
}

TunnelingSyntax ::= SEQUENCE {
   tunneling     BOOLEAN DEFAULT TRUE,
   tunnelAlgIDs  TunnelAlg
}

TunnelAlg ::= SEQUENCE OF OBJECT IDENTIFIER
```

SETExtensions

Overview

The SETExtensions private extension lists the SET message extensions for payment instructions that the Payment Gateway supports. The Cardholder checks the Payment Gateway certificate prior to including critical message extensions in the payment instructions. Message extensions are indicated by Object Identifiers.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
SETExtensions .SETExtensionsSyntax	Object Identifiers	List of Object Identifiers pointing to the message extensions the Payment Gateway supports.

ASN.1

```
setExtensions EXTENSION ::= {
   SYNTAX         SETExtensionsSyntax
   IDENTIFIED BY  { id-set-setExtensions }
}

SETExtensionsSyntax ::= SEQUENCE OF OBJECT IDENTIFIER
```

Section 4
Certificate Profiles

Certificate Types

Summary

Table 22 provides a complete list of all certificates needed in SET.

Entity	digital signature	key encipher-ment/data encipherment	keyCert signature	CRL signature
Cardholder	X			
Merchant	X	X		
Payment Gateway	X	X		
Cardholder Certificate Authority	X	X	X	
Merchant Certificate Authority	X	X	X	
Payment Certificate Authority	X	X	X	X
Geo-political Certificate Authority	X		X	X
Brand Certificate Authority			X	X
Root Certificate Authority			X	X

Table 22: Certificate Types

Combining entities

The CCA, MCA, and PCA do not necessarily require three distinct certificates if they are integrated functions. A single signature certificate could contain two or three different Certificate Types.

Combining KeyUsage functions

The various CAs do not necessarily need a different certificate for signing certificates and for signing CRLs. The KeyUsage field may contain:

- both the keyCertSign and the offLineCRLSign privilege, or
- both the keyEncipherment and dataEncipherment.

No other functions can be combined into one certificate.

Required End Entity Certificate Extensions

	Cardholder Certificate	Merchant Certificate		Gateway Certificate	
X.509 Extension	Signature	Signature	Key Encipherment	Signature	Key Encipherment
AuthorityKeyIdentifier	X	X	X	X	X
KeyUsage	X	X	X	X	X
PrivateKeyUsagePeriod	X	X		X	
CertificatePolicies	X	X	X	X	X
SubjectAltName	O	O	O	O	O
BasicConstraints	X	X	X	X	X
IssuerAltName	O	O	O	O	O
Private Extension					
HashedRootKey					
CertificateType	X	X	X	X	X
MerchantData		X	X		
CardCertRequired					X
Tunneling					X
SETExtensions					X

X = Required
O = Optional

Table 23: Required End Entity Certificate Extensions

Required CA Certificate Extensions

	CA				Root CA
X.509 Extension	Digital Signature	Certificate Signature	Key Encipher-ment	CRL Signature	Certificate & CRL Signature
AuthorityKeyIdentifier	X	X	X	X	
KeyUsage	X	X	X	X	X
PrivateKeyUsagePeriod	X	X		X	X
CertificatePolicies	X	X	X	X	X
SubjectAltName	O	O	O	O	O
BasicConstraints	X	X	X	X	X
IssuerAltName	O	O	O	O	O
Private Extension					
HashedRootKey					X
CertificateType	X	X	X	X	X
MerchantData					
CardCertRequired					
Tunneling			X		
SETExtensions					

X = Required
O = Optional

Table 24: Required CA Certificate Extensions

Chapter 5
Certificate Revocation List and BrandCRLIdentifier

Chapter Overview

Introduction

This chapter describes the use of the BrandCRLIdentifier and the X.509 Certificate Revocation List (CRL) in SET. The CRL is a mechanism defined by X.509 for publicizing and distributing lists of revoked, unexpired certificates. Each CA (except the MCA and CCA) will maintain a CRL. All CAs will distribute CRLs. The BrandCRLIdentifier (BCI) is defined by SET and contains a list of all the current CRLs within a given brand. Whenever a CA issues a new CRL, the associated BCI is updated. The BCI is distributed in all downstream messages. Possession of the BCI and the CRLs it identifies ensures that an End Entity is screening certificates against the latest revocation information.

Organization

Chapter 5 includes the following topics:

- X.509 CRL Data Definitions,
- CRL Extensions,
- CRL Validation, and
- BrandCRLIdentifier.

X.509 CRL Data Definitions

Overview

Each CA in SET other than the CCA and MCA is responsible for maintaining and distributing a CRL. A CA is responsible for revoking compromised certificates that it generated and signed. The CA will place the serial numbers of compromised certificates on its CRL. The CA is identified within the CRL by its distinguished name, and the CRL is signed by the CA.

CRL contents

Each CA CRL contains the following information:

- the CRL Number, where the number increases with each new CRL generated (included in the CRLNumber Extension),
- a list of serial numbers of revoked certificates,
- the date when each certificate was revoked,
- the dates when the CRL was generated and when it expires (and a new CRL is in effect),
- the distinguished name of the CA (that maintains this CRL and generated the revoked certificates),
- the issuer and serial number of the CA certificate that was used to sign this CRL (included in the authorityKeyIdentifier extension).

The table on the next page defines the format and value restrictions for each field in the X.509 CRL.

Continued on next page

X.509 CRL Data Definitions, continued

Name	Format and Value Restrictions	Description
CRL .version	Integer; V2	Indicates the CRL version, always 2.
CRL .signature .algorithmIdentifier	OID and type	Defines the algorithm used to sign the CRL.
CRL .Issuer	Name	Contains the subject DN of the CA that issued the revoked certificate. Shall match the value in the Subject Name in the CA Certificate.
CRL .thisUpdate	UTC Time	Specifies when the CRL was generated.
CRL .NextUpdate	UTC Time	Specifies when the CRL expires.
CRL .revokedCertificates .certSerialNumber	Integer	The serial numbers of the revoked certificates.
CRL .revokedCertificates .revocationDate	UTC Time	The date of revocation.
CRL .revokedCertificates .extensions	Extensions	Not used in SET.
CRL .extensions	Extensions	Two extensions are supported in this field: CRLNumber and AuthorityKeyIdentifier.

Table 25: X.509 CRL Data Definitions

Continued on next page

X.509 CRL Data Definitions, continued

CAs maintaining CRLs

The following CAs are required to maintain CRLs in SET:

- Root CA - To support unscheduled replacement of the Root certificates, or brand CA certificates.
- Brand CAs - To support the unscheduled replacement or termination of a CA certificate issued by the brand CA.
- Geopolitical CAs - To support the unscheduled replacement of CCA, MCA, or PCA entities.
- Payment Gateway CAs - To support the unscheduled replacement of Payment Gateway key-exchange certificates.

CA CRL Extensions

The following extensions are required in each CRL for each CA in the SET hierarchy:

	PCA	Geopolitical CA	Brand CA	Root CA
X.509 Extension				
AuthorityKeyIdentifier	X	X	X	X
CRL Number	X	X	X	X

CRL distribution to Cardholders and Merchants

CRLs are distributed to Cardholders and Merchants within the CRL field of the PKCS #7 SignedData. An entity in the SET protocol shows the CRLs it's holding by putting the Thumbprints in the first upstream request message. The recipient checks the Thumbprints and includes any missing CRLs in its downstream response message.

CRL distribution to CAs and Payment Gateways

CRLs are distributed to CAs and Payment Gateways using the distribution message specified starting on page 259.

Continued on next page

X.509 CRL Data Definitions, continued

CRL update

A new CRL is created whenever a certificate is revoked and the list shall be updated. When the new CRL is created, any certificates on the list which have expired may be removed. The updated CRL will contain the complete list of all unexpired, revoked certificates that the CA issued. The new CRL will be delivered to the brand for inclusion on the BrandCRLIdentifier as specified starting on page 256.

CRL Extensions

Overview

The following X.509 extensions are used with SET CRLs:

- AuthorityKeyIdentifier
- CRLNumber

Authority KeyIdentifier

The AuthorityKeyIdentifier extension is used the same for CRLs as for Certificates. See page 215.

CRLNumber

The CRLNumber extension contains a single integer value. The CA signing the CRL is required to increment the CRL number each time a new CRL is issued. This extension is non-critical.

CRLNumber restrictions

Name	Format and Value Restrictions	Description
cRLNumber	Integer	As defined above.

CRLNumber ASN.1

```
cRLNumber EXTENSION ::= { -- For use in CRLs only
   SYNTAX         CRLNumber
   IDENTIFIED BY id-ce-cRLNumber
}

CRLNumber ::= INTEGER (0..MAX)
```

CRL Validation

Overview

This section defines the rules for validation of CRLs.

Validation of the CRL

The following shall be verified:

1. The signature shall be validated.
    a) Use the AuthorityKeyIdentifier CRL extension to identify the correct signature certificate.
    b) The KeyUsage extension in the signing certificate shall indicate CRLsign (6).
2. The IssuerDN in the CRL shall match the subjectDN of the certificate used to verify the signature.
3. The IssuerDN and revoked certSerialNumber are compared with the certificate being validated.

Checking Certificates against a CRL

The following validation shall be performed to determine if a given certificate is included on a CRL:

1) The IssuerDN of the certificate in question shall match the IssuerDN field in the CRL.
2) The certSerialNumber shall match the revokedCertificates.certSerialNumber field in the CRL.

Replacement of older CRLs

Existing CRLs from the same IssuerDN may be deleted when a CRL with a higher value in the CRLNumber has been successfully validated.

BrandCRLIdentifier

Overview

The BrandCRLIdentifier is a structure defined by SET and used to identify all current CRLs for CAs under the domain of a given brand. The BCI is maintained by the Brand CA. The BCI contains a list of CRL numbers. It is distributed in all downstream response messages. An entity receiving the BCI shall verify that it holds all of the CRLs on the list. The BCI is updated every time a CA within the Brand's sub-trees updates a CRL. The BCI is signed by the Brand CA. Each brand shall maintain one BCI.

Contents of BCI

The BrandCRLIdentifier contains the following information:

a) the BCI Number (increases with each new BCI),

b) the Brand Name,

c) the validity period,

d) the list of CRL Numbers (from the CRL Number Extension),

e) the list of distinguished names of the CAs that issued the CRLs, and

f) the issuer and serial number of the Brand CA certificate that was used to sign this BCI (included in an extension).

The BCI is signed by the Brand CA using the private key corresponding to the CRLSign certificate.

Entities on BCI

The entries on a BCI consist of certificate subject names of the following SET entities:

a) Root CA
b) Any brand CA
c) Geopolitical CAs
d) Payment Gateway CA

Continued on next page

BrandCRLIdentifier, continued

BCI distribution to Cardholders and Merchants

BCIs are distributed to Cardholders and Merchants within downstream response messages. An entity in the SET protocol indicates which BCI it's holding by putting its Thumbprint in an upstream request message. The recipient checks the Thumbprint and includes the new BCI, if its Thumbprint is not present, in its downstream response message.

BCI distribution to CAs and Payment Gateways

BCIs are retrieved by CAs and Payment Gateways from the brand designated CA via a distribution message as specified starting on page 259.

BCI updates

BCIs are generated on a scheduled interval that will be set by the brand's policy.

BCI processing

The processing of a BCI specified CRL is only required when the certificate path goes through one of the entries in the BCI.

Continued on next page

BrandCRLIdentifier, continued

Restrictions

Name	Format and Value Restrictions	Description
BrandCRLIdentifier .version	Integer; V1	Indicates the BCI version. Always 1.
BrandCRLIdentifier .sequenceNum	Integer	Increasing sequence number. The higher the sequence number, the more recent the BCI.
BrandCRLIdentifier .brandName	Name	The BrandName of the BCI.
BrandCRLIdentifier .bciNotBefore	Generalized Time	Specifies when the BCI becomes valid.
BrandCRLIdentifier .bciNotAfter	Generalized Time	Specifies when the BCI expires.
BrandCRLIdentifier .crl-ID .issuerName	Name	The IssuerName of a CRL that needs to be used in signature validations.
BrandCRLIdentifier .crl-ID .crlNumber	Integer	The value of the CRLNumber extension of the CRL.
BrandCRLIdentifier .extensions	Extensions	The only extension used in the BCI is AuthorityKeyIdentifier. The same restrictions are applied as in its use in CRLs and certificates. See page 215.
AlgorithmIdentifier .algorithm	OID	
AlgorithmIdentifier .parameters	Null	
Hash	Bit string	The signature over the BCI.

Chapter 6
CA to CA Messages

Chapter Overview

Organization

This chapter addresses the following topics:

- CA to CA Certificate Requests and Responses
- CRL Distribution to brand designated CAs
- BCI Retrieval by a CA

Section 1
Certificate Requests and Responses

Certificate Requests and Responses

CA Certificate request/ response overview

This section defines the protocol used by CAs to request certificates from a superior CA and for the superior CA to send generated certificates to a subordinate CA. A PKCS #10 CertificationRequest is used to submit a certificate request. After a certificate is generated by the CA, it is returned to the subordinate CA in a PKCS #7 SignedData.

Responsibility for generating certificates

Different entities are responsible for signing different certificates, as described below.

Certificate for:	Is generated and signed by:
Brand CA	Root CA
Geopolitical CA	Brand CA
CCA, MCA, or PCA	Geopolitical CA, if there is a Geopolitical CA covering the subject CA, otherwise, Brand CA

Certification request format

Messages from a CA requesting a certificate shall be formatted in accordance with the CertificationRequest specified in PKCS #10, version 1.0. The PKCS #10 shall be encoded and wrapped in a manner appropriate to the agreed transport mechanism. Transport of the PKCS #10 from the subordinate CA to the superior CA shall be coordinated out of band.

Certification request Generation

The CertificationRequest is self-signed and contains the public key, subject DN, and attributes that the signing CA will certify. The subject DN shall comply with the Certificate Subject Name Formats specified on pages 210 and 211.

Continued on next page

Certificate Requests and Responses, continued

Extensions to the certificate

The certification request message includes information that is to appear in the certificate extensions; this information is carried in attributes in the PKCS #10 request. Attributes corresponding to the following extensions shall appear in the request based upon the CA type:

- Key Usage
- Certificate Type
- Tunneling
- Private Key Usage Period
- Subject Alternate name

Finally, the additional policy attribute may appear in the request to indicate policy information for the certificate. The additional policy is defined on page 221. The values for the attributes shall be set as specified starting on page 214.

Table 9 shows the attributes which are required or optional in the CertificationRequest based upon the CA type and the key usage.

	CCA, MCA, or PCA				Geopolitical or Brand CA
SET Attribute	Digital Signature	Certificate Signature	Key Encipherment	CRL Signature	Certificate & CRL Signature
KeyUsage	X	X	X	X	X
PrivateKeyUsagePeriod	X	X		X	X
AdditionalPolicy	O	O	O	O	O
SubjectAltName	O	O	O	O	O
CertificateType	X	X	X	X	X
Tunneling			X		

X = Required
O = Optional

Table 9: Required Certification Request Attributes

Certificate Requests and Responses, continued

Certification Request Processing

Upon receipt of a CertificationRequest, the CA shall verify the request and generate a response as follows:

Step	Action
1	Verify the authenticity of the CertificationRequest using the Brand specified procedure.
2	Verify the signature using the public key included in the request.
3	Verify that the subject Distinguished Name complies with the Certificate Subject Name format specified on pages 210 and 211.
4	Based on the certificate type and key usage attributes, verify that the required attributes have been included as detailed in Table 9.
5	For signature certificates, verify that the requested PrivateKeyUsagePeriod is within the Validity dates of the signing CA and that the notBefore date in the requested PrivateKeyUsagePeriod is within the PrivateKeyUsagePeriod of the signing CA.
6	If any of the above validation steps fail, the certificate shall not be generated and this shall be communicated out of band.
7	If validation is successful, the certificate shall be generated using the attributes included in the request. The generated certificate and its chain shall be placed in the certificates portion of SignedData. The content of SignedData will be empty and is therefore not signed.
8	The generated certificate and its chain are placed within SignedData which is placed within a Content wrapper. This is then encoded and wrapped in a form suitable for the agreed transport mechanism. The transport of the Certification Response is outside of the scope of the SET specification. Note: This is not a SET message and will not be wrapped in a SET Message Wrapper.

Section 2
CRL Distribution

CRL Distribution

Overview

A CRL is a mechanism defined by X.509 for publicizing and distributing lists of revoked, unexpired certificates. Each CA (except the MCA and CCA) relating to the Brand CA through the brand's designated CA is responsible for maintaining and delivering CRLs to the brand's designated CA when they are created. This section describes the mechanism for the CA to reliably deliver the CRL to the brand's designated CA.

CRL Updates

Whenever the Root CA updates its CRL, it shall distribute the CRL to each of the brands. Whenever a subordinate CA updates its CRL it shall distribute the CRL to its brand CA. The brand's designated CA shall provide an agreed transport mechanism through which the related CAs can send CRL Update messages.

Generate CRL Notification Message

CRLs shall be distributed to the brand designated CA within the SignedData portion of the CRLNotification message as follows:

Step	Action
1	Build CRLNotificationTBS: a. Populate the Date with the current date b. Populate the CRLThumbprint with the thumbprint of the included CRL
2	Sign the content using the notifying CA's digital signature certificate. Set the content type to id-set-content-CRLNoticationTBS.
3	Insert the new CRL in the CRLs portion of SignedData; insert the CRL's and the signing certificate's certificate chain in the certificate portion of the message.
4	Encode and wrap the signed CRLNotification message in a form suitable for the agreed transport mechanism. Note: this is not a SET message and will not be wrapped in a SET Message Wrapper. DER encode SignedData and place in a MIME wrapper.

Continued on next page

CRL Distribution, continued

CRL Notification Message Contents

The following fields are in the CRL Notification Message:

Field Name	Description
CRLNotification	**S(CA, CRLNotificationTBS)**
CRLNotificationTBS	**{ Date, CRLThumbprint }**
Date	*The date on which the message is generated.*
CRLThumbprint	*Thumbprint for the CRL included in the CRLs portion of the SignedData.*

Receive CRL Notification Message

Upon receipt of the CRL Notification message, the brand's designated CA validates and processes the message as follows:

Step	Action
1	If the Date is earlier than that in any previous CRL received from this CA, discard the message and respond to the issuing CA with a SET Error message with ErrorCode set to badDate.
2	If the CRLThumbprint does not match that for the CRL included in the CRLs portion of the SignedData, discard the message and respond to the issuing CA with a SET Error message with ErrorCode set to thumbsMismatch.
3	Store the modified CRL and transmit to the Brand CA for inclusion with a subsequent BCI Distribution Message.

Continued on next page

CRL Distribution, continued

Generate Response message

The brand's designated CA shall generate a CRL Notification Response message as follows:

Step	Action
1	Populate CRLNotificationResTBS: a. Include the date from the CRLNotification message.. b. Include the Thumbprint of the received CRL.
2	Sign the content using the brand designated CA's digital signature certificate. Set the contentType to id-set-content-CRLNotificationResTBS.
3	Encode and wrap the signed CRLNotificationRes message in a form suitable for the agreed transport mechanism. Note: this is not a SET message and will not be wrapped in a SET Message Wrapper. Send the CRL notification response message back to the CA.

CRL Notification Response Message Contents

The following fields are in the CRL Notification Response Message:

Field Name	Description
CRLNotificationRes	**S(CA, CRLNotificationResTBS)**
CRLNotificationResTBS	**{ Date, CRLThumbprint }**
Date	*Copied from the CRLNotification Message.*
CRLThumbprint	*Thumbprint for the CRL copied from the CRLNotification Message.*

Response message Verification

Upon receipt of the CRL Notification Response message, the CA shall verify that the date and the thumb of the received CRL match those in the corresponding CRLNotification message. If these are invalid, a SET error message shall be returned and the CRL shall be re-posted in a CRLNotification request message.

Section 3
BCI Retrieval

BCI Retrieval

Overview

A BrandCRLIdentifier is a list of the up-to-date CRLs corresponding to all of the CAs under the brand hierarchy. The brand's designated CA is responsible for maintaining an up-to-date BCI and providing a mechanism for CAs and Payment Gateways to retrieve the BCI and the associated CRLs. Each CA and Payment Gateway under a Brand CA is responsible for downloading, storing and providing access to an up-to-date BCI and the associated CRLs in their response messages.

BCI Host

Each SET brand shall maintain an up-to-date version of the BCI and all CRLs referenced by the BCI in a BCI Distribution Message, and the brand's designated CA shall provide one or mechanisms whereby this message can be downloaded by the supported CAs and the Payment Gateways. The CAs and Payment Gateway are responsible for retrieving the BCI Distribution message on a daily basis.

BCI Distribution Message

The BCI shall be provided for distribution by the brand's designated CA in a BCI Distribution message. The BCI Distribution message is a signed message containing the current date, BCI, and associated certificates and CRLs. The date ensures the current validity of the BCI. The brand's designated CA shall generate a new BCI Distribution message daily. Note: the BCI is not generated daily, just the distribution message.

Continued on next page

BCI Retrieval, continued

BCI Distribution Message Generation

The BCI Distribution message shall be generated by the brand's designated CA as follows:

Step	Action
1	Populate BCIDistributionTBS: a. Populate Date with the current date. b. Include the latest BCI.
2	Sign the content using the CA digital signature certificate. Set the contentType to id-set-content-BCIDistributionTBS. In the CRLs portion of SignedData, insert all of the CRLs listed on the BCI. In the certificates portion, insert all of the certificates necessary to verify all of the CRLs.
3	Encode and wrap the signed BCIDistribution message in a form suitable for the agreed transport mechanism. Note: This is not a SET message and will not be wrapped in a SET Message Wrapper.

BCI Distribution Message Contents

The following fields are in the BCI Distribution Message:

Field Name	Description
BCIDistribution	**S(CA, BCIDistributionTBS)**
BCIDistributionTBS	**{ Date, BrandCRLIdentifier }**
Date	*The date on which the message is generated.*
BrandCRLIdentifier	*List of current CRLs for all CAs under the Brand CA, the Brand CA itself and the Root CA.. Signed by the brand's designated CA.*

Continued on next page

BCI Retrieval, continued

BCI Distribution Message Retrieval

The BCI Distribution message shall be processed by the receiving CA or Payment Gateway as follows:

Step	Action
1	Extract the message from any transport specific wrapper and decode. Verify the message signature using the signature certificate of the brand's designated CA.
2	If the Date is earlier than that contained in a previously retrieved BCI Distribution Message, discard the message.
3	If the BrandCRLIdentifier differs from that currently stored, verify the signatures of each of the CRLs listed on the BCI. If the signatures do not verify or CRLs referenced in the BCI are not included in the message, discard the message.
4	Store the CRLs and the BrandCRLIdentifier for distribution in SET messages.

Part III
Payment System

Overview

Introduction

Part III presents the description and processing of the payment system portion of the SET protocol, including all messages related to authorization, capture, and management of the payment system.

Organization

Part III includes the following chapters:

Chapter 1
Common Data and Flows

Overview

Introduction

Chapter 1 presents the data structures common to payment messages, and presents the message flow model for the payment system.

Organization

Chapter 1 includes the following sections:

Section	Title	Contents	Page
1	Data Structures	Presents data structures common to multiple payment messages.	265
2	General Flow	Presents a summary of a typical payment flow, plus a summary of all messages which may be present in payment system flows.	301

Notation

The notation used in the tables describing structure of the messages is presented on page 59.

Section 1
Data Structures

Definition

SET messages include several data structures that bear data items which recur from message to message, representing control structures, recurring application data, etc.

The following tables define logically related groups of fields that appear in several places in the various messages. These definitions are presented here for ease of reference, and to provide the common understanding needed to understand the protocol.

Data Structure	Page
TransIDs	267
PI (Payment Instruction)	271
InstallRecurData	274
AuthToken	275
AcqCardMsg	278
CapToken	280
PANData	283
PANToken	284
SaleDetail	285
Commercial card data	288
MarketAutoCap	290
MarketHotelCap	292
Market transport data	293
Location	294
RRTags	295
BatchStatus	296
TransactionDetail	298
Amount Fields	299
Date Fields	300

Continued on next page

Overview, continued

Payloads

Payloads are message components that do not directly affect protocol logic, that is, whether and when to send messages and how to treat them cryptographically. From the protocol point of view, they are opaque fields. However, they constitute the payment content of the messages. Protocol software is concerned with encryption, decryption, signatures, checking for hash matches, and so on. Payment software is concerned with processing payloads.

TransIDs

Purpose

TransIDs provides all the information to uniquely define the transaction and transaction characteristics of which the message is a part. In particular, TransIDs enables an entity (protocol participant) to relate each message to the transaction of which it is a part, and therefore to the request/response pair (since the request/response pairs can occur only once in each transaction).

TransIDs data

TransIDs	**{LID-C, [LID-M], XID, PReqDate, [PaySysID], Language }**
LID-C	*Local ID; convenience label generated by and for Cardholder system.*
LID-M	*Local ID; convenience label generated by and for Merchant system.*
XID	*Globally unique ID.*
PReqDate	*Date of purchase request; generated by Merchant in* **PInitRes** *or by Cardholder in* **PReq**.
PaySysID	*Used by some payment card brands to label transaction from time of authorization onward*
Language	*Cardholder's natural language*

Notes

TransIDs provides a variety of identifiers for transactions. XID is described below. LID-C, LID-M, and PaySysID are identifiers which are assigned by the Cardholder, Merchant, and/or payment system infrastructure to tag transactions in a manner convenient for each of them; however, other parties may not assume characteristics of these labels. LID-M may often be used to hold the Merchant's order number associated with the transaction. PReqDate provides the date of the transaction start and Language provides the language the Cardholder requests for the transaction. They are included here for convenience so that they travel with each message.

Generating XID

XID is a transaction ID that is usually generated by the Merchant system, unless there is no PInitRes, in which case it is generated by the Cardholder system. It is a randomly generated 20 byte variable that is globally unique (statistically). Merchant and Cardholder systems shall use appropriate random number generators to ensure the global uniqueness of XID.

Continued on next page

TransIDs, continued

TransIDs field generation and usage

The following table specifies when a TransIDs field is generated and used in all the SET messages. The notation used is (notes are listed after the table):

R	Field is required; it is generated by sender of the message and copied to the MessageWrapper.
C	Field is conditional. It may be generated for this message and repeated in MessageWrapper. Otherwise, it is copied from previous message if it exists.
N/P	Not Present in the message or in the MessageWrapper.
⇩	Copied from the request or the previous message in the protocol. Repeated in MessageWrapper.
I	May be present in an "item" in a data structure of the message. Not present in the MessageWrapper.

Message	LID-C	LID-M	XID	PaySysID
PInitReq	R	C_1	N/P	N/P
PInitRes	⇩	⇩ (C_2)	R	N/P
PReq	⇩	⇩	⇩ (R_3)	N/P
PRes	⇩	⇩ (C_2)	⇩	C_4
InqReq	⇩	⇩	⇩	C_5
InqRes	⇩	⇩	⇩	C_4
AuthReq	⇩	⇩	⇩	N/P
AuthRes	⇩	⇩	⇩	C_6
AuthRevReq	⇩	⇩	⇩	C
AuthRevRes	⇩	⇩	⇩	⇩
CapReq	I	I	I	I
CapRes	I	I	I	I
CapRevReq	I	I	I	I
CapRevRes	I	I	I	I
CredReq	I	I	I	I
CredRes	I	I	I	I
CredRevReq	I	I	I	I
CredRevRes	I	I	I	I

Table 26: TransIDs Field Usage

Continued on next page

TransIDs, continued

Message	LID-C	LID-M	XID	PaySysID
PCertReq	N/P	C	N/P	N/P
PCertRes	N/P	⇩	N/P	N/P
BatchAdminReq	I	I	I	I
BatchAdminRes	I	I	I	I
CardCInitReq	R	N/P	N/P	N/P
CardCInitRes	⇩	N/P	N/P	N/P
Me-AqCInitReq	N/P	C	N/P	N/P
Me-AqCInitRes	N/P	⇩	N/P	N/P
RegFormReq	⇩	⇩	N/P	N/P
RegFormRes	⇩	⇩	N/P	N/P
CertReq	⇩	⇩	N/P	N/P
CertRes	⇩	⇩	N/P	N/P
CertInqReq	⇩	⇩	N/P	N/P
CertInqRes	⇩	⇩	N/P	N/P

Table 26: TransIDs Field Usage (continued)

Notes:
1. Copied from the SET Initiation Process if present.
2. If no previous LID-M for this transaction exists, the Merchant may generated one for this message.
3. If no PInitReq/PInitRes pair, generated by the Cardholder.
4. If sent after receiving an AuthRes with a PaySysID.
5. If sent after receiving a PRes with a PaySysID.
6. Payment Gateway may generate for this message.

Continued on next page

TransIDs, continued

Generation of TransIDs

Construct a TransIDs as follows:

Step	Action
1	If a message has been previously received for the current transaction, copy all fields which have been received.
2	If this is a new transaction, generate all required fields as specified in individual messages and the table above.
3	Populate any optional fields desired which may be generated by the current entity; these are specified in individual messages.

Processing of TransIDs

The processing of TransIDs depends on the message. The processing of each message with a TransIDs will describe its processing.

PI (Payment Instruction)

Purpose

PI (Payment Instruction) is the most central and sensitive data structure in SET. It is used to pass the data required to authorize a payment card payment from the Cardholder to the Payment Gateway, which will use the data to initiate a payment card transaction through the traditional payment card financial network. The data is encrypted by the Cardholder and sent via the Merchant, such that the data is hidden from the Merchant unless the Acquirer passes the data back to the Merchant.

Variations

There are three versions of the PI. The first two are created by Cardholders; the third is created by Payment Gateways to support split shipments and recurring payments. The variations are:

PIUnsigned	Created by a Cardholder with no signature certificate. Used in a PReqUnsigned message. Data integrity is provided through the addition of a hash of the PI data which is protected in the OAEP block. No source authentication is provided by this mechanism.
PIDualSigned	Created by a Cardholder who possesses a signature certificate. Used in a PReqDualSigned message. The cardholder's signature authenticates the source as well as providing data integrity.
AuthToken	Created by the Payment Gateway. The Merchant extracts the **PI** for later incorporation into **AuthReq**. This version is used to support split shipment, and is passed back from the Payment Gateway after initial authorization to be used to request subsequent authorizations.

Continued on next page

PI (Payment Instruction), continued

PI data

PI	**< PIUnsigned, PIDualSigned, AuthToken >** *Cardholder creates* **PIUnsigned** *or* **PIDualSigned**. *Payment gateway creates* **AuthToken** *to support split shipments or installment/recurring payments.* *Merchant shall retain the* **PI** *for later incorporation into* **AuthReq**.
PIUnsigned	**EXH(P, PI-OILink, PANToken)** *See page 284 for* **PANToken**.
PIDualSigned	**{PISignature, EX(P, PI-OILink, PANData)}** *See page 283 for* **PANData**.
AuthToken	*See page 275.*
PI-OILink	**L(PIHead, OIData)** *See page 273 for* **PIHead**. *See page 325 for* **OIData**.
PISignature	**SO(C, PI-TBS)**
PI-TBS	**{HPIData, HOIData}**
HPIData	**DD(PIData)**
HOIData	**DD(OIData)** *See page 325 for* **OIData**.
PIData	**{PIHead, PANData}** *See page 273 for* **PIHead**. *See page 283 for* **PANData**.

Table 27: PI

Continued on next page

PI (Payment Instruction), continued

PIHead

PIHead	**{TransIDs, Inputs, MerchantID, [InstallRecurData], TransStain, SWIdent, [AcqBackKeyData], [PIExtensions]}**
TransIDs	*See page 267.*
Inputs	**{HOD, PurchAmt}**
MerchantID	*Copied from Merchant signature certificate*
InstallRecurData	*See page 274.*
TransStain	**HMAC(XID, CardSecret)**
SWIdent	*String identifying the software (vendor and version) initiating the request. It is specified in the* **PI** *so the Payment Gateway knows the software of the Cardholder.*
AcqBackKeyData	**{AcqBackAlg, AcqBackKey}**
PIExtensions	*The data in an extension to the payment instructions must be financial and should be important for the processing of an authorization by the Payment Gateway, the financial network, or the issuer.*

Table 28: PIHead

Notes

The application data in this message consists of PIData, from which PANData is distinguished by being provided a stronger cryptographic treatment. PANData is the payment card information. PIData includes all other purchase data, and transaction identification and cryptographic support variables. These are detailed in the discussion of the purchase messages later in this document.

InstallRecurData

Purpose

InstallRecurData allows the Cardholder to authorize installment payments or recurring payments.

InstallRecurData data

InstallRecurData	**{InstallRecurInd, [IRExtensions]}**
InstallRecurInd	**< InstallTotalTrans, Recurring >**
IRExtensions	*The data in an extension to installment or recurring data must be financial and should relate to the processing of subsequent authorizations by the Merchant and the Payment Gateway.* *Note: The installment/recurring data is not transmitted to the issuer.*
InstallTotalTrans	*Cardholder specifies a maximum number of permitted Authorizations for installment payments.*
Recurring	**{RecurringFrequency, RecurringExpiry}**
RecurringFrequency	*The minimum number of days between Authorizations (a frequency of monthly is indicated by a value of 28), and…*
RecurringExpiry	*a final date, after which no further Authorizations are permitted.*

Table 29: InstallRecurData

Extension guideline

The split/recurring data is a component of the payment instructions that is copied into the authorization token. The data in an extension shall be financial and should relate to the processing of subsequent authorizations by the Merchant and the Payment Gateway.

Note: The split/recurring data is not transmitted to the Issuer.

AuthToken

Purpose

AuthToken represents data required by a Payment Gateway for subsequent authorizations of a transaction, which is provided by a Payment Gateway at the time of previous authorization of the transaction. The Payment Gateway updates the AuthToken as necessary, and only the Payment Gateway can read the information it contains.

AuthToken data

AuthTokenData	**{TransIDs, PurchAmt, MerchantID, [AcqBackKeyData], [InstallRecurData], [RecurringCount], PrevAuthDateTime, TotalAuthAmount, AuthTokenOpaque}**
PANToken	*Fields copied from Cardholder-produced PIHead. See page 273.*
TransIDs	
PurchAmt	
MerchantID	
AcqBackKeyData	
InstallRecurData	
RecurringCount	*Number of recurring Authorizations performed so far.*
PrevAuthDateTime	*Date and time of Merchant's last Authorization in a sequence of recurring Authorizations.*
TotalAuthAmount	*The total amount authorized so far by all Authorizations for this* **XID**.
AuthTokenOpaque	*Opaque data defined by the generating Payment Gateway.*

Table 30: AuthToken

Continued on next page

AuthToken, continued

Generation of AuthToken

1	Construct AuthTokenTBE as follows:	
	If this is the first authorization (i.e. it is being generated from a PI)	a) Populate PANToken, TransIDs, PurchAmt, MerchantID and, if present in the PI, AcqBackInfo and InstallRecurData from the PI. b) Populate RecurringCount with 1. c) Populate PrevAuthDateTime with the current date. d) Populate TotalAuthAmount with the AuthAmt from the authorization response which will contain this AuthToken.
	If this is a subsequent authorization (i.e. it is being generated from a previous AuthToken)	a) Populate PANToken, TransIDs, PurchAmt, MerchantID and, if present, AcqBackInfo and InstallRecurData from the previous AuthToken. b) Increment the RecurringCount by 1. c) Populate PrevAuthDateTime with the current date. d) Increase TotalAuthAmount by the AuthAmt from the authorization response which will contain this AuthToken.
	If this is an authorization reversal (i.e. it is being generated from a previous AuthToken)	a) Populate PANToken, TransIDs, PurchAmt, MerchantID, PrevAuthDateTime and, if present, AcqBackInfo and InstallRecurData from the previous AuthToken. b) If this is a reversal of the full authorized amount (i.e. the AuthNewAmt in the AuthRevReq is 0), decrement the RecurringCount by 1. c) Decrease TotalAuthAmount by the AuthNewAmt from the authorization reversal response which will contain this AuthToken.
2	Construct PANToken (see page 284).	
3	Envelope the data with the EncX encapsulation using P1 = P2 = Cert-PE as the s and r parameters, the AuthTokenTBE (from Step 1) as parameter t, and PANToken (from Step 2) as parameter p.	

Continued on next page

AuthToken, continued

Processing of AuthToken

Step	Action
1	Extract the AuthToken from the EncX encapsulation using the Payment Gateway's private encryption key.
2	If this is an authorization request and the AuthToken has already been used in an authorization, set the AuthCode to piPreviouslyUsed.
3	If this is an authorization reversal request and the AuthToken has not been used in an authorization, set the AuthCode to piAuthMismatch.
4	If this is an authorization request and InstallRecurData is specified with Recurring information: A. Verify that the current date is before the RecurringExpiry date. If it does not verify, set AuthCode to recurringExpired. B. Verify that the current date is later than the PrevAuthDate plus the number of days specified in RecurringFrequency. If it does not verify, set AuthCode to recurringTooSoon.
5	If this is an authorization request and InstallRecurData is specified with Installment information, perform specific payment card brand policy.
6	If AuthCode has not been set in the previous steps, forward the data from the AuthToken to the authorization process.

AcqCardMsg

Purpose

This field provides a mechanism for an Acquirer to send a message back to the Cardholder, without exposing it to the Merchant. It may be sent after the Payment Gateway has received the AuthReq message from the Merchant.

AcqCardMsg data

AcqCardMsg	**EncK(AcqBackKeyData, P, AcqCardCodeMsg)** **AcqBackKeyData** *is supplied by the Cardholder in the* **PI**. *The encrypted message is destined to the Cardholder.*
AcqBackKeyData	*Copied from* **PIHead.AcqBackKeyData**; *see page 273.*
AcqCardCodeMsg	**{AcqCardCode, AcqCardMsgData}**
AcqCardCode	*Enumerated code.*
AcqCardMsgData	**{[AcqCardText], [AcqCardURL], [AcqCardPhone]}**
AcqCardText	*Textual message to be displayed to Cardholder.*
AcqCardURL	*URL referencing HTML message to be displayed to Cardholder.*
AcqCardPhone	*Phone number to be presented to the Cardholder.*

Table 31: AcqCardMsg

Notes

This is tunneled from the Acquirer to the Cardholder through the Merchant. The Cardholder sends the symmetric key needed to decrypt it upstream in the **PI**. The Merchant receives it in **AuthRes** and is required to copy it to **PRes** and **InqRes**.

This is an optional field of the protocol, and it is available only if supported by the profile of a payment card brand via the Payment Gateway's encryption certificate (Cert-PE).

Continued on next page

AcqCardMsg, continued

AcqCardCode

The following values are defined for **AcqCardCode.**

messageOfDay	A message the Acquirer wishes to display to all users.
accountInfo	Information about the account to be passed back to the user.
callCustomerService	Prompts the application to display a message requesting that the user call Customer Service.

Figure 25: Enumerated Values for AcqCardCode

CapToken

Purpose

CapToken represents data required by a Payment Gateway for capture of a transaction, which is provided by a Payment Gateway at the time of authorization of the transaction. It is generated by the Payment Gateway when an authorization is requested without a capture.

CapToken data

CapToken	**< Enc(P1, P2, CapTokenData),** **EncX(P1, P2, CapTokenData, PANToken),** **{} >** **P1** *and* **P2** *denote Payment Gateways:* • **P1** *is the sender.* • **P2** *is the receiver.* *In this version of SET,* **P1** *and* **P2** *are always the same Payment Gateway.*
CapTokenData	**{AuthRRPID, AuthAmt, TokenOpaque}**
PANToken	*See page 284.*
AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *or* **AuthRevReq**.
AuthAmt	*Actual amount authorized, which may differ from Cardholder's* **PurchAmt**.
TokenOpaque	*Opaque data defined by the generating Payment Gateway.*

Table 32: CapToken

Continued on next page

CapToken, continued

Generating CapToken

This version of SET, only supports encryption to the same Payment Gateway; that is, P1=P2.

Step	Action
1	If generated during an authorization process, set the AuthAmt in CapTokenData equal to the AuthAmt to be returned in the AuthRes. Otherwise, if generated during an authorization reversal process, set the AuthAmt in CapTokenData equal to the AuthNewAmt to be returned in the AuthRevRes.
2	Populate the TokenOpaque in CapTokenData with private data required for clearing.
3	If the Merchant normally receives PANToken from the Acquirer, then: a) Populate PANToken from details in the PI. b) Use EncX encapsulation with CapTokenData in the normally encrypted part and PANToken in the extra encrypted part. Otherwise: a) Use Enc encapsulation with CapTokenData.

Continued on next page

CapToken, continued

Processing of CapToken

Step	Action
1	Extract the CapTokenData from the EncX or Enc encapsulation using the Payment Gateway's private encryption key.
2	If this is a capture request and the CapToken has already been used in a capture request, set the CapCode in the CapResPayload to duplicateRequest.
3	If this is a capture reversal request, credit request, or credit reversal request and the CapToken has not previously been used in a capture request, set the CapRevOrCredCode in the CapRevOrCredResPayload to originalNotFound.
4	If this is a capture reversal request and the CapToken has already been used in a capture reversal request, set the CapRevOrCredCode in the CapRevOrCredResPayload to duplicateRequest.
5	If CapCode or CapRevOrCredCode has not been set in the previous step, forward the data from the CapToken to the capture process.

PANData

Purpose

PANData contains the information identifying the specific payment card account. The structure is broken out so that it can conveniently be separated and encrypted under appropriately strong encryption for sensitive data.

PANData data

PANData	**{PAN, CardExpiry, PANSecret, EXNonce}** *Always in the extra (OAEP) slot of an encapsulation operator.*
PAN	*Primary Account Number; typically, the account number on the card.*
CardExpiry	*Expiration date on the card.*
PANSecret	*Secret value shared among Cardholder, Payment Gateway, and Cardholder CA; prevents guessing attacks on* **PAN** *in the Cardholder certificate.*
EXNonce	*A fresh nonce to foil dictionary attacks on* **PANData**.

Table 33: PANData

Generating PANData

Step	Action
1	Populate PAN with the cardholder's account number.
2	Populate CardExpiry with the cardholder's account expiration date.
3	Populate PANSecret, which was received from Certificate Authority with the Cardholder certificate. For a cardholder without a certificate, all the octets of this field are set to zero (00 hex).
4	Generate a fresh EXNonce.

PANToken

Purpose

PANToken, like PANData, contains the information identifying the specific payment card account. PANToken is used when PANSecret is not needed to provide blinding of the data.

PANToken data

PANToken	**{PAN, CardExpiry, EXNonce}** *Always in the extra (OAEP) slot of an encapsulation operator.*
PAN	*Primary Account Number; typically, the account number on the card.*
CardExpiry	*Expiration date on the card.*
EXNonce	*A fresh nonce to foil dictionary attacks on* **PANToken**.

Table 34: PANToken

Generating PANToken

Step	Action
1	Populate PAN with the cardholder’s account number.
2	Populate CardExpiry with the cardholder ’s account expiration date.
3	Generate a fresh EXNonce.

SaleDetail

Purpose

SaleDetail collects data associated with sale represented by the payment card transaction. It is generated as part of the settlement process between the Merchant and the Payment Gateway.

Brand-specific mapping

The mapping of the order identification from **SaleDetail** to the data formats used by payment card brands will be published as requirements in brand-specific documentation and does not appear in the SET version 1.0 documents.

Merchant generating SaleDetail

For AuthReq, CredReq, and CapReq, optional: Populate BatchID with the value of a currently open batch for the Brand and BIN associated with the transaction and a BatchSequenceNumber.

For AuthReqRev CapReqRev, and CredReqRev: copy BatchID and BatchSequenceNumber from corresponding AuthReqPayload (for a AuthReqReq), from corresponding CapReq (for a CapReqRev), or from corresponding CredReq (for a CredReqRev).

Payment Gateway receives SaleDetail

For AuthReq, CredReq, and CapReq: if present, verify the BatchID is an open batch for the Brand and BIN; and verify that the BatchSequenceNumber is unique within the Batch.

For AuthReqRev, CredReqRev, and CapReqRev: if present, verify that BatchID and BatchSequenceNum correspond to the AuthRRPID of the original transaction.

Extension guideline

The sale detail carries information from the Merchant necessary for the Payment Gateway to produce a clearing request message (for payment) that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the sale detail shall be financial and should be important for the processing of a capture request by the Payment Gateway, the financial network or the Issuer.

Continued on next page

SaleDetail, continued

SaleDetail data

SaleDetail	**{[BatchID], [BatchSequenceNum], [PayRecurInd], [MerOrderNum], [AuthCharInd], [MarketSpecSaleData], [CommercialCardData], [OrderSummary], [CustomerReferenceNumber], [CustomerServicePhone], OKtoPrintPhoneInd, [SaleExtensions]}** *Note: This field may appear in an* **AuthReq** *with* **CaptureNow** *set to* **TRUE** *or in the capture-related messages; when appearing in* **AuthReq**, *the fields noted as originating from* **AuthResPayload** *are not present.*
BatchID	*Identification of the settlement batch for merchant-acquirer accounting.*
BatchSequenceNum	*The sequence number of this item within the batch.*
PayRecurInd	*Enumerated transaction type.*
MerOrderNum	*Merchant order number.*
AuthCharInd	*Copied from* **AuthResPayload**; *see page 359.*
MarketSpecSaleData	**{[MarketSpecDataID], [MarketSpecCapData]}**
CommercialCardData	*Description of items for this capture; see page 288. Typically, this information is only included for commercial card products under special arrangement between the merchant and the customer.*
OrderSummary	*A summary description of the order.*
CustomerReferenceNumber	*A reference number assigned to the order by the Cardholder.*
CustomerServicePhone	*The Merchant's customer service telephone number*
OKtoPrintPhoneInd	*A Boolean value indicating if the Issuer may print the customer service telephone number on the Cardholder's statement.*

Table 35: SaleDetail

Continued on next page

SaleDetail, continued

SaleDetail data (continued)

SaleExtensions	*The data in an extension to the sale detail must be financial and should be important for the processing of a capture request by the Payment Gateway, the financial network, or the issuer.*
MarketSpecDataID	*Copied from* **AuthResPayload***; see page 359.*
MarketSpecCapData	**< MarketAutoCap, MarketHotelCap, MarketTransportCap >** *Market-specific capture data.*
MarketAutoCap	*Automobile rental charge description. See page 290.*
MarketHotelCap	*Hotel charge description. See page 287.*
MarketTransportCap	*Passenger transport data. See page 293.*

Table 35: SaleDetail, continued

PayRecurInd

The following values are defined for **PayRecurInd**.

unknown	*The type of transaction is unknown*
singleTransaction	*The transaction consists of a single authorization and capture*
recurringTransaction	*The transaction consists of multiple authorizations and captures that are repeated on a regular basis*
installmentPayment	*The transaction consists of multiple authorizations and captures that are performed a fixed number of times*
otherMailOrder	*Any other mail order transaction*

Figure 26: Enumerated Values for PayRecurInd

Continued on next page

SaleDetail, continued

Commercial card data

CommercialCardData	**{[ChargeInfo], [MerchantLocation], [ShipFrom], [ShipTo], [ItemSeq]}**
ChargeInfo	**{[TotalFreightShippingAmount], [TotalDutyTariffAmount], [DutyTariffReference], [TotalNationalTaxAmount], [TotalLocalTaxAmount], [TotalOtherTaxAmount], [TotalTaxAmount], [MerchantTaxID], [MerchantDutyTariffRef], [CustomerDutyTariffRef], [SummaryCommodityCode], [MerchantType]}**
MerchantLocation	**Location**; *see page 294*
ShipFrom	**Location**; *see page 294*
ShipTo	**Location**; *see page 294*
ItemSeq	**{Item +}** *1 to 999 item level detail records*
TotalFreightShippingAmount	*The total amount added to the order for shipping and handling.*
TotalDutyTariffAmount	*The total amount of duties or tariff for the order.*
DutyTariffReference	*The reference number assigned to the duties or tariff for the order.*
TotalNationalTaxAmount	*The total amount of national tax (sales or VAT) applied to the order.*
TotalLocalTaxAmount	*The total amount of local tax applied to the order.*
TotalOtherTaxAmount	*The total amount of other taxes applied to the order.*
TotalTaxAmount	*The total amount of taxes applied to the order.*
MerchantTaxID	*The tax identification number of the Merchant.*
MerchantDutyTariffRef	*The duty or tariff reference number assigned to the Merchant.*
CustomerDutyTariffRef	*The duty or tariff reference number assigned to the Cardholder.*

Table 36: CommercialCardData

Continued on next page

SaleDetail, continued

Commercial card data (continued)

SummaryCommodityCode	*The commodity code that applies to the entire order.*
MerchantType	*The type of merchant.*
Item	**{Quantity, [UnitOfMeasureCode], Descriptor, [CommodityCode], [ProductCode], [UnitCost], [NetCost], DiscountInd, [DiscountAmount], [NationalTaxAmount], [NationalTaxRate], [NationalTaxType], [LocalTaxAmount], [OtherTaxAmount], ItemTotalCost}**
Quantity	*The quantity for the line item.*
UnitOfMeasureCode	*The unit of measure for the line item.*
Descriptor	*A description of the line item.*
CommodityCode	*The commodity code for the line item.*
ProductCode	*The product code for the line item.*
UnitCost	*The unit cost of the line item.*
NetCost	*The net cost per unit of the line item.*
DiscountInd	*Indicates if a discount was applied.*
DiscountAmount	*The amount of discount applied to the line item.*
NationalTaxAmount	*The amount of national tax (sales or VAT) applied to the line item.*
NationalTaxRate	*The national tax (sales or VAT) rate applied to the line item.*
NationalTaxType	*The type of national tax applied to the line item.*
LocalTaxAmount	*The amount of local tax applied to the line item.*
OtherTaxAmount	*The amount of other taxes applied to the line item.*
ItemTotalCost	*The total cost of the line item.*

Table 36: CommercialCardData, continued

Continued on next page

SaleDetail, continued

MarketAutoCap

MarketAutoCap	**{[RenterName], [RentalLocation], RentalDateTime, [AutoNoShow], [RentalAgreementNumber], [ReferenceNumber], [InsuranceType], [AutoRateInfo], [ReturnLocation], ReturnDateTime, AutoCharges}**
RenterName	*The name of the person renting the vehicle.*
RentalLocation	**Location***; see page 294.*
RentalDateTime	*The date (and optionally time) the vehicle was rented.*
AutoNoShow	*Enumerated code indicating that the customer failed to show up to rent the vehicle as scheduled.*
RentalAgreementNumber	*The rental agreement number.*
ReferenceNumber	*The rental reference number.*
InsuranceType	*The type of insurance selected by the renter.*
AutoRateInfo	**{AutoApplicableRate, [LateReturnHourlyRate], [DistanceRate], [FreeDistance], [VehicleClassCode], [CorporateID]}**
ReturnLocation	**Location***; see page 294.*
ReturnDateTime	*The date (and optionally time) the vehicle was returned.*
AutoCharges	**{RegularDistanceCharges, [LateReturnCharges], [TotalDistance], [ExtraDistanceCharges], [InsuranceCharges], [FuelCharges], [AutoTowingCharges], [OneWayDropOffCharges], [TelephoneCharges], [ViolationsCharges], [DeliveryCharges], [ParkingCharges], [OtherCharges], [TotalTaxAmount], [AuditAdjustment]}**
AutoApplicableRate	**<DailyRentalRate, WeeklyRentalRate>**
LateReturnHourlyRate	*The hourly charge for late returns.*
DistanceRate	*The rate charged per mile in excess of any free distance allowance.*
FreeDistance	*The distance the vehicle can travel per day without incurring an additional charge.*

Table 37: MarketAutoCap

Continued on next page

SaleDetail, continued

MarketAutoCap (continued)

VehicleClassCode	*The class of vehicle rented.*
CorporateID	*The corporate identification number that applies to the rental rate.*
RegularDistanceCharges	*The amount of charges for the rental (excluding extras classified below).*
LateReturnCharges	*The amount of charges for returning the vehicle after the date and time due back.*
TotalDistance	*The total distance the vehicle was driven.*
ExtraDistanceCharges	*The amount of the charges resulting from exceeding the free distance allowance.*
InsuranceCharges	*The amount of charges resulting from insurance.*
FuelCharges	*The amount of refueling charges.*
AutoTowingCharges	*The amount of charges resulting from towing.*
OneWayDropOffCharges	*The amount of the drop-off charges resulting from a one-way rental.*
TelephoneCharges	*The amount of charges resulting from the use of the rental vehicle telephone.*
ViolationsCharges	*The amount of charges resulting from violations assessed during the rental period.*
DeliveryCharges	*The amount of charges resulting from the delivery of the rental vehicle.*
ParkingCharges	*The amount of charges resulting from parking the rental vehicle.*
OtherCharges	*The amount of other charges not classified elsewhere.*
TotalTaxAmount	*The total amount of taxes applied to the rental.*
AuditAdjustment	*The amount the transaction was adjusted as a result of auditing by the rental company.*
DailyRentalRate	*The daily rental rate.*
WeeklyRentalRate	*The weekly rental rate.*

Table 37: MarketAutoCap, continued

Continued on next page

SaleDetail, continued

MarketHotelCap

MarketHotelCap	**{ArrivalDate, [HotelNoShow], DepartureDate, [DurationOfStay], [FolioNumber], [PropertyPhone], [CustomerServicePhone], [ProgramCode], [HotelRateInfo], HotelCharges}**
ArrivalDate	*The date the Cardholder checked in (or was scheduled to check in) to the hotel.*
HotelNoShow	*Enumerated code indicating that the customer failed to check in to the hotel as scheduled.*
DepartureDate	*The date the Cardholder checked out of the hotel.*
DurationOfStay	*The number of days the Cardholder stayed in the hotel.*
FolioNumber	*The folio number.*
PropertyPhone	*The telephone number of the hotel.*
CustomerServicePhone	*The customer service telephone number (of the hotel or the hotel chain).*
ProgramCode	*A code indicating the type of special program that applies to the stay.*
HotelRateInfo	**{DailyRoomRate, [DailyTaxRate]}**
HotelCharges	**{RoomCharges, [RoomTax], [PrepaidExpenses], [FoodBeverageCharges], [RoomServiceCharges], [MiniBarCharges], [LaundryCharges], [TelephoneCharges], [BusinessCenterCharges], [ParkingCharges], [MovieCharges], [HealthClubCharges], [GiftShopPurchases], [FolioCashAdvances], [OtherCharges], [TotalTaxAmount], [AuditAdjustment]}**
DailyRoomRate	*The daily room rate. This value includes applicable taxes unless the* **DailyTaxRate** *is specified.*
DailyTaxRate	*The amount of taxes applied to the daily room rate.*

Table 38: MarketHotelCap

Continued on next page

SaleDetail, continued

MarketHotelCap (continued)

RoomCharges	*The total amount charged for the room (excluding extras classified below).*
RoomTax	*The amount of tax applied to the* **RoomCharges**.
PrepaidExpenses	*The total amount of pre-paid expenses.*
FoodBeverageCharges	*The total amount of food and beverage charges.*
RoomServiceCharges	*The total amount of room service charges.*
MiniBarCharges	*The total amount of mini bar charges.*
LaundryCharges	*The total amount of laundry charges.*
TelephoneCharges	*The total amount of telephone charges.*
BusinessCenterCharges	*The total amount of business center charges.*
ParkingCharges	*The total amount of parking charges.*
MovieCharges	*The total amount of in-room movie charges.*
HealthClubCharges	*The total amount of health club charges.*
GiftShopPurchases	*The total amount of gift shop purchase charges.*
FolioCashAdvances	*The total amount of cash advances applied to the room.*
OtherCharges	*The total amount of other charges (not classified above).*
TotalTaxAmount	*The total amount of taxes applied to the bill.*
Audit Adjustment	*The amount the transaction was adjusted as a result of auditing by the hotel.*

Table 38: MarketHotelCap, continued

Market transport data

MarketTransportCap	**{PassengerName, DepartureDate, OrigCityAirport, [TripLegSeq], [TicketNumber], [TravelAgencyCode], [TravelAgencyName], [Restrictions]}**
PassengerName	*The name of the passenger to whom the tickets were issued.*
DepartureDate	*The departure date.*
OrigCityAirport	*The city of origin for the trip.*
TripLegSeq	**{TripLeg +}** *1 to 16* **TripLeg** *records.*
TicketNumber	*The ticket number.*
TravelAgencyCode	*The travel agency code.*
TravelAgencyName	*The travel agency name.*

Table 39: MarketTransportCap

Continued on next page

SaleDetail, continued

(continued)

Restrictions	*Enumerated code indicating restrictions on refunds or changes.*
TripLeg	**{DateOfTravel, CarrierCode, ServiceClass, StopOverCode, DestCityAirport, [FareBasisCode], [DepartureTax]}**
DateOfTravel	*The date of travel for this trip leg.*
CarrierCode	*The carrier code for this trip leg.*
ServiceClass	*The class of service for this trip leg.*
StopOverCode	*Enumerated code indicating whether stopovers are permitted for this trip leg.*
DestCityAirport	*The destination city for this trip leg.*
FareBasisCode	*The fare basis code for this trip leg.*
DepartureTax	*The departure tax for this trip leg.*

Table 39: MarketTransportCap, continued

Location data

Location	**{CountryCode, [City], [StateProvince], [PostalCode], [LocationID]}**
CountryCode	*The ISO 3166 country code for the location.*
City	*The city name of the location.*
StateProvince	*The name or abbreviation of the state or province.*
PostalCode	*The postal code of the location.*
LocationID	*An identifier that the Merchant uses to specify one of its locations*

RRTags

Purpose

RRTags carries message identification data; in particular, RRPID serves as the unique identifier for a message pair.

RRTags data

RRTags	**{RRPID, MerTermIDs, Date}**
RRPID	*Fresh request/response pair ID.*
MerTermIDs	**{MerchantID, [TerminalID], [AgentNum], [ChainNum], [StoreNum]}**
Date	*Current date for aging logs.*
MerchantID	*Cardholder inserts this data in* **PIHead**. *It is copied from* **MerID** *in the Merchant signature certificate.*
TerminalID	*Merchant inserts this data in* **AuthReq**.
AgentNum	*Merchant inserts this data in* **AuthReq**.
ChainNum	*Merchant inserts this data in* **AuthReq**.
StoreNum	*Merchant inserts this data in* **AuthReq**.

Table 40: RRTags

Generate RRTags

Step	Action
1	Generate a fresh RRPID. Save in transaction database.
2	Populate MerTermIDs from stored data at Merchant, describing the location of the sale.
3	Populate Date with current date.

BatchStatus

Purpose

To return the status of a batch from a Payment Gateway to the Merchant or to reconcile the value of a batch from the Merchant to the Payment Gateway.

BatchStatus data

BatchStatus	**{OpenDateTime, [ClosedWhen], BatchDetails, [BatchExtensions]}**
OpenDateTime	*The date and time the batch was opened.*
ClosedWhen	**{CloseStatus, CloseDateTime}**
BatchDetails	**{BatchTotals, [BrandBatchDetailsSeq]}**
BatchExtensions	*The data in an extension to the batch administration message must be financial and should be important for the processing of the batch administration request.*
CloseStatus	*Enumerated code indicating status of batch close.*
CloseDateTime	*The date and time the batch was closed.*
BatchTotals	**{TransactionCountCredit, TransactionTotalAmtCredit, TransactionCountDebit, TransactionTotalAmtDebit, [BatchTotalExtensions]}**
BrandBatchDetailsSeq	**{BrandBatchDetails +}**
TransactionCountCredit	*The number of transactions that resulted in a credit to the Merchant's account.*
TransactionTotalAmtCredit	*The total amount credited to the Merchant's account.*
TransactionCountDebit	*The number of transactions that resulted in a debit to the Merchant's account.*
TransactionTotalAmtDebit	*The total amount debited from the Merchant's account.*

Continued on next page

BatchStatus, continued

BatchStatus data (continued)

BatchTotalExtensions	*The data in an extension to the batch administration response message must be financial and should be important for the processing of the batch administration request.* *Note: Information regarding the processing of the request itself should appear in an extension to* **BatchAdminResData***; information regarding the status of a batch should appear in an extension to BatchStatus; information regarding detail for an item within the capture batch should appear in an extension to TransactionDetail.*
BrandBatchDetails	**{BrandID, BatchTotals}**
BrandID	*Payment card brand (without product type).*

Table 41: BatchStatus

TransactionDetail

Purpose

To provide details of a transaction in a batch from a Payment Gateway to the Merchant or to reconcile the value of a transaction from the Merchant to the Payment Gateway.

TransactionDetail data

TransactionDetail	**{TransIDs, AuthRRPID, BrandID, BatchSequenceNum, [ReimbursementID], TransactionAmt, TransactionAmtType, [TransactionStatus], [TransExtensions]}**
TransIDs	*The transaction identifiers from the authorization/capture processing of the item.*
AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *or* **AuthRevReq**.
BrandID	*Payment card brand (without product type).*
BatchSequenceNum	*The sequence number of this item within the batch.*
ReimbursementID	*Enumerated code indicating the type of reimbursement for the item.*
TransactionAmt	*The amount for the item of the type indicated by* **TransactionAmtType**. *The amount is always specified as a positive value.*
TransactionAmtType	*Enumerated code indicating the type of amount (credit or debit)*
TransactionStatus	*Enumerated code indicating the result of passing the transaction to the next upstream system.*
TransExtensions	*The data in an extension to the batch administration response message must be financial and should be important for the processing of the batch administration request.* *Note: Information regarding the processing of the request itself should appear in an extension to* **BatchAdminResData**; *information regarding the status of a batch should appear in an extension to BatchStatus; information regarding detail for an item within the capture batch should appear in an extension to TransactionDetail.*

Table 42: TransactionDetail

Amount Fields

Amount format

Amounts in the SET payment messages are expressed in terms of three fields: *currency*, *amount*, and *amtExp10*. These components are specified in the field values:

