Part II
Certificate Management

Overview

Introduction

Part II defines the certificate management architecture, protocols, and concepts used in SET.

Organization

The following chapters are included:

Chapter	Title	Contents	Page
1	Certificate Management Architecture	Provides an overview of the certificate management architecture and describes the issuance and management of the Root certificates.	210
2	Certificate Request Protocols	Defines the protocols that allow Cardholders, Merchants, and Payment Gateways to obtain original certificates and to renew certificates.	218
3	Certificate Revocation or Cancellation	Describes the process of revoking or canceling a SET certificate.	303
4	Certificate Format	Describes the X.509 Version 3 certificate and certificate extensions (both X.509 and SET-specific) used in SET.	312
5	Certificate Revocation List and Brand CRL Identifier	Describes the use of the X.509 Certificate Revocation List (CRL) and the Brand CRL Identifier (BCI) in SET.	345
6	CA to CA Messages	Defines the protocols used by CAs to exchange certificates, CRLs, and BCIs.	354

Chapter 1
Certificate Management Architecture

Overview

Introduction

Chapter 1 provides an overview of the certificate management architecture and describes the issuance and management of the Root certificates.

Organization

The following sections are included:

Section	Title	Contents	Page
1	Architecture Overview	Introduces the certificate management architecture and defines each of the architectural components.	211
2	Root Certificate Distribution	Describes the issuance and management of the Root certificates.	214

Section 1
Architecture Overview

General Overview

Architecture diagram

The certificate management architecture consists of the nine components identified in Figure 1. The architecture is based on the hierarchy of trust defined for the management and verification of SET certificates by Certificate Authorities (CAs).

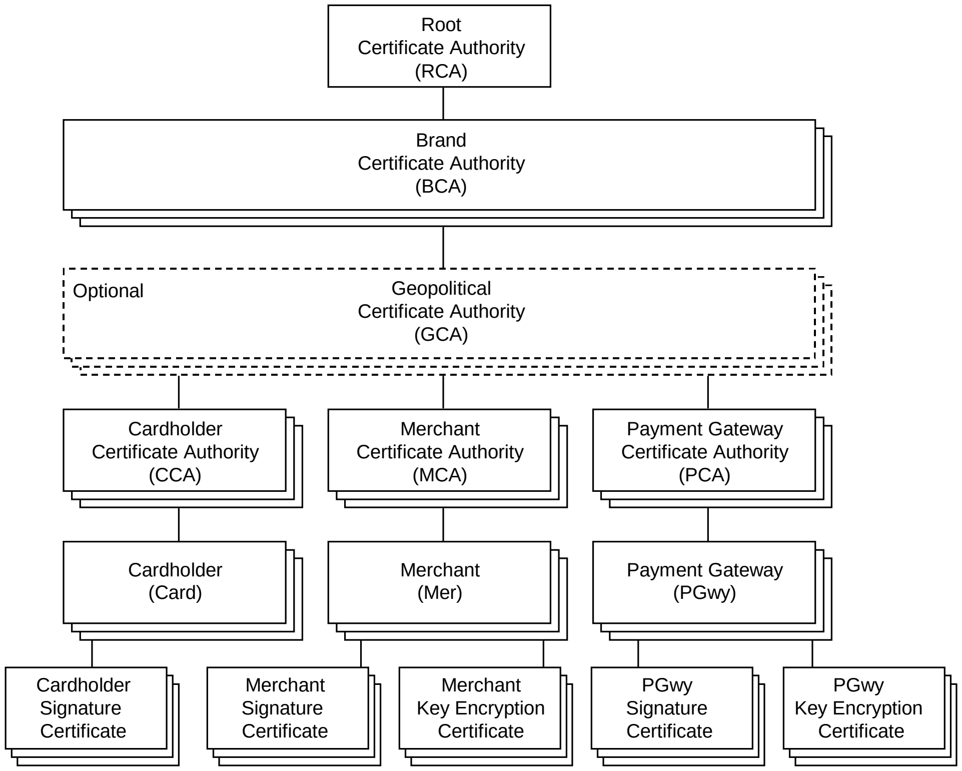

Figure 1: Certificate Management Architecture

Services

Each CA provides three basic services to the entities below it in the certificate management hierarchy: certificate issuance, renewal, and revocation. These services are described in the following chapters.

Architecture Overview

Root CA

The Root Certificate Authority (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.

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. Each BCA will issue CA certificates to the Geopolitical and/or Cardholder, Merchant, and Payment Gateway CAs immediately below it in the hierarchy, or may designate a CA to do so on its behalf.

Geopolitical CA

The Geopolitical CA (GCA) allows the brand to distribute to geographic or political regions the responsibility of managing Cardholder CA, Merchant CA, and/or Payment Gateway CA certificates. This level in the architecture allows brand policies to vary from one region to another as deemed necessary by the brands.

Cardholder CA

The Cardholder CA (CCA) issues Cardholder certificates (after the certificate request is verified and approved by the Issuer).

The CCA is operated by a card Issuer or on behalf of one or more card Issuers ~~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) issues Merchant certificates (after the certificate request is verified and approved by the Acquirer).