```
currency amount amtExp10
```

where *currency*, *amount*, and *amtExp10* are numeric ASCII strings described further below. The elements shall appear in the specified order *(currency, amount, amtExp10)*.

Field	Definition
currency	The value shall be a numeric ASCII string specifying the three-digit ISO 4217 currency code. For example, a payment denominated in U.S. currency will have a currency value of "840". The values shall be between 1 and 999 inclusive.
amount	The value shall be a numeric ASCII string representing the amount of the payment, specified in terms of the stated currency. The value shall be a non-negative integer.
amtExp10	The value shall be a numeric ASCII string representing an exponent base 10 such that `amount * (10 ** amtExp10)` shall be the value in the minor unit of the currency specified in ISO 4217. The value may be either a negative or positive integer.

Example

In order to represent US $2.50 in the PurchAmt field, the corresponding values for *currency*, *amount*, and *amtExp10* fields are 840, 250, and -2, respectively.

Date Fields

Date field format

Dates in SET are typically indicated in the form of a string representing the calendar date and UTC time, in the format:

YYYYMMDDHHMM[SS[.f[f[f]]]]Z

where Z is a literal upper-case letter Z.

That is, the string should consist of a four-digit representation of the year that includes the century, a two-digit representation of the month, a two-digit representation of the day in the month, a two-digit representation of the hour (on a 24-hour clock), a two-digit representation of the minutes after the hour, an optional representation of the seconds after the minute, and a literal upper-case letter Z.

If present, the seconds field shall be a two-digit representation of the seconds followed optionally by a representation of fractional seconds. Fractional seconds are indicated by a decimal point, followed by one to three digits.

No separators are used aside from the decimal point in the optional representations of fractional seconds.

Examples:

199602232106Z
19960223210630Z
19960223210630.123Z

Midnight shall be represented in the form: YYYYMMDD000000Z, where YYYYMMDD represents the day following the midnight in question.

Following are examples of invalid representations:

19920520240000Z (midnight represented incorrectly)
19920622123421.0Z (spurious trailing zeros)

Section 2
General Flow

Payment Flow

Collection of request /response messages

The main flow for SET payment processing involves collections of paired request/response messages between Cardholder and Merchant, and between Merchant and Payment Gateway. These pairs of messages each supports a step in the payment process. There is a basic set of required pairs, and additional sets of optional pairs.

Purchase

The PReq/PRes message provide the basic purchase process between the Cardholder and Merchant. The PRes message may be returned immediately as in the picture below, or anytime later in the protocol. The information returned will depend on the stage of the protocol at which the PRes is returned.

Authorization

Authorization is performed by the Merchant to the Payment Gateway using the AuthReq/AuthRes messages. Authorization provides the Merchant approval by the Issuer to continue processing.

Capture messages

Capture may be accomplished with the CapReq/CapRes message pair. This activity completes the purchase for the Payment Gateway, and results in the actual charge against the cardholder's account.

Continued on next page

Payment Flow, continued

Protocol summary

Figure 27 below shows a typical example of a purchase protocol flow. Optional messages are written in italics.

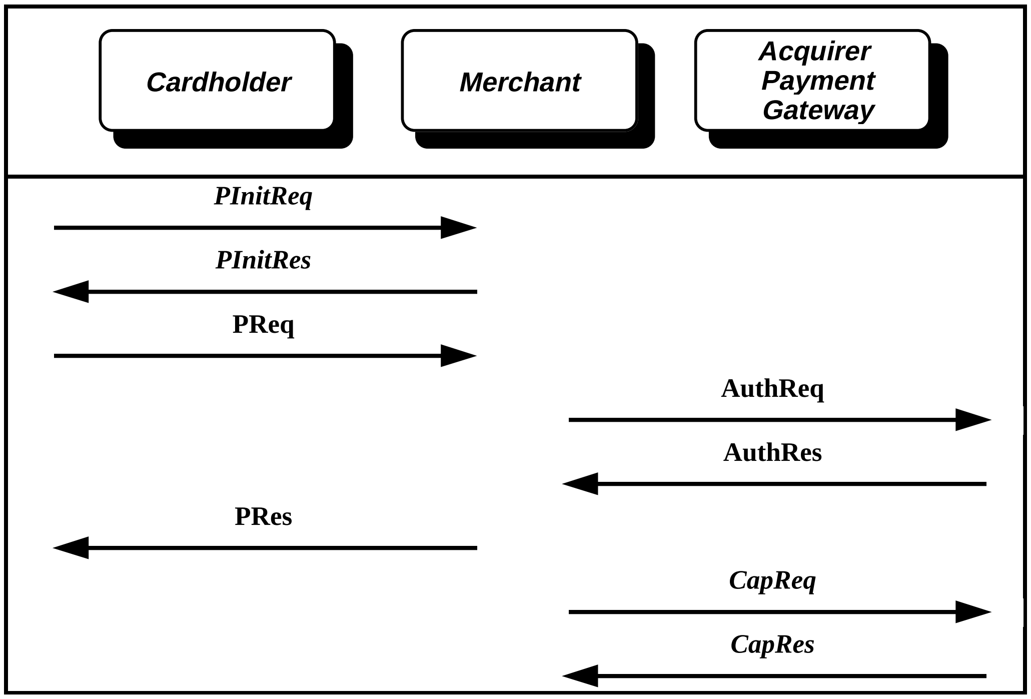


Figure 27: Purchase Protocol Flow

Basic purchase flow

Figure 1 is the basic purchase flow, with the Merchant choosing to request authorization and capture after purchase confirmation, and with initial messages to perform full initialization.

Continued on next page

Payment Flow, continued

Payment flow options

Figure 28 below (which continues on the next page) shows all the possible messages which may occur in the processing of a transaction (optional messages are in italics). All the messages shown here will be described in the following sections. Note that the ordering of the messages in this flow is permissible, but other orderings are also allowed.

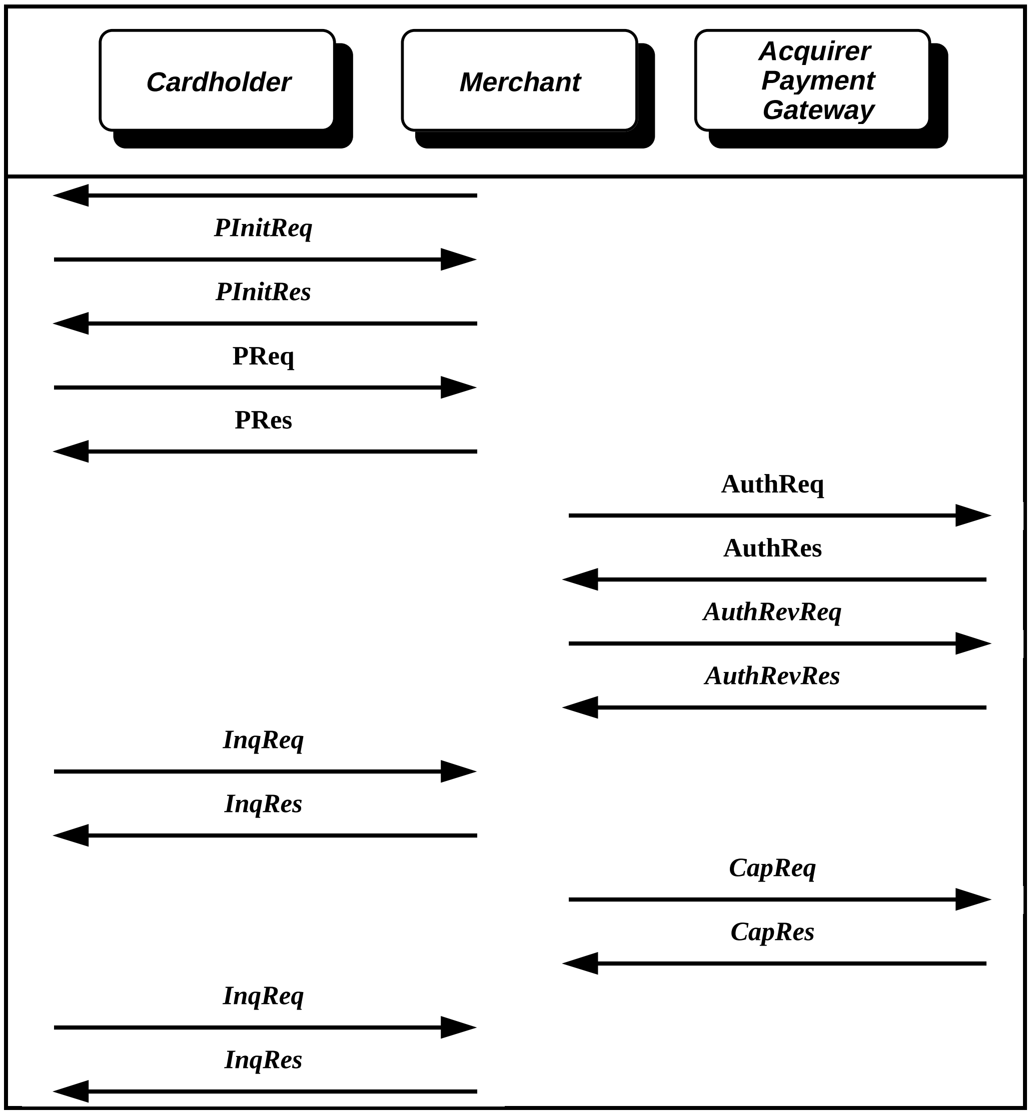


Figure 28: Payment Flow Options

Continued on next page

Payment Flow, continued

Payment flow options (continued)

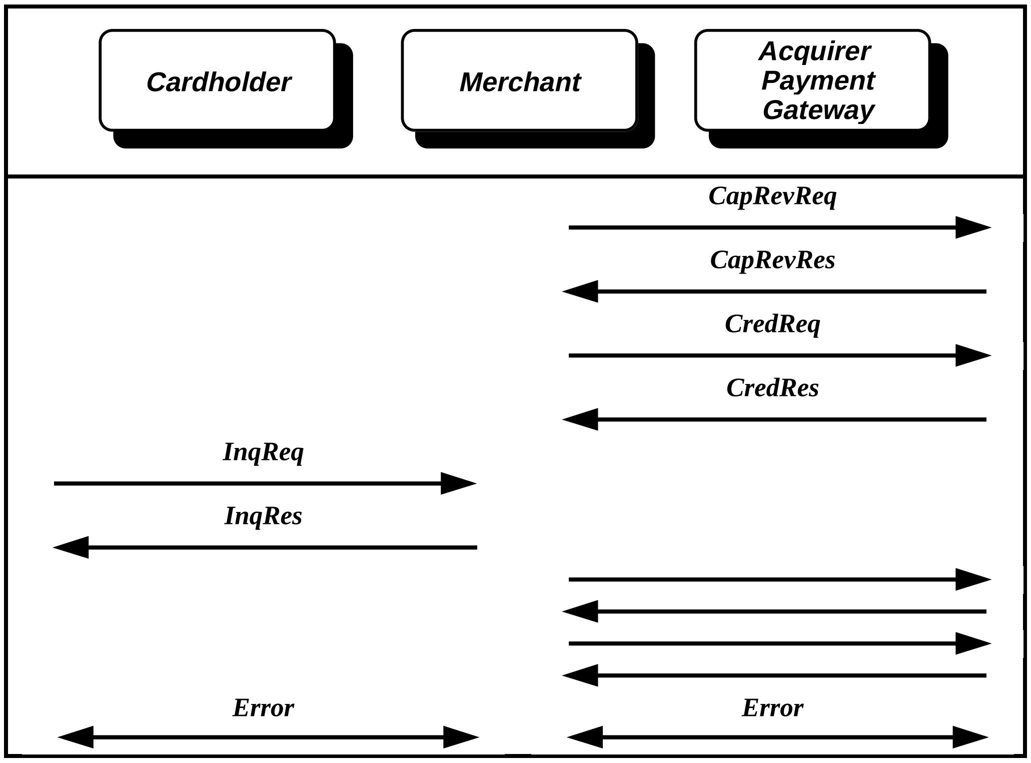


Figure 28: Payment Flow Options (continued)

Continued on next page

Payment Flow, continued

Inquiry messages

The InqReq/InqRes message pair enables the Cardholder to inquire of the status of the transaction. The InqReq can be sent anytime after the Cardholder has sent a PReq to the Merchant.

Payment messages

The PReq/PRes message pair encompass the actual purchase by the Cardholder from the Merchant.

Authorization reversal

AuthRevReq and AuthRevRes are used when authorizations need to be reversed.

Capture reversal

CapRevReq and CapRevRes comprise the capture reversal process, used to reverse a capture before settlement has occurred.

Credit

CredReq and CredRes comprise the credit process, similar to capture reversal, except that it is used after settlement.

Gateway Certificate

PCertReq/ PCertRes message pair provide a mechanism for the Merchant to obtain the Gateway Encryption Certificates which are needed before a message to the Payment Gateway can be encrypted.

Batch administration

BatchAdminReq and BatchAdminRes provide the means for the Merchant to open, close, and obtain status and transaction data on batches maintained by the Payment Gateway for managing the Merchant's transactions.

Error

A general error message is provided for reporting protocol and/or processing errors.

Chapter 2
Cardholder/Merchant Messages

Overview

Introduction

Chapter 2 describes messages exchanged between the Cardholder and Merchant.

Organization

Chapter 2 includes the following sections:

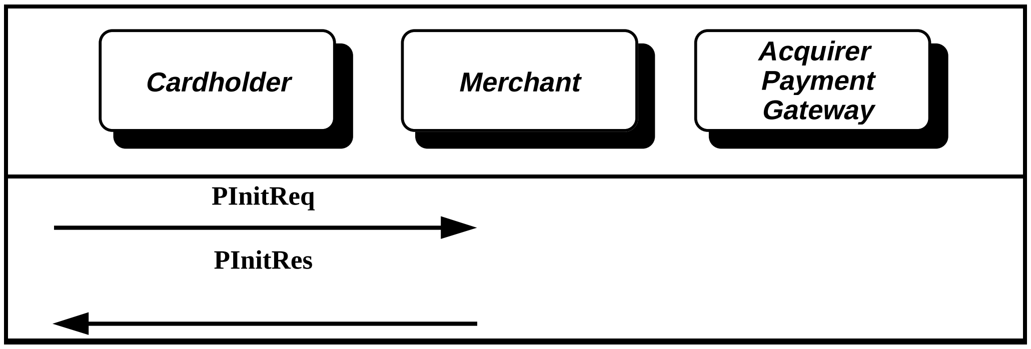
Section	Title	Contents	Page
1	Payment Initialization Request/Response	Presents the PInitReq and PInitRes messages, which support initialization of the protocol, including selection of the payment card and set up of cryptographic data.	307
2	Purchase Order Request/Response	Presents the PReq and PRes messages, which encompass the purchase transaction between the Cardholder and Merchant.	316
3	Inquiry Request/Response	Presents the InqReq and InqRes messages, enabling the Cardholder to query the Merchant as to the status of the transaction.	340

Section 1
Payment Initialization Request/Response Processing

Overview

Introduction

The initialization request processing consists of two messages, a request from a Cardholder to Merchant, and a response from the Merchant back to the Cardholder.



Purpose

The purpose of this message pair is to obtain certificates and CRLs for the Cardholder. In the absence of this message pair, this information must be obtained through some other means (such as a CD-ROM). These messages are usually preceded by a shopping phase and a SET Initiation Process.

The request message, PInitReq, identifies the Brand of the Card to be used, provides a local Cardholder defined identifier for the transaction, sends a challenge variable to ensure freshness of the response message, and sends a set of Thumbs, which identify relevant certificates and CRLs already held by the Cardholder, so that they need not be resent.

The response message, PInitRes, contains the requested data, including certificates and CRLs (included in the signature). These are provided within the signature of the response message. Also a Merchant date and an XID are established and the Merchant replies with the Cardholder's challenge, and adds a challenge of its own.

Variations

The protocol allows these messages to be omitted in non-interactive environments, with the data provided in these messages provided by off-line mechanisms (such as CD-ROM) and the challenges omitted, with less guarantee of freshness of messages.

Cardholder Generates PInitReq

Create PInitReq

The Cardholder shall perform the steps below to create a PInitReq message.

Step	Action
1	Generate RRPID for identifying message and matching response message to request.
2	Populate Language, the Cardholder's chosen language for the transaction.
3	Generate LID-C, to be identifier for message pair, since XID has not been assigned. This field may be assigned sequentially or randomly, but should not be repeated frequently.
4	If Merchant supplied a LID-M in the SET Initiation Process, copy it into the message.
5	Generate a fresh Chall-C.
6	Populate BrandID based on selected form of payment (input from shopping software or from the SET Initiation Process).
7	Populate the BIN (first 6 digits of the cardholder's account number).
8	Optional: Populate Thumbs for certificates, CRLs, and BrandCRLIdentifier held at the Cardholder. This should include the Root certificate. If these are included, fewer certificates, etc. will need to be sent in PInitRes.
9	Save RRPID, LID-C, LID-M (if available), Chall-C, and Thumbs (if available) in transaction database.
10	Optional: Populate any PInitReq extensions.
11	Invoke *Compose Message Wrapper* (see page 76) to send to Merchant.

Continued on next page

Cardholder Generates PInitReq, continued

PInitReq data

The PInitReq message provides the context for a transaction, by specifying the Cardholder's preferred language, IDs for the transaction, and specification of the Brand of the card to be used. In addition Thumbs are presented, where certificates are cached, and a cryptographic challenge is provided to ensure that the response message will be fresh.

PInitReq	**{RRPID, Language, LID-C, [LID-M], Chall-C, BrandID, BIN, [Thumbs], [PIRqExtensions]}**
RRPID	*Request/response pair ID.*
Language	*Cardholder's natural language.*
LID-C	*Local ID; convenience label generated by and for the Cardholder system.*
LID-M	*Copied from SET initiation messages (if present) described in the External Interface Guide.*
Chall-C	*Cardholder's challenge to Merchant's signature freshness.*
BrandID	*Cardholder's chosen payment card brand.*
BIN	*Bank Identification Number from the cardholder's account number (first six digits).*
Thumbs	*Lists of Certificate, CRL, and BrandCRLIdentifier thumbprints in Cardholder's cache.*
PIRqExtensions	*Note: The purchase initialization request is not encrypted, so this extension shall not contain confidential information.*

Table 43: PInitReq

Continued on next page

Cardholder Generates PInitReq, continued

Merchant processes PInitReq

When the Merchant receives a PInitReq, it shall process the message as follows.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	If LID-M is present, search for transaction record based on LID-M. If record not found: a) Return an Error message with ErrorCode set to unknownLID. b) Stop processing PInitReq.
3	If LID-M is <u>not</u> present, find the transaction record based on other criteria outside the scope of SET. Optional: If Merchant has not generated a LID-M for this transaction, generate a LID-M and store in transaction record.
4	Generate a fresh XID.
5	Save XID, RRPID, Language, LID-C, Chall-C, BrandID and BIN in transaction record.
6	If Thumbs present, save in transaction record.
7	If any PInitReq extensions exist, process them. If an extension is unknown and criticality flag is True, generate Error message, otherwise ignore extension. If extension is known, process extension.

Merchant Generates PInitRes

Create PInitRes

After the Merchant processes the PInitReq, it shall create the PInitRes as follows.

Step	Action
1	Construct PInitResData as follows: a) Construct TransIDs as follows: 1. Copy LID-C, XID, and Language from transaction record. 2. If transaction record has a LID-M, copy it. 3. Populate TransIDs.PReqDate with current date. b) Copy RRPID from transaction record. c) Copy Chall-C from transaction record. d) Generate fresh Chall-M. e) If Thumb for current BrandCRLIdentifier was not received or is not current, populate BrandCRLIdentifier with current one. f) Select the Payment Gateway based on the information from PInitReq (BrandID, BIN, and cardholder's certificate). Populate PEThumb with the thumbprint of the selected Payment Gateway's certificate (Cert-PE). g) Copy Thumbs from PInitReq if present. This enables the Cardholder to verify that the Merchant correctly received all Thumbs sent, and no others. h) Optional: add any PIRsExtensions.
2	Invoke *Compose SignedData* (see Part I, page 94). If Thumb for Cert-PE not received in PInitReq, include Cert-PE in the signature.
3	Invoke *Compose MessageWrapper* (see page 76) to send to Cardholder.

Continued on next page

Merchant Generates PInitRes, continued

PInitRes data

The PInitRes contains the following data:

PInitRes	**S(M, PInitResData)**
PInitResData	**{TransIDs, RRPID, Chall-C, Chall-M, [BrandCRLIdentifier], PEThumb, [Thumbs], [PIRsExtensions]}**
TransIDs	*See page 267.*
RRPID	*Request/response pair ID.*
Chall-C	*Copied from* **PInitReq**.
Chall-M	*Merchant's challenge to Cardholder's signature freshness.*
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PEThumb	*Thumbprint of Payment Gateway key-exchange certificate.*
Thumbs	*Copied from* **PInitReq**.
PIRsExtensions	*Note: The purchase initialization response is not encrypted, so this extension shall not contain confidential information.*

Table 44: PInitRes

Continued on next page

Merchant Generates PInitRes, continued

Cardholder processesPInit Res

When a Cardholder receives a PInitRes, it shall process the message as follows.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Invoke *Receive SignedData* (see Part I, page 77).
3	Verify TransIDs as follows: a) Search for transaction based on LID-C. If record not found: 1. Send an Error message with ErrorCode set to unknownLID. 2. Stop processing PInitRes. b) If a LID-M was sent in the SET Initiation Process, verify LID-M with LID-M in transaction record. If mismatch: 1. Send an Error message with ErrorCode set to unknownLID. 2. Stop processing PInitRes. c) If a LID-M was not sent, and there is a LID-M: 1. Send an Error message with ErrorCode set to unknownLID. 2. Stop processing PInitRes.
4	Verify RRPID against the RRPID in the transaction record. If RRPIDs different: a) Return an Error message with ErrorCode set to unknownRRPID. b) Stop processing PInitRes.
5	Verify Chall-C against the Chall-C in the transaction record. If Chall-Cs different: a) Return an Error message with ErrorCode set to challengeMismatch. b) Stop processing PInitRes.
6	As an optional controlled by the cardholder, extract the merchant's name from the merchant's certificate, and display the name to the user. Proceed if the cardholder approves of the merchant. Otherwise, stop processing PInitRes.
7	Save data, including TransIDs, and Chall-M in transaction record.
8	Process BrandCRLIdentifier if present (see page 249).
9	Use PEThumb to identify the encryption certificate (Cert-PE) to use in PReq to encrypt data for the Payment Gateway.

Continued on next page

Merchant Generates PInitRes, continued

Cardholder processesPInitRes (continued)

10	Verify that the brand of the merchant's certificate and the payment gateway's certificate (Cert-PE) matches the brand specified by the cardholder in the PInitReq. If mismatch, inform the cardholder and stop processing PInitRes.
11	If Thumbs present, verify with Thumbs sent in PInitReq. If Thumbs the same, go to Step 14, otherwise: A. Return Error message with ErrorCode set to thumbsMismatch. B. Stop processing PInitRes.
12	If Thumbs not present, verify that Thumbs was not sent in PInitReq. If Thumbs sent in PInitReq: A. Return Error message with ErrorCode set to thumbsMismatch. B. Stop processing PInitRes.
13	If PIRsExtensions exist, process them. If an extension is unknown and its criticality flag is True, generate Error message, otherwise ignore extension. If extension is known, process extension.
14	Check Cert-PE (identified by PEThumb) for unsigned transactions. If the indicator in Cert-PE does not allow unsigned transactions and the cardholder does not have a certificate, inform the cardholder that the transaction cannot proceed and stop processing.
15	Cardholder may now proceed with the purchase request.

Extension Guidelines

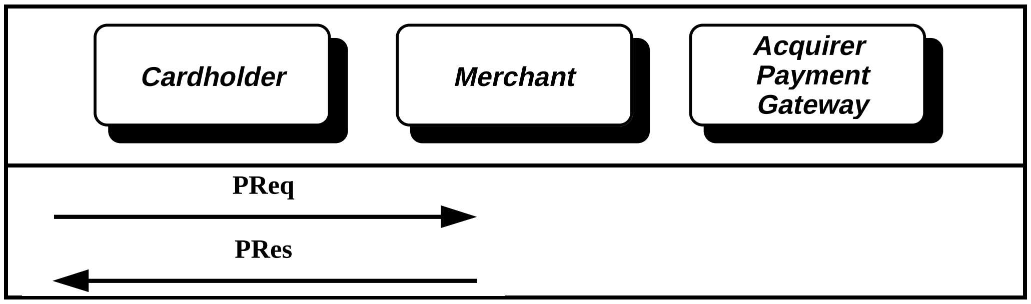
PInitReq	The purchase initialization request carries enough information about the cardholder's selection of a card for payment for the Merchant software to select an appropriate Payment Gateway certificate. Note: the purchase initialization request is not encrypted so this extension shall not contain confidential information.
PInitRes	The purchase initialization response carries copies of data from the purchase initialization request and the Merchant and Payment Gateway certificates. The information from the request is copied into the response because the response is signed by the Merchant; the signature allows to Cardholder to ensure that the request was received by the Merchant intact. Note: the purchase initialization response is not encrypted so this extension shall not contain confidential information.

Section 2
Purchase Order Request/Response Processing

Overview

Introduction

The PReq/PRes messages encompass the actual purchase by the Cardholder from the Merchant.



Purpose

The purchase order request/response messages are at the heart of the payment protocol. This is the Cardholder/Merchant payment pair that embodies the payment from the Cardholder's point of view.

Continued on next page

Overview, continued

Variations

The PReq message may or may not be preceded by a PInitReq/PInitRes message pair. PRes may be returned before authorization and capture. The Merchant-Payment Gateway processing performed affects the contents of the messages.

If a Cardholder certificate is available, a dual signature is used to provide integrity and authentication for the two parts of this message.

Introduction

The Purchase request/response messages encompass the actual payment between the Cardholder and the Merchant. PReq is the most complex message in the protocol. It consists of two parts: an Order Instruction (OI), for the Merchant, and a Payment Instruction (PI), tunneled through the Merchant to the Payment Gateway. These two items are, conceptually, separately signed. The separate signatures are combined in a provably secure optimization: a dual signature.

The Merchant is assumed to get the Order Description (OD) and PurchAmt out of band.

The salted hash of OD and PurchAmt, that is, HODInput, shall be included in the PI. The Payment Gateway verifies that the hash tunneled to him through the Merchant by the Cardholder equals the hash provided by the Merchant in AuthReq.

Some cardholders will not have certificates. Messages created by such cardholders are not signed; instead the PIHead is linked to OIData. Integrity of such messages is guaranteed by:

- OAEP used with the PI;
- H(PIHead) in OAEP block (along with PANData);
- H(OIData) with PIHead;
- Comparison by Payment Gateway of H(OIData) as supplied by the Merchant with H(OIData) with PIHead.

Cardholder Generates PReq

Create PReq

The Cardholder shall create a PReq message as follows (these steps apply to both PReqUnsigned and PReqDualSigned).

Step	Action		
1	Retrieve PurchAmt and OD from the results of the shopping phase.		
2	Construct OIData as follows:		
	a)	If PInitReq/PInitRes messages were exchanged:	Copy TransIDs from PInitRes.
		Otherwise:	To produce TransIDs, Cardholder shall generate PReqDate (current date/time), LID-C, and XID.
	b)	Generate a fresh RRPID; store to verify reply from Merchant.	
		If PInitReq/PInitRes messages were exchanged:	Copy Chall-C from PInitRes.
		Otherwise:	Generate fresh Chall-C.
	c)	Generate fresh ODSalt.	
	d)	Construct HODInput as follows:	
		1. Copy OD from the SET Initiation Process.	
		2. Populate PurchAmt with the purchase amount approved by the cardholder during the SET Initiation Process.	
		3. Copy ODSalt from Step 2.C.	
		4. If the cardholder is to make installment or recurring payments, populate InstallRecurData (see page 274).	
		5. Optional: add any ODExtenstions.	
	e)	Generate HOD with HODInput from Step 2.D.	

Continued on next page

Cardholder Generates PReq, continued

Create PReq (continued)

Step	Action
2	f) If the PInitReq/PInitRes messages were exchanged: Copy Chall-M from the PInitRes. Do not populate BrandID. Otherwise: Do not populate Chall-M. Populate BrandID to reflect desired card to be used. g) Populate BIN (with the first six numbers of the cardholder's PAN). h) If HODInput from Step 2.D has any ODExtensions, populate ODExtOIDs with all the OIDs in ODExtensions. The ODExtOIDs shall be listed in the same order as the ODExtensions so that the merchant can recompute the same hash. i) Optional: add any OIExtenstions.
3	Construct PIHead as follows: a) Copy the TransIDs computed Step 2.A.. b) Construct Inputs as follows: 1. Copy HOD from Step 2.e. 2. Copy PurchAmt from Step 2.d.2). c) Copy MerchantID from the Merchant certificate in PInitRes, or other means such as a CD-ROM catalog. d) Copy InstallRecurData from Step 2.d.4) if available. e) Generate TransStain as an HMAC using XID as the data and CardSecret as the key. If the cardholder does not have a certificate, use a CardSecret filled with all zeros. f) Populate SWIdent, which is derived from local data. This value shall match the value in MessageWrapper.

Continued on next page

Cardholder Generates PReq, continued

Create PReq (continued)

Step	Action
3	a) If the tunneling extension of Cert_PE is present , construct AcqBackInfo as follows (see Passed Keys below):: 1. Search the tunneling extension for an encryption algorithm acceptable to the Cardholder. If found, populate AcqBackAlg; otherwise, do not construct AcqBackInfo and go to Step F. 2. Generate a fresh AcqBackKey (appropriate to AcqBackAlg). b) Optional: add any PIExtentions.
4	Construct PANData as described on *page 283*.
5	Construct PI-OILink with the L operator using PIHead from Step 3 (parameter t1) and OIData from Step 2 (parameter t2).
6	Using the results of Steps 2, 3 and 4, construct a PReqDualSigned if the cardholder has a certificate or a PReqUnsigned if the cardholder does not have a certificate. Note: if the Payment Gateway certificate indicates that signed messages are required, and the cardholder does not have a certificate, the Cardholder system shall inform the cardholder that the transaction may not be accepted before composing the PReq.

Passed keys

When a Payment Gateway is prepared to encrypt data back to an end entity, its certificate lists one or more symmetric encryption algorithms that it supports in order of preference. The end entity that wants to have data encrypted should select the first algorithm for the list that it supports and generate a symmetric key. This key is passed to the Payment Gateway in a PReq.

Continued on next page

Cardholder Generates PReq, continued

Completing PReqDualSigned

Step	Action
1	Construct PISignature as follows: A. Construct PI-TBS as follows: a) Construct HPIData as digested data of PIData. b) Construct HOIData as digested data of OIData. B. Complete PISignature with the SO operator using the cardholder certificate (parameter s), and PI-TBS (parameter t).
2	Apply the EX operator using the Payment Gateway's public key (parameter r), PI-OILink from Cardholder Composes PReq Step 5 (parameter t) and PANData from Cardholder Composes PReq Step 4 (parameter p).
3	Form PIDualSigned as the concatenation of PISignature computed in Step 1 and the encrypted data computed in Step 2.
4	Construct PIData as the concatenation of PIHead from Cardholder Composes PReq Step 3 and PANData from Cardholder Composes PReq Step 4.
5	Construct OIDualSigned with the L operator using OIData from Cardholder Composes PReq Step 2 (parameter t1) and PIData from Step 4 (parameter t2).
6	Construct PReqDualSigned as the concatenation of PIDualSigned from Step 3 and OIDualSigned from Step 5.

Continued on next page

Cardholder Generates PReq, continued

Completing PReqUnsigned

Step	Action
1	Construct PIUnsigned with the EXH operator using the Payment Gateway's public key (parameter r), PI-OILink from Cardholder Composes PReq Step 5 (parameter t), and PANToken from Cardholder Composes PReq Step 4.
2	Construct PIDataUnsigned as the sequence of PIHead and PANToken.
3	Construct OIUnsigned with the L operator using OIData from Cardholder Composes PReq Step 2 (parameter t1) and PIDataUnsigned from Step 2 (parameter t2).
4	Construct PReqUnsigned as the sequence of PIUnsigned Step 1 and OIUnsigned Step 3.

Continued on next page

Cardholder Generates PReq, continued

Overall PReq data

The purchase request message supports cardholders with or without certificates. The PReq data consists of:

- Order Instruction (OI) for the Merchant, and
- Payment Instruction (PI) which is tunneled encrypted through the Merchant to the Payment Gateway.

Authentication and integrity are achieved using a dual signature if the cardholder has a certificate (PReqDualSigned). Integrity is achieved by using hashes protected in the OAEP envelope if the cardholder is operating without a signature certificate (PReqUnsigned).

PReq	**< PReqDualSigned, PReqUnsigned >**
PReqDualSigned	*See page 323.*
PReqUnsigned	*See page 324.*

Table 45: PReq

PReqDual Signeddata

The PReqDualSigned is created by cardholders with certificates.

PReqDualSigned	**{PIDualSigned, OIDualSigned}**
PIDualSigned	*See "PI (Payment Instruction)" on page 271.*
OIDualSigned	**L(OIData, PIData)**
OIData	*See page 325.*
PIData	**{PIHead, PANData}** *See page 273 for* **PIHead**. *See page 283 for* **PANData**.

Table 46: PReqDualSigned

Continued on next page

Cardholder Generates PReq, continued

PReqUnsigned data

The PReqUnsigned is created by cardholders without certificates.

PReqUnsigned	**{PIUnsigned, OIUnsigned}**
PIUnsigned	*See "PI (Payment Instruction)" on page 271.*
OIUnsigned	**L(OIData, PIDataUnsigned)**
OIData	*See page 325.*
PIDataUnsigned	**{PIHead, PANToken}** *See page 273 for* **PIHead**. *See page 284 for* **PANToken**.

Table 47: PReqUnsigned

Continued on next page

Cardholder Generates PReq, continued

Common PReq data

The following data is common to both PReqDualsigned and PReqUnsigned.

OIData	**{TransIDs, RRPID, Chall-C, HOD, ODSalt, [Chall-M], BrandID, BIN, [ODExtOIDs], [OIExtensions]}**
TransIDs	*Copied from* **PInitRes***, if present; see page 267*
RRPID	*Request/response pair ID.*
Chall-C	*Copied from corresponding* **PInitReq***; see page 309.*
HOD	**DD(HODInput)** *Links* **OIData** *to* **PurchAmt** *without copying* **PurchAmt** *into* **OIData***, which would create confidentiality problems.*
ODSalt	*Copied from* **HODInput**.
Chall-M	*Merchant's challenge to Cardholder's signature freshness.*
BrandID	*Cardholder's chosen payment card brand.*
BIN	*Bank Identification Number from the cardholder's account number (first six digits).*
ODExtOIDs	*List of object identifiers from* **ODExtensions** *in the same order as the extensions appeared in* **ODExtensions**.
OIExtensions	*The data in an extension to the* **OI** *should relate to the Merchant's processing of the order.* *Note: The order information is not encrypted so this extension shall not contain confidential information.*

Table 48: OIData

Continued on next page

Cardholder Generates PReq, continued

Common PReq data (continued)

HODInput	**{OD, PurchAmt, ODSalt, [InstallRecurData], [ODExtensions]}**
OD	*The Order Description. This information is exchanged between the Cardholder and the Merchant out-of-band to SET. The contents, which are determined by the Merchant's processing requirements, will include information such as the description of the items ordered (including quantity, size, price, etc.), the shipping address, and the Cardholder's billing address (if required).*
PurchAmt	*The amount of the transaction as specified by the Cardholder; this must match the value in* **PIHead** *on page 273.*
ODSalt	*Fresh Nonce generated by Cardholder to prevent dictionary attacks on* **HOD**.
InstallRecurData	*See page 274.*
ODExtensions	*The data in an extension to the* **OD** *should relate to the Merchant's processing of the order.* *The information in these extensions must be independently known to both the Cardholder and Merchant.*

Table 48: OIData, continued

Continued on next page

Cardholder Generates PReq, continued

Merchant processes PReq

When the Merchant receives a PReq, it shall process the message as follows.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	If a PReqDualSigned was received, verify PISignature as follows: A. Extract OIData and HPIData from OIDualsigned. B. Construct HOIData as digested data of OIData. C. Construct PI-TBS as the concatenation of HPIData from Step A and HOIData from Step B. D. Create a signature with the SO operator using the cardholder certificate (parameter s), and PI-TBS from Step C (parameter t).. E. Compare the signature from Step D with PISignature. If they are not equal, return an Error message with ErrorCode set to signatureFailure. Stop processing PReq. F. Go to Step 4.
3	If a PReqUnsigned is received, verify that Brand Certificate (Cert-PE) allows a PReqUnsigned. If not: A. Return a PRes message with CompletionCode set to signatureRequired; B. Stop processing PReq.

Continued on next page

Cardholder Generates PReq, continued

Merchant processes PReq (continued)

Step	Action
4	Process the TransIDs as follows: Search for transaction based on XID. **If record found:** Verify LID-C and LID-M with record. If mismatch: a) Return an Error message with ErrorCode set to unknownLID. b) Stop processing PReq. Otherwise, verify Chall-M with record. If mismatch: a) Return an Error message with ErrorCode set to challengeMismatch. b) Stop processing PReq. **Otherwise:** a) Create new transaction record. b) Save LID-C, PReqDate and Language. c) Optional: Generate a LID-M.
5	Verify that the BrandID in the cardholder's certificate matches the BrandID in the PInitReq and/the OIData. If failure: a) Return a PRes message with completionCode set to orderRejected. b) Stop processing PReq.
6	Store Chall-C to return later in PRes.
7	Store remaining variables from message in database.
8	Verify HOD against generated hash of OD, PurchAmt, ODSalt, InstallRecurData (if present) and ODExtensions (if present).
9	At this time, the Merchant may send the PRes to the Cardholder if desired, or wait until after the AuthRes is received from the Payment Gateway.

Merchant Generates PRes

Create PRes

After processing a PReq, the Merchant shall create a PRes as follows: (Note: an InqRes message is identical to a PRes and therefore generated with this same procedure.)

Step	Action
1	a) Construct PResData as follows: b) Populate TransIDs; include all fields of TransIDs received from the Cardholder or the Payment Gateway. c) Copy RRPID from PReq (or InqReq). d) Copy Chall-C from PReq (or InqReq). e) If Thumb for current BrandCRLIdentifier is not received or is not current, populate BrandCRLIdentifier with current one. f) Construct PResPayloadSeq as follows: 1. If Purchase Request includes a PurchAmt of 0, construct a single PResPayload with CompletionCode set to meaninglessRatio and all other fields omitted. Go to Step 2. 2. If the Payment Gateway has rejected the order, construct a single PResPayload as follows: • Set CompletionCode to orderRejected. • Copy the AcqCardMsg from the AuthRes if present. • Go to Step 2. 3. If the Payment Gateway has not yet responded to the Merchant's authorization request, generate a single PResPayload with a CompletionCode of orderReceived and all other fields omitted. Go to Step 2. 4. If this is a response to an InqReq where the transaction was not found, generate a single PResPayload with CompletionCode set to orderNotReceived and all other fields omitted. Go to Step 2. 5. If the Payment Gateway has responded to the Merchant's authorization request, construct a PResPayloadSeq as described below.
2	Invoke *Compose SignedData*
3	Invoke *Compose MessageWrapper* 77to send to Cardholder.

Continued on next page

Merchant Generates PRes, continued

Constructing PResPayload-Seq

For every authorization the Merchant has performed, and has not fully reversed, construct a PResPayload as follows:

Step	Action
1	If only an authorization has been performed: a) Set CompletionCode to authorizationPerformed. b) Construct Results as described below, omitting CapStatus and CredStatusSeq.
2	If a capture has been performed (implies an authorization): a) Set CompletionCode to capturePerformed. b) Construct Results as described below, omitting CredStatusSeq.
3	If a credit has been performed (implies a capture): a) Set CompletionCode to creditPerformed. b) Construct Results as described below.
4	Optional: add any PRsExtensions.

Continued on next page

Merchant Generates PRes, continued

Constructing Results

Construct a Results as follows:

Step	Action
1	Copy AcqCardMsg from AuthRes if present.
2	If the item was authorized, construct AuthStatus as follows: a) Copy AuthDate from transaction record. b) Copy AuthCode from transaction record. c) Compute AuthRatio as AuthReqAmt ÷ PurchAmt. d) If currency conversion data is present in AuthRes, copy it.
3	If the item was captured, construct CapStatus as follows: a) Copy CapDate from transaction record. b) Copy CapCode from transaction record. c) Compute CapRatio as CapReqAmt ÷ PurchAmt.
4	Construct CredStatusSeq as a sequence of CredStatus for each credit performed and not reversed. Construct a CredStatus as follows: a) Copy CreditDate from transaction record. b) Copy CreditCode from transaction record. c) Compute CreditRatio as CapRevOrCredReqAmt ÷ PurchAmt.

Continued on next page

Merchant Generates PRes, continued

PRes data

PRes	**S(M, PResData)**
PResData	**{TransIDs, RRPID, Chall-C, [BrandCRLIdentifier], PResPayloadSeq}**
TransIDs	*Copied from* **PReq***; see page 267.*
RRPID	*Request/response pair ID.*
Chall-C	*Copied from corresponding* **PInitReq***; see page 309.*
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PResPayloadSeq	**{PResPayload +}** *One entry per Authorization performed. Note: a reversal removes the data from* **PResPayload**. *If no authorizations have been performed, a single entry with the appropriate status appears.*
PResPayload	*See page 333.*

Table 49: PRes

Continued on next page

Merchant Generates PRes, continued

PResPayload data

PResPayload	**{CompletionCode, [Results], [PRsExtensions]}**
CompletionCode	*Enumerated code indicating completion status of transaction.*
Results	**{[AcqCardMsg], [AuthStatus], [CapStatus], [CredStatusSeq]}**
PRsExtensions	*Note: The purchase response is not encrypted so this extension shall not contain confidential information.*
AcqCardMsg	*Copied from* **AuthRes**. *See page 278.*
AuthStatus	**{AuthDate, AuthCode, AuthRatio, [CurrConv]}**
CapStatus	**{CapDate, CapCode, CapRatio}** *Data only appears if* **CapReq** *corresponding to the Authorization has been performed. Note: a* **CapRevReq** *removes the data.*
CredStatusSeq	**{CreditStatus +}** *Data only appears if* **CredReq** *corresponding to the Authorization has been performed. Note: a* **CredRevReq** *removes the data.*
AuthDate	*Date of authorization; copied from* **AuthRRTags.Date** *(see page 350).*
AuthCode	*Enumerated code indicating outcome of payment authorization processing; copied from* **AuthResPayload** *(see page 359).*
AuthRatio	**AuthReqAmt ÷ PurchAmt** *For* **AuthReqAmt**, *see "**AuthReqPayload**" on page 352 or* **AuthNewAmt**, *see "AuthRevReq" on page 368.* *For* **PurchAmt**, *see "OIData" on page 325. After a partial reversal, the new amount replaces the original amount.*
CurrConv	**{CurrConvRate, CardCurr}** *Currency conversion information; copied from* **AuthResPayload** *(see page 359).*

Table 50: PResPayload

Continued on next page

Merchant Generates PRes, continued

PResPayload data (continued)

CapDate	*Date of capture; copied from* **CapPayload** *(see page 378).*
CapCode	*Enumerated code indicating status of capture; copied from* **CapResPayload** *(see page 383).*
CapRatio	**CapReqAmt ÷ PurchAmt** *For* **CapReqAmt***, see "CapPayload" on page 378. For* **PurchAmt***, see "OIData" on page 325.*
CreditStatus	**{CreditDate, CreditCode, CreditRatio}** *Data only appears if corresponding* **CreditReq** *has been performed. Note: A* **CredRevReq** *removes the data.*
CreditDate	*Date of credit; copied from* **CapRevOrCredReqData. CapRevOrCredReqDate** *(see page 404).*
CreditCode	*Enumerated code indicating status of credit; copied from* **CapRevOrCredResPayload.CapRevOrCredCode** *(see page 394).*
CreditRatio	**CapRevOrCredReqAmt ÷ PurchAmt** *For* **CapRevOrCredReqAmt***, see "CapRevOrCredReqData" on page 389.* *For* **PurchAmt***, see "OIData" on page 325.*

Table 50: PResPayload, continued

Continued on next page

Merchant Generates PRes, continued

Completion Code

The following values are defined for CompletionCode.

meaninglessRatio	PurchAmt = 0; ratio cannot be computed
orderRejected	Merchant cannot process order
orderReceived	No authorization processing to report
orderNotReceived	Inquiry received before order
authorizationPerformed	See AuthStatus for details
capturePerformed	See CapStatus for details
creditPerformed	See CreditStatus for details

Figure 29: Enumerated Values for CompletionCode

Continued on next page

Merchant Generates PRes, continued

Cardholder processes PRes

When the Cardholder receives a PRes, it shall process the message as follows.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Invoke *Receive Signed Data* to verify the merchant's signature.
3	Search for transaction record based on Trans.LID-C. If record not found: a) Send an Error message with ErrorCode set to unknownLID. b) Stop processing PRes.
4	Verify TransIDs.XID against XID in transaction record. If mismatch: a) Send an Error message with ErrorCode set to unknownXID. b) Stop processing PRes.
5	Verify RRPID against RRPID in transaction record. If mismatch: a) Send an Error message with ErrorCode set to unknownRRPID. b) Stop processing PRes.
6	Verify Chall-C against Chall-C in transaction record. If mismatch: a) Send an Error message with ErrorCode set to challengeMismatch. b) Stop processing PRes.
7	Store BrandCRLIdentifier and verify that CRLs listed are kept in cache. If not, if CRLs listed are for elements whose certificates were used in verifying signature, back out of message; signature may not be valid.

Continued on next page

Merchant Generates PRes, continued

Cardholder processes PRes (continued)

Step	Action
8	For each PResPayload in PResPayloadSeq do the following: a) If CompletionCode indicates Credit has been completed, for each data structure in the CreditSeq, report the derived Credit Amount (Purchase Amount * CredRatio) and Credit Date to the user. Report CapCode to user, together with derived Capture Amount (Purchase Amount * CapRatio). b) Otherwise if CompletionCode indicates Capture is complete, report CapCode to user, together with derived Capture Amount (Purchase Amount * CapRatio). c) Otherwise if CompletionCode indicates Authorization complete, report AuthCode to user together with derived Authorization Amount (Purchase Amount * AuthRatio). d) Otherwise report result of transaction based on CompletionCode. e) If AcqCardMsg present, decrypt and present to the cardholder. If this contains a URL, software may invoke the appropriate Web page. There may be additional brand-dependent processing.

Extension Guidelines

PIHead

The payment instructions carry information from the Cardholder to the Payment Gateway. The data in an extension to the payment instructions shall be financial and should be important for the processing of an authorization by the Payment Gateway, the financial network or the Issuer.

OIData

The order information carries information to link the purchase request to the prior shopping and ordering dialogue between the Cardholder and the Merchant. The data in an extension should relate to the Merchant's processing of the order.

Note: The order information is not encrypted so this extension shall not contain confidential information.

HODInput

The hash of the order description provides a secure linkage of the shopping/ordering dialogue and the purchase request. All information in the hash must be exchanged between the Cardholder and the Merchant out of band to SET before the purchase request is sent.

Extensions can be included in this linkage via ODExtensions. The cardholder shall indicate in ODExtOIDS which extensions are included in HODInput, and the order that they are specified in HODInput, so that the merchant can know how to compute HOD2.

PResPayload

The purchase response carries information about the processing of the purchase request by the Merchant.

Note: The purchase response is not encrypted so this extension shall not contain confidential information.

Order description

A significant amount of information regarding an order is exchanged between the cardholder and merchant out of band to the SET protocol. This information is collectively referred to as the order description. The digital signature of the cardholder on the purchase request is generated over the hash of the order description.

To reference additional information within the protocol, extensions can be added to the HODInput field. Since this field is not transmitted within the SET message, the cardholder software places the list of extension object identifiers that were processed as part of the hash in the ODExtOIDs field of the order information. The object identifiers must be listed in the same order within the order information field as they appeared in the computation of the hash.

To verify the hash, the merchant must independently build the data of the extensions that are declared by the cardholder in the order information and put the extensions into its copy of HODInput. In order for the hash to verify, the extensions must appear in the same order that the cardholder software arranged them.

Continued on next page

Extension Guidelines, continued

Payment instruction extensions

An additional private certificate extension has been added for SET payment gateway certificates. This certificate extension will list the object identifiers of the message extensions that the gateway can process in payment instructions. Cardholder software can shall use this information to ensure that no unrecognized critical extensions are put into the payment instructions (in **PIExtensions** or **SRExtensions**).

The payment gateway certificate extension is defined as follows:

```
setExtensions EXTENSION ::= {
   SYNTAX         SETExtensionsSyntax
   IDENTIFIED BY  { id-set-setExtensions }
}

SETExtensionsSyntax ::= SEQUENCE OF OBJECT IDENTIFIER
```

Section 3
Inquiry Request/Response Processing

Overview

Introduction

The InqReq/InqRes message pair enables the Cardholder to inquire as to the status of a transaction.

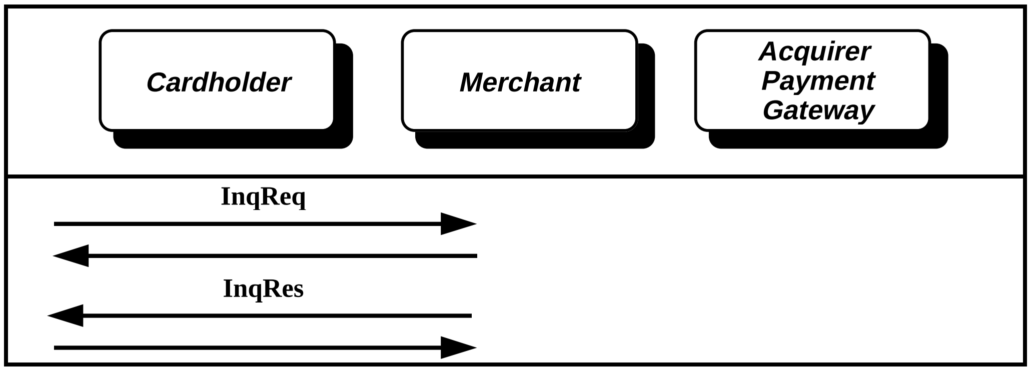


Figure 30: InqReq/InqRes Message Pair

Purpose

This sequence of messages is optional. The Inquiry Request message may be sent at any time after PReq by the Cardholder to the Merchant to inquire as to the status of a transaction. Since it may be sent repeatedly, it includes its own challenge, unique to each invocation, and TransIDs to identify the intended transaction.

The response message is of the same format as PRes, but is a distinct message, since otherwise it would signal the Merchants final report on the transaction.

The Merchant is required to verify that the certificate accompanying InqRes matches the certificate originally used with PRes. This prevents one cardholder from inquiring about another's purchases.

Cardholders without certificates do not sign inquiries, which means that the integrity of inquiry messages is not guaranteed.

Variations

Inquiry requests can be sent at any time. Multiple inquiry messages can be sent regarding the same transaction.

Cardholder Generates InqReq

Create InqReq

The Cardholder shall create an InqReq as follows.

Step	Action
1	Construct InqReqData as follows: A. Copy TransIDs from previous PReq. B. Generate fresh RRPID. C. Generate fresh Chall-C. D. Optional: add any InqRqExtentions.
2	If the Cardholder sent a signed PReq, invoke *Compose SignedData* with InqReqData.
3	Invoke *Compose MessageWrapper* (see page 76) to send to Merchant.

InqReq data

InqReq	**< InqReqSigned, InqReqData >**
InqReqSigned	**S(C, InqReqData)**
InqReqData	**{TransIDs, RRPID, Chall-C2, [InqRqExtensions]}**
TransIDs	*Copied from the most recent of the following:* **PReq**, **PRes**, **InqRes**.
RRPID	*Request/response pair ID.*
Chall-C2	*Fresh Cardholder challenge to Merchant's signature.*
InqRqExtensions	*Note: The inquiry request is not encrypted so this extension shall not contain confidential information.*

Table 51: InqReq

Continued on next page

Cardholder Generates InqReq, continued

Merchant processes InqReq

When the Merchant receives an InqReq, it process the message as follows.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	If an InqReqData is received (as opposed to a InqReqSigned), check if Payment Gateway certificate allows unsigned transactions. If it does not, then: a) Return an InqRes message with CompletionCode set to signatureRequired. b) Stop processing InqReq. Otherwise, go to Step 4.
3	If an InqReqSigned received, verify the signature. If signature fails: a) Return an Error message with ErrorCode set to signatureFailure. b) Stop processing InqReq.
4	Verify TransIDs with MessageWrapper. If mismatch: a) Return an Error message with ErrorCode set to wrapperMsgMismatch. b) Stop processing InqReq.
5	Search for transaction in database based on TransIDs.XID. If record not found: a) Return an InqRes with CompletionCode set to orderNotReceived. b) Stop processing InqReq.
6	If PReq was signed, verify that the same cardholder signed both the PReq and InqReq. If mismatch: a) Return an Error message with ErrorCode set to unknownXID. b) Stop processing InqReq.
7	Construct PResPayloadSeq

Merchant Generates InqRes

Create InqRes

The creation of an **InqRes** is the identical to the creation of a **PRes**. See page 329.

InqRes data

InqRes	*This is identical to a* **PRes***; see page 332.*

Cardholder processes InqRes

When a Cardholder receives an InqRes, it shall process the message just like a PRes. See page 336.

Extension Guidelines

InqReqData	The inquiry request carries enough information about the purchase request for the Merchant to locate the transaction and return the current transaction status. Note: the inquiry request is not encrypted so this extension shall not contain confidential information.

Chapter 3
Merchant/Payment Gateway Messages

Overview

Introduction

Chapter 3 describes messages exchanged between the Merchant and the Payment Gateway and includes the following sections:

Section	Title	Contents	Page
1	Authorization Request/Response	Presents the AuthReq and AuthRes messages, which support the authorization stage of the payment transaction.	346
2	Capture Request/Response	Presents the CapReq and CapRes messages, which support the capture stage of the payment transaction.	374
3	Authorization Reversal	Presents the AuthRevReq and AuthRevRes messages, providing for the reversal or cancellation of a previous authorization.	366
4	Capture Reversal or Credit Data	Presents the data structures used by both the Capture Reversal and Credit pairs of messages. These two message pairs have identical data structures; the processing of these messages is discussed in the next two sections.	387
5	Capture Reversal	Presents the CapRev and CapRes messages, which support the reversal of previously captured transactions.	398
6	Credit Request/Response	Presents the CredReq and CredRes messages, which support credits against transactions which have been captured and cleared.	402
7	Credit Reversal Request/Response	Presents the CredRevReq and CredRevRes messages, which support reversal of previously granted credits.	406
8	Gateway Certificate Request/Response	Presents the PCertReq and PCertRes messages, which enable a Merchant to request and receive Payment Gateway encryption certificates, which are required before the Merchant can send encrypted messages to the Payment Gateway.	410
9	Batch Administration	Presents the BatchAdminReq and BatchAdminRes messages, which enable the Merchant to request the payment to open and close capture batches, and to query their status and contents.	417

Section 1
Authorization Request/Response

Overview

Introduction

The Authorization Request/Response processing consists of two messages, a request from a Merchant to a Payment Gateway and a response from the Payment Gateway back to the Merchant. These messages are used for both authorization-only transactions, and also for authorization with capture (sale) transactions.

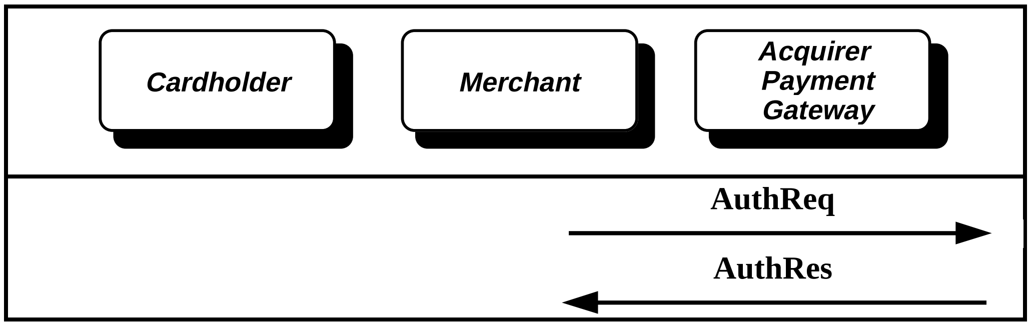


Figure 31: AuthReq/AuthRes Message Pair

Purpose

The Authorization Request and Response message pair provide the mechanism for the Merchant to obtain authorization for a purchase.

In the authorization request, the Merchant sends the data about the purchase, signed and encrypted, plus the PI received from the Cardholder. Since each contains the hash of the OD and the amount, the Payment Gateway can verify that the Merchant and Cardholder agree upon the order description and the amount to be authorized. Since the PI includes the payment card data required for the authorization, the Payment Gateway can authorize using the existing payment card financial network.

Continued on next page

Overview, continued

Split Shipments

When a merchant knows in advance that an order must be split into multiple shipments, the merchant shall perform multiple AuthReq steps – one for each shipment. The merchant shall set SubsequentAuthInd TRUE in the first AuthReq, which represents the authorization for the first shipment. The payment gateway will return an AuthToken in the AuthRes. The merchant shall perform additional AuthReqs for subsequent shipments. For each subsequent AuthReq, the merchant includes the AuthToken from the previous AuthRes in each AuthReq. In the final AuthReq, the merchant shall set the SubsequentAuthInd to FALSE.

When a merchant discovers an order must be split after an AuthReq has been processed, the merchant shall do an AuthRevReq to adjust the amount of the additional authorization to the correct amount for the initial shipment and set the SubsequentAuthInd TRUE to request an AuthToken. The AuthToken shall be used in AuthReq for subsequent AuthReqs to correspond with remaining shipments.

Merchant Generates AuthReq

Create AuthReq

Step	Action
1	Generate AuthTags.
2	Generate HOD2 by hashing OD, PurchAmt, ODSalt , ODExtensions, and, if present, InstallRecurData. The Payment Gateway will compare it to the value received in the PI.
3	Generate AuthReqPayload as described on page 349.
4	Optional: For concurrent authorization and capture: A. Set CaptureNow to TRUE. B. Generate SaleDetail as described on page 285. C. Optional: Populate BatchID with the value of a currently open batch for the BrandAndBIN associated with the transaction and a BatchSequenceNumber. *Note: In some situations, the Acquirer may not be able to perform combined authorization and capture even if CaptureNow is TRUE. When this happens, a "captureNotSupported" AuthCode will indicate authorization only; the Merchant may subsequently issue a CapReq message to capture the payment.*
5	Include CheckDigests, with Merchant computing H(OIData) and HOD2; H(PIData) is copied by Merchant from PReq. This field is omitted if this AuthReq is a subsequent Authorization based on the return of the AuthToken from the Payment Gateway, rather than an initial authorization based on PReq.
6	Recommended: Populate MThumbs by computing thumbprints of Certificates and CRLs held by the Merchant; the Merchant should populate thumbs in the message which may subsequently be needed to verify signatures and certificates provided by the Payment Gateway. *Note: Inclusion of this field is an optimization to reduce certificates and CRLs provided in the next message from the Payment Gateway.*
7	Invoke EncB encapsulation.
8	Include sending entity's signature and encryption certificates and certificate chain up to the Brand certificate. These may be elided if the merchant has evidence the gateway already has these certificates.
9	Invoke *Compose MessageWrapper* (see page 76) to send to Cardholder.

Continued on next page

Merchant Generates AuthReq, continued

Generate AuthTags

Step	Action
1	Populate AuthRRTags as described in RRTags on page 295.
2	Populate TransIDs. If this is a subsequent authorization and a PaySysID has been defined, populate PaySysID.
3	If this is split or recurring payment and acquirer has assigned an AuthRetNum to the authorization, copy AuthRetNum from preceding AuthRes.

Generate AuthReq Payload

Step	Action
1	If subsequent authorizations will be processed for the purchase and this is not the last authorization set SubsequentAuthInd TRUE, else set it FALSE.
2	If the merchant and cardholder have agreed on installment or recurring payments, populate InstallRecurData.
3	Set AuthReqAmt to the amount of the authorization.
4	Optional: set CardSuspect with an appropriate value if the Merchant is suspicious of the cardholder.
5	If MerchData is required by specific payment card brand policy generate it.
6	Generate MarketSpecAuthData if required by the payment card brand and type of purchase.
7	If AVSData is required by specific payment card brand policy, populate it with information provided by the cardholder using out of band mechanism.
8	If SpecialProcessing is required by specific payment card brand policy, generate it.
9	If merchant requires information about the payment card type, set the RequestCardTypeInd to True.

Continued on next page

Merchant Generates AuthReq, continued

AuthReq data

AuthReq	**EncB(M, P, AuthReqData, PI)**
AuthReqData	**{AuthReqItem, [MThumbs], CaptureNow, [SaleDetail]}**
PI	*See page 272.*
AuthReqItem	**{AuthTags, [CheckDigests], AuthReqPayload}**
MThumbs	*Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.*
CaptureNow	*Boolean indicating that capture should be performed if authorization is approved.*
SaleDetail	*See page 286.*
AuthTags	**{AuthRRTags, TransIDs, [AuthRetNum]}**
CheckDigests	**{HOIData, HOD2}** *Used by Payment Gateway to authenticate* **PI***. Omit if* **PI** *is an* **AuthToken***.*
AuthReqPayload	*See page 352.*
AuthRRTags	**RRTags,** *see page 295.* *Note:* **RRPID** *is needed because there may be more than one authorization cycle per* **PReq***.*
TransIDs	*Copied from corresponding* **OIData***; see page 323.*
AuthRetNum	*Identification of the authorization request used within the financial network.*

Table 52: **AuthReq**

Continued on next page

Merchant Generates AuthReq, continued

AuthReq data (continued)

HOIData	**DD(OIData)** *See page 325 for the definition of* **OIData**. *An independent hash computed by Merchant. Payment Gateway compares with Cardholder-produced copy in* **PI** *to verify linkage from* **PI** *to* **OIData**.
HOD2	**DD(HODInput)** *See "OIData" on page 325 for definition of* **HODInput**. *Independent computation by Merchant. Payment Gateway compares to Cardholder-produced copy in* **PI** *to verify out-of-band receipt by Merchant of relevant data. See "OIData" on page 325.*

Table 52: AuthReq, **continued**

Continued on next page

Merchant Generates AuthReq, continued

AuthReqPayload

AuthReqPayload	**{SubsequentAuthInd, AuthReqAmt, [AVSData], [SpecialProcessing], [CardSuspect], RequestCardTypeInd, [InstallRecurData], [MarketSpecAuthData], MerchData, [ARqExtensions]}**
SubsequentAuthInd	*Boolean indicating Merchant requests an additional authorization because of a split shipment.*
AuthReqAmt	*May differ from* **PurchAmt***; acquirer policy may place limitations on the permissible difference.*
AVSData	**{[StreetAddress], Location}** *Cardholder billing address; contents are received from cardholder using an out-of-band mechanism.* *See page 294 for definition of* **Location**.
SpecialProcessing	*Enumerated field indicating the type of special processing requested.*
CardSuspect	*Enumerated code indicating that Merchant is suspicious of the Cardholder and the reason for the suspicion.*
RequestCardTypeInd	*Indicates that the type of card should be returned in* **CardType** *in the response; if the information is not available, the value* **unavailable(0)** *is returned.*
InstallRecurData	*See page 274.*
MarketSpecAuthData	**< MarketAutoAuth, MarketHotelAuth, MarketTransportAuth >** *Market-specific authorization data.*
MerchData	**{ [MerchCatCode], [MerchGroup]}**
ARqExtensions	*The data in an extension to the authorization request must be financial and should be related to the processing of an authorization (or subsequent capture) by the Payment Gateway, the financial network, or the issuer.*
StreetAddress	*The street address of the cardholder.*

Table 53: **AuthReqPayload**

Continued on next page

Merchant Generates AuthReq, continued

AuthReqPayload (continued)

MarketAutoAuth	**{Duration}**
MarketHotelAuth	**{Duration, [Prestige]}**
MarketTransportAuth	**{}** *There is currently no authorization data for this market segment.*
MerchCatCode	*Four-byte code (defined in ANSI X9.10) describing Merchant’s type of business, product, or service.*
MerchGroup	*Enumerated code identifying the general category of the merchant.*
Duration	*The anticipated duration of the transaction (in days). This information assists the issuer by indicating how much time is likely to elapse between the authorization and the capture.*
Prestige	*Enumerated type of prestigious property; the meaning of the various levels are defined by the payment card brand.*

Table 53: AuthReqPayload, continued

Continued on next page

Merchant Generates AuthReq, continued

Payment Gateway processes AuthReq

Step	Action
1	Receive message from “Receive Message” (see page 77).
2	Decrypt PI.
3	Compare the TransIDs from AuthTags and PIHead or AuthToken: a) Verify that both XIDs match. b) Verify that both LID-Cs match. c) If present in both AuthTags and PIHead, verify that both LID-Ms match. If any of these verifications fail, reject the message by returning a “piAuthMismatch” AuthCode.
4	If PI is an AuthToken: a) Process AuthToken. Else, if PI is a signed PI: a) Verify that the brand in the cardholder signature certificate matches the brand of the payment gateway encryption certificates. If it does not, reject the authorization by returning a “CardMerchBrandMismatch” AuthCode. b) Verify PANData. c) Verify Unique Cardholder ID by using PAN, CardExpiry, and PANSecret from PANData and comparing with cardholder ID from commonName field of cardholder certificate. If they do not match, reject the AuthReq by sending a “signatureFailure” Error message. d) Verify SO encapsulation. e) Store data from PANData. Else, if PI is an unsigned PI: a) Verify that Payment Gateway does not require signatures by checking payment gateway certificate. If it does, reject the authorization by returning a “signatureRequired” AuthCode. b) Verify the hash in EXH. c) Store data from PANToken.
5	Verify the PIs authorization status. If PI was processed and not declined or subsequently reversed, reject the authorization by returning a “piPreviouslyUsed” AuthCode.

Continued on next page

Merchant Generates AuthReq, continued

Payment Gateway processes AuthReq (continued)

Step	Action
6	Process PIHead as follows: a) Verify that PiHead.MerchantID matches the merID field in the merchantData extension in the signature certificate used to verify the Merchant's signature on the AuthReq message. If it does not match, reject the authorization by returning a "piAuthMismatch" AuthCode. This prevents "substitution attacks" where a PI is swapped between merchants. b) Transmit TransStain to the issuer via the financial network for verification or save for possible off-line verification. c) Verify hashes of OIData received from Cardholder and Merchant. If they do not match, return a "piAuthMismatch" AuthCode. d) Verify that HOD from PIHead matches HOD2 from AuthReqPayload, reject the message by returning a "HODMismatch" Error message. e) Process PIExtensions. If unsupported critical extensions are found, reject message by returning an "unrecognizedExtension" Error message. f) Store data from PIHead.
7	If InstallRecurData is present in AuthReq, verify that InstallRecurData in AuthReqPayload and PIHead match. If they do not, reject the authorization by returning an "InstallRecurMismatch" AuthCode.
8	Store AcqBackInfo in secure local storage, if present.
9	If captureNow is TRUE and the Payment System does not support this mode, send capturedNotSupport AuthCode.
10	Perform authorization through existing payment card financial networks.
11	If CaptureNow is TRUE, Perform capture through existing payment card financial networks.
12	Proceed to generate AuthRes message.

Continued on next page

Payment Gateway Generates AuthRes

Create AuthRes

AuthRes is generated after the authorization through the payment card financial network is complete. AuthCode and AuthAmt are derived from the payment card financial network result.

Step	Action
1	Retrieve authorization data from authorization process.
2	Populate AuthTags from AuthReq, if required by the specific payment card financial networks populate AuthRetNum received in authorization process.
3	Populate current BrandCRLIdentifier held by Payment Gateway if thumb for current BrandCRLIdentifier was not received or is not current.
4	If MThumbs from AuthReq indicates that the Merchant needs a new Cert-PE to encrypt information to the Payment Gateway: a) Insert a Cert-PE in the PKCS#7 envelope. b) Insert GKThumb into AuthResData since Cert-PE itself is not protected by a signature.
5	Populate PaySysID in TransIDs if received from the authorization process.
6	If required in Merchant certificate, populate PANToken.
7	Populate AuthResBaggage as follows: Optional: a) Optional: PopulateInclude CapToken as described under "Generating CapToken" on page 281. b) Optional: PopulateInclude AcqCardMsg if appropriate based on payment card brand policy rules, need to send message, and receipt of key from Cardholder. c) Populate AuthToken as described on page 275 using values provided in merchant's InstallRecurData if additional authorizations are provided for, based on SubseqentAuthInd was set TRUE in preceding AuthReq. *Note: If none of these values are present, AuthResBaggage is an empty sequence.* This step constitutes AuthResBaggage.
8	Optional: Populate BatchStatus as required by payment card brand policy.
9	If PANToken is included invoke EncBX encapsulation. Else invoke EncBencapsulation.
10	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Payment Gateway Generates AuthRes, continued

Generate AuthResPayload

Step	Action
1	Generate CapResPayload.
2	Populate AuthCode and AuthAmt with results from the authorization process. a) If authorization is rejected, return the AuthAmt specified in the preceding AuthReq. b) If CaptureNow was indicated in the AuthReq but not performed, return "captureNotSupported" AuthCode for successful authorization.
3	Populate CurrConv with Cardholder's requested currency and current conversation rate between AuthAmt currency and CardCurr if a AuthAmt is specified in a currency other than the one used by the cardholder.
4	Populate ResponseData as follows: a) Populate AuthValCodes as follows: populate ApprovalCode, RespReason, AuthCharInd, Validation Code, and LogRefID if returned from authorization process. b) If RequestCardTypeInd was set TRUE in AuthReq, populate CardType with value returned from authorization process. c) Populate AuthCharInd with value returned from the authorization process.

Continued on next page

Payment Gateway Generates AuthRes, continued

AuthRes

AuthRes	**< EncB(P, M, AuthResData, AuthResBaggage), EncBX(P, M, AuthResData, AuthResBaggage, PANToken) >**
AuthResData	**{AuthTags, [BrandCRLIdentifier], [PEThumb], AuthResPayload}**
AuthResBaggage	**{[CapToken], [AcqCardMsg], [AuthToken]}**
PANToken	*See page 284. Sent if Merchant certificate indicates Merchant is entitled to the information.*
AuthTags	*Copied from corresponding* **AuthReq***;* **TransIDs** *and* **AuthRetNum** *may be updated with current information.*
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PEThumb	*Thumbprint of Payment Gateway certificate provided if* **AuthReq.MThumbs** *indicates Merchant needs one.*
AuthResPayload	*See page 359.*
CapToken	*See page 280.*
AcqCardMsg	*If Cardholder included* **AcqBackKeyData** *in* **PIHead***, the Payment Gateway may send this field to the Merchant containing a message (encrypted using the key data) for the Cardholder. The Merchant is required to copy* **AcqCardMsg** *to any subsequent* **PRes** *or* **InqRes** *sent to the Cardholder. See page 278.*
AuthToken	*Merchant uses as the* **PI** *in a subsequent* **AuthReq***. See page 275.*

Table 54: AuthRes

Continued on next page

Payment Gateway Generates AuthRes, continued

AuthResPayload

AuthResPayload	**{AuthHeader, [CapResPayload], [ARsExtensions]}**
AuthHeader	**{AuthAmt, AuthCode, ResponseData, [BatchStatus], [CurrConv]}**
CapResPayload	*See page 383.* *Returned if* **CaptureNow** *had a value of* **TRUE** *in* **AuthReq**.
ARsExtensions	*The data in an extension to the authorization response must be financial and should be important for the processing of the authorization response or a subsequent authorization reversal or capture request by the Payment Gateway, the financial network, or the issuer.*
AuthAmt	*Copied from* **AuthReqPayload.AuthReqAmt**.
AuthCode	*Enumerated code indicating outcome of payment authorization processing.*
ResponseData	**{[AuthValCodes], [RespReason], [CardType], [AVSResult], [LogRefID]}**
BatchStatus	*See page 296.*
CurrConv	**{CurrConvRate, CardCurr}**
AuthValCodes	**{[ApprovalCode], [AuthCharInd], [ValidationCode], [MarketSpecDataID]}**
RespReason	*Enumerated code that indicates authorization service entity and (if appropriate) reason for decline.*
CardType	*Enumerated code indicating the type of card used for the transaction.*
AVSResult	*Enumerated Address Verification Service response code.*
LogRefID	*Alphanumeric data assigned to the authorization transaction (used for matching to reversals).*

Table 55: **AuthResPayload**

Continued on next page

Payment Gateway Generates AuthRes, continued

AuthResPayload (continued)

CurrConvRate	*Currency Conversion Rate: value with which to multiply* **AuthReqAmt** *to provide an amount in the Cardholder's currency.*
CardCurr	*ISO 4217 currency code of Cardholder.*
ApprovalCode	*Approval code assigned to the transaction by the Issuer.*
AuthCharInd	*Enumerated value that indicates the conditions present when the authorization was performed.*
ValidationCode	*Four-byte alphanumeric code calculated to ensure that required fields in the authorization messages are also present in their respective clearing messages.*
MarketSpecDataID	*Enumerated code that identifies the type of market-specific data supplied on the authorization (as determined by the financial network).*

Table 55: AuthResPayload, continued

Continued on next page

Payment Gateway Generates AuthRes, continued

AuthCode

The following values are defined for AuthCode.

approved	*The authorization request was approved.*
unspecifiedFailure	*The authorization request could not be processed for a reason that does not appear elsewhere in this list.*
declined	*The authorization request was declined.*
noReply	*The Issuer did not respond to the authorization request. This value frequently indicates a temporary system outage in the Issuer's data processing facility.*
callIssuer	*The Issuer requests a telephone call from the Merchant.*
amountError	*The transaction amount could not be processed by an upstream system (acquirer, financial network, Issuer, etc.).*
expiredCard	*The card has expired.*
invalidTransaction	*The request could not be processed by an upstream system (acquirer, financial network, Issuer, etc.) because the type of transaction is not allowed.*
systemError	*The request could not be processed by an upstream system (acquirer, financial network, Issuer, etc.) because data in the request is invalid.*
piPreviouslyUsed	*The Payment Instructions in the authorization request have been used for a prior authorization request (Payment Gateway generated response).*
recurringTooSoon	*The minimum time between authorizations has not elapsed for a recurring transaction (Payment Gateway generated response).*
recurringExpired	*The expiration date for a recurring transaction has passed (Payment Gateway generated response).*
piAuthMismatch	*The data in the PI from the Cardholder does not correspond with the data in the OD from the Merchant.*
installRecur Mismatch	*InstallRecurData in the PI from the Cardholder does not correspond with InstallRecurData in the OD from the Merchant.*
captureNotSupported	*The Payment Gateway does not support capture.*
signatureRequired	*The unsigned PI option is not supported by the Payment Gateway for this brand.*
cardMerchBrand Mismatch	*The brand in the cardholder signature certificate does not match with the brand of the payment gateway encryption certificate.*

Continued on next page

Payment Gateway Generates AuthRes, continued

Merchant processes AuthRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Retrieve transaction record and compare to AuthTags: a) Verify that XID matches transaction. If it does not, reject the message and log an "unknownXID" Error. b) Verify that LID-M and, if present in transaction record, LID-C match transaction record. If either does not, reject message and log an "unknownLID" Error with the payment gateway.
3	If BrandCRLIdentifier is included in message, store with CRLs.
4	Process AuthResPayload as described on page 363.
5	Verify that GKThumb matches an existing Payment Gateway encryption certificate if GKThumb is present. If it does not, update certificate cache with current certificate.
6	If BatchStatus is present, process and store data.
7	Process AuthResBaggage: a) Store CapToken if present. b) If AcqCardMsg is present, store for return to Cardholder. c) Store AuthToken for subsequent authorization if present. *Note: If SubsequentAuthInd was set TRUE in AuthReq, AuthToken will be returned.*
8	If PANToken is present, store in secure local storage.
9	Proceed with Capture and/or Purchase Response, depending on the results of authorization, and merchant's time frame for return of purchase response.

Continued on next page

Payment Gateway Generates AuthRes, continued

Process AuthResPayload

Step	Action
1	Process ARsExtensions if present. If an unsupported extension is marked critical, log an "unrecognizedExtension" Error with the Payment Gateway and discard AuthRes.
2	Process CapResPayload if present: a) Process CRsPayExtensions. If unrecognized extensions are present and marked critical, reject AuthRes and log an "unrecognizedExtension" Error with Payment Gateway. b) Process CapCode to determine result. c) Process SaleDetail according to payment card brand policy and processing steps. d) For successful capture, record CapCode and CapAmt. *Note: If capture was requested, CapResPayload will be returned.*
3	If CurrConv is present, store for forwarding to Cardholder.
4	Process AuthCode, AuthAmt, and ResponseData: a) Process AuthCode to determine outcome. b) Store AuthCode and AuthAmt for successful outcome. c) Store ValidationCode for successful outcome, if present. d) Store AuthValCodes if present. e) Process RespReason, if present. f) Store AVSResult, if present. g) Store LogRefID, if present.

Referral Processing

Overview

When an Issuer processes an authorization request, the response can indicate three possible results: approved, declined or conditionally declined. This latter result is commonly called a *referral* and it is indicated by a value of *callIssuer*(4) in the AuthCode.

Upon receiving a referral response, a Merchant may call the acquirer using a telephone number supplied out-of-band; after identifying the transaction, the acquirer shall connect the Merchant to the Issuer. As a result of this phone call, the issuer may convert the authorization to an approval by providing the merchant with ApprovalCode during the call.

Merchant processing of referral

Merchant software shall allow the merchant server operator to enter an approval code. The software will then process the transaction as though the response code had been *approved*(0).

Note: the value from the response code shall not be changed.

A referred approval never results in capture. This is true even in the case of an AuthReq with CaptureNow set TRUE. In order to capture the transaction, the merchant software may issue and CapReq.

Merchant software shall not attempt to authorize referred authorization by issuing additional AuthReq messages.

Payment Gateway processing of referral

Payment Gateway software shall process referred authorizations as approved with two exceptions: The AuthCode shall be set to "callIssuer" and; Capture will not be performed until the merchant has received an ApprovalCode from the issuer. The AuthRes returned shall in all other ways be identical to the AuthRes that would be returned on an approved authorization.

Payment Gateway software shall also process all subsequent messages for the transactions as if the transaction had been approved if and only if the merchant provides a valid ApprovalCode.

Extension Guidelines

AuthReq-Payload	The authorization request carries information from the Merchant necessary for the Payment Gateway to produce an authorization request message that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the authorization request shall be financial and should be important for the processing of an authorization (or subsequent capture) by the Payment Gateway, the financial network or the Issuer.
AuthRes-Payload	The authorization response carries information from the Payment Gateway regarding the processing of the authorization request. The data in an extension to the authorization response shall be financial and should be important for the processing of the authorization response or a subsequent authorization reversal or capture request by the Payment Gateway, the financial network or the Issuer.

Section 2
Authorization Reversal

Overview

Introduction

The AuthRevReq/AuthRevRes message pair is used only to reduce or cancel a previously granted authorization. The pair may also be used to split a previously unsplit authorization.

Note: AuthRevReq/Res cannot be used to unsplit a previously split transaction.

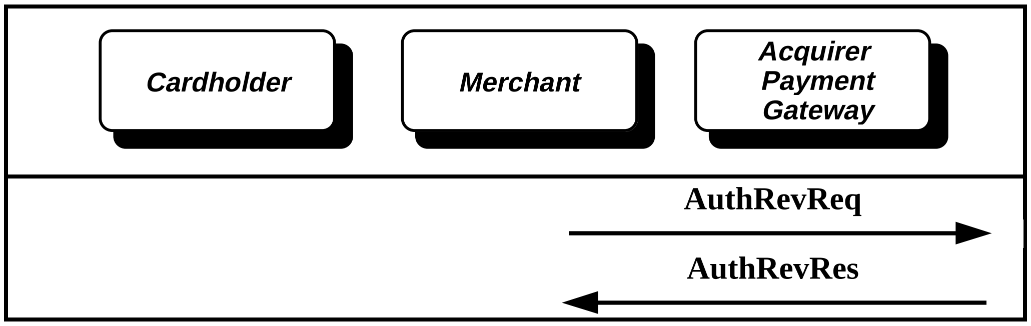


Figure 32: AuthRevReq/AuthRevRes Message Pair

Purpose

This sequence of messages is optional and is used only if change or elimination of an authorization is required. The Authorization Reversal Request message may be sent at any time after authorization and before capture has been requested to the Payment Gateway to change the amount of authorization for a transaction or even completely remove the authorization. It may also be used to split a previously unsplit transaction.

Split Shipments

If a merchant discovers after an initial, non-split AuthReq that a shipment must be split, the merchant shall use AuthRevReq to split the authorization. To do this, the merchant shall submit an AuthRevReq which reduces the AuthAmt to reflect the value of the initial shipment and set the SubsequentAuthInd TRUE. The payment gateway shall return an AuthToken in the AuthRevRes. This AuthToken will be used to authorize purchases for subsequent partial shipments.

Merchant Generates AuthRevReq

Create-AuthRevReq

In the processing steps below, "most recent authorization request" is defined as the most recently generated, AuthReq or AuthRev that is part of the corresponding transaction. Similarly, "most recent authorization response" is defined as the response to the most recently generated AuthReq or AuthRev.

Step	Action
1	Populate AuthRevTags as follows: a) Populate AuthRevRRTags as described in RRTags on page 295. b) Populate transIDs as described in TransIDs. If PaySysID has been defined, populate PaySysID. c) Copy AuthRetNum from most recent authorization request if present.
2	Populate AuthRevReqBaggage as follows: a) If an AuthToken was sent, populate PI with corresponding AuthToken. Else, copy PI from most recent authorization request. b) Copy CapToken from most recent authorization request if present.
3	Populate AuthNewAmt with requested new authorized amount. AuthNewAmt, shall be less than current authorized amount.
4	Recommended: Populate MThumbs by computing thumbprints of Certificates and CRLs held by the Merchant; the Merchant should populate thumbs in the message which may subsequently be needed to verify signatures and certificates provided by the Payment Gateway. Inclusion of this field is an optimization to reduce certificates and CRLs provided in the next message from the Payment Gateway.
5	Optional: Copy AuthReqData from most recent authorization response. To split a previously unsplit transaction, set SubsequentAuthInd in AuthReqData to TRUE.
6	Copy AuthResPayload from most recent authorization response. AuthResPayload may be omitted if CapToken is present.
7	To cancel authorization, set AuthNewAmt to zero.
8	Optional: Populate ARvRqExtensions.
9	Invoke EncB encapsulation with: a) AuthRevReqData in the ordinary slot; and b) AuthRevReqBaggage in the baggage slot.
10	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Merchant Generates AuthRevReq, continued

AuthRevReq

AuthRevReq	**EncB(M, P, AuthRevReqData, AuthRevReqBaggage)**
AuthRevReqData	**{AuthRevTags, [MThumbs], [AuthReqData], [AuthResPayload], AuthNewAmt, [ARvRqExtensions]}**
AuthRevReqBaggage	**{PI, [CapToken]}**
AuthRevTags	**{AuthRevRRTags, [AuthRetNum]}**
MThumbs	*Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.*
AuthReqData	*Copied from prior, corresponding* **AuthReq.** *Not required in message if* **CapToken** *generated by Payment Gateway contains all relevant data.*
AuthResPayload	*Copied from prior, corresponding* **AuthRes.** *Not required in message if* **CapToken** *generated by Payment Gateway contains all relevant data.*
AuthNewAmt	*New authorization amount requested. A value of zero indicates that the entire Authorization should be reversed; any other value less than the original authorized amount indicates a partial reversal. Full or partial reversals are used by Issuers to adjust the Cardholder's open to buy.*
ARvRqExtensions	*The data in an extension to the authorization reversal request must be financial and should be related to the processing of an authorization reversal (or subsequent capture) by the Payment Gateway, the financial network, or the issuer.*
PI	*Copied from prior, corresponding* **AuthReq**.
CapToken	*Copied from prior, corresponding* **AuthRes**.
AuthRevRRTags	**RRTags**. *Fresh* **RRPID** *and* **Date** *for* **AuthRev** *pair.*
AuthRetNum	*Identification of the authorization request used within the financial network.*

Table 56: AuthRevReq

Continued on next page

Merchant Generates AuthRevReq, continued

Payment Gateway processes AuthRevReq

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process ARvRqExtensions. If an unsupported extension is marked critical, reject the message by sending an "unrecognizedExtension" Error message.
3	Retrieve transaction record based on TransIDs in AuthRevTags.
4	Process MThumbs in AuthRevReqData. Ignore MThumbs in AuthReqData.
5	If transaction record indicates the transaction has been captured, reject the message by sending an "originalProcessed" Error message.
6	Verify PI by decrypting and matching against stored, previously verified PI from most recent authorization request. If they do not match, reject the reversal by sending a "piMismatch" AuthRevCode.
7	Compare the TransIDs from AuthRevTags and PIHead or AuthToken: a) Verify that both XIDs match. b) Verify that both LID-Cs match. c) If present in both AuthRevTags and PIHead, verify that both LID-Ms match. If any of these verifications fail, reject the reversal by returning a "piAuthMismatch" AuthRevCode.
8	If present, verify that AuthReqData AuthResPayload match data in transaction record. If either does not match, reject the reversal by returning and "authDataMismatch" AuthRevCode.
9	If CapToken is present: a) Verify CapToken. If CapToken is invalid, reject the message by returning a "invalidCapToken" Error message. b) Process TokenOpaque.
10	If CapToken is absent but transaction record indicates one was sent, reject the authorization by returning a "capTokenMissing" AuthRevCode.
11	Determine changes: reduction in authorization amount, addition of subsequent authorization capability, or authorization canceled.
12	Verify that AuthNewAmt is less than AuthAmt in AuthResPayload. If it is not, reject the reversal by returning an "invalidAmount" AuthRevCode.
13	Perform Reversal using existing payment card financial network.
14	Continue with Create AuthRevRes on page 370.

Payment Gateway Generates AuthRevRes

Create AuthRevRes

Step	Action
1	Retrieve authorization data from authorization reversal process.
2	Copy AuthRevTags from AuthRevReq, update with AuthRetNum received in authorization data from payment card financial network.
3	Populate current BrandCRLIdentifier held by Payment Gateway if thumb for current BrandCRLIdentifier was not received or is not current.
4	If MThumbs indicates that the Merchant needs a new Cert-PE to encrypt information to the Payment Gateway: a) Insert a Cert-PE in the PKCS#7 envelope. b) Insert GKThumb into AuthResData since Cert-PE itself is not protected by a signature.
5	Populate AuthResDataNew as follows: a) Copy TransIDs from corresponding AuthRevReq. b) Populate AuthRevCode with results from the authorization reversal process. c) Populate AuthNewAmt with results from the authorization reversal process if authorization is rejected, return the AuthNewAmt specified in the preceding AuthRevReq. d) Generate AuthResPayload as described on page 357. e) If CaptureNow is was indicated in the AuthReq but not supported, return "captureNotSupported" AuthRevCode for successful authorization.
6	Populate AuthRevResBaggage as follows: a) Optional: Populate CapTokenNew using the process for CapToken described on page 281. b) Populate AuthTokenNew using values provided in merchant's InstallRecurData if SubseqentAuthInd was set TRUE in preceding AuthRevReq.
7	Optional: Populate BatchStatus as required by payment card brand policy.
8	Optional: Populate ARvRsExtensions.
9	If CapTokenNew and/or AuthTokenNew are included, invoke EncB. Else, invoke Enc.
10	Invoke *Compose MessageWrappe* to send to Cardholder.

Continued on next page

Payment Gateway Generates AuthRevRes, continued

AuthRevRes

AuthRevRes	**< EncB(P, M, AuthRevResData, AuthRevResBaggage), Enc(P, M, AuthRevResData) >**
AuthRevResData	**{AuthRevCode, AuthRevTags, [BrandCRLIdentifier], [PEThumb], AuthNewAmt, AuthResDataNew, [ARvRsExtensions]}**
AuthRevResBaggage	**{[CapTokenNew], [AuthTokenNew]}**
AuthRevCode	*Enumerated code indicating outcome of payment authorization reversal processing.*
AuthRevTags	*Copied from corresponding* **AuthRevReq**
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PEThumb	*Thumbprint of Payment Gateway certificate provided if* **AuthRevReq.MThumbs** *indicates Merchant needs one.*
AuthNewAmt	*Copied from corresponding* **AuthRevReq.**
AuthResDataNew	**{TransIDs, [AuthResPayloadNew]}** *If* **AuthNewAmt** *is not 0, Payment Gateway creates a new instance of* **AuthResData** *(see "AuthRes" on page 358).*
ARvRsExtensions	*The data in an extension to the authorization reversal response must be financial and should be important for the processing of the authorization reversal response or a subsequent capture request by the Payment Gateway, the financial network, or the issuer.*
CapTokenNew	*New Capture Token (with updated fields), if* **AuthNewAmt** *is not 0. This replaces the* **CapToken** *returned in the corresponding* **AuthRes**.
AuthTokenNew	*New Authorization Token (with updated fields). Merchant uses as the* **PI** *in a subsequent* **AuthReq**. *See "AuthToken" on page 275.*
TransIDs	*Copied from corresponding* **AuthRevReq.**
AuthResPayloadNew	*Formally identical to* **AuthResPayload** *(see page 359); if* **AuthNewAmt** *is not 0.*

Table 57: AuthRevRes

Continued on next page

Payment Gateway Generates AuthRevRes, continued

Merchant receives AuthRevRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process ARvRsExtensions and ARsExtensions if present. If an unsupported extension is marked critical, log an "unrecognizedExtension" Error with the Payment Gateway and discard AuthRevRes.
3	Retrieve transaction record and compare against AuthRevTags. a) Verify that XID matches transaction record. If it does not log an "unknownXID" Error with the payment gateway. b) Verify that LID-M and, if present in transaction record, LID-C match values from transaction record. If they do not, log an "unknownLID" Error with the payment gateway.
4	If BrandCRLIdentifier is included in message, store with CRLs.
5	Verify that GKThumb matches an existing Payment Gateway encryption certificate if GKThumb is present. If it does not, update certificate cache with current certificate.
6	Store AuthNewAmt is not 0. If AuthNewAmt is 0, cancel the most recent authorization.
7	If BatchStatus is present, process and store data.
8	Process AuthResPayloadNew using the process for AuthResPayload described on page 363.
9	Process AuthRevResBaggage: a) Store CapTokenNew if present. b) Store AuthTokenNew for subsequent authorizations if present. *Note: If SubsequentAuthInd was set TRUE in AuthReq, AuthToken will be returned.*
10	If successful, update stored database record with new data.

Extension Guidelines

AuthRevReq-Data	The authorization reversal request carries information from the Merchant necessary for the Payment Gateway to produce an authorization reversal request message that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the authorization reversal request shall be financial and should be important for the processing of an authorization reversal (or subsequent capture) by the Payment Gateway, the financial network or the Issuer.
AuthRevRes-Data	The authorization reversal response carries information from the Payment Gateway regarding the processing of the authorization reversal request. The data in an extension to the authorization reversal response shall be financial and should be important for the processing of the authorization reversal response or a subsequent capture request by the Payment Gateway, the financial network or the Issuer.

Section 3
Capture Request/Response

Overview

Introduction

The capture request processing consists of two messages, a request from a Merchant to a Payment Gateway, and a response from the Payment Gateway back to the Merchant.

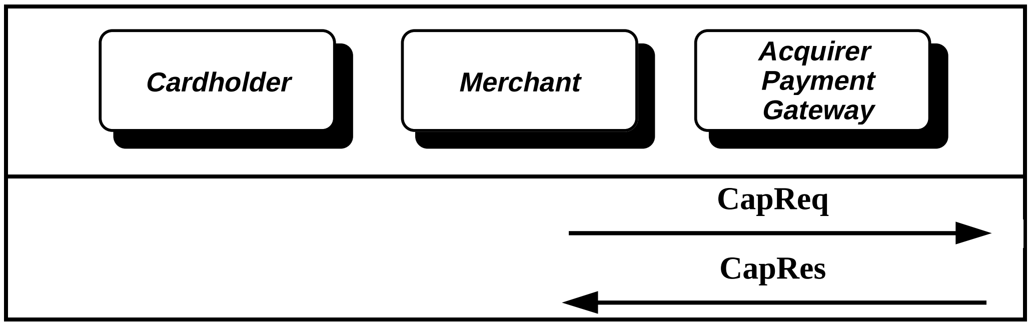


Figure 33: CapReq/CapRes Message Pair

Purpose

The Capture Request/Response message pair provides the mechanism for completing the payment of moneys previously authorized in one or more authorization transactions. Amounts captured must be previously authorized using authorization messages. A single capture message may be made up of multiple capture tokens associated with distinct transactions and their previous authorizations.

The merchant shall not send a Capture Request for previously successfully captured.

Variations

Capture may be accomplished by this protocol pair, although out-of-band methods of capture outside the scope of this protocol may also be used. The total amount captured (often by multiple capture messages) usually will be required to be less than or equal to the amount authorized by this protocol based on acquirer's rules.

Merchant Generates CapReq

Create CapReq

Step	Action
1	Populate CapRRTags as described for RRTags on page 295.
2	Optional: Populate AuthReqData, and AuthResPayload. Shall only be omitted if information is contained in CapToken.
3	Recommended: Populate MThumbs of all certificates for the Payment Gateway that the message is being sent to, for CRLs, and for BrandCRLIdentifier.
4	Populate one or more CapItems in CapSeq as follows. For each CapItem: a) Populate TransIDs and AuthRRPID for capture item from preceding messages in each transaction. b) Populate CapPayload as described on page 375. c) Populate SaleDetail as required by payment card brand policy. d) If no CapToken is available or authorization is not available to Payment gateway, copy AuthReqItem from matching AuthReqItem AuthReq and AuthResPayload from matching AuthRes.
5	Populate CapTokSeq with CapTokens from corresponding AuthRes messages in ordered one-to-one correspondence with CapItems in CapSeq. If no CapToken is available for a transaction, a null shall be used.
6	Populate the extra slot of the EncBX encapsulation with PANToken, if Merchant has received PANToken.
7	Optional: Populate CRqExtensions.
8	If PANToken is included, invoke EncBX encapsulation.
9	If PANToken is not included, invoke EncB encapsulation.
10	Invoke *Compose MessageWrapper* to send to Cardholder.

Generate CapPayload

Step	Action
1	Populate CapDate with current date.
2	Populate CapReqAmt with amount to be captured.
3	Copy AuthResPayload from corresponding AuthRes.
4	Optional: Populate CPayExtensions.

Continued on next page

Merchant Generates CapReq, continued

CapReq

The capture request message contains a signed and encrypted sequence of data elements identifying the message and providing a set of capture amounts and tokens to be processed.

CapReq	**< EncB(M, P, CapReqData, CapTokenSeq), EncBX(M, P, CapReqData, CapTokenSeq, PANToken) >** **CapTokenSeq** *is external "baggage".* *If* **PANToken** *is included, it must correspond to a single* **CapItem** *and a single* **CapToken** *in* **CapTokenSeq**.
CapReqData	**{CapRRTags, [MThumbs], CapItemSeq, [CRqExtensions]}**
CapTokenSeq	**{[CapToken] +}** *One or more* **CapTokens**, *in ordered one-to-one correspondence with* **CapItems** *in* **CapItemSeq**. *Note: Any* **CapToken** *may be omitted; that is, may be* NULL.
PANToken	*See page 284*
CapRRTags	**RRTags**. *Fresh* **RRPID** *and* **Date.**
MThumbs	*Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held in Merchant's cache.*
CapItemSeq	**{CapItem +}** *One or more* **CapItem** *in an ordered array.*

Table 58: CapReq

Continued on next page

Merchant Generates CapReq, continued

CapReq (continued)

CRqExtensions	*The data in an extension to the capture request must be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to every item in the capture request; data related to a specific item should be placed in an extension to* **CapPayload**.
CapToken	*Copied from corresponding* **AuthRes** *(see page 358) or* **AuthRevRes** *(see page 371).*
CapItem	**{TransIDs, AuthRRPID, CapPayload}**
TransIDs	*Copied from corresponding* **AuthRes** *(see page 358) or* **AuthRevRes** *(see page 371).*
AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *(see page 350)or* **AuthRevReq** *(see page 368).*
CapPayload	*See page 378.*

Table 58: CapReq, continued

Continued on next page

Merchant Generates CapReq, continued

CapPayload

CapPayload	**{CapDate, CapReqAmt, [AuthReqItem], [AuthResPayload], [SaleDetail], [CPayExtensions]}**
CapDate	*Date of capture; this is the Transaction Date that will appear on the Cardholder's statement.*
CapReqAmt	*Capture amount requested by Merchant, may differ from* **AuthAmt***; this is the Transaction Amount (before any currency conversion) that will appear on the Cardholder's statement.*
AuthReqItem	*See* "AuthReq" *on page 350.* *Required if the corresponding* **CapToken** *is not present or the Payment Gateway/acquirer systems do not contain the relevant authorization request data.*
AuthResPayload	*See page 359.* *Required if the corresponding* **CapToken** *is not present or the Payment Gateway/acquirer systems do not contain the relevant authorization response data.*
SaleDetail	*See page 286.*
CPayExtensions	*The data in an extension to the capture request payload must be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to an individual item in the capture request; data related to the entire capture request message should be placed in an extension to CapReqData.*

Table 59: CapPayload

Continued on next page

Merchant Generates CapReq, continued

Payment Gateway receives CapReq

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process CRqExtensions. If any unsupported extensions are flagged critical, reject the message by returning an "unrecognizedExtension" Error Message.
3	For each CapItem, process the capture, and create CapResItem with the amount from the capture processing and success/failure in CapCode. a) Process CapPayload as described on page **380**. b) If CapToken is present: 1. Verify CapToken. If CapToken is invalid, decline capture by returning an "invalidCapToken" CapCode for this item. 2. Verify CapToken has not already been processed. If decline capture by returning and "invalidCapToken" CapCode. 3. Process TokenOpaque. c) Else, if capture without CapToken is supported 1. If AuthReqItem and AuthResPayload are not present, decline capture by returning an "authDataMissing" CapCode. *Note: Extensions in authorization data are processed during authorization.* 2. Verify AuthReqItem and AuthResPayload against transaction records. If they do not match an existing transaction, decline capture by returning an "invalidAuthData" CapCode. d) Else, in capture without CapToken is not supported, decline capture by returning a "missingCapToken" CapCode. e) Verify TransIDs as follows: 1. Retrieve transaction record. 2. Verify XID matches transaction records. If it does not, decline capture by returning an "unknownXID" CapCode. 3. Verify LID-C and, if present, LID-M by comparing to values from transaction record. If either does not match, fail the transaction by returning an "UnknownLID" CapCode. f) Process capture for item via existing payment card financial network and store results.

Continued on next page

Merchant Generates CapReq, continued

Payment Gateway Processes CapPayLoad

Step	Action
1	Process CPayExtensions. If an unknown extension is marked critical, reject the message and return an "unrecognizedExtension" Error message.
2	Store SaleDetail.
3	Verify the BatchID is an open batch for the BrandAndBIN if present. a) If batch is unknown, decline capture by returning a "batchUnknown" CapCode. b) If not open, decline capture by returning a "batchClosed" CapCode.
4	Verify the BatchSequenceNum is unique within batch. If not unique, decline capture by returning an "batchUnknown" CapCode.

Payment Gateway Generates CapRes

Create CapRes

Step	Action
1	Retrieve capture data from capture process.
2	Copy CapRRTags from CapReq.
3	Populate current BrandCRLIdentifier held by Payment Gateway if thumb for current BrandCRLIdentifier was not received or is not current.
4	If MThumbs indicates that the Merchant needs a new Cert-PE to encrypt information to the Payment Gateway: a) Insert a Cert-PE in the PKCS#7 envelope. b) Insert GKThumb into AuthResData since Cert-PE itself is not protected by a signature.
5	Optional: Populate BatchStatusSeq with status of current batches.
6	Copy the BatchID and BatchSequenceNum from the SaleDetail to the CapResPayload.
7	Populate CapResSeq. For each CapItem in corresponding CapReq, populate a CapResItem as follows: a) Copy TransIDs from corresponding CapReqItem. b) Copy AuthRRPID from corresponding CapReqItem if present. c) Populate CapResPayload as described on page 381.
8	Optional: Populate CRsExtensions.
9	Invoke *Compose MessageWrapper* to send to Merchant.

Generate CapRes Payload

Step	Action
1	Populate CapCode and CapAmt with results from processing corresponding CapReqItem.
2	Copy BatchID and BatchSequenceNum from corresponding CapReqItem.
3	Optional: Populate CRsPayExtensions.

Continued on next page

Payment Gateway Generates CapRes, continued

CapRes

CapRes	**Enc(P, M, CapResData)**
CapResData	**{CapRRTags, [BrandCRLIdentifier], [PEThumb], [BatchStatusSeq], CapResItemSeq, [CRsExtensions]}**
CapRRTags	**RRTags***; copied from* **CapReq.**
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA . See page 249.*
PEThumb	*Thumbprint of Payment Gateway certificate provided if* **CapReqData.MThumbs** *indicates Merchant needs one.*
BatchStatusSeq	**{BatchStatus +}**
CapResItemSeq	**{CapResItem +}** *Order corresponds to* **CapReq.**
CRsExtensions	*The data in an extension to the capture response must be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to every item in the capture response; data related to a specific item should be placed in an extension to CapResPayload.*
BatchStatus	*See page 296.*
CapResItem	**{TransIDs, AuthRRPID, CapResPayload}**
TransIDs	*Copied from corresponding* **CapReq**.
AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *or* **AuthRevReq***; copied from corresponding* **CapReq**.
CapResPayload	*See page 383.*

Table 60: CapRes

Continued on next page

Payment Gateway Generates CapRes, continued

CapResPayload

CapResPayload	**{CapCode, CapAmt, [BatchID], [BatchSequenceNum], [CRsPayExtensions]}**
CapCode	*Enumerated code indicating status of capture.*
CapAmt	*Copied from corresponding* **CapReq.**
BatchID	*Identification of the settlement batch for merchant-acquirer accounting; copied from corresponding* **CapReq.**
BatchSequenceNum	*The sequence number of this item within the batch; copied from corresponding* **CapReq.**
CRsPayExtensions	*The data in an extension to the capture response payload must be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to an individual item in the capture response; data related to the entire capture response message should be placed in an extension to CapResData.*

Table 61: CapResPayload

Continued on next page

Payment Gateway Generates CapRes, continued

Merchant receives CapRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process CRsExtensions if present. If an unsupported extension is marked critical, log an "unrecognizedExtension" Error with the Payment Gateway and discard CapRes.
3	Retrieve transaction record and compare to CapRRTags: a) Verify that XID matches transaction. If it does not, reject the message and log an "unknownXID" Error with payment gateway. b) Verify that LID-M and, if present in transaction record, LID-C match transaction record. If either does not, reject message and log an "unknownLID" Error with the payment gateway.
4	If BrandCRLIdentifier is included in message, store with CRLs.
5	Verify that GKThumb matches an existing Payment Gateway encryption certificate if GKThumb is present. If it does not, update certificate cache with current certificate.
6	For each CapResItem in CapResSeq: A. Process CRsPayExtensions. If unrecognized extensions are present and marked critical, reject CapRes and log an "unrecognizedExtension" Error with Payment Gateway. B. Process CapCode to determine result. C. For successful capture, store CapCode and CapAmt, associating with AuthRRPID.
7	If BatchStatusSeq is present, process and store each BatchStatus.

Continued on next page

Payment Gateway Generates CapRes, continued

CapCode

The following values are defined for CapCode.

success	*The capture item was successfully processed by the Payment Gateway.*
unspecifiedFailure	*The reason for the failure does not appear elsewhere in this list.*
duplicateRequest	*A capture request has already been processed for this transaction (XID and AuthRRPID).*
authExpired	*The authorization request was processed too long ago. The maximum time period is defined by payment card brand rules and the Acquirer.*
authDataMissing	*The authorization information was not present in the capture request.*
invalidAuthData	*The authorization information is not valid for this transaction*
capTokenMissing	*The CapToken necessary to process this item was not present in the capture request.*
invalidCapToken	*The CapToken is not valid for this transaction*
batchUnknown	*The batch for this item is unknown to the Payment Gateway.*
batchClosed	*The batch for this item has already been closed.*
unknownXID	*The XID is not recognized*
unknownLID	*The LID is not recognized*

Extension Guidelines

CapReqData	The capture request carries information from the Merchant necessary for the Payment Gateway to produce clearing request messages (for payment) that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the capture request shall be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network or the Issuer. Note: The data in this extension applies to every item in the capture request; data related to a specific item should be placed in an extension to CapPayload.
CapPayload	The capture request payload carries information from the Merchant necessary for the Payment Gateway to produce a clearing request message (for payment) that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the capture request payload shall be financial and should be important for the processing of a capture message by the Payment Gateway, the financial network or the Issuer. Note: The data in this extension applies to an individual item in the capture request; data related to the entire capture request message should be placed in an extension to CapReqData.
CapResData	The capture response carries information from the Payment Gateway regarding the processing of the capture request. The data in an extension to the capture response shall be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network or the Issuer. Note: The data in this extension applies to every item in the capture response; data related to a specific item should be placed in an extension to CapResPayload.
CapRes Payload	The capture response payload carries information from the Payment Gateway regarding the processing of the capture request. The data in an extension to the capture response payload shall be financial and should be important for the processing of the capture response or a subsequent capture reversal or credit request by the Payment Gateway, the financial network or the Issuer. Note: The data in this extension applies to an individual item in the capture response; data related to the entire capture response message should be placed in an extension to CapResData.

Section 4
Capture Reversal or Credit Data

Overview

Introduction

Capture Reversal and Credit messages are syntactically identical and perform similar functions. Therefore, the data common to both messages are presented here and will be referenced from sections 5-7.

Merchant Generates CapRevOrCredReqData

Create CapRevOrCredReqData

Step	Action
1	Generate CapRevOrCredRRTags with fresh RRPID and current date.
2	Recommended: Populate MThumbs by computing thumbprints of Certificates and CRLs held by the Merchant; the Merchant should populate thumbs in the message which may subsequently be needed to verify signatures and certificates provided by the Payment Gateway. Inclusion of this field is an optimization to reduce certificates and CRLs provided in the next message from the Payment Gateway.
3	Populate one or more items in CredRevOrCredReqItems as follows: a) Copy TransIDs from corresponding CapRes if available. b) Copy AuthRRPID from most recent settlement request if available. c) Copy CapPayload from most recent settlement request (i.e. CapReq, CapRevReq, CredReq, or CredRevReq). d) Populate NewBatchID if transactions original batch has closed. e) Populate CapRevOrCredReqDate with current date/time. f) Optional: Populate CapRevOrReqAmt with new amount for settlement. *Note: May differ from both AuthAmt in CapToken and CapReqAmt in CapPayload.* g) Optional: Set NewAccountInd if settlement will occur on a new cardholder account as specified by included PANToken. h) Optional: Populate CRvRqItemExtensions.
4	Optional: Populate CRvRqExtensions.

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

CapRevOrCredReqData

CapRevOrCredReqData	**{CapRevOrCredRRTags, [MThumbs], CapRevOrCredReqItemSeq, [CRvRqExtensions]}**
CapRevOrCredRRTags	**RRTags,**. *Fresh* **RRPID** *and* **Date** *for this pair.*
MThumbs	*Thumbprints of certificates, CRLs, and Brand CRL Identifiers currently held Merchant's cache.*
CapRevOrCredReqItemSeq	**{CapRevOrCredReqItem +}** *One or more* **CapRevOrCredReqItem** *in an ordered array.*
CRvRqExtensions	*The data in an extension to the capture reversal or credit request must be financial and should be important for the processing of a capture reversal or credit by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to every item in the capture reversal or credit request; data related to a specific item should be placed in an extension to CapRevOrCredReqItem.*
CapRevOrCredReqItem	**{TransIDs, AuthRRPID, CapPayload, [NewBatchID], CapRevOrCredReqDate, [CapRevOrCredReqAmt], NewAccountInd, [CRvRqItemExtensions]}**
TransIDs	*Copied from the corresponding* **CapRes**. *Required if the corresponding* **CapToken** *is not present or does not contain the relevant authorization request data.*

Table 62: **CapRevOrCredReqData**

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

CapRevOrCredReqData (continued)

AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *or* **AuthRevReq.**
CapPayload	*See page 378.*
NewBatchID	*This field specifies a new batch identifier; it is used for reversal requests for items submitted in a batch that has subsequently been closed. The* **BatchID** *in* **CapPayload** *identifies the original batch.*
CapRevOrCredReqDate	*The date the request is submitted.*
CapRevOrCredReqAmt	*In credit requests, the amount of credit requested, which may differ from* **AuthAmt** *in* **CapToken** *and* **CapReqAmt** *in* **CapPayload**.
NewAccountInd	*Indicates that a new account number is specified in* **PANToken***; when this field is set, the new account number overrides the account information in the* **CaptureToken** *or authorization data retained by the acquirer. Use of this field is subject to payment card brand and acquirer policies.*
CRvRqItemExtensions	*The data in an extension to the capture reversal or credit request item must be financial and should be important for the processing of a capture reversal or credit by the Payment Gateway, the financial network or the issuer.* *Note: The data in this extension applies to an individual item in the capture reversal or credit request; data related to the entire capture reversal or credit request message should be placed in an extension to* **CapRevOrCredReqData**.

Table 62: CapRevOrCredReqData, continued

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

Payment Gateway Processes CapRevOrCredReq-Data

Step	Action
1	Process CRvRqExtensions. If an unsupported extension is marked critical, return an "unrecognizedExtension" Error message and discard message.
2	Process each CapRevOrCredItem as follows: a) Process CRvRqItemExtensions. If an unrecognized extension is marked critical, reject the message by returning an "unrecognizedExtension" Error message. b) Retrieve transaction record and compare to TransIDs in CapRevOrCredItem 1. Verify that XID matches a prior transaction. If it does not, reject message by returning an "unknownXID" Error message. 2. Verify LID-C and, if present in transaction record, LID-C match values from transaction record. If either does not, reject the message by returning an "unkownLID" Error message. c) Verify CapPayload by matching against transaction record. If it does not match, reject item by returning a "capDataMismatch" CapRevOrCredCode. *Note: CPayExtensions shall have been previously processed.* d) If NewBatchID is set, verify the BatchID is an open batch for the BrandAndBIN. If the batch has been closed, return a "batchClosed" CapRevOrCredCode. If the batch is unknown, return a "batchUnknown" CapRevOrCredCode. e) Store CapRevOrCredAmt. f) If NewAccountInd is set, use account number in PANToken for payment card financial network processing.