The MCA ~~may be~~ is operated by or on behalf of ~~a payment brand,~~ an Acquirer~~, or another party according to payment brand rules~~.

Payment Gateway CA

The Payment Gateway CA (PCA) issues certificates to SET Payment Gateways (after the certificate request is verified and approved by the Acquirer).

The PCA ~~may be~~ is operated by or on behalf of a payment brand~~, an Acquirer, or another party according to payment brand rules~~.

Cardholder

Cardholders request and receive certificates from a CCA.

Merchant

Merchants request and receive certificates from an MCA.

Payment Gateway

Payment Gateways request and receive certificates from a PCA.

Section 2
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 is usually distributed with the SET software.

Organization

This section includes the following topics:

- Initial Root Certificate Verification and Distribution
- Root Certificate Update
- Future Implications

Root certificate format

The SET Root certificate is a Version 3 X.509 certificate containing the extensions described in "CA Certificate Extensions" on page 342. 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, **HashedRootKey**, 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 333.

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 SET application software is usually delivered with:

- the initial Root :

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 **HashedRootKey**)

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

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

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).

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.

Future Implications

Expiration of initial Root certificate

When the initial Root certificate expires, all SET Version 1 applications should cease to function.

Root certificate retention

Each SET CA shall maintain a copy of all Root certificates starting with the oldest certificate that was valid with the prior major release of the specification. Root, brand, and geopolitical CAs shall send all these Root certificates in any PKCS #7 response to a lower-level CA.

Chapter 2
Certificate Request Protocols

Overview

Introduction

Certificates are issued 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 issued only signature certificates.
- Merchants and Payment Gateways ~~may be~~ are issued both signature and encryption certificates.

This chapter defines the certificate request protocols that allow Cardholders, Merchants, and Payment Gateways to obtain their original certificate(s) and to renew certificates.

Organization

This chapter includes the following sections.

Section	Title	Contents	Page
1	Protocol Overview	Describes the overall protocol for obtaining and renewing certificates.	219
2	Cardholder Certificate Initiation Request/Response Processing	Defines how the certificate request process is started for a Cardholder.	226
3	Cardholder Registration Form Request/Response Processing	Defines how the Cardholder requests and obtains a registration form.	235
4	Merchant/Payment Gateway Certificate Initiation Request/Response Processing	Defines how the certificate request process is started for a Merchant or an Payment Gateway.	250
5	Certificate Request/Response Processing	Defines the processing associated with the **CertReq**, the generation of the certificate, and the generation of the **CertRes**.	262
6	Certificate Inquiry Request/Response Processing	Defines how the end entity queries the CA to obtain the status of the certificate request.	292

Section 1
Protocol Overview

Overview

Purpose

This section defines the protocol and message processing for a Cardholder, Merchant, or Payment Gateway to request and obtain signature and/or data encryption X.509 certificates from a Certificate Authority (CA). The same protocol is used whether the end entity is requesting its first certificate or renewing a certificate.

A different protocol is used by CAs requesting certificates. See “CA to CA Certificate Requests and Responses” on page 355.

Organization

The following topics are included:

- Prerequisites
- Protocol initiation
- Cardholder certificate request protocol
- Merchant or Payment Gateway certificate request protocol

Prerequisites

Cardholder prerequisites

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

- an established valid payment card account,
- knowledge of information used to identify and authenticate the Cardholder as required by the payment card Issuer (Issuers will have different requirements for this information),
- the Universal Resource Locator (URL) or electronic mail address for the CCA, and
- a SET application with the ability to generate public/private key pairs and to securely store the private key (or interface with a hardware cryptographic device providing these functions).

Merchant prerequisites

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

- an established valid Merchant account with an Acquirer,
- knowledge of information from the agreement between the Merchant and the Acquirer (Acquirers will have different requirements for this information),
- the Universal Resource Locator (URL) or electronic mail address for the MCA, and
- a SET application with the ability to generate public/private key pairs and to securely store the private key (or interface with a hardware cryptographic device providing these functions).

Payment Gateway prerequisites

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

- an established relationship with a brand,
- its Bank Identification Number (BIN),
- knowledge of information used to identify and authenticate the Payment Gateway as required by the ~~Acquirer~~ brand (brands will have different requirements for this information),
- the Universal Resource Locator (URL) or electronic mail address for the PCA, and
- a SET application with the ability to interface with a hardware cryptographic device to generate public/private key pairs and to securely store the private key.

Protocol Initiation

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 as discussed in the *SET External Interface Guide*. (See "Related documentation" in the Preface.)
- The user of an electronic mail application shall initiate the SET application locally.

Subsequent processing

The figures on the following pages show the message exchanges required for an end entity to obtain a new or renewal SET certificate:

- Figure 2 on page 223 illustrates the exchanges between the Cardholder and the CCA.
- Figure 3 on page 225 illustrates the exchanges between the Merchant and the MCA or the Payment Gateway and the PCA.

The messages exchanged to obtain and submit a certificate registration form are different for the Cardholder than for the Merchant or Payment Gateway although the certificate request and response and the certificate inquiry request and response use the same format for all end entities.

Cardholder Certificate Request Protocol

Cardholder/ CCA processing

Figure 2 on page 223 shows the exchanges for the Cardholder to register and obtain a new certificate or to renew a certificate.

The Cardholder application sends a **CardCInitReq** to the CA, using the stored **BrandID** or one obtained from the certificate initiation message.	See also "Cardholder request via electronic mail" on page 223.
The CCA returns a **CardCInitRes** including an encryption certificate for the Cardholder to use to protect the transmission of its payment card number to the CCA.	
The Cardholder application encrypts the user's payment card number using the CCA's certificate and sends it to the CCA in a **RegFormReq**.	
The CCA sends a **RegFormRes** containing the appropriate registration template and policy statement.	
The Cardholder application displays the registration template and policy statement. The user enters the requested information and agrees to the policy statement.	Repeat if necessary to correct registration information.
The Cardholder application sends a **CertReq** to the CCA, including: • the filled-in registration form (if applicable), • a new public key, and • the certificate being renewed (if applicable).	
The CCA verifies the registration information with the Issuer. If the request is approved, the CCA generates and signs the certificate, and sends it to the Cardholder in a **CertRes**. If the request is not approved, the **CertRes** includes status information rather than a certificate.	
If the **CertRes** indicated that the certificate is not ready, the Cardholder sends a **CertInqReq** to obtain the certificate or its status.	Optional: Not necessary if certificate is received in **CertRes**.
The CCA returns a **CertInqRes** containing either the certificate or status information.	Repeat until certificate is received.

Continued on next page

Cardholder Certificate Request Protocol, continued

Cardholder certificate request exchanges

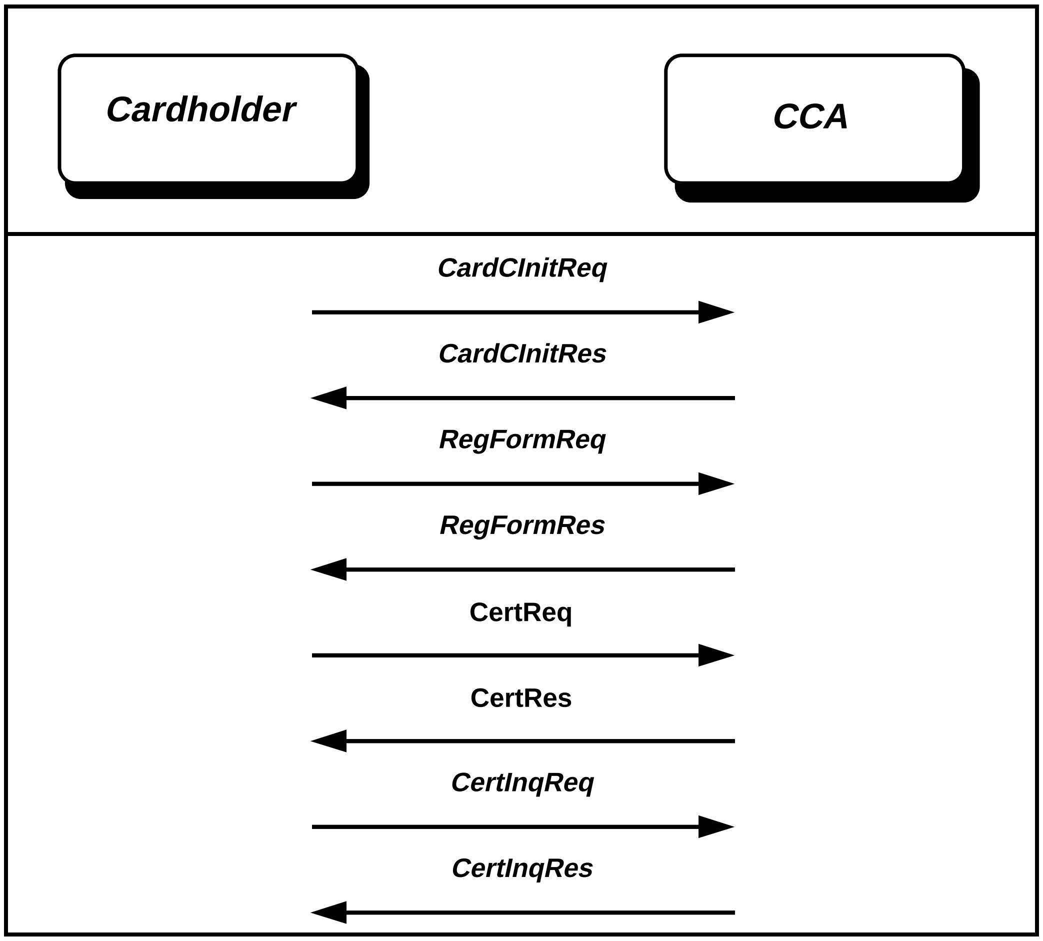

Figure 2: Cardholder Certificate Request Exchanges

Cardholder request via electronic mail

When a non-interactive communications mechanism such as electronic mail (SMTP) is used, **CardCInitReq/Res** and **RegFormReq/Res** may be omitted from the protocol, if the Cardholder already holds:

- a registration form, and
- the applicable CA certificates required to encrypt the **CertReq**.