3	Retrieve transaction record based on TransIDs in AuthRevTags.

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

Payment Gateway Generates CapRevOrCredResData

Step	Action
1	Populate CapRevOrCredTags.
2	Populate current BrandCRLIdentifier held by Payment Gateway if thumb for current BrandCRLIdentifier was not received or is not current.
3	If MThumbs indicates that the Merchant needs a new Cert-PE to encrypt information to the Payment Gateway: a) Insert a Cert-PE in the PKCS#7 envelope. b) Insert GKThumb into AuthResData since Cert-PE itself is not protected by a signature.
4	Optional: Populate BatchStatusSeq with BatchStatus for each batch for which status was requested.
5	For each item in corresponding CapRevOrCredReqItems, populate a CapRevOrCredResItem in CapRevOrCredReqItems as follows: a) Copy TransIDs from corresponding CapRevOrCredReqItem. b) If available, copy RRPID from corresponding CapRevOrCredReqItem. 1. Populate CapRevOrCredResPayload as follows: 2. Populate CapRevOrCredCode with result from reversal or credit. 3. Populate CapRevOrCredActualAmt with actual amount or reversal or credit. 4. If present, copy BatchID and BatchSequenceNum from corresponding CapRevOrCredReqItem. 5. Optional: Populate CRvRsPayExtensions.
6	Optional: Populate CRvRsExtensions.

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

CapRevOrCredResData

CapRevOrCredResData	**{CapRevOrCredRRTags, [BrandCRLIdentifier], [PEThumb], [BatchStatusSeq], CapRevOrCredResItemSeq, [CRvRsExtensions]}**
CapRevOrCredRRTags	**RRTags**; *copied* **CapRevOrCredRRTags** *from corresponding* **CapRevOrCredReqData.**
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PEThumb	*Thumbprint of Payment Gateway certificate provided if* **CapRevOrCredReq.MThumbs** *indicates Merchant needs one.*
BatchStatusSeq	**{BatchStatus +}**
CapRevOrCredResItemSeq	**{CapRevOrCredResItem +}** *One or more* **CapRevOrCredResItem** *in an ordered array.*
CRvRsExtensions	*The data in an extension to the capture reversal or credit response must be financial and should be important for the processing of the capture reversal or credit response by the Payment Gateway, the financial network, or the issuer.* *Note: The data in this extension applies to every item in the capture reversal or credit response; data related to a specific item should be placed in an extension to CapRevOrCredResPayload.*
BatchStatus	*See page 296.*
CapRevOrCredResItem	**{TransIDs, AuthRRPID, CapRevOrCredResPayload}**
TransIDs	*Copied from corresponding* **CapRevOrCredReqData.AuthReqData.AuthTags.**
AuthRRPID	*The RRPID that appeared in the corresponding* **AuthReq** *or* **AuthRevReq.**
CapRevOrCredResPayload	*See page 394.*

Table 63: CapRevOrCredResData

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

CapRevOrCredResPayload

CapRevOrCredResPayload	**{CapRevOrCredCode, CapRevOrCredActualAmt, [BatchID], [BatchSequenceNum], [CRvRsPayExtensions]}**
CapRevOrCredCode	*Enumerated code indicating capture reversal or credit status.*
CapRevOrCredActualAmt	*Copied from corresponding* **CapRevOrCredReqItem.**
BatchID	*Identification of the settlement batch for merchant-acquirer accounting.*
BatchSequenceNum	*The sequence number of this item within the batch.*
CRvRsPayExtensions	*The data in an extension to the capture reversal or credit response must be financial and should be important for the processing of the capture reversal or credit response.* *Note: The data in this extension applies to an individual item in the capture reversal or credit response; data related to the entire capture reversal or credit response message should be placed in an extension to CapRevOrCredResData.*

Table 64: CapRevOrCredResPayload

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

CapRevOrCred Code

The following values are defined for CapRevOrCredCode.

success	The item was successfully processed by the Payment Gateway.
unspecifiedFailure	The reason for the failure does not appear elsewhere in this list.
duplicateRequest	A capture reversal or credit request has already been processed for this transaction (XID and AuthRRPID).
originalProcessed	The capture request for this item has already been processed.
originalNotFound	The specified item is not found by the Payment Gateway.
capPurged	The capture information has been purged from the Payment Gateway's transaction store.
missingCapData	The capture information was not present in the capture reversal or credit request.
missingCapToken	The CapToken necessary to process this item was not present in the capture reversal or credit request.
invalidCapToken	The CapToken is not valid
batchUnknown	The batch for this item is unknown to the Payment Gateway.
batchClosed	The batch for this item has already been closed.

Continued on next page

Merchant Generates CapRevOrCredReqData, continued

Merchant Processes CapRevOrCredResData

Step	Action
1	Process CRvRsExtensions. If any unrecognized extensions are marked critical, reject the message by returning an "unrecognizedExtension" Error message and discarding message.
2	Process CapRevOrCredTags as described in RRTags.
3	Retrieve stored transaction record and process TransIDs as follows: a) Verify that XID matches data from transaction record. If it does not, reject the message and log an "unknownXID" Error with the payment gateway. b) Verify that LID-C and, if present in transaction record, LID-M match data from transaction record. If either does not, reject the message and log an "unkownLID" error with the Payment Gateway.
4	If BrandCRLIdentifier is included in message, store with CRLs.
5	Verify that GKThumb matches an existing Payment Gateway encryption certificate if GKThumb is present. If it does not, update certificate cache with current certificate.
6	For each BatchStatus in BatchStatusSeq, process BatchStatus as described on page and store results.
7	Process each CapRevOrCredResItem in CapRevOrCredResItems as follows: a) Process CRvRsPayExtensions. If any unrecognized extensions are marked critical, reject message by returning an "unrecognizedExtension" Error message. b) Retrieve transaction records using TransIDs. If no transaction with matching TransIDs can be found, reject the message and log an "unknownXID" Error with payment gateway. c) Compare LID-C and, if present in transaction record, LID-M to values in message. If either does not match, log an "unknownLID" Error with payment gateway. d) Process CapRevOrCredResPayload as follows: 1. Process CapRevOrCredCode to determine result. 2. If credit or reversal was successful, record CapCode and CapAmt. 3. Process BatchID and BatchSequenceNum if presence.

Extension Guidelines

CapRevOrCred-ReqData

The capture reversal or credit request carries information from the Merchant necessary for the Payment Gateway to reverse a prior clearing request message (for payment) or to issue a credit request that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the capture reversal or credit request shall be financial and should be important for the processing of a capture reversal or credit by the Payment Gateway, the financial network or the Issuer.

Note: The data in this extension applies to every item in the capture reversal or credit request; data related to a specific item should be placed in an extension to CapRevOrCredReqItem.

CapRevOrCred-ReqItem

The capture reversal or credit request payload carries information from the Merchant necessary for the Payment Gateway to reverse a prior clearing request message (for payment) or to issue a credit request that can be processed by the Acquirer or financial network for transmission to the Issuer. The data in an extension to the capture reversal or credit request payload shall be financial and should be important for the processing of a capture reversal or credit by the Payment Gateway, the financial network or the Issuer.

Note: The data in this extension applies to an individual item in the capture reversal or credit request; data related to the entire capture reversal or credit request message should be placed in an extension to CapRevOrCredReqData.

CapRevOrCred-ResData

The capture reversal or credit response carries information from the Payment Gateway regarding the processing of the capture reversal or credit request. The data in an extension to the capture reversal or credit response shall be financial and should be important for the processing of the capture reversal or credit response.

Note: The data in this extension applies to every item in the capture reversal or credit response; data related to a specific item should be placed in an extension to CapRevOrCredResPayload.

CapRevOrCred-ResPayload

The capture reversal or credit response payload carries information from the Payment Gateway regarding the processing of the capture reversal or credit request. The data in an extension to the capture reversal or credit response shall be financial and should be important for the processing of the capture reversal or credit response.

Note: The data in this extension applies to an individual item in the capture reversal or credit response; data related to the entire capture reversal or credit response message should be placed in an extension to CapRevOrCredResData.

Section 5
Capture Reversal

Overview

Introduction

The CapRevReq/CapRevRes message pair is used to reduce or eliminate a previously captured amount. It shall only be used after an capture has been processed and before capture records have aged of merchant or acquirer logs.

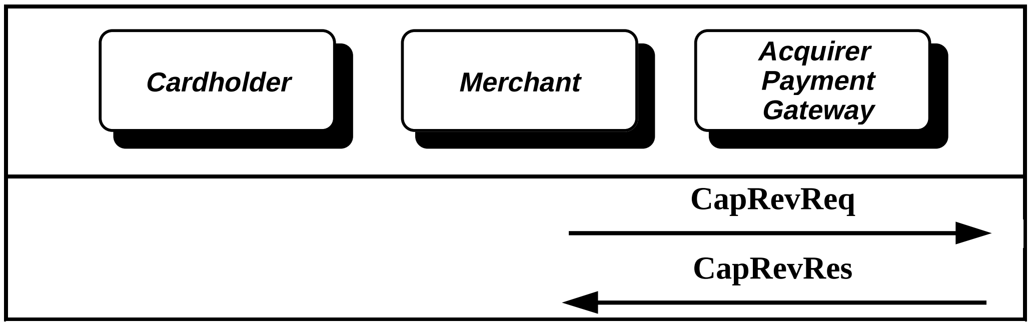


Figure 34: CapRevReq/CapRevRes Message Pair

Purpose

This sequence of messages is optional and is used only if change or elimination of a capture is required. The Capture Reversal Request message may be sent at any time after capture has been requested to the Payment Gateway to change the amount of capture for a transaction, including change to zero, which removes the capture completely.

Variations

This optional sequence may be sent repeatedly, therefore it includes its own challenge, unique to each invocation, and TransIDs in the PI to identify the intended transaction. This message is sent to reverse captures which are still in the batch queue.

If the new amount captured is zero, the CapToken is omitted.

Processing

Merchant sends CapRevReq to Payment Gateway

Step	Action
1	Generate CapRevData as described in CapRevOrCredReqData on page 388.
2	For each CapRevOrCred item in CapRevOrCredItems: Populate an item in CapTokSeq as follows: a) If available, populate with CapToken for corresponding transaction. b) Else, if unavailable, insert a NULL. *Note: The result of this step will be a CapTokSeq with a one-to-one, ordered correspondence between items in CapRevData and CapTokSeq.*
3	If available or new PAN is required, populate PANToken in extra slot of EncBX encapsulation. *Note: If PANToken is included, only one item may be present in both CapRevData and CapTokSeq.*
4	If PANToken is included, invoke EncBX encapsulation. Else, invoke EncB encapsulation.
5	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Processing, continued

CapRevReq

CapRevReq	**< EncB(M, P, CapRevData, CapTokenSeq), EncBX(M, P, CapRevData, CapTokenSeq, PANToken) >** **CapTokenSeq** *is external "baggage".* *If* **PANToken** *is included, it must correspond to a single entry in* **CapRevData.CapRevOrCredReqItemSeq** *and a single* **CapToken** in **CapTokenSeq.**
CapRevData	**CapRevOrCredReqData**; *see page 389.*
CapTokenSeq	**{[CapToken] +}** *One or more* **CapTokens,** *in ordered one-to-one correspondence with* **CapRevOrCredReqItem** *sequence in* **CapRevOrCredReqData.CapRevOrCredReqItemSeq**. *Note: Any* **CapToken** *may be omitted; that is, may be* NULL.
PANToken	*See page 284*
CapToken	*Copied from corresponding* **AuthRes** *or* **AuthRevRes.**

Table 65: CapRevReq

Payment Gateway receives CapRevReq

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	For each item for which the merchant received a CapToken: a) Verify the presence of CapToken. If CapToken is absent, reject the item by returning a "capTokenMissing" CapRevOrReqCode. b) Verify CapToken.
3	For each transaction in message, perform Reversal using existing payment card financial network as specified by contents of CapRevOrCredItem.
4	Continue with Send CapRevRes on page 401.

Continued on next page

Processing, continued

Payment Gateway sends CapRevRes

Step	Action
1	Receive response from payment card financial network.
2	Generate CapRevResData as described in CapRevOrCredResData using results from payment card financial network.
3	Invoke Enc encapsulation on result.
4	Invoke *Compose MessageWrapper* to send to Cardholder.

CapRevRes

CapRevRes	**Enc(P, M, CapRevResData)**
CapRevResData	**CapRevOrCredResData***; see page 394.*

Table 66: CapRevRes

Merchant receives CapRevRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process CapRevResData

Section 6
Credit Request/Response

Overview

Introduction

The CredReq/CredRes message pair are used to return credit on a previously captured transaction. They may be used instead of CapRevReq/Res after capture have aged off the merchant or payment gateway's logs.

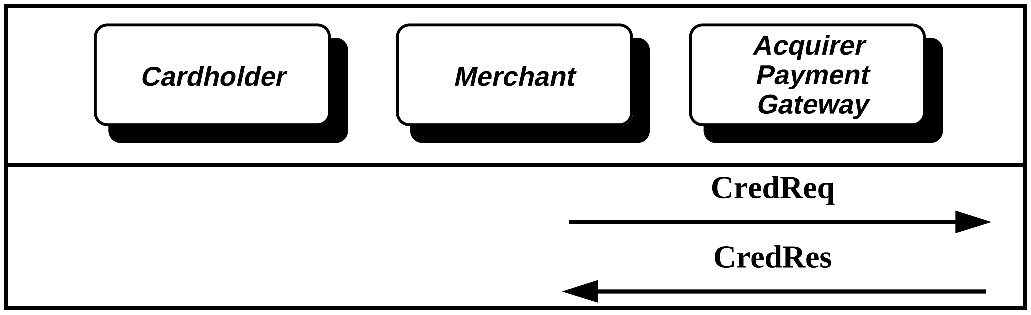


Figure 35: CredReq/CredRes Message Pair

Credit request and response

This sequence of messages is used by the Merchant to provide a credit after a prior transaction. The Credit Request message may be sent at any time. This sequence may be used after the reconciliation of Merchant account with the Acquirer.

Merchant Generates CredReq

Create CredReq

Step	Action
1	Generate CredReqData
2	For each CapRevOrCred item in CapRevOrCredItems: populate an item in CapTokSeq as follows: a) If available, populate with CapToken for corresponding transaction. b) Else, if unavailable, insert a NULL. *Note: The result of this step will be a CapTokSeq with a one-to-one, ordered correspondence between items in CredReqData and CapTokSeq.*
3	If available or new PAN is required, populate PANToken in extra slot of EncBX encapsulation. *Note: If PANToken is included, only one item may be present in both CredReqData and CapTokSeq.*
4	If PANToken is included, invoke EncBX encapsulation. Else, invoke EncB encapsulation.
5	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Merchant Generates CredReq, continued

CredReq

CredReq	**< EncB(M, P, CredReqData, CapTokenSeq), EncBX(M, P, CredReqData, CapTokenSeq, PANToken) >** **CapTokenSeq** *is external "baggage".* *If* **PANToken** *is included, it must correspond to a single entry in* **CredReqData.CapRevOrCredReqItemSeq** and a single **CapToken** in **CapTokenSeq.**
CredReqData	**CapRevOrCredReqData***; see page 389.*
CapTokenSeq	**{[CapToken] +}** *One or more* **CapTokens** *in ordered one-to-one correspondence with* **CapRevOrCredReqItem** *sequence in* **CapRevOrCredReqData.CapRevOrCredReqItemSeq**. *Note: Any* **CapToken** *may be omitted; that is, may be* NULL.
PANToken	*See page 284.*
CapToken	*Copied from corresponding* **AuthRes** *or* **AuthRevRes**.

Table 67: CredReq

Payment Gateway processes CredReq

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	For each item for which the merchant received a CapToken: a) Verify the presence of CapToken. If CapToken is absent, reject the item by returning a "capTokenMissing" CapRevOrReqCode. b) Verify CapToken
3	For each transaction in message, perform credit using existing payment card financial network as specified by contents of CapRevOrCredItem.

Payment Gateway Generates CredRes

Create CredRes

Step	Action
1	Receive response from payment card financial network.
2	Generate CredResData as described in CapRevOrCredResData on page 394 using results from payment card financial network. PopulateInclude RRTags received in request.
3	Invoke Enc encapsulation on result with results.
4	Invoke *Compose MessageWrapper* to send to Cardholder.

CredRes

CredRes	**Enc(P, M, CredResData)**
CredResData	**CapRevOrCredResData***; see page 394.*

Table 68: CredRes

Merchant processes CredRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process CredResData as described in CapRevOrCredResData on page 394. For each CapRevOrCredResItem, check the CapRevOrCredCode to determine result and record successful captured amounts.

Section 7
Credit Reversal Request/Response

Overview

Introduction

The CredRevReq/CredRevRes messages provide a mechanism for a Merchant to reverse a previous granted credit.

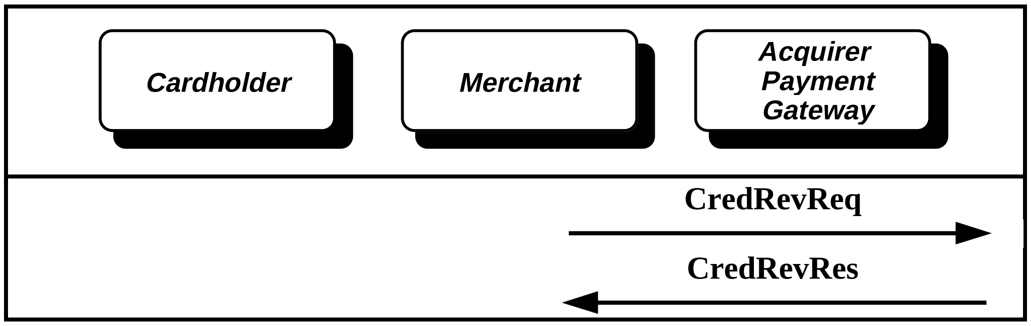


Figure 36: CredRevReq/CredRevRes Message Pair

Merchant Generates CredRevReq

Create CredRevReq

Step	Action
1	Generate CredRevReqData as described in CapRevOrCredReq on page 394.
2	For each CapRevOrCred item in CapRevOrCredItems: Populate an item in CapTokSeq as follows: a) If available, populate with CapToken for corresponding transaction. b) Else, if unavailable, insert a NULL. *Note: The result of this step will be a CapTokSeq with a one-to-one, ordered correspondence between items in CredRevReqData and CapTokSeq.*
3	If available or new PAN is required, populate PANToken in extra slot of EncBX encapsulation. *Note: If PANToken is included, only one item may be present in both CredRevReqData and CapTokSeq.*
4	If PANToken is included, invoke EncBX encapsulation. Else, invoke EncB encapsulation.
5	Invoke *Compose MessageWrapper* to send to Cardholder.

CredRevReq

CredRevReq	**< EncB(M, P, CredRevReqData, CapTokenSeq), EncBX(M, P, CredRevReqData, CapTokenSeq, PANToken) >** **CapTokenSeq** *is external "baggage".* *If* **PANToken** *is included, it must correspond to a single entry in* **CredRevReqData.CredRevReqSeq** *and a single* **CapToken** in **CapTokenSeq**.
CredRevReqData	**CapRevOrCredReqData***; see page 389.*
CapTokenSeq	**{[CapToken] +}** *One or more* **CapTokens***, in ordered one-to-one correspondence with* **CredRevReqItem** *in* **CapRevOrCredReqData.CapRevOrCredReqItemSeq**. *Note: Any* **CapToken** *may be omitted; that is, may be NULL.*
PANToken	*See page 284*
CapToken	*Copied from corresponding* **AuthRes** *or* **AuthRevRes**.

Table 69: CredRevReq

Continued on next page

Merchant Generates CredRevReq, continued

Payment Gateway processes CredRevReq

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	For each item for which the merchant received a CapToken: a) Verify the presence of CapToken. If CapToken is absent, reject the item by returning a "capTokenMissing" CapRevOrReqCode. b) Verify CapToken.
3	For each transaction in message, perform credit reversal using existing payment card financial network as specified by contents of CapRevOrCredItem.

Payment Gateway Generates CredRevRes

Create CredRevRes

Step	Action
1	Receive response from payment card financial network.
2	Generate CredRevResData as described in CapRevOrCredResData on page 394 using results from payment card financial network. PopulateInclude RRTags received in request.
3	Invoke Enc encapsulation on result with results.
4	Invoke *Compose MessageWrapper* to send to Cardholder.

CredRevRes

CredRevRes	**Enc(P, M, CredRevResData)**
CredRevResData	**CapRevOrCredResData***; see page 394.*

Table 70: CredRevRes

Merchant processes CredRevRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process CredRevResData as described in CapRevOrCredResData on page 394. For each CapRevOrCredResItem, check the CapRevOrCredCode to determine result and record successful captured amounts.

Section 8
Gateway Certificate Request/Response

Overview

Introduction

The Gateway Certificate request processing consists of two messages, a request from a Merchant to a Payment Gateway, and a response from the Payment Gateway back to the Merchant. With the request message, the Merchant requests the Payment Gateway's encryption certificate, which are needed before an encrypted message can be sent to the Payment Gateway. The certificate is returned in the response message.

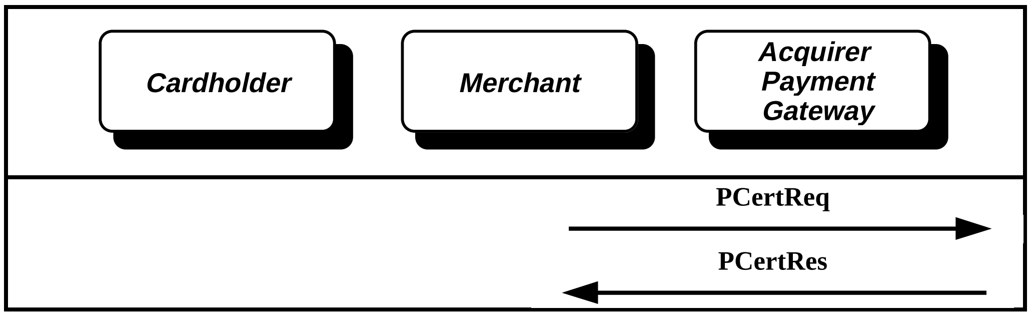


Figure 37: Gateway Certificate Request/Response

Merchant Generates PCertReq

Create PCertReq

Step	Action
1	Populate PCertTags as described in RRTags on page 295.
2	Recommended: Populate MThumbs by computing thumbprints of Certificates and CRLs held by the Merchant; the Merchant should populate thumbs in the message which may subsequently be needed to verify signatures and certificates provided by the Payment Gateway. Inclusion of this field is an optimization to reduce certificates and CRLs provided in the next message from the Payment Gateway.
3	Populate BrandIDSeq with one or more BrandIDandBINs for which certificates are required: a) Populate BrandID. b) Optional: Populate BIN
4	Optional: Populate PCRqExtensions.
5	Invoke S (see page 93).
6	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Merchant Generates PCertReq, continued

PCertReq

PCertReq	**S(M, PCertReqData)**
PCertReqData	**{PCertTags, [MThumbs], BrandAndBINSeq, [PCRqExtensions]}**
PCertTags	**RRTags**,. *Fresh* **RRPID** *for this* **PCertReq**, *Merchant-supplied* **MerTermIDs**, *and current date.*
MThumbs	*Thumbprints of Payment Gateway certificates currently in Merchant cache.*
BrandAndBINSeq	**{BrandAndBIN +}** *Merchant requests Payment Gateway certificates for these payment card brands if the thumbprint of the current certificate does not appear in* **MThumbs**.
PCRqExtensions	*Note: The Payment Gateway certificate request is not encrypted so this extension must not contain confidential information.*
BrandAndBIN	**{BrandID, [BIN]}**
BrandID	*Payment card brand (without product type).*
BIN	*Bank Identification Number for the processing of Merchant's transactions at the Payment Gateway.*

Table 71: PCertReq

Payment Gateway processes PCertReq

Step	Action
1	Gateway receives PCertReq from "Receive Message" (see page 77).
2	Process PCRqExtensions. If any unrecognized extensions are marked critical, return "unrecognizedExtension" and discard PCertReq.
3	Process BrandIDSeq and MThumbs.
4	Proceed to send PCertRes.

Payment Gateway Generates PCertRes

Create PCertRes

Step	Action
1	Copy PCertTags from PCertReq into PCertRes.
2	For each BrandAndBIN in BrandIDSeq from PCertReq, a) If a valid certificate is available: 1. Populate the corresponding Gateway Encryption Certificate Cert-PE. 2. Populate PCertCode of corresponding PCertResItem with *"success"* PCertCode. 3. Populate CertThumb with thumbprint for returned certificate. b) Else, if certificates are not available because brand in unsupported: populate PCertCode of corresponding PCertResItem with "brandUnsupported" PCertCode and omit CertThumb. c) Else, if brand is supported but BIN is unknown: populate PCertCode of corresponding PCertResItem with "unknownBIN" PCertCode and omit CertThumb. d) Else, populate PCertCode of corresponding PCertResItem with "unspecifiedFailure" and omit CertThumb.
3	For each Brand for which a certificate is returned, return current BrandCRLIdentifier unless MThumbs contained thumbprint for current BrandCRLIdentifier.
4	Optional: Populate PCrqExtensions.

Continued on next page

Payment Gateway Generates PCertRes, continued

PCertRes

PCertRes	**S(P, PCertResTBS)**
PCertResTBS	**{PCertTags, [BrandCRLIdentifierSeq], PCertResItems, [PCRsExtensions]}**
PCertTags	**RRTags**; *copied from* **PCertReq.**
BrandCRLIdentifierSeq	**{BrandCRLIdentifier +}**
PCertResItems	**{PCertResItem +}** *One or more status codes and certificate thumbprints of the certificates that are returned in a one-to-one correspondence with* **PCertReq.BrandAndBINSeq.**
PCRsExtensions	*Note: The Payment Gateway certificate response is not encrypted so this extension must not contain confidential information.*
BrandCRLIdentifier	*List of current CRLs for all CAs under a Brand CA. See page 249.*
PCertResItem	**{PCertCode, [CertThumb]}**
PCertCode	*Enumerated code indicating result of* **PCertReq.**
CertThumb	*Thumbprint of returned certificate.*

Table 72: PCertRes

Merchant processes PCertRes

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Process PCRsExtensions. If any unrecognized extensions are marked critical, log an "unrecognizedExtension" Error with Acquirer and discard PCertRes.
3	Extract certificates from Cert-PE.
4	Verify certificates in Cert-PE by matching against CertThumbs in PCertResItems. Discard all certificates that do not match a returned CertThumb.
5	Process each BrandCRLIdentifier in returned BrandCRLIdentifiers sequence.
6	Proceed to process any messages that were waiting for certificates returned in PCertRes.

Continued on next page

Payment Gateway Generates PCertRes, continued

PCertCode

The following values are defined for PCertCode.

success	The request was processed successfully.
unspecifiedFailure	The request was not processed successfully because of a failure not included elsewhere in this list.
brandNotSupported	The request was not processed successfully because a Brand specified in the **PCertReq** message was not supported.
unknownBIN	The request was not processed successfully because a BIN specified in the **PCertReq** message was not supported.

Figure 38: Enumerated Values for PCertCode

Extension Guidelines

PCertReqData	The Payment Gateway certificate request carries information to identify certificates that the Merchant desires. Note: the Payment Gateway certificate request is not encrypted so this extension shall not contain confidential information.
PCertResData	The Payment Gateway certificate response carries copies of the certificates that were requested by the Merchant. Note: the Payment Gateway certificate response is not encrypted so this extension shall not contain confidential information.

Section 9
Batch Administration

Overview

Introduction

The Merchant sends Batch Administration Requests to the Payment Gateway to administer batches of capture transactions. The batch administration processing consists of two messages, a request from a Merchant to a Payment Gateway, and a response from the Payment Gateway back to the Merchant. The request may include instructions to open, purge or close a batch and/or may transmit or request information about the batch contents. The response returns status and/or requested information.

A batch is set up by a Payment Gateway when a Merchant opens the batch to accumulate transactions and amounts captured in a specific category. All transactions which are captured (or reversed, etc.) between the Merchant and Acquirer are included in the specified batch, or in a distinguished batch called the default batch, if no batch is explicitly identified in the capture transaction. Batches enable the Merchant and Payment Gateway to reconcile collections of transactions, and identify any discrepancies. Note that a batch can support transactions for multiple brands within the batch.

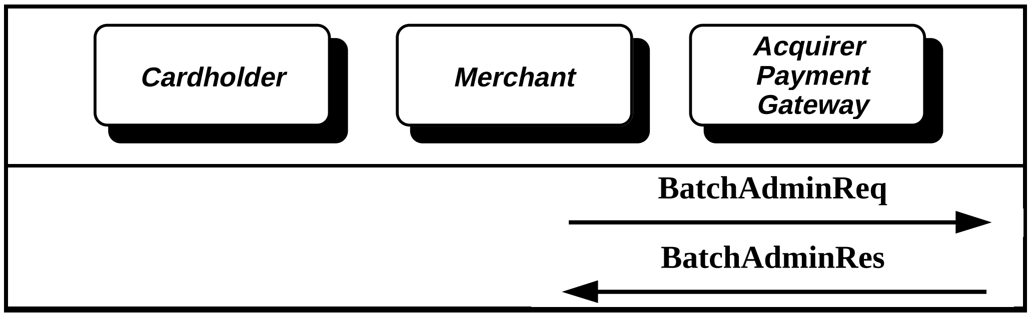


Figure 39: BatchAdminReq/BatchAdminRes Message Pair

Batch Operations

If the Merchant is controlling the content of batches, each batch shall be opened before transactions are associated with it by including a BatchID and BatchSequenceNum in capture transactions. The Merchant may also close batches in accordance with business requirements. Transactions shall not be associated with batches after closure. The Merchant also has the capability to purge all transactions from an open batch. Purging transactions shall not cause the batch to be closed.

If the Payment Gateway is controlling the content of batches, the Merchant shall not provide BatchID and BatchSequenceNum in capture transactions. The opening and closing of batches shall be controlled by the Payment Gateway who may include BatchID and BatchSequenceNum in the returned SaleDetail. The Merchant shall not be allowed to purge or close BatchIDs generated by the Payment Gateway.

Continued on next page

Overview, continued

Batch Status

The Merchant can request status information for any batch that has a known BatchID. The Payment Gateway shall return BatchStatus for the requested batch. The Merchant may request the BatchStatus to be provided for specific brands, or may request only summary totals for the batch.

If the Merchant is controlling the content of batches, then the Merchant may provide BatchStatus information to the Payment Gateway for any BatchID. The Payment Gateway shall check the Merchant provided BatchStatus against that accumulated by the Payment Gateway and return a status indicating whether or not the totals reconcile.

Batch Item Reconciliation

In a similar manner to the request for and transmission of BatchStatus, the Merchant can request or transmit transaction details associated with a particular BatchID. To allow the Merchant to request or transmit transactions for large batches, the Merchant may submit a sequence of BatchAdmin requests until the whole batch is requested or transmitted. If the Merchant requests transaction information from the Payment Gateway, then the Merchant can use this to reconcile the batch totals. If the Merchant supplies transaction information to the Payment Gateway, then the Payment Gateway shall reconcile the batch and provide a status in the BatchAdminRes which follows the BatchAdminReq containing the end of batch indication.

Merchant Generates BatchAdminReq

Create BatchAdminReq

Step	Action
1	If this is the first message to the Payment Gateway since new private keys were received, or if this is the first message of the day, include in the envelope of this message the certificates for the private keys and the certificates in their chain to the Brand certificate chosen by the Merchant for signature and encryption of BatchAdmin messages.
2	Generate RRTags as described on page 295 as BatchAdminRRTags.
3	If a new batch is to be opened: a) Set BatchOperation to open. b) Populate BatchID with the identification for an unused batch. c) Optionally, populate BrandAndBIN with a sequence of BrandIDs and, optionally, BINs to constrain the transactions that can appear in the batch. d) Set ReturnBatchSummaryIndicator to FALSE. e) Omit all other fields from the message.
4	If a batch is to be purged: a) Set BatchOperation to purge. b) Populate BatchID with the identification for an open batch. c) Optionally, populate BrandandBIN with a sequence of BrandIDs and, optionally, BINs to restrict the transaction that are purged from the batch. d) Set ReturnBatchSummaryIndicator to FALSE. e) Omit all other fields from the message.
5	If batch is to be closed: a) Set BatchOperation to close. b) Populate BatchID with the identification for an open batch. c) Set ReturnBatchSummaryIndicator to FALSE. d) Omit all other fields from the message.
6	If batch status is to be requested from the Payment Gateway: a) Omit BatchOperation. b) Populate BatchID with the identification of the batch. c) Optionally, populate BrandandBIN with a sequence of BrandIDs and, optionally, BINs to restrict the scope of the status returned in the BatchAdminRes. d) Set ReturnBatchSummaryInd to TRUE. e) Omit all other fields from the message.

Continued on next page

Merchant Generates BatchAdminReq, continued

Create BatchAdminReq (continued)

Step	Action
7	If batch detail is to be requested from the Payment Gateway: a) Omit BatchOperation. b) Populate BatchID with the identification of the batch. c) Optionally, populate BrandandBIN with a sequence of BrandIDs and, optionally, BINs to restrict the transaction that are purged from the batch. d) If batch summary information is also required set ReturnBatchSummaryInd to TRUE, otherwise set to FALSE. e) If this is the first (or only) request in the sequence, set StartingPoint to 0 otherwise set StartingPoint to the opaque NextSequence value received in the previous BatchAdminRes for this sequence. f) Populate MaximumItems with the largest number of items to be sent in the BatchAdminRes. g) Omit all other fields from the message.
8	If batch status is to be transmitted to the Payment Gateway: a) Omit BatchOperation. b) Populate BatchID with the identification of the batch. c) Omit BrandandBIN. d) Set ReturnBatchSummaryIndicator to FALSE. e) Build BatchStatus: 1. Populate BatchTotals with values for all transactions in the batch. 2. Optionally, populate BrandBatchDetails with BrandID and BatchTotals for one or more brands included in the batch. f) Omit all other fields from the message.

Continued on next page

Merchant Generates BatchAdminReq, continued

Create BatchAdminReq (continued) (continued)

Step	Action
9	If batch detail is to be transmitted to the Payment Gateway: a) Omit BatchOperation. b) Populate BatchID with the identification of the batch. c) Omit BrandandBIN. d) Set ReturnBatchSummaryIndicator to FALSE. e) If this is the last (or only) request in the sequence, set NextStartingPoint to 0 otherwise set NextStartingPoint to an opaque value to allow the Payment Gateway to check that the batch is received in the correct sequence. f) Populate TransactionDetail for a set of items from the batch and maintain a record of the items sent. g) If NextStartingPoint is 0, optionally, build BatchStatus: 1. Populate BatchTotals with values for all transactions in the batch. 2. Optionally, populate BrandBatchDetails with BrandID and BatchTotals for one or more brands included in the batch. h) If the Merchant wishes to abort the transmission of BrandBatchDetails, set the MaximumItems field to 0, otherwise omit this field. Omit all other fields from the message.
10	Invoke Signature operation using a Merchant Signature certificate for any of the brands known to the Payment Gateway (see page 93) on result of steps 1-9.
11	Include the Merchant Encryption certificate for the same brand as the Merchant Signature certificate chosen in the previous step. This certificate's public key will be used by the Payment Gateway to encrypt the BatchAdminRes.
12	Encrypt BatchAdminReqTBE using the Payment Gateway certificate and set the content type equal to id-set-content-BatchAdminReqTBE.
13	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Merchant Generates BatchAdminReq, continued

BatchAdminReq

BatchAdminReq	**Enc(M, P, BatchAdminReqData)**
BatchAdminReqData	**{BatchAdminRRTags, [BatchID], [BrandAndBINSeq], [BatchOperation], ReturnBatchSummaryInd, [ReturnTransactionDetail], [BatchStatus], [TransDetails], [BARqExtensions]}**
BatchAdminRRTags	**RRTags**,. *Fresh* **RRPID** *and* **Date.**
BatchID	*Identification of the settlement batch for merchant-acquirer accounting.*
BrandAndBINSeq	**{BrandAndBIN +}**
BatchOperation	*Enumerated value indicating the action to be performed on the batch.*
ReturnBatchSummaryInd	*Indicates batch summary data is to be returned in* **BatchAdminRes.**
ReturnTransactionDetail	**{StartingPoint, MaximumItems, ErrorsOnlyInd, [BrandID]}** *If* **BrandID** *is specified, only items for that payment card brand are returned.*
BatchStatus	*See page 296.*
TransDetails	**{NextStartingPoint, TransactionDetailSeq}**
BARqExtensions	*The data in an extension to the batch administration message must be financial and should be important for the processing of the batch administration request.*

Table 73: BatchAdminReq

Continued on next page

Merchant Generates BatchAdminReq, continued

BatchAdminReq (continued)

BrandAndBIN	**{BrandID, [BIN]}**
StartingPoint	*Zero indicates to send detail for the first group of items; otherwise,* **NextStartingPoint** *from a prior* **BatchAdminRes**.
MaximumItems	*The maximum number of items to be returned in this group of items.*
ErrorsOnlyInd	*Boolean indicating if only items with an error status should be returned.*
BrandID	*Payment card brand (without product type).*
NextStartingPoint	*Zero indicates that this is the last group of items; otherwise, an opaque value used to identify the starting point of the next group of items.*
TransactionDetailSeq	**{TransactionDetail +}**
BIN	*Bank Identification Number for the processing of Merchant's transactions at the Acquirer.*
TransactionDetail	*See page 298.*

Table 73: BatchAdminReq, continued

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq,

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Verify the signature. If it does not verify, return an Error Message with ErrorCode set to signatureFailure.
3	Verify that the RRPID in the BatchAdminReq matches the RRPID in the Message Wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
4	If BatchOperation is set to open: a) Verify that BatchID is not already open. If it does not verify, set BAStatus to batchAlreadyOpen. b) Verify that BatchID is available. If it does not verify, set BAStatus to batchIDUnavailable. c) If BrandIDSeq is present: 1. Verify that each BrandID is supported. If it does not verify, set BAStatus to brandNOTSupported. 2. Verify that each BIN is supported. If it does not verify, set BAStatus to unknownBIN. 3. Store brands and BINs which can be used with this batch. d) Open batch for use by Merchant and set BAStatus to success. e) Proceed to send BatchAdminRes. *Note: Any other fields present in the BatchAdminReq message shall be ignored, when the BatchOperation is set to open.*

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq, (continued)

Step	Action
5	If BatchOperation is set to purge: a) Verify that BatchID is already open. If it does not verify, set BAStatus to unknownbatchID. b) If BrandIDSeq is present: 1. Verify that each BrandID relates to this batch. If it does not verify, set BAStatus to brandBatchMismatch. 2. Verify that each BIN relates to this batch. If it does not verify, set BAStatus to unknownBIN. 3. Purge all transactions in the batch associated with the specified brand and BIN. c) Otherwise, purge all transactions from the batch. d) Set BAStatus to success. e) Proceed to send BatchAdminRes. *Note: Any other fields present in the BatchAdminReq message shall be ignored, when the BatchOperation is set to purge.*
6	If BatchOperation is set to close: a) Verify that BatchID is already open. If it does not verify, set BAStatus to unknownbatchID. b) Set BAStatus to success. c) Proceed to send BatchAdminRes. *Note: Any other fields present in the BatchAdminReq message shall be ignored, when the BatchOperation is set to close.*

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq, (continued)

Step	Action
7	If BatchOperation is omitted and ReturnBatchSummaryInd is TRUE: a) Verify that BatchID is available. If it does not verify, set BAStatus to batchIDUnavailable. b) If BrandAndBIN are included: 1. Verify that each BrandID relates to this batch. If it does not verify, set BAStatus to brandBatchMismatch. 2. Verify that each BIN relates to this batch. If it does not verify, set BAStatus to unknownBIN. 3. Calculate BatchTotals and populate a BrandBatchDetails data structure for each BrandAndBIN specified. c) Calculate BatchTotals for the brands included in BrandAndBIN, or for all transactions if the BrandAndBIN is omitted. Populate the BatchTotals in the BatchStatus data structure with the calculated values. d) If TransmissionStatus and SettlementInfo is available from an upstream clearing system used by the Payment Gateway, populate the BatchAdminRes with this information. e) If StartingPoint is omitted, set BAStatus to success and proceed to send BatchAdminRes, otherwise proceed to the next step. *Note: The NextStartingPoint and TransactionDetailSeq are ignored if ReturnBatchSummaryInd is TRUE.*

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq, (continued)

Step	Action
8	If StartingPoint is included: a) If MaximumItems is set to 0, discard any previous information for this batch and set the BAstatus to success. Proceed to send BatchAdminRes. b) Verify that BatchID is available. If it does not verify, set BAStatus to batchIDUnavailable. c) If StartingPoint is non-zero, verify that the StartingPoint equals the NextStartingPoint returned in the previous BatchAdminRes. If it does not verify, set BAStatus to unknownStartingPoint. d) If StartingPoint is zero, set the batch position to beginning of the batch, otherwise set the position in the batch as specified by StartingPoint. e) If BrandAndBIN are included, 1. Verify that each BrandID relates to this batch. If it does not verify, set BAStatus to brandBatchMismatch 2. Verify that each BIN relates to this batch. If it does not verify, set BAStatus to unknownBIN 3. If MaximumItems is specified, populate TransactionDetail for at most MaximumItems from the current position and set NextStartingPoint to the position from which further transactions can be obtained. If the end of the batch is reached set NextStartingPoint to 0. The choice of items is constrained by BrandandBIN and by the ErrorsOnlyInd f) Set BAStatus to success proceed to send BatchAdminRes. *Note: The NextStartingPoint and TransactionDetailSeq are ignored if StartingPoint is included.*

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq, (continued)

Step	Action
9	If BatchOperation is omitted and BatchStatus is included: a) Verify that BatchID is available. If it does not verify, set BAStatus to batchIDUnavailable. b) If BrandBatchDetails are included: 1. Verify that each BrandID relates to this batch. If it does not verify, set BAStatus to brandBatchMismatch. 2. Verify that each BIN relates to this batch. If it does not verify, set BAStatus to unknownBIN. 3. Calculate BatchTotals and populate a BrandBatchDetails data structure for each BrandAndBIN specified. c) Calculate BatchTotals for the brands included in BrandAndBIN, or for all transactions if the BrandAndBIN is omitted. d) For any set of BatchTotals that do not match those given in the BatchAdminReq message, populate the BatchTotals in the BatchStatus data structure with the calculated values. e) If any of the totals does not balance, set BAStatus to totalsOutOfBalance and proceed to the next step. f) If TransactionDetails are omitted, set BAStatus to success and proceed to send BatchAdminRes, otherwise proceed to the next step. *Note The BrandAndBIN sequence is ignored*.

Continued on next page

Merchant Generates BatchAdminReq, continued

Payment Gateway processes BatchAdminReq, (continued)

Step	Action
10	If BatchOperation is omitted and TransactionDetails are included: a) Verify that BatchID is available. If it does not verify, set BAStatus to batchIDUnavailable. b) If StartingPoint is non-zero and does not match the NextStartingPoint from the previous BatchAdminReq set BAStatus to unknownStartingPoint. c) If NextStartingPoint is non-zero, store the TransactionDetails, copy the NextStartingPoint to the BatchAdminRes message and set BAStatus to success. Proceed to send BatchAdminRes. d) Reconcile the transactions received against those held at the PaymentGateway. If any differences are found, set the BAStatus to totalsOutOfBalance. Proceed to send BatchAdminRes. e) Optionally set BAStatus to stopItemDetail to inform Merchant that the Payment Gateway is unwilling to receive more items in the batch. Proceed to send BatchAdminRes. f) Set BAStatus to success proceed to send BatchAdminRes. *Note:. The BrandAndBIN sequence is ignored.*

Payment Gateway Generates BatchAdminRes

Create BatchAdminRes

Step	Action
1	If the BAStatus is not set to success or MaximumItems in BatchAdminReq is set to 0, discard any batch information for this sequence of Batch Admin requests previously transmitted from the Merchant.
2	Invoke Signature operation using the Payment Gateway Signature certificate. (see page 93) on BatchAdminResData.
3	Encrypt BatchAdminResTBE using the provided Merchant encryption certificate and set the content type equal to id-set-content-BatchAdminResTBE.
4	Invoke *Compose MessageWrapper* to send to Cardholder.

Continued on next page

Payment Gateway Generates BatchAdminRes, continued

BatchAdminRes

BatchAdminRes	**Enc(P, M, BatchAdminResData)**
BatchAdminResData	**{BatchAdminTags, BatchID, [BAStatus], [BatchStatus], [TransmissionStatus], [SettlementInfo], [TransDetails], [BARsExtensions]}**
BatchAdminTags	**RRTags***; copied from prior* **BatchAdminReq.**
BatchID	*Identification of the settlement batch for merchant-acquirer accounting.*
BAStatus	*Enumerated code indicating status of batch open.*
BatchStatus	*See page 296.*
TransmissionStatus	*Enumerated value indicating the status of the transmission from the gateway to the next upstream system.*
SettlementInfo	**{SettlementAmount, SettlementType, SettlementAccount, SettlementDepositDate}**
TransDetails	**{NextStartingPoint, TransactionDetailSeq}**
BARsExtensions	*The data in an extension to the batch administration response message must be financial and should be important for the processing of the batch administration request.* *Note: Information regarding the processing of the request itself should appear in an extension to BatchAdminResData; information regarding the status of a batch should appear in an extension to BatchStatus; information regarding detail for an item within the capture batch should appear in an extension to TransactionDetail.*
SettlementAmount	*The net settlement amount to the Merchant's account.*
SettlementType	*Enumerated code indicating the type of amount.*
SettlementAccount	*The Merchant's account.*
SettlementDepositDate	*The date that the* **SettlementAmount** *will be credited to/debited from the Merchant's account.*

Table 74: BatchAdminRes

Continued on next page

Payment Gateway Generates BatchAdminRes, continued

BatchAdminRes (continued)

NextStartingPoint	*Zero indicates that this is the last group of items; otherwise, an opaque value used to identify the starting point of the next group of items.*
TransactionDetailSeq	**{TransactionDetail +}**
TransactionDetail	*See page 298.*

Table 74: BatchAdminRes, continued

ReimbursementID

The following values are defined for ReimbursementID.

unspecified	*Unknown or does not appear elsewhere in this list.*
standard	*Standard interchange rate.*
keyEntered	*Interchange rate for key-entered transactions.*
electronic	*Interchange rate for electronic transactions.*
additionalData	*Interchange rate for transactions that include additional clearing data.*
enhancedData	*Interchange rate for transactions that include data enhancements (such as additional authorization-related data).*
marketSpecific	*Interchange rate for transactions within a specific market segment (such as Passenger Transport).*

Continued on next page

Payment Gateway Generates BatchAdminRes, continued

Merchant processes BatchAdminRes

The Merchant receives and processes BatchAdminRes.

Step	Action
1	Receive message from "Receive Message" (see page 77).
2	Verify the signature. If it does not verify, return an Error Message with ErrorCode set to signatureFailure.
3	Verify that the RRPID in the BatchAdminReq matches the RRPID in the Message Wrapper. If it does not verify, return an Error Message with ErrorCode set to unknownRRPID.
4	If the BAStatus is not success and the Merchant is transmitting or requesting batch details, discard any information stored for this batch and restart from the beginning if details are still required.
5	If the Merchant is receiving batch details, store the NextStartingPoint for use in a subsequent BatchAdminRes. A value of zero indicates that all batch details have been transmitted.
6	If the Merchant is transmitting batch details, verify that the NextStartingPoint matches that sent in the BatchAdminReq. If it does not verify, send a BatchAdminReq with MaximumItems set to 0 to inform the Payment Gateway to discard the batch details previously sent and then resend the batch details to the Payment Gateway in a subsequent series of BatchAdmin requests.
7	Store details from the BatchAdmin request and forward to Merchant batch procedures.

Extension Guidelines

BatchAdmin-ReqData	The batch administration request carries information from the Merchant to control capture batches. The data in an extension to the batch administration message shall be financial and should be important for the processing of the batch administration request.
BatchAdmin-ResData	The batch administration response carries information from the Payment Gateway regarding the processing of the batch administration response. The data in an extension to the batch administration response message shall be financial and should be important for the processing of the batch administration request. Note: Information regarding the processing of the request itself should appear in an extension to BatchAdminResData; information regarding the status of a batch should appear in an extension to BatchStatus; information regarding detail for an item within the capture batch should appear in an extension to TransactionDetail.
BatchStatus	See BatchAdminResData.
Transaction Detail	See BatchAdminResData.

Appendices

Overview

Introduction

This is a collection of supplementary information that is applicable to the SET specification. The appendices are categorized according to the type of information discussed:

- Normative — standards and OIDs relevant to SET (A, C, E-M, R)
- Informative — including industry algorithms and guidelines (B, D, N, P, S)
- Examples — durations, message and certificate content (T,U,V)

All of the information provided in the normative category shall be included as part of the SET specification.

Organization

The following appendices are included:

Appendix	Title	Contents	Page
A	External Standards	Lists the standards supported by these the SET specification.	437
B	Terminology	Lists acronyms and definitions.	439
C	SET Messages	Provides information about when messages are used.	454
D	SET Fields	Provides an alphabetic list of fields to assist developers who will implement cardholder and merchant systems.	464
E	Field Support Requirements	Describes the support requirements for the SET fields defined as OPTIONAL in the ASN.1	466
F	Logo Display During Certificate Registration	Discusses the display of logos during certificate registration process.	482
G	Object Identifiers	Describes the types of object identifiers.	485
H	Extension Mechanism for SET Messages	Describes a mechanism to extend SET payment messages to support additional business functions.	492
K	Object Identifiers under {id-set}	Describes SET OID usage and OID management.	499

Continued on next page

Overview, continued

Organization (continued)

Appendix A
External Standards

Overview

SET design is based on standards established by industry (existing infrastructure), Internet, and international organizations as defined in ISO, IEFT, PKCS, and ANSI standards. The remainder of this appendix identifies the distinctive standards, algorithms, and certificates supported by the SET specifications.

ASN.1

Abstract Syntax Notation

ASN.1 is the notation used by SET for specifying messages. The 1995 version of the ASN.1 specification is described in ISO/IEC 8824-1, 8824-2, 8824-3, and 8824-4 documents.

DER

Distinguished Encoding Rules

Support the encoding of protocol data in unambiguous fashion both in payment messages and in certificates (as specified in X.509). The 1995 version of the DER specification is described in ISO/IEC 8825-1.

DES

Data Encryption Standard

Standard for data encryption (as specified in FIPS PUB 46-2). The DES key is distributed in an encrypted form within a digital envelope using public key encryption.

HMAC

Keyed-hashing mechanism for shared secret and blinding function

HTTP

Hyper-Text Transport Protocol

This World Wide Web Transport Protocol supports existing WWW browsers and servers (as specified in RFC 1945).

ISO 3166:1993

Codes for the representation of names of countries

ISO 4217:1995

Codes for the representation of currencies and funds

Continued on next page

Overview, continued

ISO 7812:1985	Identification Cards - Numbering system and registration procedure for issuer identifiers which includes the definition for computing for check digit
ISO 8583:1993	Financial Transaction Card Originated Messages - Interchange Message Specifications
ISO 9594-8:1997	ITU-T Recommendation X.509 (1997), Information Technology - Open Systems Interconnection - The Directory:Authentication.Framework The certificate format supported by the SET specification.
ISO 9834-7	Provides an international registration authority for object identifier arcs.
MIME	Multipurpose Internet Message Extensions Used for encoding of envelopes for payment messages, supports browser recognition of payment messages, and supports electronic mail based commerce.
PKCS	Public Key Cryptography Standards Define the cryptographic message syntax (PKCS #7) and certificate request message syntax (PKCS #10).
RFC 1766	Language tag standard
SHA-1	Secure Hashing Algorithm Developed jointly by NIST and NSA (as specified in FIPS 180-1). SET uses SHA-1 for all digital signatures.
TCP/IP	Protocol family supporting Internet communication
X.509	ITU-T Recommendation X.509 (1997)|ISO/IEC 9594-8:1997 Standard for encoding of Public Key Certificates The certificate format supported by the SET specification is defined in the ISO standard X.509 version 3; ANSI X9.57 (also ISO/IEC 9594-8:1993).

Appendix B
Terminology

Organization

This appendix includes the following lists:

- Acronyms
- Glossary
- ASN.1 Symbols Used by SET

Acronyms

ADB	Actual Data Block (OAEP)
ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
AVS	Address Verification Service
Base64	Encoding scheme used with MIME mapping to map full 8-bit bytes into 64-character set
BC	Block Contents (OAEP)
BCA	Brand Certificate Authority
BCI	Brand CRL Identifier
BID	Business Identification
BIN	Bank Identification Number
BT	Block Type (OAEP)
C	Cardholder
CA	Certificate Authority
CBC	Cipher Block Chaining
CCA	Cardholder Certificate Authority
CD-ROM	Compact Disk Read Only Memory
CDMF	Commercial Data Masking Facility
C*n*	Certificate for *n*th generation of the Root Signature Key
CPS	Custom Payment Service
CRL	Certificate Revocation List
CRLF	Carriage Return, Line Feed

Continued on next page

Acronyms, continued

DB	Data Block (OAEP)
DD	Digested Data
DEA	Data Encryption Algorithm
DEK	Data Encryption Key (OAEP)
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DN	Distinguished Name
E	Asymmetric Encryption operator
EE	End Entity: Cardholder (C), Merchant (M), or Payment Gateway (P)
EH	Integrity Encryption operator
EK	Symmetric Encryption with a provided key operator
Enc	Simple Encapsulation with signature
EncB	Simple Encapsulation with signature and encrypted baggage
EncBX	Extra Encapsulation with signature and encrypted baggage
EncK	Simple Encapsulation with signature and a provided key
EncX	Extra Encapsulation with signature
E-Salt	A fresh, random salt (OAEP)
EX	Extra Encryption operator
EXH	Extra Encryption with Integrity operator
FIPS PUB	Federal Information Processing Standards Publication

Continued on next page

Acronyms, continued

GCA	Geo-political Certificate Authority
GIF	Graphics Interchange Format
H	SHA-1 Hash operator
H*n*	Hash (SHA-1) of Certificate for *n*th generation of the Root Signature Key
H1	Hashing #1 operator for OAEP (returns leading bytes)
H2	Hashing #2 operator for OAEP (returns trailing bytes)
HDC	Host Data Capture
HMAC	Keyed Hashing Message Authentication Code
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IIC	Institution Identification Code
IIN	Institution Identification Number
IP	Internet Protocol
ISO	International Organization for Standardization
ITU	International Telecommunication Union
L	Linkage operator
M	Merchant
MAC	Message Authentication Code
MCA	Merchant Certificate Authority
MCC	Merchant Category Code
MD5	Message Digest (Version 5)
MIME	Multipurpose Internet Message Extensions
MOTO	Mail Order/Telephone Order
NIST	National Institute of Standards and Technology
NSA	National Security Agency

Continued on next page

Acronyms, continued

OAEP	Optimal Asymmetric Encryption Padding
OD	Order Description: out-of-band shopping data exchanged between cardholder and merchant
OI	Order Instruction (or Information)
OID	Object Identifier
OLE	Object Linking and Embedding
P	Payment Gateway
PAN	Primary Account Number
PCA	Payment Gateway Certificate Authority
PDB	Padded Data Block (OAEP)
PGWY	Payment Gateway
PI	Payment Instruction (or Information)
PIN	Personal Identification Number
PKCS	Public Key Cryptography Standards
PK-E	Public Key for Encryption
PK-S	Public Key for Signature
POS	Point of Sale
RA	Registration Authority
R*n*	Root Signature Key #*n*
RCA	Root Certificate Authority
RFC	Request For Comments
RRPID	Request/Response Pair Identifier
RSA	Rivest Shamir Adleman (RSA is also an example of public key cryptography)
RSADSI	RSA Data Security Incorporated
S	Signed Message operator
SAIC	Science Applications International Corporation
SET	Secure Electronic Transaction
SHA-1	Secure Hash Algorithm

Continued on next page

Acronyms, continued

SMTP	Simple Mail Transfer Protocol
SO	Signature Only operator
TBE	To Be Enveloped
TBL	To Be Linked
TBS	To Be Signed
TCP	Transmission Control Protocol
TDC	Terminal Data Capture
UDP	User Datagram Protocol
URL	Universal Resource Locator
USA	United States of America
UTC	Universal Time Coordinated
V	Verification String (OAEP)
W3C	World Wide Web Consortium
WWW	World Wide Web
XOR	Exclusive "or" bit operation

Glossary

Acquirer	The financial institution (or its agent) that acquires from the card acceptor the financial data relating to the transaction and initiates that data into an interchange system.
Array	A logical grouping of fields or data structures that may be repeated multiple times in a message.
Authentication	The process that seeks to validate identity or to prove the integrity of information. Authentication in public key systems uses digital signatures.
Authorization	The process by which a properly appointed person or persons grants permission to perform some action on behalf of an organization. This process assesses transaction risk, confirms that a given transaction does not raise the account holder's debt above the account's credit limit, and reserves the specified amount of credit. (When a merchant obtains authorization, payment for the authorized amount is guaranteed — provided, of course, that the merchant followed the rules associated with the authorization process.)
Authorization Reversal	A transaction sent when a previous authorization needs to be canceled (a full reversal) or decreased (a partial reversal). A partial reversal contains one additional field, the replacement amount, which will be less than the authorized amount. A full reversal will be used when the transaction can not be completed, such as when the cardholder cancels the order or the merchant discovers that the goods are no longer available (discontinued). A partial reversal will be used when the authorization was for the entire order and some of the goods cannot be shipped (resulting in a split shipment).
Baggage	A term denoting an opaque encrypted tuple, which is included in a SET message but appended as external data to the PKCS encapsulated data. This avoids super-encryption of the previously encrypted tuple, but guarantees linkage with the PKCS portion of the message.

Continued on next page

Glossary, continued

Bolt-on Application	A "helper" application that provides easy technique for supporting additional functionality to a mail-reader or browser using the MIME technique for specifying a relationship between data types and application programs.
Browser	Software running on the cardholder processing system that provides an interface to public data networks.
Capture	A transaction sent after the merchant has shipped the goods. This transaction will trigger the movement of funds from the Issuer to the Acquirer and then to the merchant's account.
Capture Reversal	A transaction sent when the information in a previous capture message was incorrect or should never have been sent (such as when the goods were not actually shipped). If the capture reversal is the result of incorrect information, it will be followed by a new capture message with the corrected information.
Cardholder	The holder of a valid payment card account and user of software supporting electronic commerce.
Certificate	A special kind of digitally signed data structure that contains information about a public key and the owner of the public key. SET defines the following certificate types: signature, key encipherment, certificate signature, and CRL signature.
Certificate Authority	An entity trusted by one or more users to create and assign certificates.
Certificate Chain	An ordered grouping of digital certificates, including the Root certificate, that are used to validate a specific certificate.
Certificate Renewal	The process by which a new certificate is created for an existing public key.
Certificate Revocation	The process of revoking an otherwise valid certificate by the entity that issued the certificate.
Certificate Revocation List	A list of certificate serial numbers previously issued by a certificate authority that indicate the certificates that are invalid prior to normal expiration due to compromise, disaffiliation, or some other unusual circumstance.
Certification	The process of ascertaining that a set of requirements or criteria has been fulfilled and attesting to that fact to others, usually with some written instrument. A system that has been inspected and evaluated as fully compliant with the SET protocol by duly authorized parties and process would be said to have been certified compliant.

Continued on next page

Glossary, continued

Confidentiality	The protection of sensitive and personal information from unintentional and intentional attacks and disclosure.
Credit	A transaction sent when the merchant needs to return money to the cardholder (via the Acquirer and the Issuer) following a valid capture message, such as when goods have been returned or were defective.
Credit Reversal	A transaction sent when the information in a previous credit transaction was incorrect or should never have been sent.
Cryptographic Key	A value which is used to control a cryptographic process, such as encryption or authentication. Knowledge of an appropriate key allows correct decryption or validation of a message.
Cryptography	Mathematical process used for encryption or authentication of information. The discipline which embodies principles, means, and methods for the transformation of data in order to hide its information content, prevent its undetected modification and unauthorized use, or a combination thereof.
Cryptoperiod	The time span during which a specific key is authorized for use or in which the keys for a given system may remain in effect.
Customer Agreement	Contract with a customer that sets forth the customer's responsibilities and governs which security process will be used in the conduct of business between the institution and customer.
Destruction of Information	Any condition that renders information unusable, regardless of cause.
Dictionary Attack	A cryptographic attack where the attacker builds a dictionary by encrypting known data with all possible keys so that the key for any given message may be easily obtained by looking up the encrypted data in the dictionary.
Digital Envelope	A cryptographic technique to encrypt data and send the encryption key along with the data. Generally, a symmetric algorithm is used to encrypt the data and an asymmetric algorithm is used to encrypt the encryption key.
Digital Signature	Information encrypted with an entity's private key, which is appended to a message to assure the recipient of the authenticity and integrity of the message. The digital signature proves that the message was signed by the entity owning, or having access to, the private key.
Disclosure of Information	Any condition that results in unauthorized viewing or potential viewing of information.
Dual Signature	A digital signature that covers two or more data structures by including secure hashes for each data structure in a single encrypted block. Dual signing is done for efficiency, that is, to reduce the number of public key encryption operations.

Continued on next page

Glossary, continued

Electronic Commerce	The exchange of goods and services for payment between the cardholder and merchant when some or all of the transaction is performed via electronic communication.
Encryption	The process of converting information in order to render it into a form unintelligible to all except holders of a specific cryptographic key. Use of encryption protects information between the encryption process and the decryption process (the inverse of encryption), against unauthorized disclosure.
Financial Institution	An establishment responsible for facilitating customer-initiated transactions or transmission of funds for the extension of credit or the custody, loan, exchange, or issuance of money.
Fresh	A property associated with value for a field (for example, symmetric keys, nonces and transaction identifiers) which guarantees that the value is unique.
Goods and Services Order	The price, currency, payment method, number of payments, and other terms of the transaction (also referred to as the "Order Description" in SET).
Hardware Token	A portable device (for example, smart card, and PCMCIA cards) specifically designed to store cryptographic information and possibly perform cryptographic functions in a secure manner.
Hash	A function that maps values from a large (possibly very large) domain into a smaller range. It may be used to reduce a potentially long message into a "hash value" or "message digest", which is sufficiently compact to be input into a digital signature algorithm. A "good" hash is such that the results of applying the function to a (large) set of values in the domain will be evenly (and randomly) distributed over the range.