Merchant or Payment Gateway Certificate Request Protocol

Merchant or Payment Gateway processing

Figure 3 on page 225 shows the exchanges for the Merchant or Payment Gateway to register and obtain a new certificate or to renew a certificate.

The SET application sends a **Me-AqCInitReq** to the CA, using the stored **BrandID** or one obtained from the certificate initiation message, as well as the end entity's BIN and other ID obtained from the Merchant or Payment Gateway system administrator.	
The CA returns a **Me-AqCInitRes** containing the registration template and policy statement.	
The SET application displays the registration template and policy statement. The user enters the requested information and agrees to the policy statement.	
The SET application sends a CertReq to the CA, including: • the filled in registration form (if applicable), • new public key(s), and • the certificate(s) being renewed (if applicable).	Repeat if necessary to correct registration information.
The CA verifies the registration information with the Acquirer. If the request if approved, the CA generates and signs the certificate(s), and sends it/them to the Merchant or Payment Gateway in a **CertRes**. If the request is not approved, the **CertRes** includes status information rather than certificates.	
If the **CertRes** indicates that the certificate is not ready, the Merchant or Payment Gateway sends a **CertInqReq** to obtain the certificate or its status.	Optional: Not necessary if certificate is received in **CertRes**.
The CA returns a **CertInqRes** containing either the certificate or status information.	Repeat until certificate is received.

Continued on next page

Merchant or Payment Gateway Certificate Request Protocol, continued

Merchant or Payment Gateway certificate request exchanges

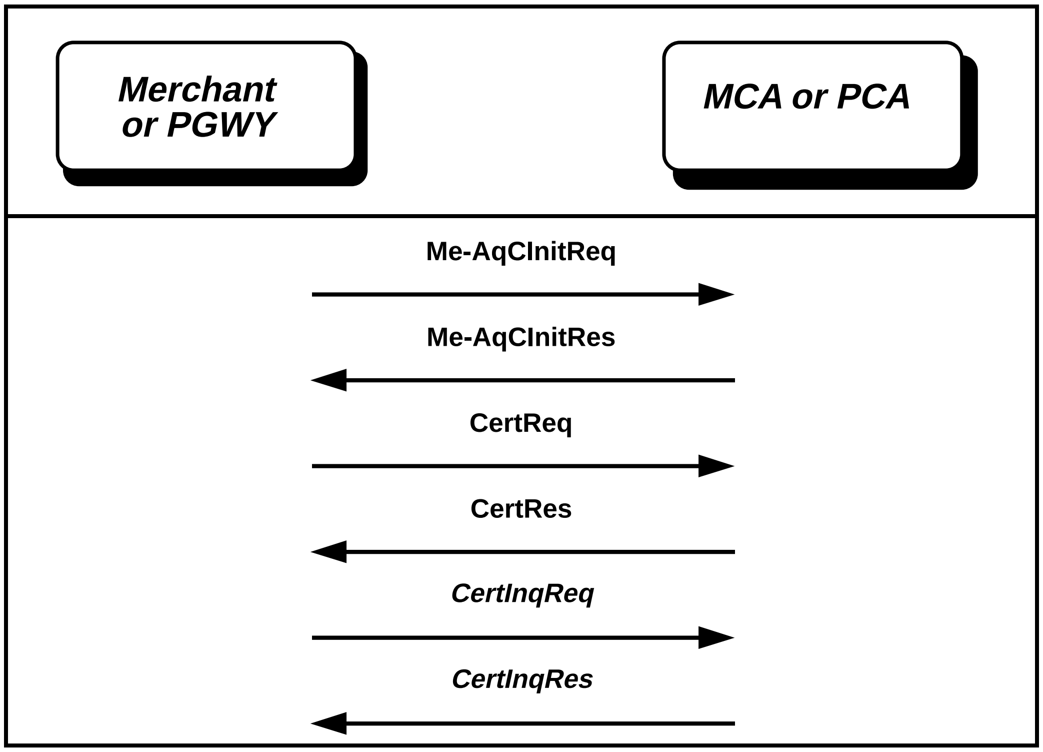

Figure 3: Merchant/Payment Gateway Certificate Request Exchanges

Section 2
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 BCI that are contained in its certificate cache. The CCA responds with a **CardCInitRes** containing any certificates, CRLs, and BCI that the Cardholder will need for signature verification, as well as an encryption certificate to use for subsequent messages.

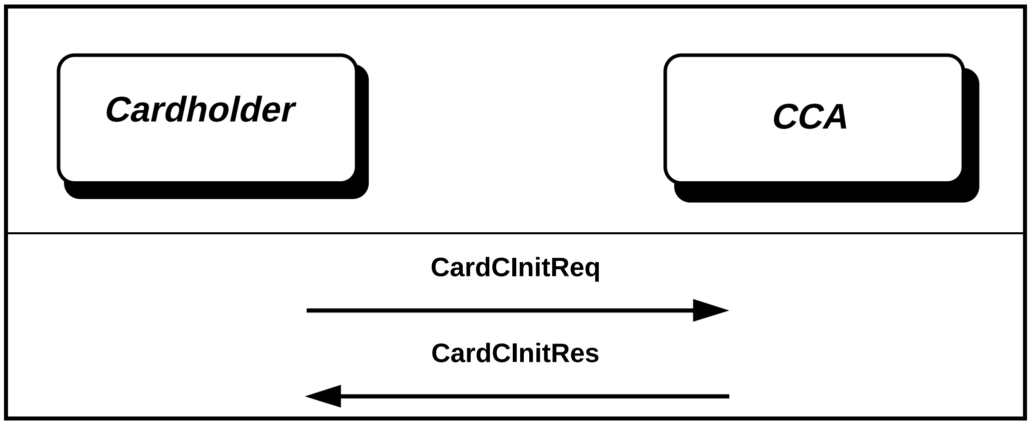

Figure 4: Cardholder Certificate Initiation Process

E-mail initiation

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

World Wide Web initiation

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

Cardholder Generates CardCInitReq

Create CardCInitReq

Step	Action
1	Construct *CardCInitReq*:

rrpid	a fresh statistically unique **RRPID**
lid-EE	a unique local identifier
chall-EE	a fresh random challenge
brandID	the **BrandID** that is stored or that was received in the initiation message
thumbs	the result of "Create **Thumbs**" on page **Error! Bookmark not defined.** (optional)

Step	Action
2	Store the result of Step 1 in the message database.
3	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input:

recip	the CCA
msg	the result of Step 1
ext	any message extension(s) required to support additional business functions (optional)
rrpid	*rrpid* from Step 1
lid-C	*lid-EE from Step 1*

CardCInitReq data

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 BCI Thumbprints currently held by Cardholder.*

Table 1: CardCInitReq Data

CCA Processes CardCInitReq

Process CardCInitReq

Step	Action
1	Receive as input: **_hdr_** — an instance of *MessageHeader* **_msg_** — an instance of *CardCInitReq* **_ext_** — any message extension(s) required to support additional business functions (optional)
2	Validate the following contents of **_msg_**: *rrpid* — **_hdr_.rrpid** *lid-EE* — **_hdr_.lid-c** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — ~~*unknownRRPID*~~ *wrapperMsgMismatch*
3	If the CA does not process requests (or make referrals) for **_msg_.brandID**, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — *unsupportedBrand*
4	Invoke "Create **CardCInitRes**" on page 229 with the following input: **_req_** — **_msg_** **_ext_** — **_ext_**

CCA Generates CardCInitRes

Create CardCInitRes

Step	Action	
1	Receive as input:	
	req	an instance of *CardCInitReq*
	ext	any message extension(s) required to support additional business functions (optional)
2	Construct *CardCInitResTBS*:	
	rrpid	***req*.rrpid**
	lid-EE	***req*.lid-EE**
	chall-EE	***req*.chall-EE**
	lid-CA	a unique local identifier (optional)
	caeThumb	the Thumbprint of the CCA key encryption certificate
	brandCRLIdentifier	the current **BrandCRLIdentifier** (if not specified in ***req*.thumbs**)
	thumbs	***req*.thumbs**
3	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input:	
	s	the CCA signature certificate Note: If the CA only makes referrals for ***req*.brandID**, select any valid signature certificate.
	t	the result of Step 2
	type	*id-set-content-CardCInitResTBS*
	certs	the CCA key encryption certificate
4	Optionally store the result of Step 2 in the message database. Note: If **lid-CA** is included, the CA must confirm the value in subsequent messages.	

Continued on next page

CCA Generates CardCInitRes, continued

Create CardCInitRes (continued)

Step	Action
5	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input: ***recip*** — the Cardholder ***msg*** — the result of Step 3 ***ext*** — any message extension(s) required to support additional business functions (optional) ***rrpid*** — ***req*.rrpid** ***lid-C*** — ***req*.lid-EE**

Continued on next page

CCA Generates CardCInitRes, continued

CardCInitRes data

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 351.*
Thumbs	*Copied from* **CardCInitReq**.

Table 2: CardCInitRes Data

Cardholder Processes CardCInitRes

Process CardCInitRes

Step	Action
1	Receive as input: <table><tr><td>***hdr***</td><td>an instance of *MessageHeader*</td></tr><tr><td>***msg***</td><td>an instance of *SignedData*</td></tr><tr><td>***ext***</td><td>any message extension(s) required to support additional business functions (optional)</td></tr></table>
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: <table><tr><td>***d***</td><td>***msg***</td></tr><tr><td>***type***</td><td>*id-set-content-CardCInitResTBS*</td></tr></table>Designate the value of ***t*** returned as ***res***.
3	Validate the following contents of ***res***: <table><tr><td>*rrpid*</td><td>***hdr*.rrpid**</td></tr><tr><td>*lid-EE*</td><td>***hdr*.lid-EE**</td></tr></table>If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: <table><tr><td>***errorCode***</td><td>*wrapperMsgMismatch*</td></tr></table>
4	From the message database, retrieve the instance of *CardCInitReq* whose **RRPID** matches ***msg*.rrpid**. If found, designate it as ***req***; if not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: <table><tr><td>***errorCode***</td><td>*unknownRRPID*</td></tr></table>
5	Validate the following contents of ***res***: <table><tr><td>*lid-EE*</td><td>***req*.lid-EE**</td></tr><tr><td>*chall-EE*</td><td>***req*.chall-EE**</td></tr><tr><td>*thumbs*</td><td>***req*.thumbs**</td></tr></table>If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed: <table><tr><td rowspan="3">***errorCode***</td><td>*lid-EE*</td><td>*unknownLID*</td></tr><tr><td>*chall-EE*</td><td>*challengeMismatch*</td></tr><tr><td>*thumbs*</td><td>*thumbsMismatch*</td></tr></table>