Host Data Capture	This is a processing option under which an Acquirer host system stores or "captures" merchant transactions for payment. Depending on the operation of the system, the agreement between the merchant and the Acquirer, and the type of transactions involved, the merchant may send a combined authorization request and capture request in a single message known as a "sale" request. Other options support the separate authorization and capture transactions, as well as a number of post-processing options for capture batch balancing.
Idempotency	The property whereby one can repeat an operation and the result is the same. In terms of protocol, sending an idempotent message repeatedly would result in no change of outcome.
Installment Payments	A type of payment transaction negotiated between the merchant and the cardholder which permits the merchant to process multiple authorizations. Cardholder specifies a maximum number of permitted Authorizations for paying through installment payments.

Continued on next page

Glossary, continued

Integrity	The quality of information or a process that is free from error, whether induced accidentally or intentionally.
Interactive	A generic class for a network transport mechanism that is dependent on a logical session being maintained during the message exchange (for example, World Wide Web sessions).
Internet	The largest collection of networks in the world, interconnected in such a way as to allow them to function as a single virtual network.
Interoperability	The ability to exchange messages and keys, both manually and in an automated environment, with any other party implementing this standard, provided that both implementations use compatible options of this standard and compatible communications facilities.
Issuer	The financial institution or its agent that issues the unique primary account number (PAN) to the cardholder for the payment card brand.
Language	The designation of the language preference to be used when initiating a certificate or purchase request.
Mail Order/Telephone Order	The type of payment card transaction where the order and payment information are transmitted to the merchant either by mail or by telephone, in contrast to a "card present" or face-to-face transaction, in which the customer makes a purchase at the merchant's store. This type of transaction is also referred to as a "MOTO transaction."
Merchant	A seller of goods, services, and/or other information who accepts payment for these items electronically. The merchant may also provide electronic selling services and/or electronic delivery of items for sale.
Message Authentication Code	The code, appended to a message by the sender, which is the result of processing the message through a cryptographic process. If the receiver can generate the same code, confidence is gained that the message was not modified and that it originated with the holder of the appropriate cryptographic key.
Message Digest	The fixed-length result when a variable-length message is fed into a one-way hashing function. Message digests help verify that a message has not been altered because altering the message would change the digest.
Message Wrapper	A common set of data elements that is prefixed to each SET message to identify the particular protocol version, revision, date/time, transaction identifiers, and request/response pair identifiers (RRPID) for this cycle.
Modification of Information	The unauthorized or accidental change in information, whether detected or undetected.
Network	A collection of communication and information processing systems that may be shared among several users.

Continued on next page

Glossary, continued

Non-interactive	A generic class for a network transport mechanism that is not dependent on a logical session being maintained during the message exchange (for example, electronic mail sessions).
Non-repudiation	The proof of the integrity and origin of data – both in an unforgeable relationship – which can be verified by any party. SET does not support non-repudiation.
Nonce	A randomly generated value used to defeat "playback" attacks.
OAEP	Optimal Asymmetric Encryption Padding is a method, developed by M. Bellare and P. Rogaway, for securely padding public-key encrypted data.
Opaque	Data whose format and content is specified outside of this specification.
Optional	A term used to indicate that a field, data structure or message is not universally required. Optional fields, data structures or messages may be required in specific contexts.
Order Inquiry	The request made by the cardholder to the merchant to determine the status of a purchase request.
Ordered Array	A logical grouping of fields or data structures which may be repeated multiple times in a message and for which the relative ordering of each occurrence is significant.
Out-of-band	Information exchanged using a means of communication that is independent of the SET protocol specification.
Payload	The information sent with a message that encompasses the business data.
Payment Card	A term used in SET to collectively refer to credit cards, debit cards, charge cards, and bank cards issued by a financial institution and which reflects a relationship between the cardholder and the financial institution.
Payment Gateway	A system operated by an Acquirer for the purpose of providing electronic commerce services to the merchants in support of the Acquirer, and which interfaces to the Acquirer to support the authorization and capture of transactions.
Policy	A data element defined in the X.509 certificate that designates the brand's policy on how the certificate will be used.
Primary Account Number (PAN)	The assigned number that identifies the card issuer and cardholder. This account number is composed of an issuer identification number, an individual account number identification, and an accompanying check digit, as defined by ISO 7812-1985.

Continued on next page

Glossary, continued

Private Key	A cryptographic key used with a public key cryptographic algorithm, uniquely associated with an entity and not made public. This key is used to create digital signatures or to decrypt messages or files.
Pseudo-random	A value that is statistically random and essentially unpredictable although generated by an algorithmic process.
Public Key	A cryptographic key used with a public key cryptographic algorithm, uniquely associated with an entity publicly available publicly. It is used to verify signatures that were created with the matched private key. Public keys are also used to encrypt messages or files that can only be decrypted using the matched private key.
Public Key Certificate	Public key and identification data signed by a certificate authority to provide authentication and integrity of the key.
Public Key Cryptography	A field of cryptography invented in 1976 by Whitfield Diffie and Martin Hellman. Public key cryptography depends on a matched pair of inverse keys. Information encrypted with one key can only be decrypted with the other. This public key provides a user with the facility to both encrypt and decrypt data or text.
Random	A value in a set that has equal probability of being selected from the total population of possibilities and is hence unpredictable.
Recurring Payments	A type of payment transaction initiated by the cardholder that permits the merchant to process multiple authorizations. There are two kinds of recurring payments: multiple payments for a fixed amount (for example, four easy payments of $19.95) or repeated billings (for example, a monthly bill from an Internet service provider).
Registration Authority	An independent third-party organization that processes payment card applications for multiple payment card brands and forwards applications to the appropriate financial institutions.
Replay	An attack in which a valid message is repeated, either by the authorized originator (that is, the merchant replays a message) or by an adversary posing as the originator. Therefore, the replayed message appears valid, except for refresh nonces.
Request-Response Pair	A pair of messages flowing in opposite directions between the same parties and sharing the same RRPID.
Risk	The possibility of loss because of one or more threats to information (not to be confused with financial or business risk).
Root Certificate	The certificate at the top of the certificate hierarchy.

Continued on next page

Glossary, continued

Sales Transaction	A payment authorization transaction that allows a merchant to authorize a transaction and request payment in a single message to the Acquirer.
Sequence	An abstract grouping of zero or more data elements. Also called a "tuple."
Server	A computer that acts as a provider of some service to other computers, such as processing communications, interface with file storage, or printing facility.
Spamming	An inappropriate use of a networked communications facility in which the same message is sent simultaneously to many recipients.
Split Shipment	Occurs when the merchant is unable to provide or deliver one or more of the requested goods and services to the cardholder, most likely due to insufficient inventory. The merchant indicates an intention to perform a subsequent authorization request to the Acquirer for the backordered goods and services.
Substitution Attack	An attack in which the attacker substitutes one message for another
Terminal Data Capture	A processing option in which authorized transactions are held in a merchant-based system and submitted to the Acquirer as a capture transaction at a time controlled and specified by the merchant. Under this option, the merchant controls the contents of the "transaction batch", as well as the time of capture transaction submission. The Acquirer is not required to maintain capture files on behalf of the merchant.
Threat	A condition which may cause information or information processing resources to be intentionally or accidentally lost, modified, exposed, made inaccessible, or otherwise affected to the detriment of the institution.
Thumbprint	The hash calculated over an item to generate or verify the signature.
Thumbs	An instance of one or more thumbprints.
Transaction	A sequence of one or more related messages.
Trust Chain	A synonym for certificate chain.
Tuple	An abstract grouping of zero or more data elements. Also called a "sequence."
Unavailability of Service	The inability to access information or information processing resources for any reason, such as disaster, power failure, or malicious actions.

ASN.1 Symbols Used by SET

BIT STRING	A field containing a sequence of zero or more bits
BMPString	A field containing a sequence of Basic Multilingual Plane (BMP) characters
BOOLEAN	A field containing a value of either TRUE or FALSE
CHOICE	A field containing a union of one or more types
CLASS	An intrinsic type used to define additional data types using simple type definitions and constraint rules
ENUMERATED	A field whose value is bound to pre-defined identifiers
GeneralizedTime	A field containing a string a calendar date and time
IA5String	A field containing a sequence of IA5 (ASCII subset) characters
INTEGER	A field containing a integer number value
NULL	A field containing a null value
NumericString	A field containing a string of digits or space
OBJECT IDENTIFIER	A sequence of integers that identify an object
OCTET STRING	A field containing a sequence of 8-bit values
PrintableString	A field containing a sequence of printable characters
REAL	A field containing a real number value
SEQUENCE	A type containing an ordered collection of one or more fields
SET	A type containing an unordered collection of one or more fields
TYPE-IDENTIFIER	An intrinsic type used to refer to the value of an OBJECT IDENTIFIER type by its unique identifier
UniversalString	A field containing a sequence of universal characters
UTCTime	A field containing a string a calendar date and time using two digit year
VisibleString	A field containing a string of visible characters

Appendix C
SET Messages

Overview

Purpose

SET supports variations in the message protocol, the message data, and the protocol implementation. These variations are necessary in order to satisfy the business requirements and policies established by the payment card brand, Acquirer, and merchant.

This appendix summarizes the messages, including variations.

Appendix D describes the message data structures and fields.

Appendix E describes the field support requirements for the SET data structure and fields that are described as OPTIONAL in the ASN.1.

Appendix S describes implementation variations.

Organization

This appendix includes the following topics:

Topic	Contents	Page
Certificate Request Messages	Lists all certificate request/response pairs, indicates which are optional, and under what circumstances.	455
Payment System Messages	Lists all payment system request/response pairs, indicates which are optional (and under what circumstances), and provides information about processing variations.	458

Error message

In addition to the certificate request messages and payment system messages, SET includes an error message.

This message is used to notify the sender when a receiver of a SET message determines that it is corrupted or cannot be processed by the recipient to the point of sending a response message.

Certificate Request Messages

Messages

SET includes the following certificate request messages:

CardCInitReq **CardCInitRes**	Cardholder's certificate initialization messages
Me-AqCInitReq **Me-AqCInitRes**	Merchant or Acquirer Certificate Initialization messages
RegFormReq **RegFormRes**	Cardholder's Registration Form messages
CertReq **CertRes**	Certificate request and response messages
CertInqReq **CertInqRes**	Certificate Inquiry messages

The following pages indicate which of these messages are optional, and under what circumstances.

Continued on next page

Certificate Request Messages, continued

Requirements for message support

Table 75 defines the requirements for support of SET certificate request messages in applications created for various end entities.

An M in the column for an end entity indicates that support for the message is mandatory; applications created for that end entity shall support the message.

An O in the column for an end entity indicates that support for the message is optional; a vendor creating applications for that end entity may choose not to support the message (within limitations described below).

Either of these indicators may be followed by one or more numbers in parentheses that specify conditions for support of the message. The conditions are described on page 457.

An x in the Idempotent column indicates that the recipient of the message shall preserve the idempotency property presented in Part I on page 69.

Message	Cardholder	Merchant	Payment Gateway	CA	Idempotent
CardCInitReq	O (1, 2)			M	
CardCInitRes	O (1, 2)			M	
Me-AqCInitReq		O (1)	O (1)	M	
Me-AqCInitRes		O (1)	O (1)	M	
RegFormReq	O (1, 2)			M	
RegFormRes	O (1, 2)			M	
CertReq	M	M	M	M	x
CertRes	M	M	M	M	x
CertInqReq	O (1, 3)	O (1, 3)	O (1, 3)	M	
CertInqRes	O (1, 3)	O (1, 3)	O (1, 3)	M	

Table 75: Certificate Request Message Summary

Continued on next page

Certificate Request Messages, continued

Support conditions

The following are conditions that apply to the support of various certificate request messages. These numbers are used as codes in Table 75 on page 456.

1. An application that ships with a registration form and certificates for the financial institution may support only the CertReq/CertRes pair. Note: Such an application will not function for any other financial institution and shall cease to function when the CA certificate expires. For example, an issuing bank can ship a wallet to its cardholders that is pre-loaded with an appropriate registration form.
2. Cardholder software need not support these message pairs if there exists an out-of-band alternative for obtaining the CA's public key and the registration template specified by the payment card brand's Issuing financial institution. CardCInitReq/Res and RegFormReq/Res are intended to support on-line requests for certificates in environments that require cardholder certificates.
3. Support of CertInqReq and CertInqRes depends upon the characteristics of the intended communications environment.
   - When direct communications exists between the cardholder and merchant using an interactive transport (as in the WWW), cardholder, Merchant, and Payment Gateway software shall support CertInqReq and CertInqRes.
   - When the communications between cardholder and merchant occurs indirectly through a non-interactive transport (as with e-mail), neither of those entities is required to support this message pair.

Payment System Messages

Messages

SET includes the following payment system messages:

PInitReq **PInitRes**	Payment initialization messages
PReq **PRes**	Purchase request and response messages
InqReq **InqRes**	Inquiry messages
AuthReq **AuthRes**	Authorization request and response messages
CapReq **CapRes**	Capture request and response messages
AuthRevReq **AuthRevRes**	Authorization reversal messages
CapRevReq **CapRevRes**	Capture reversal messages
CredReq **CredRes**	Credit messages
CredRevReq **CredRevRes**	Credit reversal messages
PCertReq **PCertRes**	Fetch certificate messages
BatchAdminReq **BatchAdminRes**	Batch administration messages

The following pages provide information about processing variations. Table 76 on page 462 indicates which of these messages are optional, and under what circumstances.

Continued on next page

Payment System Messages, continued

PInitReq
PInitRes

Payment initialization messages.

Enables the merchant to send encryption certificate(s) to the cardholder. Optional depending upon the characteristics of the intended communications environment. See page 463.

PReq
PRes

Purchase request and response messages.

In order to permit maximum flexibility needed to support the variety of business models, some merchants may elect to send the PRes message at different stages during the transaction's processing using SET. Depending upon the operating guidelines established by the brand as well as the relationship between a specific merchant and its Acquirer, some merchants may wait until authorization and capture responses are received from the Acquirer before sending PRes. A merchant may elect to send the PRes message after an authorization-only response has been received from the Acquirer.

There can also be variations due to the environment used to send the PReq to the merchant. In a non-interactive transport environment, the merchant software may be able to wait until inventory has been checked and the authorization has been obtained. The Inquiry message (InqReq) has been provided so that the cardholder can determine the status of an order at any time. Note that the Purchase Response and the Inquiry Response are identical (with the details changing over time as the order is processed).

The preferred implementation is to send back PRes after authorization; however, this is dependent upon the processing environment.

Continued on next page

Payment System Messages, continued

InqReq
InqRes

Inquiry messages.

Enables the cardholder to obtain status of a transaction. Multiple inquiry messages can be sent referencing the same transaction after the PInitRes optional message has been received or PReq sent. Inquiries originating from cardholders without certificates are not signed and their integrity may not be guaranteed.

The format of this message is dependent upon whether the payment card brand requires a signature certificate for its cardholders. If cardholder certificates are required, the signed variant of InqReq is used, otherwise its unsigned format is used.

The merchant is required to verify that the certificate accompanying the InqReq signed format matches the certificate originally used with PReq. This prevents one cardholder from inquiring about another cardholder's purchases.

AuthReq
AuthRes

Authorization request and response messages used by the merchant to request authorization from its Acquirer.

CapReq
CapRes

Capture request and response messages.

Not required if capture is performed outside of SET. When capture is performed using SET and depending upon the merchant's relationship with the Acquirer, full capture capability allows the merchant to request a capture after authorization or alternatively using an authorization-and-capture request or "sale transaction". In addition, full capture includes the capture reversal message pair.

AuthRevReq
AuthRevRes

Authorization reversal messages.

Supports the merchant's need to cancel an authorization or to reduce the amount of a prior authorization request. Depending upon the payment card brand's policy, the merchant may reduce the amount of a prior authorization by sending a replacement amount in the AuthRevReq message. This replacement amount may be less than the amount of the original authorization.

CapRevReq
CapRevRes

Capture reversal messages.

Supports the merchant's need to change or eliminate a previous capture request sent to the Acquirer. It is used when there has been an error in processing of the capture request, such as when the merchant inputs the incorrect amount for shipping. In general, this message will be processed the same business day as the capture and the cardholder will never be aware of its existence.

Continued on next page

Payment System Messages, continued

CredReq **CredRes**	Credit messages. Supports the merchant's need to refund money to the cardholder, such as in the case of damaged goods, on a previously captured transaction. In general, this message will not be processed the same business day as the capture and the credit will appear on the cardholder's statement.
CredRevReq **CredRevRes**	Credit reversal messages. Supports the merchant's need to reverse a previous, erroneously granted credit.
PCertReq **PCertRes**	Fetch certificate messages. Used by the merchant to obtain current certificate and CRLs on-demand from its Acquirer.
BatchAdminReq **BatchAdminRes**	Batch administration messages. Depending upon the relationship between the merchant and Acquirer, the merchant can assign an identifier to a batch of capture items to be processed together. This message pair can then be used by the merchant to facilitate end-of-day processing functions and manage capture token batches that result from full capture processing

Continued on next page

Payment System Messages, continued

Requirements for message support

Table 76 defines the requirements for support of SET payment system messages in applications created for various end entities.

An M in the column for an end entity indicates that support for the message is mandatory; applications created for that end entity shall support the message.

An O in the column for an end entity indicates that support for the message is optional; a vendor creating applications for that end entity may choose not to support the message (within limitations described below).

Either of these indicators may be followed by one or more numbers in parentheses that specify conditions for support of the message. The conditions are described on page 463.

An x in the Idempotent column indicates that the recipient of the message shall preserve the idempotency property presented in Part I on page 69.

Message	Cardholder	Merchant	Payment Gateway	CA	Idempotent
PInitReq	**O (1)**	**M (1)**			
PInitRes	**O (1)**	**M (1)**			
PReq	**M**	**M**			**x**
PRes	**M**	**M**			**x**
InqReq	**M**	**M**			
InqRes	**M**	**M**			
AuthReq		**M**	**M**		**x**
AuthRes		**M**	**M**		**x**
CapReq		**O (2)**	**O (2)**		**x**
CapRes		**O (2)**	**O (2)**		**x**
AuthRevReq		**M**	**M**		**x**
AuthRevRes		**M**	**M**		**x**
CapRevReq		**O (2)**	**O (2)**		**x**
CapRevRes		**O (2)**	**O (2)**		**x**
CredReq		**O**	**O**		**x**
CredRes		**O**	**O**		**x**
CredRevReq		**O**	**O**		**x**
CredRevRes		**O**	**O**		**x**
PCertReq		**M**	**M**		
PCertRes		**M**	**M**		
BatchAdminReq		**O**	**O**		**x**
BatchAdminRes		**O**	**O**		**x**

Table 76: Payment System Message Summary

Continued on next page

Payment System Messages, continued

Support conditions

The following are conditions that apply to the support of various certificate request messages. These numbers are used as codes in Table 76 on page 462.

1. Support of **PInitReq** and **PInitRes** by the cardholder and merchant depends upon the characteristics of the intended communications environment.
   - When direct communications exists between the cardholder and merchant using an interactive transport (as in the WWW), then both cardholder and merchant should implement **PInitReq** and **PInitRes**.
   - When the communications between cardholder and merchant occurs indirectly through a non-interactive transport (as with e-mail), then both cardholder and merchant need not support **PInitReq** and **PInitRes**.
2. The merchant and payment gateway systems are not required to support capture related messages. If such support is not included, the merchant software must have an alternative interface for sending transactions to the Acquirer.
3. The merchant and payment gateway systems are not required to support the **BatchAdminReq** and **BatchAdminRes** message if an alternative method is agreed upon between the Merchant and Acquirer.

Appendix D
SET Fields

Overview`

This appendix provides an alphabetic list of fields to assist developers who will implement cardholder and merchant systems. For each field, the following information is provided:

- definition,
- names of the messages in which the field occurs,
- whether the field is required, omitted, or optional in various contexts.

Note: as of the publication date, the editing of this appendix had not been completed. This appendix will be published at a later date.

Brand payment system fields

Each brand will separately publish a mapping between SET fields and the fields used by the brand's payment system.

Continued on next page

Payment System Messages, continued

This page reserved for SET field information.

Appendix E
Field Support Requirements

Introduction

Support for optional fields

Many SET fields are described as OPTIONAL in the ASN.1.

This appendix documents the support requirements of the optional fields. That is, certain fields shall be supported by the applications created for various end entities, regardless of the choices the end entities might make about using the fields in various contexts.

For example:

- The field might not be required by a particular Acquirer, and might therefore be omitted by a Merchant sending transactions to that Acquirer.
- Certain fields are required in a response message only if they were provided in the corresponding request message.

Organization

This appendix includes the following topics:

Support Conditions	A numbered list of conditions related to support of certain fields. The numbers are used in the Field Support Summary to link specific fields with specific conditions.
Support Requirements	A table listing all optional fields and indicating the conditions for support of each.

Compatibility and interoperability

The recipient of a SET message shall bear the burden of compatibility/interoperability when generating responses. Messages subsequently generated by the recipient shall be an accurate reflection of the information associated with the transaction already received.

Support Conditions

Support conditions

The following are conditions that apply to the support of various fields. These numbers are used as codes in the Field Support Summary that begins on page 468.

- An application shall recognize the presence of a message extension, test the **critical** flag and return an **Error** message if the flag is critical and the extension identified by the **extnID** is not supported.
- An application may optionally include its own local identifier in a request or response message; recipients of the message shall copy the local identifier into any subsequent message generated for the same transaction.
- An application with a valid signature certificate shall sign the **Error** message.
- Inclusion of thumbprints in a request is at the discretion of the application generating the request; omitting or including certificates, CRLs and BrandCRLIdentifiers in a response based on the thumbprints is at the discretion of the application generating the response.
- The application generating the response shall copy the thumbprints from the request message; the application receiving the response shall verify that the thumbprints in the response match the request.
- If the **Me-AqCInitRes** message includes **AcctDataField**, the **EncX** choice of **CertReq** shall be used; otherwise, the **Enc** choice shall be used.
- A Cardholder CA is not required to support this option.
- A Merchant or Payment Gateway CA is not required to support this option.
- A Payment Gateway that supports **CapReq** shall perform capture processing for the transaction if the authorization request is approved; a Payment Gateway that does not support **CapReq** shall return a value of **captureNotSupported** in the **AuthCode** field.
- Merchant software intended to be used in a Travel and Entertainment market segment (Auto Rental, Hotel, or Passenger Transport) shall support the additional data requirements of the market segment; all other **SaleDetail** fields may be supported at the discretion of the application vendor.
- Support for this **SaleDetail** field is recommended.
- Support for CommercialCardData is only necessary if
  • the merchant supports commercial card products and
  • enhanced data is not transmitted through other channels (out of band to SET).
- The requirements for inclusion of this field are specified by each brand.
- The requirements for inclusion of this field are determined by the Acquirer operating the Payment Gateway.
- The requirements for inclusion of this certificate or CRL field are determined by each brand.

Support Requirements

Using the Field Support Summary

An **M** in the column for an end entity indicates that support for the field is mandatory; applications created for that end entity shall support the field.

An **O** in the column for an end entity indicates that support for the field is optional; a vendor creating applications for that entity may choose not to support the field.

Either of these indicators may be followed by one or more numbers in parentheses that specify conditions for support of the field. Shading is used to highlight the fields or data structures that are common to multiple messages/data structures. The conditions are described on page 467.

Fields listed for one entity

Some fields are listed for only one entity. If a different entity receives a SET message that includes one of these fields, the entity may have to copy the field and send it back in a subsequent message.

Field Support Summary

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
AcctInfo	acctData		**M**	**M**	**M (7)**
Me-AqCInitResTBS	acctDataField		**M**	**M**	**M**
PIHead	acqBackInfo	**M**		**O**	
AuthTokenData	acqBackKeyData			**M**	
AuthResBaggage	acqCardMsg		**M**	**O**	
Results	acqCardMsg	**M**	**M**		
AcqCardMsgData	acqCardPhone	**M**		**O**	
AcqCardMsgData	acqCardText	**M**		**O**	
AcqCardMsgData	acqCardURL	**M**		**O**	
AcquirerID	acquirerBusinessID		**M**	**M**	**M**
SetPolicyQualifier	additionalPolicies	**O (15)**	**O (15)**	**O (15)**	**O (15)**
MerTermIDs	agentNum		**M**	**M**	
AuthValCodes	approvalCode		**M**	**M**	
AuthReqPayload	aRqExtensions		**O**	**M (1)**	
AuthResPayload	aRsExtensions		**M (1)**	**O**	
AuthRevReqData	aRvRqExtensions		**O**	**M (1)**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
AuthRevResData	aRvRsExtensions		**M (1)**	**O**	
AutoCharges	auditAdjustment		**O (10)**	**O**	
HotelCharges	auditAdjustment		**O (10)**	**O**	
AuthValCodes	authCharInd		**O (13)**	**O (13)**	
SaleDetail	authCharInd		**O (13)**	**M (13)**	
SignerInfo	authenticatedAttributes	**M**	**M**	**M**	**M**
AuthorityKeyIdentifier	authorityCertIssuer	**O (15)**	**O (15)**	**O (15)**	**O (15)**
AuthorityKeyIdentifier	authorityCertSerialNumber	**O (15)**	**O (15)**	**O (15)**	**O (15)**
AuthRevReqData	authReqData		**M**	**M**	
CapPayload	authReqItem		**M**	**M**	
CapPayload	authResPayload		**M**	**M**	
AuthRevReqData	authResPayload		**M**	**M**	
AuthResDataNew	authResPayloadNew		**M**	**M**	
AuthTags	authRetNum		**M (13)**	**M (13)**	
AuthRevTags	authRetNum		**M (13)**	**M (13)**	
CapItem	authRRPID		**M (14)**	**M**	
CapResItem	authRRPID		**M (14)**	**M**	
CapRevOrCredReqItem	authRRPID		**M (14)**	**M**	
CapRevOrCredResItem	authRRPID		**M (14)**	**M**	
TransactionDetail	authRRPID		**O**	**O**	
Results	authStatus	**M**	**M**		
PI	authToken		**M**	**M**	
AuthResBaggage	authToken		**M**	**M**	
AuthRevResBaggage	authTokenNew		**M**	**M**	
AuthTokenData	authTokenOpaque			**O**	
ResponseData	authValCodes		**M**	**M**	
MarketAutoCap	autoRateInfo		**O (10)**	**O**	
AutoCharges	autoTowingCharges		**O (10)**	**O**	
AuthReqPayload	avsData		**O (13)**	**O (13)**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
ResponseData	avsResult		**O (13)**	**O (13)**	
ErrorMsg	badWrapper	**M**	**M**	**M**	**M**
BatchAdminReqData	baRqExtensions		**O**	**M (1)**	
BatchAdminResData	baRsExtensions		**M (1)**	**O**	
BatchAdminResData	baStatus		**M**	**M**	
BatchStatus	batchExtensions		**M (1)**	**M (1)**	
CapResPayload	batchID		**O**	**O**	
CapRevOrCredResPayload	batchID		**O**	**O**	
BatchAdminReqData	batchID		**O**	**O**	
SaleDetail	batchID		**O**	**O**	
BatchAdminReqData	batchOperation		**M**	**M**	
CapResPayload	batchSequenceNum		**O**	**O**	
CapRevOrCredResPayload	batchSequenceNum		**O**	**O**	
SaleDetail	batchSequenceNum		**O**	**O**	
AuthHeader	batchStatus		**O**	**O**	
BatchAdminReqData	batchStatus		**O**	**O**	
BatchAdminResData	batchStatus		**O**	**O**	
CapResData	batchStatusSeq		**O**	**O**	
CapRevOrCredResData	batchStatusSeq		**O**	**O**	
BatchTotals	batchTotalExtensions		**M (1)**	**M (1)**	
BrandAndBIN	bin		**M**	**M**	
DirectoryString	bmpString	**M**	**M**	**M**	**M**
SETString	bmpString	**M**	**M**	**M**	**M**
BatchDetails	brandBatchDetailsSeq		**O**	**O**	
CardCInitResTBS	brandCRLIdentifier	**M**			**M**
Me-AqCInitResTBS	brandCRLIdentifier		**M**	**M**	**M**
RegFormTBS	brandCRLIdentifier	**M**			**M**
PInitResData	brandCRLIdentifier	**M**	**M**		
PResData	brandCRLIdentifier	**M**	**M**		
AuthResData	brandCRLIdentifier		**M**	**M**	
CapResData	brandCRLIdentifier		**M**	**M**	
AuthRevResData	brandCRLIdentifier		**M**	**M**	
CapRevOrCredResData	brandCRLIdentifier		**M**	**M**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
PCertResTBS	brandCRLIdentifierSeq		**M**	**M**	
Thumbs	brandCRLIdThumbs	**M (4,5)**	**M (4,5)**	**M (4,5)**	**M (4,5)**
ReturnTransactionDetail	brandID		**O**	**M**	
BatchAdminReqData	brandIDSeq		**O**	**O**	
CA-Msg	brandLogoURL	**M**			**M (8)**
RegTemplate	brandLogoURL	**M**	**M**	**M**	**M**
HotelCharges	businessCenterCharges		**O (10)**	**O**	
BasicConstraintsSyntax	cA	**M**	**M**	**M**	**M**
CertReqData	caBackKeyData	**M**			**M (8)**
RegFormTBS	caeThumb	**M**			**M**
CertStatus	caMsg	**M**			**M (8)**
AuthResPayload	capResPayload		**O**	**O**	
CapRevOrCredReqItem	capRevOrCredReqAmt		**O**	**O**	
Results	capStatus	**M**	**M**		
AuthResBaggage	capToken		**M (14)**	**O**	
AuthRevReqBaggage	capToken		**M (14)**	**O**	
AuthRevResBaggage	capTokenNew		**M (14)**	**O**	
AuthReqData	captureNow		**O**	**O (9)**	
CA-Msg	cardCurrency	**M**			**M (8)**
CA-Msg	cardholderMsg	**M**			**M (8)**
CA-Msg	cardLogoURL	**M**			**M (8)**
RegTemplate	cardLogoURL	**M**	**M**	**M**	**M**
AuthReqPayload	cardSuspect		**O**	**M (13)**	
ResponseData	cardType		**O**	**M**	
CertReqData	caTags2	**M**	**M**	**M**	**M**
SignedData	certificates	**M**	**M**	**M**	**M**
CertRes	certResTBS		**M**	**M**	**M (7)**
CertRes	certResTBSK	**M**			**M (8)**
Thumbs	certThumbs	**M (4,5)**	**M (4,5)**	**M (4,5)**	**M (4,5)**
CertResData	certThumbs	**M (5)**	**M (5)**	**M (5)**	**M (5)**
PCertResItemSeq	certThumb		**M**	**M**	
MerTermIDs	chainNum		**M**	**M**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
OIData	chall-M	**M**	**M**		
CommercialCardData	chargeInfo		**O (12)**	**O**	
AuthReqItem	checkDigests		**M**	**M**	
Location	city	**M**	**M**	**M**	**M**
MerNames	city	**O (15)**	**O (15)**	**O (15)**	**O (15)**
BatchStatus	closedWhen		**O**	**O**	
SaleDetail	commercialCardData		**O (12)**	**O**	
Item	commodityCode		**O (12)**	**O**	
ContentInfo	content	**M**	**M**	**M**	**M**
AutoRateInfo	corporateID		**O (10)**	**O**	
MerNames	countryName	**O (15)**	**O (15)**	**O (15)**	**O (15)**
CapPayload	cPayExtensions		**O**	**M (1)**	
Results	credStatus	**M**	**M**		
UnsignedBrandCRLIdentifier	crlIdentifierSeq	**M**	**M**	**M**	**M**
CRLEntry	crlEntryExtensions	**O**	**O**	**O**	**O**
UnsignedCertificateRevocationList	crlExtensions	**O (15)**	**O (15)**	**O (15)**	**O (15)**
SignedData	crls	**M**	**M**	**M**	**M**
Thumbs	crlThumbs	**M (4,5)**	**M (4,5)**	**M (4,5)**	**M (4,5)**
CapReqData	cRqExtensions		**O**	**M (1)**	
CapResData	cRsExtensions		**M (1)**	**O**	
CapResPayload	cRsPayExtensions		**M (1)**	**O**	
CapRevOrCredReqData	cRvRqExtensions		**O**	**M (1)**	
CapRevOrCredReqItem	cRvRqItemExtensions		**O**	**M (1)**	
CapRevOrCredResData	cRvRsExtensions		**M (1)**	**O**	
CapRevOrCredResPayload	cRvRsPayExtensions		**M (1)**	**O**	
AuthStatus	currConv		**M**	**M**	
ChargeInfo	custDutyTariffRef		**O (12)**	**O**	
SaleDetail	customerReferenceNumber		**O (12)**	**O**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
SaleDetail	customerServicePhone		**O (11)**	**M (13)**	
MarketHotelCap	customerServicePhone		**O (11)**	**M (13)**	
HotelRateInfo	dailyTaxRate		**O (10)**	**O**	
AutoCharges	deliveryCharges		**O (10)**	**O**	
TripLeg	departureTax		**O (10)**	**O**	
ATTRIBUTE	derivation	**M**	**M**	**M**	**M**
GeneralName	directoryName	**M**	**M**	**M**	**M**
Item	discountAmount		**O (12)**	**O**	
Item	discountInd		**O (12)**	**O**	
AutoRateInfo	distanceRate		**O (10)**	**O**	
Name	distinguishedName	**M**	**M**	**M**	**M**
GeneralName	dNSName	**O (15)**	**O (15)**	**O (15)**	**O (15)**
MarketHotelCap	durationOfStay		**O (10)**	**O**	
ChargeInfo	dutyTariffReference		**O (12)**	**O**	
CertStatus	eeMessage	**M**			**M (8)**
CertReqData	eeThumb	**O (4)**	**O (4)**	**O (4)**	**O (4)**
CertReq	enc		**M (6)**	**M (6)**	**M (7)**
AuthRevRes	enc		**M**	**M**	
CapToken	enc		**M**	**O**	
AuthRes	encB		**M**	**M**	
CapReq	encB		**M**	**M**	
AuthRevRes	encB		**M**	**M**	
CapRevReq	encB		**M**	**M**	
CredReq	encB		**M**	**M**	
CredRevReq	encB		**M**	**M**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
AuthRes	encBX		**M**	**M**	
CapReq	encBX		**M**	**M**	
CapRevReq	encBX		**M**	**M**	
CredReq	encBX		**M**	**M**	
CredRevReq	encBX		**M**	**M**	
EncryptedContentInfo	encryptedContent	**O**	**M**	**M**	**M**
CertReq	encx	**M**	**M (6)**	**M (6)**	**M**
CapToken	encX		**M**	**O**	
ErrorTBS	errorOID	**M**	**M**	**M**	**M**
ErrorTBS	errorThumb	**M**	**M**	**M**	**M**
AutoCharges	extraDistanceCharges		**O (10)**	**O**	
TripLeg	fareBasisCode		**O (10)**	**O**	
RegField	fieldDesc	**M**	**M**	**M**	**M**
RegField	fieldId	**M**	**M**	**M**	**M**
RegField	fieldInvisible	**M**	**M**	**M**	**M**
RegField	fieldLen	**M**	**M**	**M**	**M**
RegField	fieldRequired	**M**	**M**	**M**	**M**
HotelCharges	folioCashAdvances		**O (10)**	**O**	
MarketHotelCap	folioNumber		**O (10)**	**O**	
HotelCharges	foodBeverageCharges		**O (10)**	**O**	
AutoRateInfo	freeDistance		**O (10)**	**O**	
AutoCharges	fuelCharges		**O (10)**	**O**	
HotelCharges	giftShopPurchases		**O (10)**	**O**	
AuthResData	peThumb		**M**	**M**	
CapResData	peThumb		**M**	**M**	
AuthRevResData	peThumb		**M**	**M**	
CapRevOrCredResData	peThumb		**M**	**M**	
HotelCharges	healthClubCharges		**O (10)**	**O**	
MarketHotelCap	hotelRateInfo		**O (10)**	**O**	
CertReqData	idData		**M**	**M**	**M (7)**

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
InqReq	inqReqSigned	**M**	**M**		
InqReq	inqReqUnsigned	**O**	**M**		
InqReqData	inqRqExtensions	**O**	**M (1)**		
PIHead	installRecurData	**M**		**M**	
AuthReqPayload	installRecurData		**O**	**M**	
AuthTokenData	installRecurData			**M**	
HODInput	installRecurData	**O**	**O**	**O**	
InstallRecurInd	installTotalTrans	**M**	**O**	**M**	
AutoCharges	insuranceCharges		**O (10)**	**O**	
MarketAutoCap	insuranceType		**O (10)**	**O**	
InstallRecurData	irExtensions	**O**	**O**	**M (1)**	
CommercialCardData	itemSeq		**O (12)**	**O**	
MerNames	language	**M**	**M**	**M**	**M**
AutoCharges	lateReturnCharges		**O (10)**	**O**	
AutoRateInfo	lateReturnHourlyRate		**O (10)**	**O**	
HotelCharges	laundryCharges		**O (10)**	**O**	
CA-Tags	localID-CA	**M (2)**	**M (2)**	**M (2)**	**M (2)**
MessageIDs	localID-C	**O**	**M (2)**	**M (2)**	**M (2)**
MessageIDs	localID-M	**M (2)**	**O**	**M (2)**	**M (2)**
TransIDs	localID-M	**M (2)**	**O**	**M (2)**	**M (2)**
PInitReq	localID-M	**M (2)**	**O**		
Item	localTaxAmount		**O (12)**	**O**	
Location	locationID	**M**	**M**	**M**	**M**
ResponseData	logRefID		**M**	**M**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
AuthValCodes	marketSpec		**O**	**O**	
MarketSpecSaleData	marketSpecCapData		**O (10)**	**O**	
AuthReqPayload	marketSpecAuthData		**O (10)**	**O**	
MarketSpecSaleData	marketSpecDataID		**O (10)**	**O**	
SaleDetail	marketSpecSaleData		**O (10)**	**O**	
MerchantDataSyntax	merAuthFlag		**M**	**M**	**M**
CommercialCardData	merchantLocation		**O (12)**	**O**	
ChargeInfo	merchantTaxID		**O (12)**	**O**	
MerchData	merchCatCode		**M**	**M**	
MerchData	merchGroup		**M**	**M**	
ChargeInfo	merDutyTariffRef		**O (12)**	**O**	
SaleDetail	merOrderNum		**O (11)**	**M**	
ChargeInfo	merType		**O (12)**	**O**	
ErrorMsg	messageHeader	**M**	**M**	**M**	**M**
MessageHeader	messageIDs	**M**	**M**	**M**	**M**
HotelCharges	miniBarCharges		**O (10)**	**O**	
HotelCharges	movieCharges		**O (10)**	**O**	
AuthReqData	mThumbs		**O (4)**	**O (4)**	
CapReqData	mThumbs		**O (4)**	**O (4)**	
AuthRevReqData	mThumbs		**O (4)**	**O (4)**	
CapRevOrCredReqData	mThumbs		**O (4)**	**O (4)**	
PCertReqData	mThumbs		**O (4)**	**O (4)**	
MessageWrapper	mwextension	**M (1)**	**M (1)**	**M (1)**	**M (1)**
MerNames	name		**M**	**M**	**M**
Item	nationalTaxAmount		**O (12)**	**O**	
Item	nationalTaxRate		**O (12)**	**O**	
Item	nationalTaxType		**O (12)**	**O**	
Item	netCost		**O (12)**	**O**	
CapRevOrCredReqItem	newBatchID		**O**	**O**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
CertStatus	nonce-CCA	**M**			**M (8)**
MarketAutoCap	autoNoShow		**O (10)**	**O**	
MarketHotelCap	hotelNoShow		**O (10)**	**O**	
PrivateKeyUsagePeriod	notAfter	**O (15)**	**O (15)**	**O (15)**	**M**
PrivateKeyUsagePeriod	notBefore	**O (15)**	**O (15)**	**O (15)**	**M**
CapToken	null		**M**	**O**	
HODInput	odExtensions	**O**	**O**	**O**	
OIData	odExtOIDs	**O**	**M (1)**		
OIData	oiExtensions	**O**	**M (1)**		
SaleDetail	okToPrintPhoneInd		**O (11)**	**M (13)**	
AutoCharges	oneWayDropOffCharges		**O (10)**	**O**	
SaleDetail	orderSummary		**O (11)**	**M (13)**	
AutoCharges	otherCharges		**O (10)**	**O**	
HotelCharges	otherCharges		**O (10)**	**O**	
Item	otherTaxAmount		**O (12)**	**O**	
AcctInfo	panData0	**M**			**M (8)**
AlgorithmIdentifier	parameters	**M**	**M**	**M**	**M**
AutoCharges	parkingCharges		**O (10)**	**O**	
HotelCharges	parkingCharges		**O (10)**	**O**	
BasicConstraintsSyntax	pathLenConstraint	**M**	**M**	**M**	**M**
SaleDetail	payRecurInd		**O (13)**	**M (13)**	
TransIDs	paySysID	**M (2)**	**M (2)**	**M (2)**	**M (2)**
PCertReqData	pcRqExtensions		**O**	**M (1)**	
PCertResTBS	pcRsExtensions		**M (1)**	**O**	
PI	piDualSigned	**M**	**M**	**M**	
PIHead	piExtensions	**O**		**M (1)**	
PInitReq	piRqExtensions	**O**	**M (1)**		
PInitResData	piRsExtensions	**M (1)**	**M (1)**		

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
PI	piUnsigned	**O**	**M**	**M**	
SETQualifier	policyDigest	**O (15)**	**O (15)**	**O (15)**	**O (15)**
SETQualifier	policyEmail	**O (15)**	**O (15)**	**O (15)**	**O (15)**
AdditionalPolicy	policyOID	**O (15)**	**O (15)**	**O (15)**	**O (15)**
AdditionalPolicy	policyQualifier	**O (15)**	**O (15)**	**O (15)**	**O (15)**
PolicyInformation	policyQualifiers	**O (15)**	**O (15)**	**O (15)**	**O (15)**
SETQualifier	policyURL	**O (15)**	**O (15)**	**O (15)**	**O (15)**
Location	postalCode	**M**	**M**	**M**	**M**
MerNames	postalCode	**O**	**O**	**O**	**O**
HotelCharges	prepaidExpenses		**O (10)**	**O**	
PReq	pReqDualSigned	**M**	**M**		
PReq	pReqUnsigned	**O**	**M**		
MarketHotelAuth	prestige		**O (10)**	**O**	
DirectoryString	printableString	**M**	**M**	**M**	**M**
Item	productCode		**O (12)**	**O**	
MarketHotelCap	programCode		**O (10)**	**O**	
MarketHotelCap	propertyPhone		**O (10)**	**O**	
PResPayload	pRsExtensions		**M (1)**	**O**	
PublicKeySorE	publicKeyE		**M**	**M**	**M (7)**
PublicKeySorE	publicKeyS	**M**	**M**	**M**	**M**
PolicyQualifierInfo	qualifier	**O (15)**	**O (15)**	**O (15)**	**O (15)**
Item	quantity		**O (12)**	**O**	
ReferralData	reason	**M**	**M**	**M**	**M**
InstallRecurInd	recurring	**M**	**O**	**M**	
AuthTokenData	recurringCount			**M**	
MarketAutoCap	referenceNumber		**O (10)**	**O**	
ReferralData	referralURLSeq	**M**	**M**	**M**	**M**
GeneralName	registeredID	**O (15)**	**O (15)**	**O (15)**	**O (15)**
TransactionDetail	reimbursementID		**O**	**O**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
MarketAutoCap	rentalAgreementNumber		**O (10)**	**O**	
MarketAutoCap	rentalLocation		**O (10)**	**O**	
MarketAutoCap	renterName		**O (10)**	**O**	
ResponseData	respReason		**M**	**M**	
MarketTransportCap	restrictions		**O (10)**	**O**	
PResPayload	results		**M**	**M**	
BatchAdminReqData	returnBatchSummaryInd		**O**	**O**	
MarketAutoCap	returnLocation		**O (10)**	**O**	
BatchAdminReqData	returnTransactionDetail		**O**	**O**	
UnsignedCertificateRevocationList	revokedCertificates	**M**	**M**	**M**	**M**
HotelCharges	roomServiceCharges		**O (10)**	**O**	
HotelCharges	roomTax		**O (10)**	**O**	
MessageHeader	rrpid	**M**	**M**	**M**	**M**
AuthReqData	saleDetail		**O (10)**	**M**	
CapPayload	saleDetail		**O (10)**	**M**	
SaleDetail	saleExtensions		**O**	**M (1)**	
BatchAdminResData	settlementInfo		**O**	**O**	
CommercialCardData	shipFrom		**O (12)**	**O**	
CommercialCardData	shipTo		**O (12)**	**O**	
Error	signedError	**M (3)**	**M (3)**	**M (3)**	**M (3)**
AuthReqPayload	specialProcessing		**O (13)**	**O (13)**	
Location	stateProvince	**M**	**M**	**M**	**M**
MerNames	stateProvince	**M**	**M**	**M**	**M**
MerTermIDs	storeNum		**M**	**M**	
AVSData	streetAddress		**O (13)**	**O (13)**	
AuthReqPayload	subsequentAuthInd		**M**	**M**	
ChargeInfo	summaryCommodityCode		**O (12)**	**O**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
AutoCharges	telephoneCharges		**O (10)**	**O**	
HotelCharges	telephoneCharges		**O (10)**	**O**	
MerTermIDs	terminalID		**M**	**M**	
SETQualifier	terseStatement	**O (15)**	**O (15)**	**O (15)**	**O (15)**
CardCInitReq	thumbs	**O (4)**			**O (4)**
CardCInitResTBS	thumbs	**M (5)**			**M (5)**
Me-AqCInitReq	thumbs		**O (4)**	**O (4)**	**O (4)**
Me-AqCInitResTBS	thumbs		**M (5)**	**M (5)**	**M (5)**
RegFormData	thumbs	**O (4)**			**O (4)**
RegFormTBS	thumbs	**M (5)**			**M (5)**
CertReqData	thumbs	**O (4)**	**O (4)**	**O (4)**	**O (4)**
CertResData	thumbs	**M (5)**	**M (5)**	**M (5)**	**M (5)**
PInitReq	thumbs	**O (4)**	**O (4)**		
PInitResData	thumbs	**M (5)**	**M (5)**		
MarketTransportCap	ticketNumber		**O (10)**	**O**	
CapTokenData	tokenOpaque			**O**	
AutoCharges	totalDistance		**O (10)**	**O**	
ChargeInfo	totaldutyTariffAmount		**O (12)**	**O**	
ChargeInfo	totalfreightShippingAmount		**O (12)**	**O**	
ChargeInfo	totalLocalTaxAmount		**O (12)**	**O**	
ChargeInfo	totalNationalTaxAmount		**O (12)**	**O**	
ChargeInfo	totalOtherTaxAmount		**O (12)**	**O**	
AutoCharges	totalTaxAmount		**O (10)**	**O**	
HotelCharges	totalTaxAmount		**O (10)**	**O**	
ChargeInfo	totalTaxAmount		**O (12)**	**O**	
TransactionDetail	transactionStatus		**O**	**O**	
BatchAdminReqData	transDetails		**O**	**O**	
BatchAdminResData	transDetails		**O**	**O**	
TransactionDetail	transExtensions		**O**	**O**	
BatchAdminResData	transmissionStatus		**O**	**O**	

Continued on next page

Support Requirements, continued

Field Support Summary (continued)

Message/Data Structure	Field/Option	Cardholder	Merchant	Payment Gateway	CA
MarketTransportCap	travelAgencyCode		**O (10)**	**O**	
MarketTransportCap	travelAgencyName		**O (10)**	**O**	
MarketTransportCap	tripLegs		**O (10)**	**O**	
TunnelingSyntax	tunneling	**M**	**M**	**M**	**M**
ATTRIBUTE	Type	**M**	**M**	**M**	**M**
GeneralName	uniformResourceIdentifier	**O (15)**	**O (15)**	**O (15)**	**M**
Item	unitCost		**O (12)**	**O**	
Item	unitOfMeasureCode		**O (12)**	**O**	
Error	unsignedError	**M**	**M**	**M**	**O**
AuthValCodes	validationCode		**O (13)**	**O (13)**	
AutoRateInfo	vehicleClassCode		**O (10)**	**O**	
AutoCharges	violationsCharges		**O (10)**	**O**	
SETString	visibleString	**M**	**M**	**M**	**M**
MessageIDs	xID	**M**	**M**	**M**	**M**

Appendix F
Logo Display During Certificate Registration

Overview

Optional logos

The certificate registration process includes optional logos for the brand and the financial institution.

To promote SET's interoperability, it is suggested that all implementations support at least the GIF format for logos.

For URLs referencing textual information (for example, policy, instructions), the URL should be constrained to be in the natural-language character set so that it can be directly displayed to the cardholder.

Subsequent versions of the SET specification may be moving towards supporting a broader range of formats over time as business needs and technologies evolve.

Contents

This appendix includes the following information:

Logo specifications	Explains standard logo sizes, the color palette, and how to specify and display a logo.
Sample logos	Provides examples of the five standard logo sizes.

Logo Specifications

Logo sizes

Five standard sizes for logos have been defined. The sizes in pixels and the corresponding file names are given in the following table:

Size in pixels	Name	File name
32 x 32 or 32 x 20	Extra Small	exsmall.gif
53 x 33	Small	small.gif
103 x 65	Medium	medium.gif
180 x 114	Large	large.gif
263 x 166	Extra Large	exlarge.gif

Table 77: Logo Sizes

To maintain a constant ratio, designers of an extra small logo may choose to limit the size to 32 x 20 pixels. Sample logos of these sizes appear on the following page.

Color palette

The color palette of the logos needs to be designed to work within the limited palette of colors typically used by most browsers within an 8-bit or 256-color environment or greater (the "Netscape palette").

Specifying a logo

Each logo URL specified with the registration form will either be:

- The name of a directory (ending with a virgule "/") that contains one or more logos named according to the table above; at least one of the following files must appear in the directory: small.gif, medium.gif,

  or

- The name of a file containing a *Small* or *Medium* logo; the name may be different than those shown in the table; however, if one of these names is used, the size of the icon and the file name must be consistent with those shown in the table.

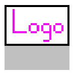
Displaying a logo

Applications that display logos on the registration form must reserve space for both brand and financial institution logos; at a minimum, the registration form must be able to support *Small* or *Medium* logos. The ability to display logos of larger sizes is at the discretion of the software vendor.

Note: *Extra Small* logos will never be displayed on a registration form.

Sample Logos

exsmall.gif

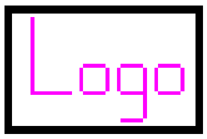


The *Extra Small* logo is the same size as a Windows icon. Unlike the other logos, it is square. To account for this difference, the sample icon shown here has the logo sized to 32 x 20 pixels, with the portion beneath the logo shaded gray. This size logo is only used by brands.

small.gif

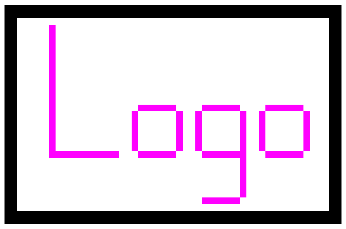
This is a sample *Small* logo.

medium.gif



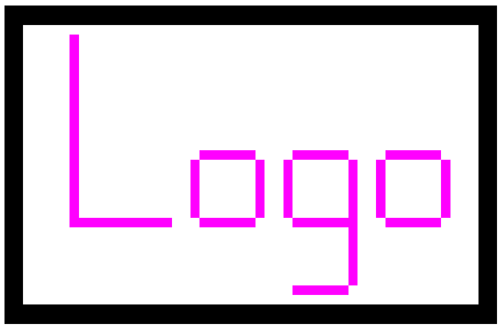
This is a sample *Medium* logo.

large.gif



This is a sample *Large* logo.

exlarge.gif



This is a sample *Extra Large* logo.

Appendix G
Object Identifiers

Introduction

Purpose

Object identifiers are used in SET to uniquely identify a particular type of object. Several of the object identifiers are standard values used throughout the industry for identifying the type of cryptographic functions being used to encapsulate a message. Other object identifiers are standard values used to uniquely an extension in a certificate. SET defines several new object identifiers..

Organization

This appendix summarizes the types of object identifiers used by SET into the following classes:

- Algorithm
- Content
- Extension
- Attribute
- ASN.1
- External

OID as SET field values

The following table summaries the fields that are defined solely for the purpose of exchanging an object identifier as its value within a SET message, exclusive of the PKCS message formatting.

Field	Usage
ErrorOID	Identify the specific object (extension, content type, attribute, etc) that caused the error condition
FieldID	Identify the specific field during certificate registration
PolicyOID	Identify the specific brand certificate policy qualifier

Algorithm

Algorithm OIDs

The following table lists the object identifiers used to uniquely identify the cryptographic operator being applied to a SET message (or data structure within a message). There are no algorithm OIDs defined specifically for SET.

Object Identifier	Usage
rsaOAEPEncryptionSET	Public key encryption RSA with OAEP for SET
id-rsaEncryption	Public key encryption using RSA
id-sha1-with-rsa-signature	Digital signature using SHA-1 as digest function
id-sha1	Digest function using SHA-1
id-desCBC	Symmetric key encryption using DES with CBC mode
id-desCDMF	Symmetric key encryption using CDMF

Content

Content OIDs

The following table lists the object identifiers used to uniquely identify the general class for the content types used in a SET message (or data structure within a message). The OIDs defined under {id-set-content} are used to further refine the specific content type of SET message or data structure. A summary of the SET related content type extensions is provided in "ContentTypes" on page 512.

Object Identifer	Usage
data	PKCS data content
signedData	PKCS signed data content
envelopedData	PKCS enveloped data content
digestedData	PKCS digested data content
encryptedData	PKCS encrypted data content
messageDigest	PKCS message digest content
contentType	PKCS content type

Extension

Extension OIDs

The following table lists the object identifiers used to uniquely identify the general class for the certificate extensions used in a SET message (or data structure within a message). Each of the OIDs in this table are defined relative to {id-ce}. OIDs under {id-set-cert} are defined as private extensions specific for SET. In addition, a SET message and data structure extension mechanism is provided under {id-set-msgExt} to facilitate supporting payment card brand and nation business requirements. A summary of the SET message and data structures is provided in "Object Identifiers under {id-set}" on page 499.

Object Identifer	Usage
id-ce-subjectKeyIdentifier	X.509 certificate subject key identifier
id-ce-keyUsage	X.509 certificate key usage
id-ce-privateKeyUsagePeriod	X.509 certificate private key usage period
id-ce-subjectAltName	X.509 certificate subject alternate name
id-ce-issuerAltName	X.509 certificate issuer alternate name
id-ce-basicConstraints	X.509 certificate basic constraints
id-ce-cRLNumber	X.509 certificate revocation list number
id-ce-certificatePolicies	X.509 certificate policies
id-ce-authorityKeyIdentifier	X.509 certificate authority key identifier
id-set-hashedRootKey	X.509 SET hash of next generation root key
id-set-certificateType	X.509 SET certificate type
id-set-merchantData	X.509 SET Merchant identification
id-set-cardCertRequired	X.509 SET Cardholder certificate required
id-set-tunneling	X.509 SET Acquirer -to-Cardholder tunneling message
id-set-setExtensions	X.509 SET message extensions support
id-set-setQualifier	X.509 SET policy qualifier

Attribute

Attribute OIDs

The following table lists the object identifiers used to uniquely identify the general class for the attribute types used in a SET message (or data structure within a message).

Object Identifer	Usage
id-set-attribute-cert	SET certificate
id-set-rootKeyThumb	SET root certificate thumbprint
id-set-additionalPolicy	SET additional policy
id-at-commonName	X.509 relative distinguished common name
id-at-countryName	X.509 relative distinguished country name
id-at-organizationName	X.509 relative distinguished organizational name
id-at-organizationalUnitName	X.509 relative distinguished origanizational unit name

ASN.1

ASN.1 OIDs

The following table lists the object identifiers used to uniquely identify each of the ASN.1 modules for SET under the {id-set-module}. These OIDs are for informational purposes.

Object Identifer	Usage
SetMessage	SET common types
SetCertMsgs	SET certificate message types
SetPayMsgs	SET payment message types
SetCertificate	SET CRL and X.509 certificate support types
SetCertificateExtensions	SET X.509 certificate extension support types
SetCRL	SET CRL support types
SetPKCS7Plus	SET PKCS #7 support types
SetAttribute	SET attribute support types
SetMarketData	SET market-specific data support types
SetPKCS10	SET PKCS #10 support types

External

External OIDs

The following table lists the object identifiers used to uniquely identify the higher-level OIDs referenced by SET and previously established by other organizations.

Object Identifer	Usage
secsig	Security functions
pkcs-1	PKCS-1
pkcs-7	PKCS-7
pkcs-9	PKCS-9
ds	Directory service
id-at	Distinguished name class

Appendix H
Extension Mechanism for SET Messages

Overview

Explanation

The scope of SET for Version 1 was intentionally limited to the minimum functionality necessary to support cardholders and merchants doing business on the Internet. Consequently, some business functions are not included in the definition of SET payment messages.

Furthermore, it is unlikely that SET could ever be robust enough to cover the business practices of every national market and every Acquirer. Therefore, it is necessary to provide a mechanism to extend SET payment messages.

An example of a business function that is not supported by the SET messages is Japanese payment options. Issuers in Japan have options for payment that are selected by the consumer at the time of the purchase. Since there is no place in the SET message to carry this information, an extension to the protocol is necessary.

Mechanism

The mechanism used to extend SET messages parallels the way that X.509 certificates are extended.

Impacted data structures

The data structures listed below have an extensions field.

Note: The extensions field is always the last item in a data structure.

MessageWrapper	**CapReqData**	**PCertReqData**
PInitReq	**CapPayload**	**PCertResTBS**
PInitResData	**CapResData**	**BatchAdminReqData**
PIHead	**CapResPayload**	**BatchAdminResData**
InstallRecurData	**CapRevOrCredReqData**	**BatchStatus**
OIData	**CapRevOrCredReqItem**	**BatchTotals**
HODInput	**CapRevOrCredResData**	**TransactionDetail**
PResPayload	**CapRevOrCredResPayload**	
InqReqData		
AuthReqPayload		
SaleDetail		
AuthResPayload		
AuthRevReqData		
AuthRevResData		

Extension Components

Contents of an extension

An extension will consist of three components:

object identifier	Must uniquely identify the extension. This identification will permit SET applications to recognize an extension and process the data contained within it
criticality flag	Indicates whether the recipient shall understand the extension to process the message containing the extension.
data	Provides the additional business information that necessitated the definition of the extension. The layout of the data (that is, the ASN.1 code) will be defined by the organization creating the extension

Criticality flag

The criticality flag is a boolean value. An extension is considered critical if this value is TRUE; otherwise, the extension is considered non-critical.

Note: The default value is FALSE.

When an extension is critical, recipients of the message must recognize and be able to process the extension.

When an extension is non-critical, recipients that cannot process the extension can safely ignore the data in the.

Information object sets

An information object set has been created for each data structure that can contain an extension. These sets define the extensions that are permitted to be processed in the data structure. Each of these sets ends with an ASN.1 extension marker ("..."). The extension marker indicates that additional extensions may be defined that the application does not recognize.

Application developers who support specific extensions should add the list of supported extensions after the extension marker.

Note: A comma should separate the extension marker from the information objects that follow.

Minimum Extension Support

Minimum implementation requirements

SET applications will fit into one of two categories:

- the application does not support any message extensions, or
- the application supports some selected set of message extensions.

An application that does not support any message extensions shall recognize the presence of an extension in a message and examine the criticality flag. If the extension is critical, the application must not process the message and shall reject it with an error code of *unrecognizedExtension*. If the extension is non-critical, the application can safely ignore the data in the extension and proceed to process the remainder of the message.

An application that supports message extensions shall recognize the presence of an extension in a message and process the extension as follows:

- If the application supports the object identifier for this message, the data within the extension is processed.
- If the application does not support the object identifier for this message and the extension is critical, the application shall not process the message and shall reject it with an error code of *unrecognizedExtension*.
- If the application does not support the object identifier for this message and the extension is non-critical, the application can safely ignore the data in the extension and proceed to process the remainder of the message.

Defining Extensions

Requirements for defining an extension

Organizations defining an extension shall register the object identifier and the data content of the extension with MasterCard and Visa (or their designee) prior to deploying software that transmits messages (including test messages) over an open network. The data in the extension shall conform to the requirements described below.

- Data fields that are defined to appear within the data block (DB) of the OAEP block shall never appear within the data of any extension. Note: While it is never appropriate to put the PAN into the definition of an extension, it is permissible to put the first six digits of the PAN, the Bank Identification Number (BIN), into the data of an extension.
- No fields may be added to the DB of the OAEP block and no new values may be defined for the block contents byte (BC).
- Data that is always encrypted by the protocol shall not appear in an unencrypted form in an extension.

Unencrypted extensions

Financial information must be protected. For example, transaction amounts are always encrypted and shall not appear unencrypted in an extension.

The following data structures contain extensions that are not encrypted:

- **MessageWrappe**r
- **PInitReq**
- **PInitResData**
- **OIData**
- **PResPayload**
- **InqReqData**
- **PCertReqData**
- **PCertResTBS**

Approval of extensions

MasterCard and Visa will be the final arbiters of whether a given data element may appear in its unencrypted form within a message.

MasterCard and Visa reserve the right to disapprove the use of any extension that contains data that may compromise the integrity of the SET protocol. For example, any extension that contains more than six digits of the PAN will not be approved for use in a SET message.

Continued on next page

Defining Extensions, continued

Export and import approval

Software vendors are responsible for obtaining export and import approval for any data added to a SET message (using extensions) by their application.

Organizations defining an extension should consider export and import requirements when defining the data that appears within an extension. The guidelines that appear later in this section provide suggestions on the content of extensions; these suggestions take export and import requirements into account.

Example

Under Martian law, a merchant on the planet Mars must send the shipment weight for orders being shipped off-planet to the Acquirer, who must authorize the shipment with the Martian Shipping Authority. To support this additional data, Martian software developers might get together and define an extension.

Defining the extension

The definition of the extension consists of the three components: object identifier (id-mars-data), criticality flag (TRUE), and syntax for the data (MarsData).