Continued on next page

Cardholder Processes CardCInitRes, continued

Process CardCInitRes (continued)

Step	Action
6	Search the trusted certificate cache for a certificate whose Thumbprint matches ***res*.caeThumb**. • If found, continue with Step 8. • Otherwise, search the untrusted certificate cache for it. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *missingCertificateCRLorBCI*
7	Invoke "Verify certificate" on page **Error! Bookmark not defined.** with the following input: ***cert*** — the result of Step 6
8	Verify that the Cardholder application supports one of the algorithms indicated in the **Tunneling** extension of the certificate found in Step 6. If the Cardholder application does not support a common encryption algorithm with the CA, notify the user and abort further CA message processing.
9	Store ***res*** in the message database.
10	Invoke "Create **RegFormReq**" on page 236 with the following input: ***initReq*** — ***req*** ***initRes*** — ***res***

Section 3
Cardholder Registration Form Request/Response Processing

Overview

After receiving the appropriate certificates, CRLs, and BCI, the Cardholder can 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, additional information concerning the service request denial is indicated in the **RegFormRes**.

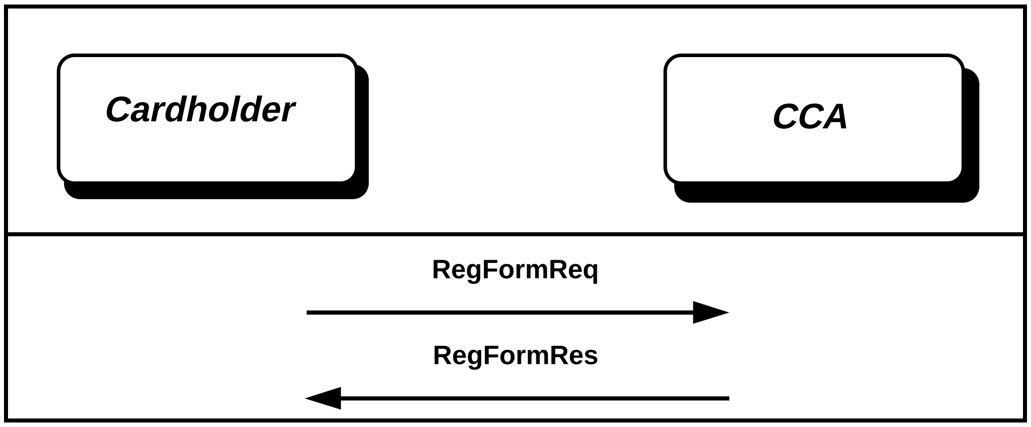

Figure 5: Cardholder Registration Form Processing

Cardholder Generates RegFormReq

Create RegFormReq

Step	Action	
1	Receive as input:	
	initReq	an instance of *CardCInitReq*
	initRes	an instance of *CardCInitResTBS*
2	Construct *RegFormReqData*:	
	rrpid	a fresh statistically unique **RRPID**
	lid-EE	***initReq*.lid-EE**
	chall-EE2	a fresh random challenge
	lid-CA	***initRes*.lid-CA** (if present)
	requestType	Populate according to Table 5 on page 239.
	language	the user's preferred language
	thumbs	the result of "Create **Thumbs**" on page **Error! Bookmark not defined.** (optional)
3	Construct the following components of **PANOnly**:	
	pan	the Primary Account Number of the cardholder
	Note: The account number must be obtained from the cardholder or from a device that is capable of reading the information from the card; the expiration date should be collected at the same time for use in "Create **CertReq**".	
4	Invoke "Compose *EXH*" on page **Error! Bookmark not defined.** with the following input:	
	r	a certificate from the trusted certificate cache whose Thumbprint matches ***initRes*.caeThumb**
	t	the result of Step 2
	p	the result of Step 3
	type-t	*id-set-content-RegFormReqTBE*
	type-p	*id-set-content-PANOnly*
5	Store the result of Step 3 in secure data storage and the result of Step 2 in the message database.	

Continued on next page

Cardholder Generates RegFormReq, continued

Create RegFormReq (continued)

Step	Action
6	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input:

recip	the CCA
msg	the result of Step 4
ext	any message extension(s) required to support additional business functions (optional)
rrpid	*rrpid* from Step 2
lid-C	***initReq*.lid-EE**

Continued on next page

Cardholder Generates RegFormReq, continued

RegFormReq data

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 239.*
Language	*Desired natural language for the rest of this flow.*
Thumbs	*Lists of Certificate (including Root), CRL, and BCI currently held by Cardholder.*

Table 3: RegFormReq Data

PANOnly data

The **PANOnly** is comprised of the following fields:

PAN	*Cardholder's payment card number.*
EXNonce	*Random number used to mask the PAN.*

Table 4: PANOnly Data

Continued on next page

Cardholder Generates RegFormReq, continued

RequestType values (Cardholder)

The following values are defined for **RequestType** for Cardholder certificate requests. (Values for Merchant and Payment Gateway certificate requests are shown on page 253.)

RequestType	Signature Certificate only
Cardholder Initial	**1**
Cardholder Renewal	**10**

Table 5: Enumerated Values for RequestType (Cardholder)

Additional RequestType restrictions

The following additional restrictions apply to the **RequestType** values in Table 5.

RequestType Value	Request Type	Restrictions
10	Renewal of signature certificate	The **CertReq** shall be signed with both: • the private key corresponding to the certificate being renewed, and • the private key of the new signature certificate.

CCA Processes RegFormReq

Process RegFormReq

Step	Action
1	Receive as input: **_hdr_** — an instance of *MessageHeader* **_msg_** — an instance of *EnvelopedData* **_ext_** — any message extension(s) required to support additional business functions (optional)
2	Invoke "Verify *EXH*" on page **Error! Bookmark not defined.** with the following input: **_d_** — **_msg_** **_type-t_** — *id-set-content-RegFormReqTBE* **_type-p_** — *id-set-content-PANOnly* Designate the value of ***t*** returned as ***req***.
3	Validate the following contents of ***req***: *rrpid* — ***hdr*.rrpid** *lid-EE* — ***hdr*.lid-c** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — ~~*unknownRRPID*~~ *wrapperMsgMismatch*
4	From the message database, retrieve the instance of *CardCInitResTBS* whose **lid-CA** matches ***req*.lid-CA**. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — *unknownLID*
5	Invoke "Create **RegFormRes**" on page 241 with the following input: **_req_** — **_req_** **_p_** — the value of ***p*** returned in Step 2 **_ext_** — **_ext_**

CCA Generates RegFormRes

Create RegFormRes

Step	Action	
1	Receive as input:	
	req	an instance of *RegFormReqData*
	p	an instance of *PANOnly*
	ext	any message extension(s) required to support additional business functions (optional)
2	Locate a registration form that corresponds to ***p*.pan**, ***req*.language** and ***req*.requestType**. If not found, continue with Step 5. Note: If **req.requestType** is any variety of renewal and if this Issuer (as indicated by ***p*.pan**) does not require a registration form, continue with Step 3.	
3	Construct *RegTemplate*:	
	regFormID	the ID of the registration form identified in Step 2
	brandLogoURL	the URL of the brand logo (optional)
	cardLogoURL	the URL of the financial institution logo (optional)
	regFieldSeq	the fields of the registration form identified in Step 2 (optional for renewal)
	See Appendix F: Logo Display during Certificate Registration for additional information on logos. See "Registration Form Creation" on page 302 for additional information on creating the fields of the registration form.	
4	Construct *RegFormData*:	
	regTemplate	the result of Step 3
	policyText	the policy that corresponds to the registration form identified in Step 2
	Continue with Step 7.	

Continued on next page

CCA Generates RegFormRes, continued

Create RegFormRes (continued)

Step	Action		
5	Identify one or more URLs of alternate CAs that can probably process requests for ***p*.pan**. If not found, identify a URL where the user can obtain more information concerning the service denial. Note: The URL should correspond to the transport mechanism of the request (electronic mail or World Wide Web).		
6	Construct *ReferralData*:		
		reason	the service denial information that will be displayed to the Cardholder (optional)
		referralURLSeq	the result of Step 5 (if found)
	Note: Either *reason* or *referralURLSeq* or both must be included.		
7	Construct *RegFormResTBS*:		
		rrpid	***req*.rrpid**
		lid-EE	***req*.lid-EE**
		chall-EE2	***req*.chall-EE2**
		lid-CA	***req*.lid-CA** if present; if not, ~~optionally~~ generate a unique local identifier.
		chall-CA	a fresh random challenge
		caeThumb	the Thumbprint of the CCA key encryption certificate (optional; must be included if *RegTemplate* was constructed in Step 3 and a different key is to be used to encrypt **CertReq** than was used to encrypt **RegFormReq**)
		requestType	***req*.requestType**
		formOrReferral	the result of Step 4 or Step 6 (as appropriate)
		brandCRLIdentifier	the current **BrandCRLIdentifier** (if not specified in ***req*.thumbs**)
		thumbs	***req*.thumbs**

Continued on next page

CCA Generates RegFormRes, continued

Create RegFormRes (continued)

Step	Action
8	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: **_s_**: CA's signature certificate. Note: If the CA only makes referrals for **_req_.brandID**, select any valid signature certificate. **_t_**: result of Step 7 **_type_**: *id-set-content-RegFormResTBS* **_certs_**: CA's key-encryption certificate
9	Optionally store the result of Step 7 in the message database. Note: If any portion of *formOrReferral* is generated specifically for this request, the CA must retain the information for subsequent messages.
10	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input: **_recip_**: the Cardholder **_msg_**: the result of Step 8 **_ext_**: any message extension(s) required to support additional business functions (optional) **_rrpid_**: **_req_.rrpid** **_lid-C_**: **_req_.lid-EE**

Continued on next page

CCA Generates RegFormRes, continued

RegFormRes data

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 239.*
RegFormOrReferral	*See page 245.*
BrandCRLIdentifier	*See page 351.*
Thumbs	*Copied from* **RegFormReq**.