```
marsData EXTENSION ::= {
   SYNTAX MarsData
   CRITICAL TRUE
   IDENTIFIED BY id-mars-data
}
```

The symbol *marsData* is an ASN.1 identifier that is the name of the extension. The symbol *MarsData,* which is an ASN.1 type reference that provides the data layout, is described below.

Identifying the extension

The object identifier, id-mars-data, is a series of integer numbers. Each number in the series provides a more specific designation.

```
id-mars-data  OBJECT IDENTIFIER ::=
   {uso(3) member-planet(0) mars(4) set-def(12) marsData(0)}
```

The first value designates the fictitious "Universal Standards Organization" (USO).

The second value designates that what follows is assigned to a member planet of the USO.

The third value designates the fourth planet, Mars.

The fourth value designates data defined for use within SET by Mars.

The fifth value designates the marsData extension.

Specifying the criticality

The Martian software developers have determined that their extension is critical. In other words, if the payment gateway does not understand and therefore cannot process the data within the extension, the message should be rejected.

Continued on next page

Defining Extensions, continued

Defining the data syntax

The Martian software developers have defined the data syntax for the marsData extension as follows:

```
MarsData ::= SEQUENCE {
   offPlanet    BOOLEAN,
   shippingInfo MarsShippingInfo OPTIONAL
}

MarsShippingInfo ::= SEQUENCE {
   weight       INTEGER,     -- Weight in kilograms
   planet       INTEGER,     -- Destination using USO-3166
                             -- planetary codes
   shipperID    OBJECT IDENTIFIER OPTIONAL
}
```

Updating the information object sets

The Martian software developers have determined that the marsData extension can appear in the authorization request and the capture request. This results in the following information object set definitions:

```
ARqExtensionsIOS EXTENSION ::= {
   ... ,
   marsData
}

CPayExtensionsIOS EXTENSION ::= {
   ... ,
   marsData
}
```

Note that locally-defined extensions appear after the extension marker "..." and are separated from it by a comma.

Continued on next page

Defining Extensions, continued

ASN.1 extension marker "..."

The inclusion of the marsData extension in the information object sets for the authorization and capture requests indicates that the extension is only intended to appear in those data structures. However, since all of the information object sets defined in the specification contain the ASN.1 extension marker "...", the marsData extension could in fact appear in any data structure. Nonetheless, adding the extension to the information object sets makes the intent clear.

The marker allows each SET application to understand different sets of objects and still interoperate. The marker indicates that each application should expect to decode objects that it does not understand. After decoding an object, such as marsData, the application must decide how to proceed.

Once an extension is decoded, the application determines whether or not it recognizes the object. If processing a capture request, it must determine if the object identifier is in the definition of CPayExtensionsIOS. If the application recognizes the object, normal processing continues.

If the application does not recognize the object and the criticality flag is TRUE, then the application shall not process the message and shall reject it with an error code of *unrecognizedExtension*. Otherwise, the information in the extension is ignored and normal processing of the message continues.

Appendix K
Object Identifiers under {id-set}

Introduction

Explanation

ISO and ITU-T have recently adopted ISO/IEC 9834-7 and ITU-T X.666, which provides for an international registration authority (RA). The arc for this RA is {joint-iso-itu-t (2) internationalRA (23)}.

MasterCard and Visa will apply for an object identifier for use by SET applications under this arc as soon as the registration authority is named. The ultimate definition of {id-set} will be of the form:

```
id-set OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
          internationalRA(23) set(42)}
```

Organization

This appendix addresses the following topics:

- OID Usage
- OID Management

OID Usage

Arc division

The {id-set} arc has been divided as follows with usage of these values described in Table 78 on page 501.

0	content type
1	message extension
2	field
3	attribute
4	algorithm
5	policy
6	module
7	certificate extension
8	brand
9	vendor
10	national market

Continued on next page

OID Usage, continued

Arc division (continued)

Symbol	Value	Used to Identify
content type	{id-set contentType (0)}	Data elements that are globally defined for use by SET applications
message extension	{id-set msgExt (1)}	Message extensions that are globally defined for use by SET applications
field	{id-set field (2)}	Registration form fields that are globally defined for use by SET applications
attribute	{id-set attribute (3)}	Attributes that are globally defined for use by SET applications
algorithm	{id-set algorithm (4)}	Algorithms that are globally defined for use by SET applications (a)
policy	{id-set policy (5)}	Policies that are globally defined for use by SET applications
module	{id-set module (6)}	SET ASN.1 modules
certificate extension	{id-set certExt (7)}	Certificate extensions that are globally defined for use by SET applications
brand	{id-set brand (8)}	Payment card brands(b)
vendor	{id-set vendor (9)}	SET application vendors(c)
national market	{id-set national (10)}	National market information(d)

Table 78: Arc Division for {id-set} OID

(a) All algorithms currently used by SET have externally defined object identifiers.

(b) Each payment card brand manages the assignment of OIDs beneath its brand OID. It is strongly recommended that the next level under the brand OID follow the conventions described below under Private OID management.

The assignment of OID values is roughly based on the values assigned to the brand by ISO using the shortest BIN prefix that is uniquely assigned to the brand.

(c) Each vendor manages the assignment of OIDs beneath its vendor OID. It is strongly recommended that the next level under the vendor OID follow the conventions described below under Private OID management.

The assignment of vendor OID values is made on a first come-first served basis.

(d) Each national market manages the assignment of OIDs beneath its national market OID. It is strongly recommended that the next level under the national market OID follow the conventions described below under Private OID management.

The assignment of national market OID values corresponds to ISO-3166 numeric codes.

OID Management

Private OID management

Each payment brand, vendor and national market manages its own arc. However it is strongly recommended that the next level in that arc follow the same conventions as followed by the {id-set} arc:

0	content type
1	message extension
2	field
3	attribute
4	algorithm
5	policy
6	module

Private OID restrictions

Private OID extensions may be defined for use in the appropriate extension fields of a SET message. Furthermore, no brand, vendor, or national market is permitted to define new certificate extensions for use within SET.

Obtaining an OID

The current Registration Authority for {id-set} is Tony Lewis. To request that an OID be assigned under any of the arcs for {id-set}, send an e-mail message to tlewis@visa.com with the following information:

- Contact name, electronic mail address, telephone and facsimile numbers
- Organization's name and address
- Usage of OID (for example, company identification, algorithm identification, policy identification, etc.)
- Plans for assignment of OIDs under the requested arc (for example, if the OID is to identify a company, whether lower-level OIDs will be created to identify fields, policies, etc.)

Appendix L
Object Identifiers for Registration Form Fields

Introduction

Explanation

An object identifier under {id-set} has been assigned to provide an arc for the data content associated with registration form fields. The definition of this identifier is:

```
id-set-field OBJECT IDENTIFIER ::= {id-set 23}
```

Organization

This appendix addresses the following topics:

- application usage
- field types

Application Usage

Options

An application can take the following optional behaviors, which are described in more detail below:

- constrain the data in the field (such as limiting the characters to those that can be entered for the type of field);
- retain the information in a database (such as the cardholder wallet) for use on future registration attempts;
- pre-fill the information on the registration form based on responses provided by the user during prior registrations.

Constraints on the data in the field

If the application recognizes the object identifier for a registration form field, it can constrain the user's input. For example, it can warn the user when the field contains characters that it believes to be invalid for the field—such as a field for telephone numbers limited to numbers, spaces, hyphens, and parentheses.

The user should always be allowed to submit the registration form, even if the application detects what it believes to be an input error. This capability is necessary because the application's edits might otherwise prevent a valid registration form from being submitted. For example, if the application limits names to alphabetic characters, then users whose names are represented using Chinese characters would not be able to submit registration forms.

Continued on next page

Application Usage, continued

Retain the information

An application may choose to retain the information from the registration form in a database (such as the cardholder wallet) for use on future registration attempts. All of the information in a registration form shall be considered sensitive and protected with at least the same level of protection as the private keys and account information stored in the database.

The application shall provide the user with the ability to:

- prevent this information from being written to the database, and
- purge this information from the database.

The application can store information even for unrecognized object identifiers. For example, an application that does not recognize the specific object identifiers for home address and billing address can still store the information using the object identifier as the key for retrieval.

Some information, such as the last payment amount, will age quickly and may be of no practical use to the user even if the application retains it.

Pre-fill the information

The application can pre-fill the fields in the registration form when it recognizes the object identifier and can retrieve the appropriate information. For example, if the cardholder has customized their wallet with name and address information, the application can pre-fill these fields.

The application can also retrieve information for prior registration requests, even if it is not otherwise able to process the field. For example, an application may not know the format of a passport number for a given country, but it can still pre-fill the information if it has processed the same object identifier in a prior registration form.

The application may be able to retrieve information from another application. For example, the last payment amount may be available from a personal finance program.

Field Types

Types of fields

The following types of fields, each represented as {id-set-field *number*}, have been identified:

0	full name
1	given name (or first name)
2	family name (or surname or last name)
3	family name at birth (or maiden name)
4	place name
5	identification number
6	month
7	date
8	address
9	telephone
10	amount
11	account number
12	pass phrase

Table 79 on page 510 shows the types of fields, and for each the OID, a description, and additional qualifiers. If used, the qualifiers must appear sequentially in the same order as they appear in the table (where they are separated by a rule). Use of one or more qualifiers is optional, but each qualifier, if included, must be in the specified sequence. Skipping one additional qualifier and including the rest is not permitted.

Note: Additional types of fields and qualifiers may be added to this list at any time.

Continued on next page

Field Types, continued

Field Type/OID	Description	Additional Qualifiers
full name {id-set field fullName (0)}	Indicates a field that contains a person's given name and family name	0 primary cardholder 1 secondary cardholder[(a)]
given name (or first name) {id-set-field givenName (1)}	Indicates a field that contains a person's given name	0 primary cardholder 1 secondary cardholder[(a)]
family name (or surname or last name) {id-set-field familyName (2)}	Indicates a field that contains an individual's family name	0 primary cardholder 1 secondary cardholder[(a)]
family name at birth (or maiden name) {id-set-field birthFamilyName (3)}	Indicates a field that contains an individual's family name at birth (for example, maiden name)	0 primary cardholder 1 secondary cardholder[(a)]
place name {id-set-field placeName (4)}	Indicates a field that contains the name of a place	To identify the specifics of the place: 0 birth ——— To indicate a place of significance to an individual: [(a)] 0 primary cardholder 1 secondary cardholder
identification number {id-set-field identificationNumber (5)}	Indicates a field that contains an identification number assigned to an individual.	To identify the type of identification: 0 passport 1 national identity 2 voter registration 3 driver license 4 business license ——— To indicate the nation from which the identification is expected to originate: the ISO-3166 numeric country code. ——— To indicate a place of significance to an individual: [(a)] 0 primary cardholder 1 secondary cardholder

(a) A further qualification to indicate a relative of one of these individuals can be specified—see *Relatives* below.

Table 79: Field Types for {id-set-field}

Continued on next page

Field Types, continued

Field Type/OID	Description	Additional Qualifiers
month {id-set-field Month (6)}	Indicates a field that contains a month of significance to an individual.	To indicate the significance of the month: 0 birth To indicate a place of significance to an individual: (a) 0 primary cardholder 1 secondary cardholder
date {id-set-field date (7)}	Indicates a field that contains a date of significance to an individual.	To indicate the significance of a date as follows: 0 birth 1 last payment 2 last transaction 3 account opened 4 PAN expiration To indicate a data of significance to an individual: (a) 0 primary cardholder 1 secondary cardholder
address {id-set-field address (8)}	Indicates a field that contains an address.	To indicate the type of address: 0 current home 1 current billing 2 current work 3 prior home 4 prior billing 5 prior work To indicate the nation from which the address is expected to originate: ISO-3166 country code
telephone {id-set-field telephone (9)}	Indicates a field that contains a telephone number.	To indicate the type of number: 0 home 1 business To indicate the nation from which the address is expected to originate: ISO-3166 country code

(a) A further qualification to indicate a relative of one of these individuals can be specified—see *Relatives* below.

Table 79: Field Types for {id-set-field}, continued

Continued on next page

Field Types, continued

Field Type/OID	Description	Additional Qualifiers
amount {id-set-field amount (10)}	Indicates a field that contains an amount.	To indicate the type of amount: 0 last payment 1 last transaction 2 account balance 3 line of credit To indicate the currency code: the ISO-4217 currency code.
account number {id-set-field amount (11)}	Indicates a field that contains an account number.	To indicate the type of account: 0 checking 1 savings 2 card verification code [(b)]
pass phrase {id-set-field amount (12)}	Indicates a field that contains a pass phrase.[(c)]	

(b) The card verification code is a number printed on the payment card; because this number is not embossed and does not appear on the magnetic stripe, it is only available to someone with physical access to the card.

(c) Typically, this will be exchanged between the individual and the financial institution over a trusted medium of exchange.

Table 79: Field Types for {id-set-field}, continued

Field Qualifier

Relatives

The relatives of an individual can be specified by adding an additional qualifier to the end of an OID as follows:

0	spouse
1	mother
2	father
3	maternal grandmother
4	maternal grandfather
5	paternal grandmother
6	paternal grandfather

Hierarchical assignments

The object identifiers are assigned in a hierarchical manner with the information becoming more specific with each number in the series providing a more specific designation.

Examples

The following are examples of object identifiers for various fields:

{id-set-field 0 1}	secondary cardholder's full name
{id-set-field 3 0 1}	primary cardholder's mother's maiden name
{id-set-field 5 0 392 0}	primary cardholder's Japanese passport number
{id-set-field 5 1 840 1}	secondary cardholder's U.S. social security number
{id-set-field 9 1 840}	business telephone number in U.S. format: (800) 555-1212
{id-set-field 10 0}	amount of last payment
{id-set-field 10 0 826}	amount of line of credit specified in Pound Sterling (UK)

Continued on next page

Field Qualifier, continued

Primary cardholder's mother's maiden name

To clarify the usage of the relative qualifier in the hierarchical manner for identifying information, the assignment given for the primary cardholder's mother's maiden name example is defined as follows:

```
{id-set-field 3 0 1}
              | | |
              | | |
              | | +--mother relative qualifier
              | |
              | |
              | +--primary cardholder qualifier
              |
              |
              +--family name at birth (or maiden name)
```

Primary cardholder's Japanese passport number

To further elaborate on usage of multiple qualifiers for a field in the hierarchical manner for identifying information, the assignment given for the primary cardholder's Japanese passport number example is defined as follows:

```
{id-set-field 5 0 392 0}
              | |  |   |
              | |  |   |
              | |  |   +--primary cardholder qualifier
              | |  |
              | |  |
              | |  +--nation qualifier
              | |
              | |
              | +--passport qualifier
              |
              |
              +--identification number
```

Appendix M
ContentTypes

Introduction

Organization

This appendix includes tables that list SET messages (or data structures within messages), content, and contentType for:

- SignedData
- DigestedData
- EnvelopedData

SignedData

SignedData

The following table shows the SET message (or data structure within a message), the content and the contentType for signed data.

Message /Data Structure	**Content**	**Content Type**
AcqCardMsg	AcqCardCodeMsg	id-set-content-AcqCardCodeMsg
AuthReq	AuthReqTBS	id-set-content-AuthReqTBS
AuthRes	AuthResTBS	id-set-content-AuthResTBS
AuthRes	AuthResTBSX	id-set-content-AuthResTBSX
AuthRevReq	AuthRevReqTBS	id-set-content-AuthRevReqTBS
AuthRevRes	AuthRevResData	id-set-content-AuthRevResData
AuthRevRes	AuthRevResTBS	id-set-content-AuthRevResTBS
AuthToken	AuthTokenTBS	id-set-content-AuthTokenTBS
BatchAdminReq	BatchAdminReqData	id-set-content-BatchAdminReqData
BatchAdminRes	BatchAdminResData	id-set-content-BatchAdminResData
CapReq	CapReqTBS	id-set-content-CapReqTBS
CapReq	CapReqTBSX	id-set-content-CapReqTBSX
CapRes	CapResData	id-set-content-CapResData
CapRevReq	CapRevReqTBS	id-set-content-CapRevReqTBS
CapRevReq	CapRevReqTBSX	id-set-content-CapRevReqTBSX
CapRevRes	CapRevResData	id-set-content-CapRevResData
CapToken	CapTokenData	id-set-content-CapTokenData
CapToken	CapTokenTBS	id-set-content-CapTokenTBS
CardCInitRes	CardCInitResTBS	id-set-content-CardCInitResTBS
CertInqReq	CertInqReqTBS	id-set-content-CertInqReqTBS
CertReq	CertReqData	id-set-content-CertReqData
CertReq	CertReqTBS	id-set-content-CertReqTBS
CertRes	CertResData	id-set-content-CertResData
CredReq	CredReqTBS	id-set-content-CredReqTBS
CredReq	CredReqTBSX	id-set-content-CredReqTBSX
CredRes	CredResData	id-set-content-CredResData
CredRevReq	CredRevReqTBS	id-set-content-CredRevReqTBS
CredRevReq	CredRevReqTBSX	id-set-content-CredRevReqTBSX
CredRevRes	CredRevResData	id-set-content-CredRevResData
InqReqSigned	InqReqData	id-set-content-InqReqData
Me-AqCInitRes	Me-AqCInitResTBS	id-set-content-Me-AqCInitResTBS
PCertReq	PCertReqData	id-set-content-PCertReqData
PCertRes	PCertResTBS	id-set-content-PCertResTBS
PInitRes	PInitResData	id-set-content-PInitResData
PISignature	PI-TBS	id-set-content-PI-TBS
PRes	PResData	id-set-content-PResData
RegFormRes	RegFormTBS	id-set-content-RegFormResTBS
SignedError	ErrorTBS	id-set-content-ErrorTBS

DigestedData

DigestedData

The following table shows the SET message (or data structure within a message), the content and the contentType for digested data.

Message / Data Structure	**Content**	**Content Type**
AuthReq	PI	id-set-content-PI
AuthRes	AuthResBaggage	id-set-content-AuthResBaggage
AuthRes	PANToken	id-set-content-PANToken
AuthRevReq	AuthRevReqBaggage	id-set-content-AuthRevReqBaggage
AuthRevRes	AuthRevResBaggage	id-set-content-AuthRevResBaggage
AuthToken	PANToken	id-set-content-PANToken
CapReq	CapTokenSeq	id-set-content-CapTokenSeq
CapReq	PANToken	id-set-content-PANToken
CapRevReq	CapTokenSeq	id-set-content-CapTokenSeq
CapRevReq	PANToken	id-set-content-PANToken
CapToken	PANToken	id-set-content-PANToken
CertReq	AcctInfo	id-set-content-AcctInfo
CredReq	CapTokenSeq	id-set-content-CapTokenSeq
CredReq	PANToken	id-set-content-PANToken
CredRevReq	CapTokenSeq	id-set-content-CapTokenSeq
CredRevReq	PANToken	id-set-content-PANToken
HOD	HODInput	id-set-content-HODInput
HOIData	OIData	id-set-content-OIData
HPIData	PIData	id-set-content-PIData
OIDualSigned	PIData	id-set-content-PIData
OIUnsigned	PIDataUnsigned	id-set-content-PIDataUnsigned
PIDualSigned	PANData	id-set-content-PANData
PI-OILink	OIData	id-set-content-OIData
PIUnsigned	OIData	id-set-content-OIData
PIUnsigned	PANToken	id-set-content-PANToken
RegFormReq	PANOnly	id-set-content-PANOnly
RootKeyThumb	SubjectPublicKeyInfo	id-set-rootKeyThumb

EnvelopedData

EnvelopedData

The following table shows the SET message (or data structure within a message), the content and the contentType for enveloped data.

Message / Data Structure	**Content**	**Content Type**
AcqCardMsg	AcqCardCodeMsgTBE	id-set-content-AcqCardCodeMsgTBE
AuthReq	AuthReqTBE	id-set-content-AuthReqTBE
AuthRes	AuthResTBE	id-set-content-AuthResTBE
AuthRes	AuthResTBEX	id-set-content-AuthResTBEX
AuthRevReq	AuthRevReqTBE	id-set-content-AuthRevReqTBE
AuthRevRes	AuthRevResTBE	id-set-content-AuthRevResTBE
AuthRevRes	AuthRevResTBEB	id-set-content-AuthRevResTBEB
AuthToken	AuthTokenTBE	id-set-content-AuthTokenTBE
BatchAdminReq	BatchAdminReqTBE	id-set-content-BatchAdminReqTBE
BatchAdminRes	BatchAdminResTBE	id-set-content-BatchAdminResTBE
CapReq	CapReqTBE	id-set-content-CapReqTBE
CapReq	CapReqTBEX	id-set-content-CapReqTBEX
CapRes	CapResTBE	id-set-content-CapResTBE
CapRevReq	CapRevReqTBE	id-set-content-CapRevReqTBE
CapRevReq	CapRevReqTBEX	id-set-content-CapRevReqTBEX
CapRevRes	CapRevResTBE	id-set-content-CapRevResTBE
CapToken	CapTokenTBE	id-set-content-CapTokenTBE
CapToken	CapTokenTBEX	id-set-content-CapTokenTBEX
CertReq	CertReqTBE	id-set-content-CertReqTBE
CertReq	CertReqTBEX	id-set-content-CertReqTBEX
CertRes	CertResTBE	id-set-content-CertResTBE
CredReq	CredReqTBE	id-set-content-CredReqTBE
CredReq	CredReqTBEX	id-set-content-CredReqTBEX
CredRes	CredResTBE	id-set-content-CredResTBE
CredRevReq	CredRevReqTBE	id-set-content-CredRevReqTBE
CredRevReq	CredRevReqTBEX	id-set-content-CredRevReqTBEX
CredRevRes	CredRevResTBE	id-set-content-CredRevResTBE
PIDualSigned	PIDualSignedTBE	id-set-content-PIDualSignedTBE
PIUnsigned	PIUnsignedTBE	id-set-content-PIUnsignedTBE
RegFormReq	RegFormReqTBE	id-set-content-RegFormReqTBE

Appendix N
Check Digit Algorithm

Purpose

This appendix describes the standard algorithm used by the payment card industry for check digits. It is provided as information only, and is not specific to SET.

Calculating a check digit

To calculate the check digit for a payment card account number:

Step	Action
1	Number the account number digits from right to left, starting with the check digit as number one.
2	Ignoring the check digit, multiply the even numbered digits by two (2), and the odd-numbered digits by one (1).
3	Calculate the sum of all the digits of the products as a string without regard to boundaries. That is, if a product from Step 2 was a 2-digit number, add each of the digits individually.
4	Subtract the result from the next higher multiple of ten (10). The resulting mod-10 residue is the check digit.

Example

The table below shows the results of applying the algorithm described above to the example account number 4287 9478.

Step									
1	Digit number:	8	7	6	5	4	3	2	1
	Example account number:	4	2	8	7	9	4	7	8
2	Multiplier:	x2	x1	x2	x1	x2	x1	x2	
	Products:	8	2	16	7	18	4	14	
3	Sum:	8+ 2+ 1+ 6+ 7+ 1+ 8+ 4+ 1+ 4 = 42							
4	Check digit	50 - 42 = 8							

Appendix P
Guidelines for Secure Implementation of SET

Overview

Introduction

This appendix is based on an RSA Laboratories Technical Note prepared by Burton S. Kaliski Jr., Ph.D. and M.J.B. Robshaw, Ph.D. of RSA Laboratories.

The authors have provided this overview of techniques for the secure implementation of cryptographic systems, which is intended to assist developers and engineers in developing applications that provide the full security envisaged by the designers of SET. It covers general issues, such as random number generation and protection of keys, and addresses their particular application to SET.

Organization

This appendix includes the following topics:

Topic	Page
Introduction	518
Background on Cryptography Used in SET	520
Assumptions about the SET Environment	522
General Implementation Guidelines	524
Applying these Guidelines to SET	538
Additional Cryptographic Issues	544
Conclusion	545
References	546
Secure Implementation Checklists	548

Introduction

Need for secure transaction processing

The world is witnessing a vast explosion in the popularity and availability of the Internet and the World Wide Web. With the expanded access to these networks and their ever-increasing popularity, there are new business opportunities. The World Wide Web in particular offers considerable business potential, as it gives vendors access to a much wider customer base than previous geographical constraints permitted.

Perhaps the chief obstacle to using the Internet or the World Wide Web as a medium for financial transactions is the reasonable belief that without adequate safeguards, personal and sensitive information such as payment card numbers and transaction amounts are vulnerable to interception by potentially malicious entities.

The cryptographic technology that can provide such safeguards has been available for many years and the dynamics of the market make it necessary to standardize on a means of securely enabling card payments over the Internet and the World Wide Web.

The SET specification

Visa and MasterCard have addressed this need with the co-development of the SET Secure Electronic Transaction specification. Now in the final stages of development, this specification is sufficiently broad and well-defined that developers and implementers can work on producing SET-compliant software and applications. This work will continue to find its way into a wide range of future products.

There is, however, a potentially great gulf between developing an application that starts with the SET specification and successfully producing an application that provides the full security that is intended by the designers of SET. These guidelines are intended to:

- highlight some of the important security issues that face application developers and implementers as they design a SET-enabled application;
- provide general guidance on issues such as key generation and the protection of keys and data—these issues as they apply to SET are discussed in more detail later in this document; and
- offer information in the form of checklists to assist in developing and validating implementations of the SET protocol.

These guidelines are not intended as a comprehensive description of how to obtain a secure implementation of SET. Rather, they advocate general principles for a secure implementation. It is for the implementer and the validator of a SET implementation to demonstrate compliance between specific implementation techniques and the outlined principles.

Continued on next page

Introduction, continued

Related work

These guidelines are intended to be complementary to other publications such as “Good Programming Practices,” published as part of FIPS 140-1 [NIS95]. The reader is encouraged to refer to this and other such publications for more information and general guidance on good programming techniques for cryptographic applications.

Background on Cryptography Used in SET

Introduction

The SET Specification provides a mechanism to enable secure payment card transactions over unsecured networks.

Data confidentiality

The idea of data confidentiality is fundamental to protecting transactions, as described earlier in this book (Part I, Chapter 3, Confidentiality):

> *Data confidentiality is the protection of sensitive and personal information from unintentional and intentional attacks and disclosure. Securing such data requires data encryption and associated key management in uncontrolled environments, such as unsecured networks.*

Data confidentiality and methods of authentication can be provided by the field of cryptography [RSA96]. The SET Specification calls on a variety of techniques, some of which are highlighted in this section as background to the guidelines that follow. These techniques include:

- symmetric encryption,
- asymmetric encryption,
- digital signatures, and
- hash function.

Symmetric encryption

Symmetric encryption is a method of converting data into an unintelligible form so that only participants who possess the secret key can recover the meaningful data. Two symmetric ciphers are used in SET: DES [NIS94a] and CDMF [JMLW93].

With symmetric encryption, both the sender and the receiver of the encrypted message must possess the same key.

Asymmetric encryption

Asymmetric encryption is also a method of converting data into an unintelligible form. It differs from symmetric encryption in that the sender of the message and the recipient use different keys. In fact, the public key, used to encrypt the message, is published (as the name implies); therefore, anyone can send an encrypted message to the intended receiver.

However, only the receiver is able to decrypt the enciphered message, since only the receiver possesses the corresponding private key. This private key must be kept secret.

Continued on next page

Background on Cryptography Used in SET, continued

Digital envelope

In SET, asymmetric and symmetric encryption techniques are used together for a *digital envelope*, in which:

- first, the message is encrypted using a symmetric cipher with a randomly generated key
- then, the key used to encrypt the message is itself encrypted using public-key encryption.

In SET, RSA [RSA78].is the only specified method of public-key encryption.:

Digital signatures

A public/private key pair can also be used to generate a *digital signature*. To provide a digital signature, a document is transformed in a way that depends on the private key of the signer. Since the corresponding public key is widely available, this transformation can be reversed by anyone using the public key. In essence, public-key cryptography is being used to perform an operation that can only be successfully completed by one person, yet can be verified by anyone. These properties are analogous to the conventional handwritten signature.

Digital signatures are also widely used to provide *public-key certificates*. Such certificates are used to assure users that a specific public key is legitimate and indeed belongs to the correct user.

Hash function

Another primitive widely used in SET is a *hash function*. Hash functions have a wide variety of properties. The specific hash function used in SET, SHA-1 [NIS94b], is used at different places for different reasons.

Two of the main properties of SHA-1 are:

- collision-resistance—meaning that it is difficult to find two inputs that produce the same output
- one-wayness—meaning that it is difficult to find any input that produces a given output.

SHA-1 is also used because it is believed that:

- the output from an iteration of SHA-1 is not easily distinguished from a truly random string
- the outputs from two closely related, but distinct, inputs to SHA-1 will appear to be unrelated.

Assumptions about the SET Environment

Ideal environment for SET

In an ideal environment, with complete physical and logical access control, a SET-compliant implementation with local key storage and strong random number generation will be secure, to the extent that SET itself is secure. Since keys are stored locally, the physical access control to the environment prevents attacks on the keys; the strong random number generation ensures that the keys are sufficiently unpredictable.

In such an environment, there is no need for further implementation guidelines. Any errors in the implementation are concealed by the access control to the environment; no opponent can exploit them.

On the other hand, in the *ideal* environment for an opponent, everything is visible, including all keys, and no implementation can be secure. Again there is no need for implementation guidelines, as even correct operation can be exploited by the opponent.

In practice, environments fall somewhere between the extremes of complete control and total visibility; and for such environments, implementation guidelines are quite helpful. Therefore, it is important to define this practical environment as a context for these guidelines.

A typical environment for SET

A typical environment to consider is one where the SET implementation runs as a process on a workstation, with a standard operating system, potentially shared access to files, and possibly peripheral cryptographic hardware. More specifically, the following assumptions can be made:

- operating system integrity—that the operating system software is in its original state and cannot be modified. Configurable aspects of the operating system may be changed, but it is also assumed that these changes have no impact on security.
- SET application integrity—that the SET application software is in its original state and cannot be modified
- process memory integrity and confidentiality—that an opponent cannot examine or alter the internal memory space of a process during the execution of a process.

  However, no assumptions are made about the exposure of the memory space after the process executes, or about portions of allocated memory that are freed or swapped to disk space while the process is executing. Also, it should be assumed that if cryptographic hardware is present, the path between the process and the hardware has integrity and confidentiality; however, the hardware itself may be available to the opponent at other times.
- clock integrity—that an opponent cannot alter the local clock and that the clock is sufficiently accurate for the purposes of SET.

Continued on next page

Assumptions about the SET Environment, continued

Primary concerns

Given these assumptions, there is not a concern with attacks where an opponent modifies the operating system or the SET application—obviously, such attacks could potentially reveal keys and other quantities. More specifically, there is not a concern with computer viruses or other forms of software intrusion by which an opponent can alter the operation of the SET application.

Instead, it is assumed that it is normal operating practice to detect and remove such intrusions. It is acknowledged that there are other intrusions such as keystroke recorders which can intercept passwords, and but again assume that such intrusions are detected and removed. The primary interest is in:

- attacks on stored data and on the remnants of the execution of a SET process, including:
  - files stored by the SET application, which may contain keys, and
  - processor memory and disk swap files on release by the SET application
- attacks based on weaknesses in random number generation are also of interest.

Assumptions about the opponent

It is assumed that the opponent has:

- full knowledge of the SET software and its external behavior, but no knowledge of its internal behavior, except possibly its timing characteristics
- complete control over the inputs to the SET software, whether supplied over the network, in files, or through a user interface.

However, the opponent is not expected to have knowledge of secret user information such as PINs and passwords, nor of any secret or private keys or sensitive data.

At this point, there is not a concern with attacks where the opponent attempts to introduce errors in the SET application through physical intrusion and thereby compromise security—as described, for example, in a recent security announcement by Bellcore [Bel97]. While such attacks are theoretically possible, they can be prevented by physical means. Also not addressed are attacks based on the analysis of timing characteristics of SET processing [Koc96]; these can be handled through means such as those described in RSA Laboratories' bulletin on timing attacks [Kal96].

General Implementation Guidelines

Introduction

This section provides general implementation guidelines, which are based on the assumptions discussed in the previous section. These guidelines are intended to ensure the security of a cryptographic system, including SET. For each guideline, the rationale is given, followed by implementation considerations.

These guidelines cover the following topics:

- random number generation
- public asymmetric keys
- private asymmetric and secret symmetric keys
- sensitive data
- memory leakage

Guideline 1: Random Number Generation

Guideline

Random number generation must be cryptographically strong:

- It must be computationally infeasible for an opponent who knows partial information about the random number generator to determine any other information about the output, with a better success rate than would be achieved by guessing.
- The generator's level of cryptographic strength must be at least as great as that of the algorithms for which keys and other quantities are generated. In other words, the random number generator must not be the weakest link.

Rationale

Some outputs of a random number generator may be transmitted in the clear in some applications; others may be accessible to the opponent, such as symmetric keys distributed to the opponent acting as a valid participant. Still others may be compromised through cryptanalysis or implementation failures by other participants. Thus it is reasonable to assume that an opponent has partial information about the output of the random number generator. Despite this information, the opponent should have no advantage over guessing in determining other outputs from the random number generator.

The overall security of a system depends not only on key sizes, but also on the quality of the generation of the keys. Thus it is important that the random number generator provide sufficient quality keys, and that it be at least as difficult to attack as the keys themselves. Indeed, the more keys that are derived from a random number generator, the greater its strength must be, since compromise of the generator can potentially compromise all the keys derived from it.

Implementation considerations

Other implementations may well be possible, in addition to the following, which are discussed here:

- hardware random number generators
- pseudorandom number generators
- multiple random number generators

Further discussion on random number generation can be found in [ECS94].

Hardware random number generator

A high-quality hardware random number generator, for instance within a cryptographic module, is perhaps the best source of randomness.

Note that with a hardware random number generator, it would be prudent to ensure that checks are in place to detect hardware failures.

Continued on next page

Guideline 1: Random Number Generation, continued

Pseudorandom number generator

A cryptographically strong pseudorandom number generator with a seed of sufficient unpredictability can also provide a good source of randomness.

While no generator has been proved cryptographically strong without assumptions, cryptographically strong pseudorandom number generators have been shown to exist under assumptions such as the difficulty of factoring [BBS86]. Constructions based on hash functions have also been given, which have a heuristic basis for security [NIS94c]; some require a secret key as input in addition to the seed.

Note that standard library functions, such as C's rand(), are not cryptographically strong.

A sufficiently unpredictable seed can generally be obtained by appropriate sampling of system events which are individually unpredictable; see [Mat96].

One type of seed input that is particularly interesting is a random "pool," obtained from sampling of previous system events which is stored after encryption with a secret key—it may also be helpful to authenticate it. This provides a continuity between instances of the random number generator, and reduces the number of new seed samples that might be required.

Note that system information such as the time in seconds is not sufficiently unpredictable, though the microsecond portion of the time can be taken as part of a random seed on some systems.

A variation on a pseudorandom generator where additional seed material can be mixed in after the output has been generated may be helpful, since this would provide further protection to later outputs, should earlier outputs be compromised.

Multiple random number generators

It may be appropriate in some cases to have more than one random number generator in an implementation, where, for instance, keys and nonces are derived from different generators. In addition, it may be helpful to have different seeds for different quantities, even with the same generator. While in principle the strongest of the generators should suffice for all quantities, it may be more efficient in practice to have different generators for different purposes. A similar observation can be made for seeds.

As examples:

- keys may be generated within a cryptographic module with a high-quality hardware random number generator, and nonces generated within software; or
- keys may be derived from a generator that is slower but provably secure (under assumptions), while nonces are derived from a generator that is faster but only heuristically secure.

Guideline 2: Public Asymmetric Keys

Guideline

Public asymmetric keys must be protected from modification and must retain their binding to any associated attributes.

It must not be possible for an unauthorized party to alter any of the following without detection:

- the value of a public asymmetric key
- the value of its associated attributes, or
- the binding between the key and its attributes.

When a key is in processor memory, it is assumed to be sufficiently protected. Thus, the primary issue is how keys are stored outside processor memory.

Rationale

Protection of public keys from disclosure is not a concern, since a public key is *prima facie* assumed to be known. For similar reasons, protection against misuse is not a direct concern—of course, it is a concern with respect to the data that is being encrypted with the public key.

However, public keys must be protected from modification. Otherwise, certain cryptographic attacks may be possible—for example, those that might involve the substitution of different keys. Similarly, keys must retain their binding to associated attributes, such as their lifetime and ownership, to prevent attacks where an old key is reused or where the wrong key is used. Not only must a public key be protected from modification; in addition, the meaning of the key must be preserved.

Implementation considerations

This section addresses the following implementation options:

- read-only storage
- data integrity mechanism
- a combination of physical and logical protection
- hierarchical protection
- specifying attributes implicitly.

Continued on next page

Guideline 2: Public Asymmetric Keys, continued

Read-only storage

The basic *physical* means for protection against modification is read-only storage—that is, read-only to outsiders, though potentially writeable by the application software. Read-only storage can be accomplished in a variety of ways, including:

- operating system controls;
- storage within a cryptographic module whose access is controlled; and
- storage within the application software itself, which is assumed to have integrity.

Storage controlled by the operating system is perhaps the most practical form of physical protection for most keys, given the limited storage in a typical cryptographic module and the difficulty of updating application software with new keys.

Data integrity mechanisms

The basic *logical* means for protection is a data integrity mechanism, of which there are two types:

- digital signatures and
- message authentication codes.

Public-key certificates are a common means of protecting public keys and their attributes with digital signatures. For this approach, an implementation would verify the digital signature before operating with a public key. Note that the verification key itself would first need to be verified.

Message authentication codes are generated with a secret symmetric key and verified with the same key; two popular examples of message authentication codes are described in [NIS85] and [BCK96]. For this approach, an implementation would verify the message authentication code before operating with a public key. The verification key would need to be verified first, and would also need to be protected from disclosure.

The data integrity mechanism should be applied both to the key and to its attributes.

The strength of the integrity mechanism should generally correspond to that of the class of key being protected, to prevent attacks based on substitution of keys. Note that key size is a separate issue from the size of a message authentication code; the latter determines the probability of undetected forgery, rather than the difficulty of key recovery. The acceptable probability will depend on the key being protected, and the ease with which an opponent can have the application check a message authentication code.

Continued on next page

Guideline 2: Public Asymmetric Keys, continued

A combination of physical and logical protection

A combination of physical and logical protection can sometimes be beneficial. For instance, the binding of certain attributes could be accomplished through read-only storage, while the binding of other attributes is accomplished with a message authentication code based on a key that is stored in a cryptographic module. The requirements of the implementation will determine which combinations are more appropriate.

Hierarchical protection

Hierarchical protection is often quite practical. If a root public key is stored in application software, then certificates signed with the root private key are protected from modification. Protection is similarly extended to the next level of keys through another level of certificates.

Another way to begin the chain of trust is to protect a small set of public keys from modification with a message authentication code whose verification key is derived from or protected by a user's password.

Specifying attributes implicitly

In some cases it is possible to specify certain attributes implicitly—for example, through the name of the file in which the protected public key is stored, where the file name might give the name of the owner. If this is the case, then it is important that the implications cannot be modified. In the example stated, it would be important that the file be protected from renaming; read-only access alone may not be enough.

Guideline 3: Private Asymmetric and Secret Symmetric Keys

Guideline

Private asymmetric keys and secret symmetric keys must be protected from modification, disclosure, and misuse; and they must retain their binding to any associated attributes.

- It must not be possible for an unauthorized party to obtain the value of a private asymmetric key or a secret symmetric key, or to alter its value, the value of its associated attributes, or the binding between the key and its attributes without detection. As a particular case of unauthorized disclosure, it must not be possible for a party to obtain the encryption of a private asymmetric key or a secret symmetric key with another key that is not trusted by the application.
- It must not be possible to use a private asymmetric key or a secret symmetric key in a manner other than that for which it was intended. For instance, it should not be possible to use a private signature key as a decryption key, and more generally, it should not be possible to use one application's key for the purpose of a different application.

When a key is in processor memory, it is assumed to be sufficiently protected, provided that the processor memory is protected against leakage of data (see the section on memory leakage later in this section).

Continued on next page

Guideline 3: Private Asymmetric and Secret Symmetric Keys, continued

Rationale

By definition, private asymmetric keys and secret symmetric keys must be protected from:

- disclosure, or otherwise they would no longer be private or secret
- modification, or otherwise certain cryptographic attacks might become possible

Note that these are independent issues—a key may be protected from disclosure by encrypting it, but unless the encryption also offers integrity, the key will not be protected from modification.

Furthermore, the keys must retain their binding to associated attributes, such as their lifetime and ownership, to prevent attacks where an old key is reused, or where the wrong key is used. It is not enough just that a key cannot be modified; the meaning of the key must also be preserved.

Controlling key usage is important as a general principle to avoid weaknesses arising from interactions between multiple uses. This is a particularly important issue if keys are shared among multiple applications, since even if one application is internally consistent in its use of keys, it may conflict with the other.

Continued on next page

Guideline 3: Private Asymmetric and Secret Symmetric Keys, continued

Examples

As one example, suppose a cryptographic module permits a private key both to decrypt encrypted symmetric keys, where the symmetric keys remain within the module, and to decrypt encrypted data, where the data may leave the module. Then an opponent can obtain a symmetric key by treating its encryption as encrypted data.

As another example, if a key is used to sign messages in two applications, and the signatures involve the same data formats and cryptographic techniques, then it will be possible to substitute messages signed in one application with those in the other. The differing interpretations of the same message bytes can lead to potential attacks. These attacks can all be avoided if a key's use is restricted.

If a private asymmetric key or a secret symmetric key is encrypted with another key, such as a public key, in the process of distributing a secret symmetric key to another party, it should first be verified that the public key is authorized for that operation—based on its ownership and other attributes. Otherwise, the key being distributed may be compromised by the party with the corresponding private key. Presumably, if the public key is authorized for the operation, the party with the private key can be trusted not to compromise the key that has been distributed. The concern over disclosure is balanced in this way with the need to distribute the key to another party.

Implementation considerations

This section addresses the following implementation considerations:

- read-only storage
- unreadable storage
- data integrity mechanisms
- data compatibility mechanisms
- a combination of physical and logical protection
- hierarchical protection
- key usage

Continued on next page

Guideline 3: Private Asymmetric and Secret Symmetric Keys, continued

Read-only storage

The basic physical means for protection against modification is read-only storage. Also refer to the previous section on public asymmetric keys for further discussion of read-only storage.

Unreadable storage

The basic physical means for protection against disclosure is unreadable storage—unreadable to outsiders, though readable to the application software.

Unreadable storage can be accomplished in a variety of ways, including operating system controls in some systems and storage within a cryptographic module with controlled access.

Again, storage controlled by the operating-system is perhaps the most practical form of physical protection, given the limited storage in a typical cryptographic module.

Data integrity and data confidentiality mechanisms

The basic logical means for protection against modification is a data integrity mechanism. Again, refer to the previous section on public asymmetric keys for further discussion.

The basic logical means for protection against disclosure is a data confidentiality mechanism of which there are two types (as discussed in the section on cryptography):

- public-key encryption
- symmetric encryption.

Typically, symmetric encryption is a more practical approach for stored data, since both types require the storage of a decryption key, which must be protected from disclosure and symmetric techniques are generally more efficient.

The order in which data integrity and data confidentiality are applied can be important, depending on the specific techniques. A generally good approach is:

- first, apply data integrity first to both the key and the attributes and
- then apply data confidentiality to the key and the integrity check value (signature or message authentication code).

In this way the attributes remain in the clear and no partial information about the key is available through the integrity check value. If an attribute needs to be protected from disclosure, it can be encrypted along with the other parts.

Continued on next page

Guideline 3: Private Asymmetric and Secret Symmetric Keys, continued

A combination of physical and logical protection

A combination of physical and logical protection can sometimes be beneficial (also refer to the previous section public asymmetric keys). For example, protection against modification could be accomplished through read-only storage, while protection against disclosure is accomplished with symmetric encryption based on a key that is stored in a cryptographic module. The requirements of the implementation will determine which combinations are more appropriate.

Hierarchical protection

Hierarchical protection is often quite practical (again refer to the previous section). If a master symmetric key is stored in a cryptographic module, then keys encrypted with the master key are protected from disclosure. Protection is similarly extended to the next level of keys through another level of encryption.

Another way to begin the chain of protection is to protect a small set of keys from disclosure with a symmetric key derived from or protected by a user's password.

Key usage

Key usage can be specified as an attribute of the key; the implementation would check the key usage attribute before performing a cryptographic operation.

A particularly sensitive key, such as a certificational private key, can be protected against disclosure by encrypting it with a key that is "secret-shared" among several trustees, or even by "secret-sharing" the key itself [Sha85]. In such an implementation, the cooperation of a threshold number of trustees (say, three out of five) is required to reconstruct the key. Any sufficiently large subset can reconstruct the key, but no smaller subset can do so. A variation on this theme is for the cryptographic module itself to hold one of the shares, so that the trustees can only reconstruct the key with a particular module, allowing more control and auditing in the system.

Recently, techniques have been proposed for private asymmetric keys where the private key itself is not even reconstructed, but rather the results of a private-key operation are obtained by the results of the trustees' operations with secret values derived from the private key [GJKR96]. Such techniques are worth considering further for sensitive keys.

Guideline 4: Sensitive Data

Guideline

Sensitive data items must be protected from modification and/or disclosure depending on the type of data, and must retain their binding to any other data items or attributes with which they are associated.

Rationale

Certain data may need to be protected from various threats; the rationale and implementation is similar to that for keys (refer to the previous two sections on public asymmetric keys, and private asymmetric and secret symmetric keys).

Data items that might need protection include:

- key identification information and counters, which must be protected from unauthorized modification
- data such as account numbers, which must be protected from unauthorized disclosure
- intermediate results of computation that are stored in a file.

A critical attribute of such a result is its place in the current computation, which must be protected to prevent substitution of old results for new ones.

Guideline 5: Memory Leakage

Guideline

Processor memory must be protected against leakage of sensitive data.

Rationale

Processor memory is assumed to have confidentiality only until it is released by a process, so memory that is allocated and later freed may be at risk. Also, depending on the system, memory that is copied to a disk file, as in a virtual memory system, may be at risk.

Note: Leakage is defined as the release of sensitive data, through errors in programming or mishandling of memory.

Intermediate results of cryptographic computations, as well as inputs and outputs, should be considered potentially sensitive. For example, the intermediate products in a private-key operation can give sufficient information for an opponent to determine the private key.

As another example, the data array in a SHA-1 computation will contain sensitive data if the input to SHA-1 is a key, and the state variables in SHA-1 may contain sensitive data if SHA-1 is called as part of random number generation that outputs a key.

Implementation considerations

This section addresses:

- zeroization of memory
- scrambling memory

Continued on next page

Guideline 5: Memory Leakage, continued

Zeroization of memory

Zeroization of memory containing any sensitive or potentially sensitive data, before that memory is released, is an essential practice.

Any data related to a cryptographic operation is potentially sensitive. For instance, the intermediate results of a hash function computation may be sensitive.

Both memory allocated from the heap and memory allocated on the stack should be zeroized.

It may be necessary to *lock* memory containing sensitive data to prevent it from being swapped to a disk file, in a virtual memory system, if that disk file could later be examined by an opponent. Alternatively, if memory needs to be swapped, then it should be encrypted first with a key that remains in process memory.

Particular attention should be paid to error returns—preferably, all returns from a procedure should pass through a common exit that zeroizes memory.

Storage of sensitive data in registers may be a concern. For instance, if the last operation performed by a process before it exits is an encryption, and the key is stored in registers, then it is possible that some data in those registers might remain available for examination by another process - it is assumed that the registers cannot be examined during the execution of the process. In this case, it is important to zeroize the registers before the process exits.

Note that compiler optimizations may sometimes eliminate zeroization at the end of a procedure, since the zeroized memory is not referenced again within the procedure. If this is the case, zeroization should be done with a procedure call, which the compiler will not eliminate. It is always important to verify not only that the source code is correct, but also that the compiled code is correct; but zeroization is a particular case where the compiler's definition of correctness may be different than expected.

Scrambling memory

Scrambling memory containing sensitive data, for example, by encrypting with a key stored elsewhere in memory, is a helpful barrier against failures in the assumption that process memory is private. It is also helpful in a case where the contents of memory are revealed in a way that is beyond the control of the application; for example, if an error crashes the system before the application has an opportunity to zeroize memory. Similarly, to limit exposure in the case of a crash, memory might be zeroized as soon as the data stored in it is no longer needed.

Applying these Guidelines to SET

Overview

This section applies the general guidelines to SET, by posing general questions to the SET developer or validator. More detailed checklists are provided following this section.

As a starting point for any developer, the cryptographic boundaries of the trusted environment within which SET will reside must be clearly documented. Any divergence from the assumptions made at the beginning of this appendix must be noted, and their effect on the general guidelines and their application to SET .

Then, the adherence of the SET implementation to the general principles for secure implementation can be documented. This information is presented in the form of five questions.

Continued on next page

Applying these Guidelines to SET, continued

Question 1

For each random number generator in the SET implementation:

a) describe the requirements for its strength, based on the quantities that are derived from it

b) justify how the required strength is achieved, listing all assumptions on which the claimed strength is based.

Assumptions

Assumptions may include the presumed security of certain cryptographic transformations, such as the pseudorandomness of a hash function.

The SET Specification is very clear about the importance of good random number generation. The following is from Book 1: Business Description (Section 3.2, Cryptography):

> *To provide the highest degree of protection, it is essential that the programming methods and random number generation algorithms generate keys such that keys cannot be easily reproduced using information about either the algorithms or the environment in which the keys are generated.*

In SET, cryptographically strong random number generation is required for all keys, as well as certain data items. These include generation of the following:

- all the message-encrypting public/private key pairs that are used by all the SET participants;
- all certificational public/private key pairs used within SET;
- all DES and CDMF keys;
- all random nonces, challenges and other random quantities such as the unique transaction number;
- all the random data required for the OAEP encryption method—the following quotation appears in this book (Part 1, Chapter 4, Section 2, Cryptography):

  > *Poor key generation and seeding methods due to using weak random numbers are common downfalls of cryptographic implementations.*

Continued on next page

Applying these Guidelines to SET, continued

Assumptions, continued

For SET, the random number generator shall have a level of security at least as great as 1024-bit RSA, 2048-bit RSA, DES or CDMF, depending on the algorithms for which keys are generated. As a general principle, random data generated by SET should be derived from sufficient truly random seed so that the intended cryptographic strength of the random quantities as used within the SET specification is not being undermined by the generation process itself.

This book of the SET specification (see Part 1, Chapter 4, Section 2) addresses the issue of using different random number generators and different seeds as follows:

For cryptographic purposes, once a strong seed is collected, it shall either be used one time only or it shall be used exclusively in cryptographically secure random number generators. Also, each instance of cryptographic algorithm shall have its own independent key-generation seed.

Continued on next page

Applying these Guidelines to SET, continued

Question 2

For each public asymmetric key in SET, describe how its required protection is achieved by the SET implementation, listing all assumptions on which the claimed protection is based

Required protection of public keys includes, at a minimum, protection from modification, and may also include usage controls and binding to other attributes.

Assumptions may include the presumed security of certain cryptographic transformations such as RSA signatures.

All public keys in SET and their attributes require protection from modification or replacement. One straightforward way of achieving this is by the use of certificates. At any level in a certification hierarchy the certificate binds the public key and attributes to some user at some lower level in the hierarchy. If each public key is certified (by means of the certification chain back to the Root CA) whenever it is used, then the threat of attack on the integrity and use of some public key is greatly reduced.

In certain situations it may well be preferable to verify a public-key certificate once and then to cache the public key in storage for later use. If such techniques are used then the proper cryptographic safeguards must be provided so that the integrity of the public key and its attributes are protected since these properties might not be checked again via certificates before use.

Note: The authentication of the Root CA certificational and CRL public keys requires special consideration since these keys cannot be authenticated by certificates. Instead the Root CA public keys are first distributed in the form of a self-signed certificate and they must then be stored in a properly authenticated manner.

Continued on next page

Applying these Guidelines to SET, continued

Question 3

For each private asymmetric key and secret symmetric key in SET, describe how its required protection is achieved by the SET implementation, listing all assumptions on which the claimed protection is based.

Required protection includes, at a minimum, protection from disclosure and modification, and also may include usage controls and binding to other attributes.

Assumptions may include the presumed security of certain cryptographic transformations such as DES encryption.

All private keys in SET are sensitive and need to be stored in a way that guarantees their secrecy and maintains the integrity of both the keys and their attributes. They also must be used according to their intended usage. Note that the number of different private keys that might need to be stored will differ greatly depending on which participant we consider in the SET transaction.

As remarked in the section on private asymmetric and secret symmetric keys, keys that are generated for symmetric encryption are also sensitive. For the most part, these keys are intended to be used on a per-message basis. They are to be generated fresh and then included within an RSA digital envelope that is sent to the intended recipient. There is often no need for this symmetric key to be stored. Usually any message incoming to a participant will contain the relevant (fresh) symmetric key within the envelope. There are exceptional circumstances however, such as the cardholder offering the payment gateway the possibility of secure communication back to the cardholder, when a symmetric encryption key that has already been generated, might have to be used. In such cases the storage of symmetric keys becomes an issue.

Continued on next page

Applying these Guidelines to SET, continued

Question 4

For each sensitive data item in SET, describe how its required protection is achieved by the SET implementation, listing all assumptions on which the claimed protection is based.

Required protection includes protection from disclosure and/or modification, depending on the data, and also include binding to attributes and other data items.

Assumptions may include the presumed security of certain cryptographic transformations such as DES encryption.

Data that is considered sensitive within the SET specification includes the PAN data, transaction information, payment card information and perhaps more depending on the participants and the messages being transmitted. If such information needs to be temporarily stored, and it is not directly protected by the message encryption techniques used during transmission, then the data must be protected against disclosure and modification by other cryptographic mechanisms as described in the previous section on sensitive data. It might well be a wise precaution to treat all data, as a default, as being as sensitive as key material and to take adequate precautions in its handling and storage.

Note: Certificates and CRLs do not need any protection, as their protection is *built-in*: by definition they are verified before use. In SET, the concept of a Brand CRL Identifier (BCI) is introduced to identify the CRLs that will be needed as part of signature verification. The BCI is itself authenticated using digital signature techniques.

Question 5

Describe how processor memory is protected against leakage of sensitive data.

This is a highly implementation-dependent issue and does not have any particular considerations that are a result of the SET specification.

Additional Cryptographic Issues

Introduction

The public key capabilities in SET are built around the use of RSA encryption and digital signatures. An important aspect of any implementation of SET will be the way that the RSA keys are actually generated. While this topic may be outside the scope of the SET Specification, it is mentioned here because this issue is so fundamental to security.

The RSA public key

The RSA public key consists of an RSA modulus and a public exponent. The RSA modulus is generated so that it consists of the products of two primes that are of roughly equal size. The security of RSA depends on the fact that it is particularly difficult to factor this type of modulus.

When generating primes of the size required to form the RSA modulus, it is common to use what are termed primality tests. These are efficient tests that indicate whether a given number is composite or prime with some level of confidence. By repeating such tests often enough, the probability that a composite number passes all the tests can be made arbitrarily small—for certain tests, the possibility of error can be eliminated entirely.

Strong primes

Some mention has been made in the literature to *strong primes.* These are prime numbers that are generated so that they have some particular properties. These properties were historically intended to provide protection against some of the earlier factoring algorithms. Today, however, it is widely believed that the use of strong primes is not required for RSA moduli of the size used in SET.

Choice of public exponent

The choice of public exponent, at this stage in the development of the SET specification, is left open to developers. With the OAEP format of RSA encryption that has been adopted within SET, there are no known bad public exponents. For reasons of good performance some common choices for the public encryption exponent are 3 or $2^{16}+1$ (also known as F_4) and there is no reason to question the security offered by either of these choices.

The issue of DES weak keys

Another cryptographic issue that may be of concern to developers is that of DES weak keys. It is well known that DES has what are termed *weak keys*. There are four of these keys and they are characterized by the fact that encrypting twice with one of these four keys will produce the initial input. The choice of whether to test for the presence of DES weak keys is left open to the developer. However, note that the probability of using one of these weak keys (or one of the 12 related semi-weak keys) is so exceptionally small when DES keys are generated at random, that a test for DES weak keys is very unlikely ever to detect one.

Conclusion

By standardizing on security techniques, the SET Specification lays the groundwork for secure payment card transactions over the Internet and the World Wide Web. The next challenge is to successfully produce applications that provide the full security as envisaged by the SET designers. The checklists that follow the next section on references are intended to assist developers and engineers in accomplishing this objective.

The authors have provided this overview of techniques for the secure implementation of cryptographic systems with the aim of helping to meet the stated challenge. Their goal is that these guidelines will assist those developing and validating SET implementations.

The authors welcome comments and suggestions.

References

Bibliography

[Bel97]	D Boneh, R. DeMillo and R. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy (ed.), Advances in Cryptology - Eurocrypt '97. Lecture Notes in Computer Science, vol. 1233, pages 37-51, Springer-Verlay, 1997.
[BBS86]	L. Blum, M. Blum and M. Shub. A simple unpredictable random number generator. *SIAM Journal on Computing,* 15:364-383, 1986.
[BCK96]	M. Bellare, R. Canetti and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz (ed.) *Advances in Cryptology - Crypto '96. Lecture Notes in Computer Science,* vol. 1109, pages 1-15, Springer-Verlag, 1996.
[BR94]	M. Bellare and P. Rogaway, Optimal asymmetric encryption. In *Advances in Cryptology - Eurocrypt '94. Lecture Notes in Computer Science,* vol. 950, pages 92-111, Springer-Verlag, 1994.
[ECS94]	D. Eastlake, S. Crocker and J. Schiller. RFC 1750: Randomness Recommendations for Security, December, 1994. Available from ftp://ds.internic.net/rfc/rfc1750.txt.
[GJKR96]	R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In N. Koblitz (ed.) *Advances in Cryptology - Crypto '96. Lecture Notes in Computer Science,* vol. 1109, pages 157-172, Springer-Verlag, 1996.
[JMLW93]	D. Johnson, S. Matyas, A. Le, and J. Wilkins. Design of the commercial data masking facility data privacy algorithm. In *Proceedings of 1st ACM Conference on Computer and Communications Security*, pages 93-96, ACM, 1993.
[Kal96]	B. Kaliski, Timing Attacks on Cryptosystems, *RSA Laboratories Bulletin,* No. 2, January 23, 1996. Available from http://www.rsa.com.
[Koc96]	P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz (ed.) *Advances in Cryptology - Crypto '96. Lecture Notes in Computer Science,* vol. 1109, pages 104-113, Springer-Verlag, 1996.
[Mat96]	T. Matthews, Suggestions for Random Number Generation in Software. *RSA Laboratories Bulletin,* No. 1, January 22, 1996. Available from http://www.rsa.com.
[NIS85]	National Institute of Standards and Technology (NIST). *FIPS Publication 113: Computer Data Authentication.* May 1985. Available from http://csrc.ncsl.nist.gov/fips/.
[NIS94a]	National Institute of Standards and Technology (NIST). *FIPS Publication 46-2: Data Encryption Standard.* February 1994. Available from http://csrc.ncsl.nist.gov/fips/.

Continued on next page

References, continued

[NIS94b]	National Institute of Standards and Technology (NIST). *FIPS Publication 180-1: Secure Hash Standard.* April 1994. Available from http://csrc.ncsl.nist.gov/fips/.
[NIS94c]	National Institute of Standards and Technology (NIST). *FIPS Publication 186: Digital Signature Standard (DSS).* May 1994. Available from http://csrc.ncsl.nist.gov/fips/.
[NIS95]	National Institute of Standards and Technology (NIST). *FIPS Publication 140-1: Security Requirements for Cryptographic Modules.* October 1995. Available from http://csrc.ncsl.nist.gov/fips/.
[RSA78]	R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM,* 21(2): 120-126, February 1978.
[RSA96]	RSA Laboratories. *Answers to Frequently Asked Questions about Today's Cryptography*. Version 3.0, 1996. Available from http://www.rsa.com.
[SET96]	*SET Secure Electronic Transaction Specification,* Latest revisions on August 1, 1996. Available from http://www.visa.com/ or http://www.mastercard.com/.
[Sha79]	A. Shamir. How to share a secret. *Communications of the ACM,* 22: 612-613, 1979.

Secure Implementation Checklists

Preface

These checklists are for developers of SET-enabled applications. They combine simple questionnaires with more detailed questions requiring longer answers.

Compliance with this checklist alone is not sufficient to guarantee the security of an implementation of SET. However, it does cover issues that are pertinent to the development of any cryptographic application, and in particular, to those following the SET Specification. In this sense, compliance with the checklist should be viewed as a necessary requisite for the development of a secure SET implementation.

One intention of these checklists is to present questions in a way that will cause the developer to pause and consider exactly how the SET Specification has evolved into a working product. These checklists are not meant to be exhaustive in terms of every potential flaw in a given application, a goal that is arguably unattainable anyway.

However, they are intended to be sufficiently general and relevant to provide the reviewer with a good overall view of the integrity of the development work that has gone into the particular application under review.

The questions are divided into sections following the organization of the associated report. For further discussion on any of the issues raised in this checklist, see the main report for more detail.

Checklist: Assumptions about the SET Environment

Number	Question
1.	Describe and document the features of the anticipated running environment such as how and where the SET implementation is anticipated to run, what operating system might be used, what files will be shared and by whom and what peripheral cryptographic hardware will be required.
2.	There are a number of issues that may affect the security of the SET application in an actual running environment. These issues are addressed in the questions below. • Is the operating system software in its original state? Is it feasible to modify the operating system? • Is the SET application software in its original state? Is it feasible to modify the SET application software? • Is it feasible for an opponent to examine or alter the internal memory space of a process during the execution of a process? • If cryptographic hardware is present, can the integrity and confidentiality of the path between the process and the hardware be vouched for? • Is it feasible for an opponent to alter the local clock? Is the local clock sufficiently accurate for the purpose of SET? • Will normal operating practice detect and remove intrusions such as computer viruses and other forms of software intrusion by which an opponent can alter the operation of the SET application? • Are the cryptographic boundaries of the trusted environment within which SET will reside clearly documented?
3.	To what extent are these issues a concern for the security of your application? To what extent are they uncertain in the actual running environments described in your answer to the previous question? Describe how any such uncertainties are addressed and what instructions if any are given to users of the application in dealing with them.

Checklist: Random number generation

Number	Question
1.	How many random number generators are used in the SET implementation? Which generators are used for the generation of which cryptographic quantities? For each random number generator in the SET implementation, describe the requirements for its strength based on the quantities that are derived from it and justify how the required strength is achieved, listing all assumptions on which the claimed strength is based.
2.	Describe exactly how the SET application will obtain the random bits necessary within SET. How are the different seeds used for the generation of different cryptographic quantities? Describe completely the methods by which any required seed is obtained. If the sampling of system events is required, provide documentary evidence that the individual events used are sufficiently unpredictable and independent as to provide the expected security.
3.	Complete the following checklist:

	yes	**no**
If a hardware source of random numbers is used, has its performance been assessed and certified?	☐	☐
If a hardware source of random numbers is used, are there checks to detect hardware failures?	☐	☐
If the technique of a random "pool" is used, are the confidentiality and integrity of the random pool guaranteed?	☐	☐
If this guarantee is provided cryptographically, are the necessary keys protected from disclosure or modification?	☐	☐

Note: If you answer "no" to any of the above questions, quantify and summarize the effect of this divergence from the general guidelines and their application to SET.

Checklist: Public Asymmetric Keys

Number	Question
1.	For each public asymmetric key in SET, indicate below what method or methods are used to protect it from modification. The available options are: • operating system controls (OSC), • message authentication codes (MAC), • digital signatures; for example, certificates, (DS), • some other method, or • not applicable; for example, for keys not handled by the implementation. Note that certificates are a natural way of protecting public keys in SET, since public keys are distributed in certificates, but that other ways are possible.

Public Key	OSC	MAC	DS	Other	N/A
Cardholder message signature	☐	☐	☐	☐	☐
Merchant message signature	☐	☐	☐	☐	☐
Merchant key-exchange	☐	☐	☐	☐	☐
Payment gateway message signature	☐	☐	☐	☐	☐
Payment gateway key-exchange	☐	☐	☐	☐	☐
Cardholder CA message signature	☐	☐	☐	☐	☐
Cardholder CA key-exchange	☐	☐	☐	☐	☐
Cardholder CA certificate	☐	☐	☐	☐	☐
Merchant CA message signature	☐	☐	☐	☐	☐
Merchant CA key-exchange	☐	☐	☐	☐	☐
Merchant CA certificate	☐	☐	☐	☐	☐
Payment gateway CA message signature	☐	☐	☐	☐	☐
Payment gateway CA key-exchange	☐	☐	☐	☐	☐
Payment gateway CA certificate issuing	☐	☐	☐	☐	☐
Payment gateway CA CRL	☐	☐	☐	☐	☐
Geo-political CA certificate issuing	☐	☐	☐	☐	☐
Geo-political CA CRL	☐	☐	☐	☐	☐
Brand CA certificate	☐	☐	☐	☐	☐
Brand CA CRL	☐	☐	☐	☐	☐
Root CA certificate	☐	☐	☐	☐	☐
Root CA CRL	☐	☐	☐	☐	☐

Continued on next page

Checklist: Public Asymmetric Keys, continued

Number	Question
2.	For those public keys protected by operating system controls in your implementation, • describe those controls; • describe the process by which the controls are applied and enforced, and • describe when the steps of application and enforcement of controls occur in the life cycle of the public keys.
3.	For those public keys protected by message authentication codes in your implementation, • describe the message authentication techniques used; • describe any assumptions on which their cryptographic strength is based; • describe the process by which the message authentication codes are generated and verified; • describe when the steps of generation and verification of the message authentication codes occur in the life cycle of the public keys; and • describe how the message authentication keys involved in those techniques are managed.
4.	For those public keys protected by digital signatures (certificates) in your implementation, • describe the digital signature techniques used; • describe any assumptions on which their cryptographic strength is based; • describe the process by which the digital signatures are generated and verified; • describe when the steps of generation and verification of the digital signatures occur in the life cycle of the public keys, and • describe how the signature generation and verification keys involved in those techniques are managed.

Continued on next page

Checklist: Public Asymmetric Keys, continued

Number	Question
5.	For those public keys that are protected by other techniques, describe the other techniques in detail and provide justification that the method chosen is both suitable and adequate for this particular application.
6.	There are a number of issues that may affect the security of the SET application in the way that public keys are protected irrespective of the method used. These issues include the following: • Are the data integrity mechanisms applied to the public key and its attributes together? • Is the strength of the integrity mechanism adequate for the class of key being protected? • If public keys are cached, are the proper cryptographic safeguards provided to ensure the integrity of public keys and attributes even if these properties are not checked again before use? To what extent are these issues a concern for the security of your application? Describe how any such issues are addressed in your application.

Checklist: Private Asymmetric Keys

Number	Question
1.	For each private asymmetric key in SET, indicate below what method or methods are used to protect it from disclosure or modification. The valid options are: • operating system controls (OSC) • symmetric encryption (SE) • asymmetric encryption (AE) • message authentication codes (MAC) • digital signatures (DS) • some other mechanism, or • not applicable; for example, for keys not handled by the implementation.

Private Asymmetric Key	OSC	SE	AE	MAC	DS	Other	N/A
Cardholder message signature	☐	☐	☐	☐	☐	☐	☐
Merchant message signature	☐	☐	☐	☐	☐	☐	☐
Merchant key-exchange	☐	☐	☐	☐	☐	☐	☐
Payment gateway message signature	☐	☐	☐	☐	☐	☐	☐
Payment gateway key-exchange	☐	☐	☐	☐	☐	☐	☐
Cardholder CA message signature	☐	☐	☐	☐	☐	☐	☐
Cardholder CA key-exchange	☐	☐	☐	☐	☐	☐	☐
Cardholder CA certificate	☐	☐	☐	☐	☐	☐	☐
Merchant CA message signature	☐	☐	☐	☐	☐	☐	☐
Merchant CA key-exchange	☐	☐	☐	☐	☐	☐	☐
Merchant CA certificate	☐	☐	☐	☐	☐	☐	☐
Payment gateway CA message signature	☐	☐	☐	☐	☐	☐	☐
Payment gateway CA key-exchange	☐	☐	☐	☐	☐	☐	☐
Payment gateway CA certificate issuing	☐	☐	☐	☐	☐	☐	☐
Payment gateway CA CRL	☐	☐	☐	☐	☐	☐	☐
Geo-political CA certificate issuing	☐	☐	☐	☐	☐	☐	☐
Geo-political CA CRL	☐	☐	☐	☐	☐	☐	☐
Brand CA certificate	☐	☐	☐	☐	☐	☐	☐
Brand CA CRL	☐	☐	☐	☐	☐	☐	☐
Root CA certificate	☐	☐	☐	☐	☐	☐	☐
Root CA CRL	☐	☐	☐	☐	☐	☐	☐

Continued on next page

Checklist: Private Asymmetric Keys, continued

Number	Question
2.	For those private asymmetric keys protected by operating system controls in your implementation, • describe those controls, • describe the process by which the controls are applied and enforced, and describe when the steps of application and enforcement of controls occur in the life cycle of the public keys.
3.	For those private asymmetric keys protected by symmetric encryption techniques in your implementation, • describe the symmetric encryption techniques used, • describe any assumptions on which their cryptographic strength is based; • describe the process by which the symmetric encryption is performed; • describe when the steps of encryption and decryption of the private asymmetric key occur in the life cycle of the keys, and describe how the secret keys involved in the symmetric encryption mechanisms are themselves managed.
4.	For those private asymmetric keys protected by asymmetric encryption techniques in your implementation, • describe the asymmetric encryption techniques used; • describe any assumptions on which their cryptographic strength is based; • describe the process by which the asymmetric encryption is performed; • describe when the steps of encryption and decryption of the private asymmetric key occur in the life cycle of the keys, and • describe how the encryption and decryption keys involved in the asymmetric encryption mechanisms are themselves managed.

Continued on next page

Checklist: Private Asymmetric Keys, continued

Number	Question
5.	For those private asymmetric keys protected by message authentication codes in your implementation, • describe the message authentication techniques used; • describe any assumptions on which their cryptographic strength is based; • describe the process by which the message authentication codes are generated and verified; • describe when the steps of generation and verification of the message authentication codes occur in the life cycle of the keys, and • describe how the message authentication keys involved in those techniques are managed.
6.	For those private asymmetric keys that are protected by digital signatures (certificates) in your implementation, • describe the digital signature techniques used; • describe any assumptions on which their cryptographic strength is based; • describe the process by which the digital signatures are generated and verified; • describe when the steps of generation and verification of the digital signatures occur in the life cycle of the keys, and • describe how the signature generation and verification keys involved in these digital signature techniques are themselves managed.
7.	For those private asymmetric keys that are protected by techniques not directly listed in the table above, describe these other techniques in detail and provide justification that the method chosen is both suitable and adequate for this particular application.

Checklist: Secret Symmetric Keys

Number	Question
1.	For each secret symmetric key in SET, indicate how it is protected from disclosure or modification. List all assumptions on which the claimed protection is based. Use the questions and tables provided in this Appendix as a guide. Note that in the expected environment of a SET implementation secret DES and CDMF keys are expected to be used once only on a per-message basis, so many of the issues pertinent to the protection of private asymmetric keys are unlikely to be applicable to this particular class of keys.

Checklist: All Cryptographic Keys

Number	Question
1.	If keys are shared among multiple applications, what techniques are in place to control key usage to avoid weaknesses arising from interactions between multiple uses? Even though one application might be internally consistent in its use of keys, are there any conflicts with another application?
2.	If secret-sharing techniques, or other threshold techniques such as signature sharing, are used to protect and control access to some cryptographically sensitive quantity, describe the exact circumstances under which the secret can be reconstructed or operations with the sensitive quantity can be performed. Justify the particular choice of access structure used and supply supporting documentation for the belief that the technique chosen offers adequate safeguards.

Checklist: Sensitive Data

Number	Question
1.	For each sensitive data item in SET, describe how the required protection is achieved, listing all assumptions on which the claimed protection is based. Among the data items that should be considered sensitive are: • key identification information; • the value of counters; • sensitive personal or purchase related data; and • the intermediate results from all computations.
2.	Indicate the methods by which sensitive data is protected in your implementation. Among the methods that should be considered are: • operating system controls; • symmetric encryption; • asymmetric encryption; • message authentication codes; • digital signatures, and • storage in secure memory.
3.	When sensitive data items are not protected by the techniques listed above, describe in detail the techniques that you have used in your particular application and provide justification that the method chosen is both suitable and adequate for this particular application.

Checklist: Memory Leakage

Number	Question
1.	Complete the following checklist.

	yes	**no**
Is all data related to a cryptographic operation treated as being potentially sensitive?	☐	☐
Is both memory allocated from the heap and memory allocated on the stack zeroized?	☐	☐
Are all registers zeroized before the process exits?	☐	☐
Is memory zeroized as soon as the data stored in it is no longer needed?	☐	☐
Is zeroization done with a procedure call which the compiler will not eliminate?	☐	☐
Do all returns from a procedure pass through a common exit that zeroizes memory?	☐	☐
Is memory containing sensitive data "locked' to prevent it from being swapped to a disk file, in a virtual memory system, if that disk file could later be examined by an opponent?	☐	☐
If memory need to be swapped, is it encrypted first with a key that remains in processor memory?	☐	☐

2.	If you answer "no" to any of the above questions quantify and summarize the effect of this divergence from the general guidelines and their application to SET.
3.	Provide full and thorough documentation on the way in which errors are handled within a SET application, and their potential impact on memory leakage.
4.	Describe any techniques that have been used as safeguards (for instance, scrambling memory containing sensitive data by encrypting with a key stored elsewhere in memory) against failure in the assumption that process memory is private.

Appendix R
Root Key

Root Key

Publication Note

As of the publication date of this specification, the Root Key has not yet been generated.

This appendix will be updated with the value of the Root Key and other authentication information as soon as it is available.

Continued on next page

Root Key, continued

This page reserved for Root key information.

Appendix S
Variations

Supported Variations

Overview

This appendix describes high-level processing variations (for example, optional cardholder certificates) that may exist in systems that support SET, in order to satisfy different business models and operating guidelines established by the brand, Acquirer, and merchant.

SET is designed to accommodate the following variations.

Cardholder certificates

For supporting initial acceptance, it was agreed that cardholder certificates may be optional at the discretion of an individual payment card brand.

Cardholder certificates will be required in the future:
The optional nature of the cardholder certificate is provided for initial implementation and acceptance only.

The payment gateway certificate will include an indicator as to whether a cardholder certificate is required for the selected **BrandID**; this will allow cardholder and payment gateway software to ensure a certificate is included with the purchase request when required. This approach also means that only payment gateways needed to be notified, via their certificates, when brand policies regarding the necessity of cardholder certificate changes.

Tunneled messages

Depending upon the operating guidelines specified by the brand, some brands may require the capability to send information back to cardholder via the merchant. This feature is intended for Issuers to communicate back to cardholders about the reason that a transaction is being declined or to request that the cardholder call the Issuer.

Capture token

Whether or not capture tokens are used depends on the business model and policy of the Acquirer. In addition, there are many variations in the business processing that can take place between the merchant and Acquirer depending upon their relationship and operating guidelines specified by the brand. A capture token may be included by the payment gateway when generating the **AuthRes** message.

Continued on next page

Supported Variations, continued

Returning PAN

Depending upon the relationship between the merchant and Acquirer and operating guidelines specified by the brand, this is an optional field that may be included by the Acquirer in several response messages to the merchant, if and only if, the merchant's certificate indicates it is authorized to receive cardholder's information. Merchants may include the PAN using "extra encryption" in the capture and credit messages and their reversals.

Batch capture processing

Depending upon the relationship between the Merchant and Acquirer, the Merchant can request capture processing for a batch of capture items to be processed together.

Split shipments, recurring, and installment payments

Subsequent authorizations due to split shipments originate from the merchant; recurring payments originate from the cardholder. The authorization response and authorization reversal response messages may contain an optional authorization token that the merchant can use for subsequent authorizations to support both split shipments and recurring payments. This variation in payment processing has implications on the order information and payment instructions and the hash computations performed by the cardholder and merchant.

Certificate and CRL signing

A CA may choose to use the same signature certificate to sign the certificates and CRLs that it generates, or may choose to use separate signature certificates for these two activities. When a separate certificate is used to sign CRLs, the **BasicConstraints.cA** field in this signature certificate is set to false and the **KeyUsage** field is set to 6. The complementary certificate used to sign certificates has **BasicConstraints.cA** set to true and **KeyUsage** set to 5.

Variations Not Addressed By SET

Variations outside the scope of SET

The variations in the table below are solely at the discretion of the brand or the financial institution, and SET makes neither provisions nor recommendations for them.

Certificate request verification	The method used to verify the information provided in a request for a SET certificate is specific to each financial institution's policies.
Certificate inquiry retention period	The maximum retention period for certificate authorities to re-send certificates in response to a certificate inquiry request message may vary according to system configuration dependencies.
SET initiation mechanism	The mechanism used to initiate a SET certificate or payment request transaction may varying depending on the network transport.
Communication of shopping-related information	The method used by the Cardholder and Merchant to accumulate the order description, its format and exchange mechanism is specific to the shopping application.
Confidentiality of purchase information	The protection of the purchase information (such as order description) using channel encryption mechanisms such as SSL.

Appendix T
Private Key and Certificate Duration

Overview

Introduction

The cryptoperiods for both certificate and private key durations are examples of security-related system configuration parameters that need to be considered when deploying SET systems.

Constraint

The private key associated with the certificate should expire before the certificate expires, allowing the public key in the certificate to be used to verify signatures after the private key has expired.

Private key duration

Example

Table 80 provides an example for maximum private-key duration.

Entity	Signature	Key-Encipherment	Certificate Signature	CRL Signature
Cardholder	3 years			
Merchant	1 year	1 year		
Payment Gateway	1 year	1 year		
Cardholder CA	1 year	1 year	1 year	
Merchant CA	1 year	1 year	1 year	
Payment Gateway CA	1 year	1 year	1 year	1 year
Geopolitical CA	1 year	1 year	1 year	1 year
Brand CA	1 year	1 year	1 year	1 year
Root CA	1 year	1 year	1 year	1 year

Table 80: Private Key Duration Example

Continued on next page

Private key duration, continued

Example

Table 81 provides an example for maximum certificate duration.

Entity	Signature	Key-Encipherment	Certificate Signature	CRL Signature
Cardholder	3 years			
Merchant	1 year	1 year		
Payment Gateway	1 year	1 year		
Cardholder CA	1 year	1 year	4 years	
Merchant CA	1 year	1 year	2 years	
Payment Gateway CA	1 year	1 year	2 years	2 years
Geopolitical CA			5 years	2 years
Brand CA			6 years	2 years
Root CA			7 years	2 years

Table 81: Certificate Duration Example

Duration Scenarios

Example

The following charts are examples of how the maximum duration for private keys and certificates is determined. The private-key duration Scenario takes the maximum private-key duration periods stated above, and maps the duration if each CA issues a certificate to an End Entity on the last day that the CA's private key can be used for signing a certificate.

The Certificate Duration Scenario maps the required certificate duration required to meet the certificate chain validation criteria, based on the given private-key duration.

Private Key Duration Scenario							
	Years						
Entity	1	2	3	4	5	6	7
Root CA	▨						
Brand CA		▨					
Geopolitical CA			▨				
Cardholder CA				▨			
Cardholder					▨	▨	▨
Merchant CA				▨			
Merchant					▨		
Payment Gateway CA				▨			
Payment Gateway					▨		

Certificate Duration Scenario							
	Years						
Entity	7	6	5	4	3	2	1
Root CA	▨	▨	▨	▨	▨	▨	▨
Brand CA		▨	▨	▨	▨	▨	▨
Geopolitical CA			▨	▨	▨	▨	▨
Cardholder CA				▨	▨	▨	▨
Cardholder					▨	▨	▨
Merchant CA				▨	▨		
Merchant					▨		
Payment Gateway CA				▨	▨		
Payment Gateway					▨		

Appendix U
Certificate Examples

Introduction

Purpose

This appendix includes an example cardholder certificate.

Format

For each message, the following information is provided:

- The data structures/fields name. Preceding dots (.) are used to show nesting.
- The content where applicable. (If the Data Structures/Fields is a construct, the contents octet(s) are not shown.)
- DER encoding.

Cardholder Certificate

This is a UnsignedCertificate data structure. The total length of the data structure is 693 bytes.

Data Structures/Fields	Content	DER encoding
`UnsignedCertificate`		`30 82 02 B1`
`.version`	`ver3(2)`	`A0 03 02 01 02`
`.serialNumber`	`22`	`02 01 16`
`.signature`		`30 0D`
`..algorithm`	`id-sha1-with-rsa-signature`	`06 09 2A 86 48 86 F7 0D 01 01 05`
`..parameters`	`null`	`05 00`
`.issuer`		`30 41`
`..countryName`		`31 0B 30 09`
`...type`	`id-at-countryName`	`06 03 55 04 06`
`...value`	`US`	`13 02 55 53`
`..organizationName`		`31 0E 30 0C`
`...type`	`id-at-organizationName`	`06 03 55 04 0A`
`...value`	`Brand`	`13 05 42 72 61 6E 64`
`..organizationUnitName`		`31 22 30 20`
`...type`	`id-at-organizationUnitName`	`06 03 55 04 0B`
`...value`	`Cardholder Certificate CA`	`13 19 43 61 72 64 68 6F 6C 64 65 72 20 43 65 72 74 69 66 69 63 61 74 65 20 43 41`
`.validity`		`30 1E`
`..notBefore`	`961126222439Z`	`17 0D 39 36 31 31 32 36 32 32 32 34 33 39 5A`
`..notAfter`	`971126235900Z`	`17 0D 39 37 31 31 32 36 32 33 35 39 30 30 5A`
`.subject`		`30 53`
`..countryName`		`31 0B 30 09`
`....type`	`id-at-countryName`	`06 03 55 04 06`
`....value`	`US`	`13 02 55 53`
`..organizationName`		`31 0E 30 0C`
`...type`	`id-at-organizationName`	`06 03 55 04 0A`
`...value`	`Brand`	`13 05 42 72 61 6E 64`
`..organizationUnitName`		`31 0D 30 0B`
`...type`	`id-at-organizationUnitName`	`06 03 55 04 0B`
`...value`	`Bank`	`13 04 42 61 6E 6B`
`..commonName`		`31 25 30 23`
`...type`	`id-at-commonName`	`06 03 55 04 06`
`...value`		`13 1C 43 61 72 64 68 6F 6C 64 65 72 39 39 39 39 39 39 30 31 32 33 34 35 36 37 38 38 43 65`

Continued on next page

Cardholder Certificate, continued

Data Structures/Fields	Content	DER encoding
`.subjectPublicKeyInfo`		`30 82 01 22`
`..algorithm`		`30 0D`
`...algorithm`	`id-rsaEncryption`	`06 09 2A 86 48 86 F7 0D 01 01 01`
`...parameters`	`null`	`05 00`
`..subjectPublicKey`		`03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 AC 0B 1D 55 77 4D 23 DE F7 0A 26 C6 BE 64 9E 9C 4F 0E B6 9B D2 19 43 95 3A 86 A0 D1 9A D4 FF 99 63 0D A3 F5 68 7D 5E F5 6C 9E 34 F5 ED 75 5C 47 FB 53 FE 9F 92 F0 E5 CE 95 60 44 EC D0 BA 25 A6 1F D1 65 7A BE B0 4D D6 85 97 AB 7D 2C AE FA 59 71 A1 AE 3C CD E9 DF 33 27 39 02 36 83 8E AE AB 8C 3F A0 C7 61 8D 78 22 24 CD 46 A1 25 84 43 B1 F7 5F B5 78 73 EE 1A 3E 4D D1 BB BA 06 64 D1 A4 FD 67 65 4D 06 F9 CA 28 AD 24 76 E3 99 7B 5F D1 A8 A0 3D 73 45 AB 52 30 53 02 1D 61 12 F1 F5 CA 94 97 FE 5C 15 DA F3 4A B0 5B 1F 9B 65 54 09 4A C1 EB AE D1 B7 6D E2 47 34 B5 C1 A1 49 A2 2D A5 76 F2 BD 02 0D D5 FF 9C 40 0E 34 CB A2 B1 D8 B0 BF 2C 2E 9B 11 C5 DD BB A6 5A 21 37 78 33 32 D3 DB 09 04 21 1F 65 04 25 FC CB A4 91 14 A4 09 E7 81 99 BD CF 4A C3 45 57 7E 59 B9 AE DB F5 74 A5 02 03 01 00 01`

Continued on next page

Cardholder Certificate, continued

Data Structures/Fields	Content	DER encoding
.extensions		A3 81 B9 30 81 B6
..keyUsage		30 0E
...extrnID	id-ce-keyUsage	06 03 55 1D 0F
...critical	TRUE	01 01 FF
...extnValue	digitalSignature(0)	04 04 03 02 01 80
..privateKeyUsagePeriod		30 2B
...extrnID	id-ce-privateKeyUsagePeriod	06 03 55 1D 10
...extnValue		04 24 30 22
....notBefore	19961126221453Z	80 0F 31 39 39 36 31 31 32 36 32 32 31 34 35 33 5A
....notAfter	19970826221453Z	81 0F 31 39 39 37 30 38 32 36 32 32 31 34 35 33 5A
..certificatePolicies		30 14
...extrnID	id-ce-certificatePolicies	06 03 55 1D 20
...critical	TRUE	01 01 FF
...extnValue		04 0A 30 08 30 06
....policyIdentifier	id-set-setQualifier	06 04 70 2A 07 06
..certificateType		30 10
...extrnID	id-set-certificateType	06 04 70 2A 07 01
...critical	TRUE	01 01 FF
...extnValue		04 05
	card(0)	03 03 07 80 00
..basicConstraints		30 0A
...extrnID		06 03 55 1D 13
...critical	TRUE	01 01 FF
...extnValue		04 00
..authorityKeyIdentifier		30 43
...extrnID		06 03 55 1D 23
...extnValue		04 3C 30 3A
....authorityCertIssuer		A1 34
.....directoryName		A4 32 30 30
......countryName		31 0B 30 09
......type	id-at-countryName	06 03 55 04 06
......value	US	13 02 55 53
.....organizationName		31 0E 30 0C
......type	id-at-organizationName	06 03 55 04 0A
......value	Brand	13 05 42 72 61 6E 64
.....organizationUnit Name		31 11 30 0F
......type	id-at-organizationUnitName	06 03 55 04 0B
......value	Brand CA	13 08 42 72 61 6E 64 20 43 41
....authorityCertSerial Number	4660	82 02 12 34

Appendix V
Message Examples

Introduction

Example messages

This appendix includes examples of the following constructs:

- **PInitReq**
- **PInitResData**
- **InqReqData**
- **OIData**
- **PIData**
- **PResData**
- **AuthReqData**
- **AuthResData**
- **AuthRevReqData**
- **AuthRevResData**
- **CapReqData**
- **CapResData**
- **CapRevData**
- **CapRevResData**
- **CredReqData**
- **CredResData**
- **CredRevReqData**
- **CredRevResData**
- **PCertReqData**
- **PCertResTBS**
- **BatchAdminReqData**
- **BatchAdminResData**
- **CardCInitReq**
- **CardCInitResTBS**
- **Me-AqCInitReq**
- **Me-AqCInitResTBS**
- **RegFormReqData**
- **RegFormTBS**
- **CertReqData**
- **CertResData**
- **CertInqReqTBS**
- **ErrorTBS**

Format

For each message, the following information is provided:

- The Data Structures/Fields name. Preceding dots (.) are used to show nesting.
- The content where applicable. (If the Data Structures/Fields is a construct, the contents octet(s) are not shown.)
- DER encoding.

PInitReq

This is a **PInitReq** message. The total length of the message is 214 bytes.

Data Structures/Fields	Content	DER encoding
MessageWrapper		30 81 D3
.header		30 5D
..version	setVer1(1)	02 01 01
..revision	0	02 01 00
..date	19970514041853Z	18 0F 31 39 39 37 30 35 31 34 30 34 31 38 35 33 5A
..messageIDs		A0 16
...localID-C		80 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..rrpid		81 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..swIdent	SET Specification v1.0	1A 16 53 45 54 20 53 70 65 63 69 66 69 63 61 74 69 6F 6E 20 76 31 2E 30
.message		A0 72
..purchaseInitRequest		A0 81 6F 30 6D
...rrpid		04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
...language	en	1A 03 65 6E 20
...localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
...chall-C		04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
...brandID	Brand:Product	1A 0D 42 72 61 6E 64 3A 50 72 6F 64 75 63 74
...bin	999999	12 06 39 39 39 39 39 39
...thumbs		A1 0D 30 0B
....digestAlgorithm		30 09
.....algorithm	id-sha1	06 05 2B 0E 03 02 1A
.....parameters	null	05 00

PInitResData

This is a **PInitResData** data structure to be signed in the **PInitRes** message. The total length of the data structure is 189 bytes.

Data Structures/Fields	Content	DER encoding
PInitResData		30 81 BA
.transIDs		30 42
..localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
..pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
..language	en	1A 03 65 6E 20
.rrpid		04 14 C9 36 C4 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.chall-C		04 14 CA 36 C4 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
. peThumbs		A1 23 30 21
..digestAlgorithm		30 09
...algorithm	id-sha1	06 05 2B 0E 03 02 1A
...parameters	null	05 00
..thumbprint		04 14 A6 A3 30 4C BC 0E BE 1F 85 A9 56 14 77 7D 8D 25 1F EF 06 02
.thumbs		A2 0D 30 0B
..digestAlgorithm		30 09
...algorithm	id-sha1	06 05 2B 0E 03 02 1A
...parameters	null	05 00

InqReqData

This is an **InqReqData** data structure to be signed/included in the **InqReq** message. The total length of the data structure is 114 bytes.

Data Structures/Fields	Content	DER encoding
`InqReqData`		`30 70`
`.transIDs`		`30 42`
`..localID-C`		`04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D`
`..xID`		`04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D`
`..pReqDate`	`19970509175416Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A`
`..language`	`en`	`1A 03 65 6E 20`
`.rrpid`		`04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.chall-C2`		`04 14 CB 37 D4 14 62 62 63 64 65 66 67 68 69 6A 6C 6C 6D 6E 6F 70`

OIData

This is an **OIData** data structure to be included in the **PReq** message. The total length of the data structure is 220 bytes.

Data Structures/Fields	Content	DER encoding
OIData		30 81 D9
.transIDs		30 42
..localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
..pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
..language	en	1A 03 65 6E 20
.rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.chall-C		04 14 CA 36 C4 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.hod		30 2C
..ddVersion	ddVer0(0)	02 01 00
..digestAlgorithm		30 09
...algorithm	id-sha1	06 05 2B 0E 03 02 1A
...parameters	null	05 00
..contentInfo		30 06
...contentType	id-set-content-OIData	06 04 70 2A 00 04
..digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
.odSalt		04 14 D2 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.chall-M		04 14 D3 66 CF 65 62 63 73 54 75 66 57 69 19 6E 60 2C 4D 2E 6F 10
.brandID	Brand:Product	1A 0D 42 72 61 6E 64 3A 50 72 6F 64 75 63 74

PIData

This is a **PIData** data structure to be encrypted in the **PReq** message. The total length of the data structure is 299 bytes.

Data Structures/Fields	Content	DER encoding
PIData		30 82 01 27
.piHead		30 81 DC
..transIDs		30 42
...localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
...xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	en	1A 03 65 6E 20
..inputs		30 3B
...hod		30 2C
....ddVersion	ddVer0(0)	02 01 00
....digestAlgorithm		30 09
.....algorithm	id-sha1	06 05 2B 0E 03 02 1A
.....parameters	null	05 00
....contentInfo		30 06
.....contentType	id-set-content-HODInput	06 04 70 2A 00 08
....digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
...purchAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
..merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
..transStain		04 14 18 29 34 4D 58 69 74 2D 38 49 24 D2 86 96 46 D2 88 79 74 3D
..swIdent	SET Specification v1.0	1A 16 53 45 54 20 53 70 65 63 69 66 69 63 61 74 69 6F 6E 20 76 31 2E 30
..acqBackInfo		A1 1F 30 1D
...backAlgID		30 11
....algorithm	id-desCBC	06 05 2B 0E 03 02 07
....parameters		04 08 CE 64 61 62 63 64 65 66
...backKey		04 08 42 52 69 1F 4C A7 9B 0E

Continued on next page

PIData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER Encoding
.panData		30 46
..pan	9999990123456788	12 10 39 39 39 39 39 39 30 31 32 33 34 35 36 37 38 38
..cardExpiry	199901	12 06 31 39 39 39 30 31
..panSecret		04 14 70 61 6E 73 65 63 72 65 74 70 61 6E 73 65 63 72 65 74 70 61
..exNonce		04 14 D3 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70

PResData

This is a **PResData** data structure to be signed the **PRes** message. The total length of the data structure is 178 bytes.

Data Structures/Fields	Content	DER encoding
PResData		30 81 AF
.transIDs		30 42
..localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
..pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
..language	en	1A 03 65 6E 20
.rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.chall-C		04 14 CA 36 C4 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.pResPayloadSeq		30 3D 30 3B
..completionCode	capturePerformed(4)	0A 01 04
..results		30 36
...authStatus		A1 19
....authDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
....authCode	approved(0)	0A 01 00
....authRatio	1	09 03 80 00 01
...capStatus		A2 19
....capDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
....capCode	success(0)	0A 01 00
....capRatio	1	09 03 80 00 01

AuthReqData

This is an **AuthReqData** data structure to be encrypted in the **AuthReq** message. The total length of the data structure is 258 bytes.

Data Structures/Fields	Content	DER encoding
AuthReqData		30 81 FF
.authReqItem		30 81 FC
..authTags		30 7B
...authRRTags		30 35
....rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
....merTermIDs		30 0C
.....merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
....currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...transIDs		30 42
....localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
....xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
....pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
....language	en	1A 03 65 6E 20
..checkDigests		A0 5C
...hOIData		30 2C
....ddVersion	ddVer0(0)	02 01 00
....digestAlgorithm		30 09
.....algorithm	id-sha1	06 05 2B 0E 03 02 1A
.....parameters	null	05 00
....contentInfo		30 06
.....contentType	id-set-content-OIData	06 04 70 2A 00 04
....digest		04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2
...hod2		30 2C
....ddVersion	ddVer0(0)	02 01 00
....digestAlgorithm		30 09
.....algorithm	id-sha1	06 05 2B 0E 03 02 1A
.....parameters	null	05 00
....contentInfo		30 06
.....contentType	id-set-content-HODInput	06 04 70 2A 00 08
....digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03

Continued on next page

AuthReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
..mThumbs		A1 0B
...digestAlgorithm		30 09
....algorithm	id-sha1	06 05 2B 0E 03 02 1A
....parameters	null	05 00
..authReqPayload		30 12
...subsequentAuthInd	FALSE	01 01 00
...authReqAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...merchData		30 00

AuthResData

This is an **AuthResData** data structure to be encrypted in the **AuthRes** message. The total length of the data structure is 163 bytes.

Data Structures/Fields	Content	DER encoding
AuthResData		30 81 A0
.authTags		30 7B
..authRRTags		30 35
...rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
...merTermIDs		30 0C
....merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
...currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
..transIDs		30 42
...localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
...xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	en	1A 03 65 6E 20
.authResPayload		30 21
..authHeader		30 1F
...authAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...authCode	approved(0)	0A 01 00
...responseData		30 0D
....authValCodes		A0 08
.....approvalCode	567891	80 06 35 36 37 38 39 31
....respReason	issuer(0)	81 01 00

AuthRevReqData

This is an **AuthRevReqData** data structure to be encrypted in the **AuthRevReq** message. The total length of the data structure is 354 bytes.

Data Structures/Fields	Content	DER encoding
AuthRevReqData	@826	30 82 01 5E
.authRevTags		30 37
..authRevRRTags		30 35
...rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
...merTermIDs		30 0C
....merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
...currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
.AuthReqData		A1 81 FF
..authReqItem		30 81 FC
...authTags		30 7B
....authRRTags		30 35
.....rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.....merTermIDs		30 0C
......merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
.....currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
....transIDs		30 42
.....localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
.....xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
.....pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
.....language	en	1A 03 65 6E 20
...checkDigests		A0 5C
....hOIData		30 2C
.....ddVersion	ddVer0(0)	02 01 00
.....digestAlgorithm		30 09
......algorithm	id-sha1	06 05 2B 0E 03 02 1A
......parameters	null	05 00
.....contentInfo		30 06
......contentType	id-set-content-OIData	06 04 70 2A 00 04
.....digest		04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2

Continued on next page

AuthRevReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
....hod2		30 2C
.....ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
......algorithm	id-sha1	06 05 2B 0E 03 02 1A
......parameters	null	05 00
.....contentInfo		30 06
......contentType	id-set-content-HODInput	06 04 70 2A 00 08
.....digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
...mThumbs		A1 0B
....digestAlgorithm		30 09
.....algorithm	id-sha1	06 05 2B 0E 03 02 1A
.....parameters	null	05 00
...authReqPayload		30 12
....subsequentAuthInd	FALSE	01 01 00
....authReqAmt		30 0B
.....currency	840(US)	02 02 08 40
.....amount	3059	02 02 0B B3
.....amtExp10	-2	02 01 FE
....merchData		30 00
.authResPayload		A2 21
..authHeader		30 1F
...authAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...authCode	approved(0)	0A 01 00
...responseData		30 0D
....authValCodes		A0 08
.....approvalCode	567891	80 06 35 36 37 38 39 31
....respReason	issuer(0)	81 01 00

AuthRevResData

This is an **AuthRevResData** data structure to be encrypted in the **AuthRevRes** message. The total length of the data structure is 177 bytes.

Data Structures/Fields	Content	DER encoding
AuthRevResData	@857	30 81 AE
.authRevTags		30 37
..authRevRRTags		30 35
...rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
...merTermIDs		30 0C
....merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
...currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
.authNewAmt		30 0A
..currency	840(US)	02 02 08 40
..amount	0	02 01 00
..amtExp10	0	02 01 00
.authResDataNew		30 67
..transIDs		30 42
...localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
...xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	en	1A 03 65 6E 20
..authResPayload		30 21
...authHeader		30 1F
....authAmt		30 0B
.....currency	840(US)	02 02 08 40
.....amount	3059	02 02 0B B3
.....amtExp10	-2	02 01 FE
....authCode	approved(0)	0A 01 00
....responseData		30 0D
.....authValCodes		A0 08
......approvalCode	567891	80 06 35 36 37 38 39 31
.....respReason	issuer(0)	81 01 00

CapReqData

This is a **CapReqData** data structure to be encrypted in the **CapReq** message. The total length of the data structure is 477 bytes.

Data Structures/Fields	Content	DER encoding
CapReqData		30 82 01 D9
.capRRTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.mThumbs		A0 0B 30 09
..digestAlgorithm		06 05 2b 0e 03 02 1a
..parameters	null	05 00
.capItemSeq		30 82 01 91 30 82 01 8D
..transIDs		30 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capPayload		30 82 01 45
...capDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...capReqAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE

Continued on next page

CapReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
...authReqItem		A0 81 FF 30 81 FC
....authTags		30 7B
.....authRRTags		30 35
......rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
......merTermIDs		30 0C
.......merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
......currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
.....transIDs		30 42
......localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
......xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
......pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
......language	en	1A 03 65 6E 20
....checkDigests		A0 5C
.....hOIData		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-OIData	06 04 70 2A 00 04
......digest		04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2
.....hod2		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-HODInput	06 04 70 2A 00 08
......digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
....mThumbs		A1 0B
.....digestAlgorithm		30 09
......algorithm	id-sha1	06 05 2B 0E 03 02 1A
......parameters	null	05 00

Continued on next page

CapReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
`....authReqPayload`		`30 12`
`.....subsequentAuthInd`	`FALSE`	`01 01 00`
`.....authReqAmt`		`30 0B`
`......currency`	`840(US)`	`02 02 08 40`
`......amount`	`3059`	`02 02 0B B3`
`......amtExp10`	`-2`	`02 01 FE`
`.....merchData`		`30 00`
`...authResPayload`		`A1 23 30 21`
`....authHeader`		`30 1F`
`.....authAmt`		`30 0B`
`......currency`	`840(US)`	`02 02 08 40`
`......amount`	`3059`	`02 02 0B B3`
`......amtExp10`	`-2`	`02 01 FE`
`.....authCode`	`approved(0)`	`0A 01 00`
`.....responseData`		`30 0D`
`......authValCodes`		`A0 08`
`.......approvalCode`	`567891`	`80 06 35 36 37 38 39 31`
`......respReason`	`issuer(0)`	`81 01 00`

CapResData

This is a **CapResData** data structure to be encrypted in the **CapRes** message. The total length of the data structure is 152 bytes.

Data Structures/Fields	Content	DER encoding
CapResData		30 81 95
.capRRTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capResItemSeq		30 5C 30 5A
..transIDs		30 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capResPayload		30 14
...capCode	success(0)	0A 01 00
...capAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...batchID	102	80 02 01 02

CapRevData

This is a **CapRevData** data structure to be encrypted in the **CapRevReq** message. The total length of the data structure is 485 bytes.

Data Structures/Fields	Content	DER encoding
CapRevData		A0 82 01 DF 30 82 01 DB
.capRevOrCredTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capRevOrCredReqItems		30 82 01 A0 30 82 01 9C
..transIDs		A0 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capPayload		30 82 01 45
...capDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...capReqAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...authReqItem		A0 81 FF 30 81 FC
....authTags		30 7B
.....authRRTags		30 35
......rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
......merTermIDs		30 0C
.......merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
......currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A

Continued on next page

CapRevData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
`.....transIDs`		`30 42`
`......localID-C`		`04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D`
`......xID`		`04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D`
`......pReqDate`	`19970509175416Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A`
`......language`	`en`	`1A 03 65 6E 20`
`....checkDigests`		`A0 5C`
`.....hOIData`		`30 2C`
`......ddVersion`	`ddVer0(0)`	`02 01 00`
`......digestAlgorithm`		`30 09`
`.......algorithm`	`id-sha1`	`06 05 2B 0E 03 02 1A`
`.......parameters`	`null`	`05 00`
`......contentInfo`		`30 06`
`.......contentType`	`id-set-content-OIData`	`06 04 70 2A 00 04`
`......digest`		`04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2`
`.....hod2`		`30 2C`
`......ddVersion`	`ddVer0(0)`	`02 01 00`
`......digestAlgorithm`		`30 09`
`.......algorithm`	`id-sha1`	`06 05 2B 0E 03 02 1A`
`.......parameters`	`null`	`05 00`
`......contentInfo`		`30 06`
`.......contentType`	`id-set-content-HODInput`	`06 04 70 2A 00 08`
`......digest`		`04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03`
`....mThumbs`		`A1 0B`
`.....digestAlgorithm`		`30 09`
`......algorithm`	`id-sha1`	`06 05 2B 0E 03 02 1A`
`......parameters`	`null`	`05 00`
`....authReqPayload`		`30 12`
`.....subsequentAuthInd`	`FALSE`	`01 01 00`
`.....authReqAmt`		`30 0B`
`......currency`	`840(US)`	`02 02 08 40`
`......amount`	`3059`	`02 02 0B B3`
`......amtExp10`	`-2`	`02 01 FE`
`.....merchData`		`30 00`

Continued on next page

CapRevData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
...authResPayload		A1 23 30 21
....authHeader		30 1F
.....authAmt		30 0B
......currency	840(US)	02 02 08 40
......amount	3059	02 02 0B B3
......amtExp10	-2	02 01 FE
.....authCode	approved(0)	0A 01 00
.....responseData		30 0D
......authValCodes		A0 08
.......approvalCode	567891	80 06 35 36 37 38 39 31
......respReason	issuer(0)	81 01 00
..capRevOrCredReqDate	19970509175417Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 37 5A

CapRevResData

This is a **CapRevResData** data structure to be encrypted in the **CapRevRes** message. The total length of the data structure is 148 bytes.

Data Structures/Fields	Content	DER encoding
CapRevResData		30 81 91
.capRevOrCredTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capRevOrCredResItems		30 58 30 56
..transIDs		A0 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capRevOrCredResPayload		30 10
...capRevOrCredCode	success(0)	02 01 00
...capRevOrCredActualAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE

CredReqData

This is a **CredReqData** data structure to be encrypted in the **CredReq** message. The total length of the data structure is 485 bytes.

Data Structures/Fields	Content	DER encoding
CredReqData		A1 82 01 DF 30 82 01 DB
.capRevOrCredTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capRevOrCredReqItems		30 82 01 A0 30 82 01 9C
..transIDs		A0 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capPayload		30 82 01 45
...capDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...capReqAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...authReqItem		A0 81 FF 30 81 FC
....authTags		30 7B
.....authRRTags		30 35
......rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
......merTermIDs		30 0C
.......merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
......currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A

Continued on next page

CredReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
.....transIDs		30 42
......localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
......xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
......pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
......language	en	1A 03 65 6E 20
....checkDigests		A0 5C
.....hOIData		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-OIData	06 04 70 2A 00 04
......digest		04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2
.....hod2		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-HODInput	06 04 70 2A 00 08
......digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
....mThumbs		A1 0B
.....digestAlgorithm		30 09
......algorithm	id-sha1	06 05 2B 0E 03 02 1A
......parameters	null	05 00
....authReqPayload		30 12
.....subsequentAuthInd	FALSE	01 01 00
.....authReqAmt		30 0B
......currency	840(US)	02 02 08 40
......amount	3059	02 02 0B B3
......amtExp10	-2	02 01 FE
.....merchData		30 00

Continued on next page

CredReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
...authResPayload		A1 23 30 21
....authHeader		30 1F
.....authAmt		30 0B
......currency	840(US)	02 02 08 40
......amount	3059	02 02 0B B3
......amtExp10	-2	02 01 FE
.....authCode	approved(0)	0A 01 00
.....responseData		30 0D
......authValCodes		A0 08
.......approvalCode	567891	80 06 35 36 37 38 39 31
......respReason	issuer(0)	81 01 00
..capRevOrCredReqDate	19970509175417Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 37 5A

CredResData

This is a **CredResData** data structure to be encrypted in the **CredRes** message. The total length of the data structure is 148 bytes.

Data Structures/Fields	Content	DER encoding
`CredResData`		30 81 91
`.capRevOrCredTags`		30 35
`..rrpid`		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
`..merTermIDs`		30 0C
`...merchantID`	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
`..currentDate`	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
`.capRevOrCredResItems`		30 58 30 56
`..transIDs`		A0 42
`...localID-C`		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
`...xID`		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
`...pReqDate`	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
`...language`	19970509175416Z	1A 03 65 6E 20
`..capRevOrCredResPayload`		30 10
`...capRevOrCredCode`	success(0)	02 01 00
`...capRevOrCredActualAmt`		30 0B
`....currency`	840(US)	02 02 08 40
`....amount`	3059	02 02 0B B3
`....amtExp10`	-2	02 01 FE

CredRevReqData

This is a **CredRevReqData** data structure to be encrypted in the **CredRevReq** message. The total length of the data structure is 485 bytes.

Data Structures/Fields	Content	DER encoding
CredRevReqData		A2 82 01 DF 30 82 01 DB
.capRevOrCredTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capRevOrCredReqItems		30 82 01 A0 30 82 01 9C
..transIDs		A0 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	19970509175416Z	1A 03 65 6E 20
..capPayload		30 82 01 45
...capDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...capReqAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE
...authReqItem		A0 81 FF 30 81 FC
....authTags		30 7B
.....authRRTags		30 35
......rrpid		04 14 D1 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
......merTermIDs		30 0C
.......merchantID	MerchantID	13 0A 4D 65 72 63 68 61 6E 74 49 44
......currentDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A

Continued on next page

CredRevReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
.....transIDs		30 42
......localID-C		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
......xID		04 14 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D 78 69 64 2D
......pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
......language	en	1A 03 65 6E 20
....checkDigests		A0 5C
.....hOIData		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-OIData	06 04 70 2A 00 04
......digest		04 14 8F 34 3E AC 28 EB BF 6C B0 38 CD C0 93 79 E1 23 70 85 3C A2
.....hod2		30 2C
......ddVersion	ddVer0(0)	02 01 00
......digestAlgorithm		30 09
.......algorithm	id-sha1	06 05 2B 0E 03 02 1A
.......parameters	null	05 00
......contentInfo		30 06
.......contentType	id-set-content-HODInput	06 04 70 2A 00 08
......digest		04 14 FB 7C C8 2F 80 B3 00 86 D2 60 84 29 36 69 05 70 CD CB 61 03
....mThumbs		A1 0B
.....digestAlgorithm		30 09
......algorithm	id-sha1	06 05 2B 0E 03 02 1A
......parameters	null	05 00
....authReqPayload		30 12
.....subsequentAuthInd	FALSE	01 01 00
.....authReqAmt		30 0B
......currency	840(US)	02 02 08 40
......amount	3059	02 02 0B B3
......amtExp10	-2	02 01 FE
.....merchData		30 00

Continued on next page

CredRevReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
...authResPayload		A1 23 30 21
....authHeader		30 1F
.....authAmt		30 0B
......currency	840(US)	02 02 08 40
......amount	3059	02 02 0B B3
......amtExp10	-2	02 01 FE
.....authCode	approved(0)	0A 01 00
.....responseData		30 0D
......authValCodes		A0 08
.......approvalCode	567891	80 06 35 36 37 38 39 31
......respReason	issuer(0)	81 01 00
..capRevOrCredReqDate	19970509175417Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 37 5A

CredRevResData

This is a **CredRevResData** data structure to be encrypted in the **CredRevRes** message. The total length of the data structure is 148 bytes.

Data Structures/Fields	Content	DER encoding
CredRevResData		30 81 91
.capRevOrCredTags		30 35
..rrpid		04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..merTermIDs		30 0C
...merchantID	MerchantID	13 0A 4d 65 72 63 68 61 6e 74 49 44
..currentDate	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
.capRevOrCredResItems		30 58 30 56
..transIDs		A0 42
...localID-C		04 14 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d 6c 69 64 63 2d
...xID		04 14 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d 78 69 64 2d
...pReqDate	19970509175416Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 34 31 36 5A
...language	en	1A 03 65 6E 20
..capRevOrCredResPayload		30 10
...capRevOrCredCode	success(0)	02 01 00
...capRevOrCredActualAmt		30 0B
....currency	840(US)	02 02 08 40
....amount	3059	02 02 0B B3
....amtExp10	-2	02 01 FE

PCertReqData

This is a **PCertReqData** data structure to be signed in the **PCertReq** message. The total length of the data structure is 84 bytes.

Data Structures/Fields	Content	DER encoding
`PCertReqData`		`30 52`
`.pCertTags`		`30 35`
`..rrpid`		`04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`..merTermIDs`		`30 0C`
`...merchantID`	`MerchantID`	`13 0A 4d 65 72 63 68 61 6e 74 49 44`
`..currentDate`	`19970509175510Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a`
`.brandIDSeq`		`30 19 30 17`
`..brandID`	`Brand:Product`	`1A 0D 42 72 61 6E 64 3A 50 72 6F 64 75 63 74`
`..bin`	`999999`	`12 06 39 39 39 39 39 39`

PCertResTBS

This is a **PCertResTBS** data structure to be signed in the **PCertRes** message. The total length of the data structure is 60 bytes.

Data Structures/Fields	Content	DER encoding
`PCertResTBS`		`30 3A`
`.pCertTags`		`30 35`
`..rrpid`		`04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`..merTermIDs`		`30 0C`
`...merchantID`	`MerchantID`	`13 0A 4d 65 72 63 68 61 6e 74 49 44`
`..currentDate`	`19970509175510Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a`
`.pCertCode`	`success(0)`	`02 01 00`

BatchAdminReqData

This is a **BatchAdminReqData** data structure to be encrypted in the **BatchAdminReq** message. The total length of the data structure is 64 bytes.

Data Structures/Fields	Content	DER encoding
`BatchAdminReqData`		`30 3E`
`.batchAdminRRTags`		`30 35`
`..rrpid`		`04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`..merTermIDs`		`30 0C`
`...merchantID`	`MerchantID`	`13 0A 4d 65 72 63 68 61 6e 74 49 44`
`..currentDate`	`19970509175510Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a`
`.batchID`	`103`	`80 02 01 03`
`.batchOperation`	`open(0)`	`82 01 00`

BatchAdminResData

This is a **BatchAdminResData data** structure to be encrypted in the **BatchAdminRes** message. The total length of the data structure is 64 bytes.

Data Structures/Fields	Content	DER encoding
`BatchAdminResData`		`30 3E`
`.batchAdminRRTags`		`30 35`
`..rrpid`		`04 14 D1 65 FB 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`..merTermIDs`		`30 0C`
`...merchantID`	`MerchantID`	`13 0A 4d 65 72 63 68 61 6e 74 49 44`
`..currentDate`	`19970509175510Z`	`18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a`
`.batchID`	`103`	`02 02 01 03`
`.baStatus`	`success(0)`	`02 01 00`

CardCInitReq

This is a **CardCInitReq** message. The total length of the message is 100 bytes.

Data Structures/Fields	Content	DER encoding
`CardCInitReq`		`30 62`

`.rrpid`		`04 14 87 FB 2B 3D 61 62`
		`63 64 65 66 67 68 69 6A`
		`6B 6C 6D 6E 6F 70`

`.eeTags`		`30 2C`

`..localID-EE`		`04 14 6C 69 64 63 2D 6C`
		`69 64 63 2D 6C 69 64 63`
		`2D 6C 69 64 63 2D`

`..chall-EE`		`04 14 88 FB 2B 3D 61 62`
		`63 64 65 66 67 68 69 6A`
		`6B 6C 6D 6E 6F 70`

`.brandID`	`Brand:Product`	`1A 0D 42 72 61 6E 64 3A`
		`50 72 6F 64 75 63 74`

`.thumbs`		`A0 0D 30 0B`
`..digestAlgorithm`		`30 09`
`...algorithm`	`id-sha1`	`06 05 2B 0E 03 02 1A`
`...parameters`	`null`	`05 00`

CardCInitResTBS

This is a **CardCInitResTBS** data structure to be signed in the **CardCInitRes** message. The total length of the data structure is 116 bytes.

Data Structures/Fields	Content	DER encoding
`CardCInitResTBS`		`30 72`

`.rrpid`		`04 14 87 FB 2B 3D 61 62`
		`63 64 65 66 67 68 69 6A`
		`6B 6C 6D 6E 6F 70`

`.eeTags`		`30 2C`

`..localID-EE`		`04 14 6C 69 64 63 2D 6C`
		`69 64 63 2D 6C 69 64 63`
		`2D 6C 69 64 63 2D`

`..chall-EE`		`04 14 88 FB 2B 3D 61 62`
		`63 64 65 66 67 68 69 6A`
		`6B 6C 6D 6E 6F 70`

`.caTags`		`30 2C`

`..lID-CA`		`80 14 7C 79 74 73 3D 7C`
		`6A 65 64 2E 6D 6A 65 64`
		`1D 5C 59 54 53 1D`

`..chall-CA`		`04 14 18 AB 2D 4D 51 62`
		`62 64 67 76 77 18 29 3A`
		`64 65 66 68 8F 90`

Me-AqCInitReq

This is a **Me-AqCInitReq** message. The total length of the message is 115 bytes.

Data Structures/Fields	Content	DER encoding
`Me-AqCInitReq`		`30 71`
`.rrpid`		`04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.eeTags`		`30 2C`
`..localID-EE`		`04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D`
`..chall-EE`		`04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.requestType`	`merInitialSig(4)`	`02 01 04`
`.idData`		`A0 14`
`..merchantBIN`	`999999`	`12 06 39 39 39 39 39 39`
`..merchantID`	`MerchantID`	`13 0A 4D 65 72 63 68 61 6E 74 49 44`
`.brandID`	`Brand:Product`	`1A 0D 42 72 61 6E 64 3A 50 72 6F 64 75 63 74`
`.language`	`en`	`1A 03 65 6E 20`

Me-AqCInitResTBS

This is a **Me-AqCInitResTBS** data structure to be signed in the **Me-AqCInitRes** message. The total length of the data structure is 256 bytes.

Data Structures/Fields	Content	DER encoding
`Me-AqCInitResTBS`		`30 81 FD`
`.rrpid`		`04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.eeTags`		`30 2C`
`..localID-EE`		`04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D`
`..chall-EE`		`04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.caTags2`		`30 2C`
`..lID-CA`		`80 14 7C 79 74 73 3D 7C 6A 65 64 2E 6D 6A 65 64 1D 5C 59 54 53 1D`
`..chall-CA`		`04 14 18 AB 2D 4D 51 62 62 64 67 76 77 18 29 3A 64 65 66 68 8F 90`
`.requestType`	`merInitialSig(4)`	`02 01 04`
`.regFormOrReferral`		`A0 81 85`
`..reqTemplate`		`30 75`
`...regFormID`	`1`	`02 01 01`
`...brandLogoURL`	`http://www.brand.com/~SET/logo.gif`	`1A 22 68 74 74 70 3A 2F 2F 77 77 77 2E 62 72 61 6E 64 2E 63 6F 6D 2F 7E 53 45 54 2F 6C 6F 67 6F 2E 67 69 66`
`...regFieldSeq`		`30 4C`
`....RegField`		`30 19`
`.....fieldID`	`{id-set-givenName 0}`	`80 05 70 2A 02 01 00`
`.....fieldName`	`First Name`	`1A 0A 46 69 72 73 74 20 4E 61 6D 65`
`.....fieldLen`	`20`	`02 01 14`
`.....fieldRequired`	`TRUE`	`82 01 FF`
`....RegField`		`30 18`
`.....fieldID`	`{id-set-familyName 0}`	`80 05 70 2A 02 02 00`
`.....fieldName`	`Last Name`	`1A 09 4C 61 73 74 20 4E 61 6D 65`
`.....fieldLen`	`20`	`02 01 14`
`.....fieldRequired`	`TRUE`	`82 01 FF`
`....RegField`		`30 15`
`.....fieldID`	`{id-set-identificationNumber 1 840 0}`	`80 08 70 2A 02 05 01 86 48 00`
`.....fieldName`	`SSN`	`1A 03 53 53 4E`
`.....fieldLen`	`9`	`02 01 09`
`.....fieldRequired`	`TRUE`	`82 01 FF`
`..policy`	`Brand Policy`	`1A 0C 42 72 61 6E 64 20 50 6F 6C 69 63 79`

RegFormReqData

This is a **RegFormReqData** data structure to be encrypted in the **RegFormReg** message. The total length of the data structure is 124 bytes.

Data Structures/Fields	Content	DER encoding
RegFormReqData		30 7A
.rrpid		04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.requestType	cardInitialSig(1)	02 01 01
.eeTags		30 2C
..localID-EE		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..chall-EE		04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.caTags2		30 2C
..lID-CA		80 14 7C 79 74 73 3D 7C 6A 65 64 2E 6D 6A 65 64 1D 5C 59 54 53 1D
..chall-CA		04 14 18 AB 2D 4D 51 62 62 64 67 76 77 18 29 3A 64 65 66 68 8F 90
.language	en	1A 03 65 6E 20

RegFormTBS

This is a **RegFormTBS** data structure to be signed in the **RegFormRes** message. The total length of the data structure is 256 bytes.

Data Structures/Fields	Content	DER encoding
RegFormTBS		30 81 FD
.rrpid		04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.eeTags2		30 2C
..localID-EE		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..chall-EE		04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.caTags2		30 2C
..lID-CA		80 14 7C 79 74 73 3D 7C 6A 65 64 2E 6D 6A 65 64 1D 5C 59 54 53 1D
..chall-CA		04 14 18 AB 2D 4D 51 62 62 64 67 76 77 18 29 3A 64 65 66 68 8F 90
.requestType	cardInitialSig(1)	02 01 01
.formOrReferral		30 81 85
..reqTemplate		30 75
...regFormID	1	02 01 01
...brandLogoURL	http://www.brand.com/~SET/logo.gif	1A 22 68 74 74 70 3A 2F 2F 77 77 77 2E 62 72 61 6E 64 2E 63 6F 6D 2F 7E 53 45 54 2F 6C 6F 67 6F 2E 67 69 66
...regFieldSeq		30 4C
....RegField		30 19
.....fieldID	{id-set-givenName 0}	80 05 70 2A 02 01 00
.....fieldName	First Name	1A 0A 46 69 72 73 74 20 4E 61 6D 65
.....fieldLen	20	02 01 14
.....fieldRequired	TRUE	82 01 FF
....RegField		30 18
.....fieldID	{id-set-familyName 0}	80 05 70 2A 02 02 00
.....fieldName	Last Name	1A 09 4C 61 73 74 20 4E 61 6D 65
.....fieldLen	20	02 01 14
.....fieldRequired	TRUE	82 01 FF
....RegField		30 15
.....fieldID	{id-set-identificationNumber 1 840 0}	80 08 70 2A 02 05 01 86 48 00
.....fieldName	SSN	1A 03 53 53 4E
.....fieldLen	9	02 01 09
.....fieldRequired	TRUE	82 01 FF
..policy	Brand Policy	1A 0C 42 72 61 6E 64 20 50 6F 6C 69 63 79

CertReqData

This is a **CertReqData** data structure to be encrypted in the **CertReq** message. The total length of the data structure is 503 bytes.

Data Structures/Fields	Content	DER encoding
`CertReqData`		30 82 01 F3
`.rrpid`		04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
`.requestType`	cardInitialSig(1)	02 01 01
`.requestDate`	19970509175510Z	18 0F 31 39 39 37 30 35 30 39 31 37 35 35 31 30 5a
`.eeTags3`		30 2C
`..localID-EE`		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
`..chall-EE`		04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
`.caTags2`		30 2C
`..lID-CA`		80 14 7C 79 74 73 3D 7C 6A 65 64 2E 6D 6A 65 64 1D 5C 59 54 53 1D
`..chall-CA`		04 14 18 AB 2D 4D 51 62 62 64 67 76 77 18 29 3A 64 65 66 68 8F 90
`.regFormID`	121	02 01 79
`.regForm`		30 3A
`..RegFormItems`		30 12
`...fieldName`	First Name	1A 0A 46 69 72 73 74 20 4E 61 6D 65
`...fieldValue`	Tony	1A 04 54 6F 6E 79
`..RegFormItems`		30 12
`...fieldName`	Last Name	1A 09 4C 61 73 74 20 4E 61 6D 65
`...fieldValue`	Lewis	1A 05 4C 65 77 69 73
`..RegFormItems`		30 10
`...fieldName`	SSN	1A 03 53 53 4E
`...fieldValue`	111223333	1A 09 31 31 31 32 32 33 33 33 33

Continued on next page

CertReqData, continued

DER encoding (continued)

Data Structures/Fields	Content	DER encoding
.publicKeySorE		30 82 01 2A
..publicKeyS		A0 82 01 26 30 82 01 22
..algorithm		30 0D
...algorithm	id-rsaEncryption	06 09 2A 86 48 86 F7 0D 01 01 01
...parameters	null	05 00
..subjectPublicKey		03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 AC 0B 1D 55 77 4D 23 DE F7 0A 26 C6 BE 64 9E 9C 4F 0E B6 9B D2 19 43 95 3A 86 A0 D1 9A D4 FF 99 63 0D A3 F5 68 7D 5E F5 6C 9E 34 F5 ED 75 5C 47 FB 53 FE 9F 92 F0 E5 CE 95 60 44 EC D0 BA 25 A6 1F D1 65 7A BE B0 4D D6 85 97 AB 7D 2C AE FA 59 71 A1 AE 3C CD E9 DF 33 27 39 02 36 83 8E AE AB 8C 3F A0 C7 61 8D 78 22 24 CD 46 A1 25 84 43 B1 F7 5F B5 78 73 EE 1A 3E 4D D1 BB BA 06 64 D1 A4 FD 67 65 4D 06 F9 CA 28 AD 24 76 E3 99 7B 5F D1 A8 A0 3D 73 45 AB 52 30 53 02 1D 61 12 F1 F5 CA 94 97 FE 5C 15 DA F3 4A B0 5B 1F 9B 65 54 09 4A C1 EB AE D1 B7 6D E2 47 34 B5 C1 A1 49 A2 2D A5 76 F2 BD 02 0D D5 FF 9C 40 0E 34 CB A2 B1 D8 B0 BF 2C 2E 9B 11 C5 DD BB A6 5A 21 37 78 33 32 D3 DB 09 04 21 1F 65 04 25 FC CB A4 91 14 A4 09 E7 81 99 BD CF 4A C3 45 57 7E 59 B9 AE DB F5 74 A5 02 03 01 00 01

CertResData

This is a **CertResData** data structure to be encrypted or signed in the **CertRes** message. The total length of the data structure is 153 bytes.

Data Structures/Fields	Content	DER encoding
CertResData		30 81 96
.rrpid		04 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.eeTags3		30 2C
..localID-EE		04 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D
..chall-EE		04 14 88 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
.caTags3		30 2C
..lID-CA		80 14 7C 79 74 73 3D 7C 6A 65 64 2E 6D 6A 65 64 1D 5C 59 54 53 1D
..chall-CA		04 14 18 AB 2D 4D 51 62 62 64 67 76 77 18 29 3A 64 65 66 68 8F 90
.certStatus		30 22
..certStatusCode	requestComplete(1)	02 01 01
..nonceCCA		04 14 D3 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
..failedItems		30 07
...FailedItem		30 05
....itemNumber	0	02 01 00
....reason		1A 00

CertInqReqTBS

This is a **CertInqReqTBS** data structure to be signed in the **CertInqReq** message. The total length of the data structure is 70 bytes.

Data Structures/Fields	Content	DER encoding
`CertInqReqTBS`		`30 44`

`.rrpid`		`04 14 87 FB 2B 3D 61 62`
		`63 64 65 66 67 68 69 6A`
		`6B 6C 6D 6E 6F 70`

`.caTags3`		`30 2C`

`..lID-CA`		`80 14 7C 79 74 73 3D 7C`
		`6A 65 64 2E 6D 6A 65 64`
		`1D 5C 59 54 53 1D`

`..chall-CA`		`04 14 18 AB 2D 4D 51 62`
		`62 64 67 76 77 18 29 3A`
		`64 65 66 68 8F 90`

ErrorTBS

This is an **ErrorTBS** data structure to be signed in the **Error** message. The total length of the data structure is 126 bytes.

Data Structures/Fields	Content	DER encoding
`ErrorTBS`		`30 7C`
`.errorCode`	`badMessageHeader(7)`	`02 01 07`
`.errorNonce`		`04 14 D3 65 CE 64 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`.errorMsq`		`A2 61`
`..messageHeader`		`A0 5F 30 5D`
`...version`	`2`	`02 01 02`
`...revision`	`0`	`02 01 00`
`...date`	`19970514041853Z`	`18 0F 31 39 39 37 30 35 31 34 30 34 31 38 35 33 5A`
`...messageIDs`		`A0 16`
`....localID-C`		`80 14 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D 6C 69 64 63 2D`
`...rrpid`		`81 14 87 FB 2B 3D 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70`
`...swIdent`	`SET Specification v2.0`	`1A 16 53 45 54 20 53 70 65 63 69 66 69 63 61 74 69 6F 6E 20 76 32 2E 30`

Index