Table 6: RegFormRes Data

Continued on next page

CCA Generates RegFormRes, continued

RegFormOrReferral data

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 7: RegFormOrReferral Data

Cardholder Processes RegFormRes

Process RegFormRes

Step	Action		
1	Receive as input:		
		hdr	an instance of *MessageHeader*
		msg	an instance of *SignedData*
		ext	any message extension(s) required to support additional business functions (optional)
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input:		
		d	***msg***
		type	*id-set-content-RegFormResTBS*
	Designate the value of ***t*** returned as ***res***.		
3	Validate the following contents of ***res***:		
		rrpid	***hdr*.rrpid**
		lid-EE	***hdr*.lid-C**
	If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input:		
		errorCode	~~*unknownRRPID*~~ *wrapperMsgMismatch*
4	From the message database, retrieve the instance of *RegFormReqData* whose **RRPID** matches ***res*.rrpid**.		
	• If found, designate it as ***req*** and retrieve the corresponding entries for *CardCInitResTBS* from the message database (and designate it as ***initRes***) and **PANOnly** from secure data storage (and designate it as ***PANOnly***).		
	• If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input:		
		errorCode	*unknownRRPID*

Continued on next page

Cardholder Processes RegFormRes, continued

Process RegFormRes (continued)

Step	Action
5	Validate the following contents of ***res***:

lid-EE	***req*.lid-EE**
chall-EE2	***req*.chall-EE2**
requestType	***req*.requestType**
thumbs	***req*.thumbs**

If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed:

errorCode	*lid-EE*	*unknownLID*
	chall-EE2	*challengeMismatch*
	requestType	*requestTypeMismatch*
	thumbs	*thumbsMismatch*

Step	Action
6	If ***res*.formOrReferral** contains *RegFormData*, continue with Step 7; otherwise, display a message to the user that contains: • if ***res*.referralData.reason** is specified, the reason text; • if ***res*.referralData.referralURLSeq** is specified, the alternate URL(s). and allow the user to abort processing or to select an alternate URL. If the user selects an alternate URL, restart processing with the new CA using "Create **CardCInitReq**" on page 227.
7	If ***res*.caeThumb** is not specified, continue with Step 9; otherwise, search the trusted certificate cache for a certificate whose Thumbprint matches ***res*.caeThumb**. If found, continue with Step 8. Otherwise, search the untrusted certificate cache for it. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input:

errorCode	*missingCertificateCRLorBCI*

Step	Action
8	Invoke "Verify certificate" on page **Error! Bookmark not defined.** with the following input:

cert	the result of Step 7

Continue with Step 10.

Continued on next page

Cardholder Processes RegFormRes, continued

Process RegFormRes (continued)

Step	Action
9	~~If ***res*.caeThumb** is specified, select the result of Step 7; otherwise,~~ Search the trusted certificate cache for a certificate whose Thumbprint matches ***initRes*.caeThumb**.
10	~~Select, from the **Tunneling** private extension in the CA key exchange certificate, a common preferred encryption algorithm for the CA to use to encrypt the **CertRes**. If a common algorithm is not found, abort processing and notify the user.~~ Determine whether a supported algorithm appears in the **Tunneling** private extension of ***r***. If not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: <table><tr><td>***errorCode***</td><td>*unsupportedAlgorithm*</td></tr></table>
11	~~If a field is required and invisible and the application cannot populate the field, leave the field empty and populate the remainder of the registration form and transmit in the **CertReq** as specified.~~ Determine whether ***res*.regFormData.regTemplate.regFieldSeq** contains any fields that are invisible and required. If not, continue with Step 12. For each invisible and required field, determine if the application is capable of generating the data. If not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: <table><tr><td>***errorCode***</td><td>*unrecognizedField*</td></tr></table>
12	Invoke "Create **CertReq**" on page 265 with the following input: <table><tr><td>***r***</td><td>the result of Step 9</td></tr><tr><td>***p***</td><td>***PANOnly***</td></tr><tr><td>***res***</td><td>***res***</td></tr></table>

Section 4 Merchant/Payment Gateway Certificate Initiation Request/Response Processing

Overview

Introduction

This section describes the certificate initiation process for the Merchant or Payment Gateway. After the SET application has been started, the Merchant or Payment Gateway sends a **Me-AqCInitReq** to the CA, which include identifying information and indicates via Thumbprints the certificates, CRLs, and BCI that are contained in its certificate cache. If the CA successfully validates the request, it returns a registration form in the **Me-AqCInitRes**. If the CA does not have a registration form for the request, additional information concerning the service request denial is indicated in the **Me-AqCInitRes**.

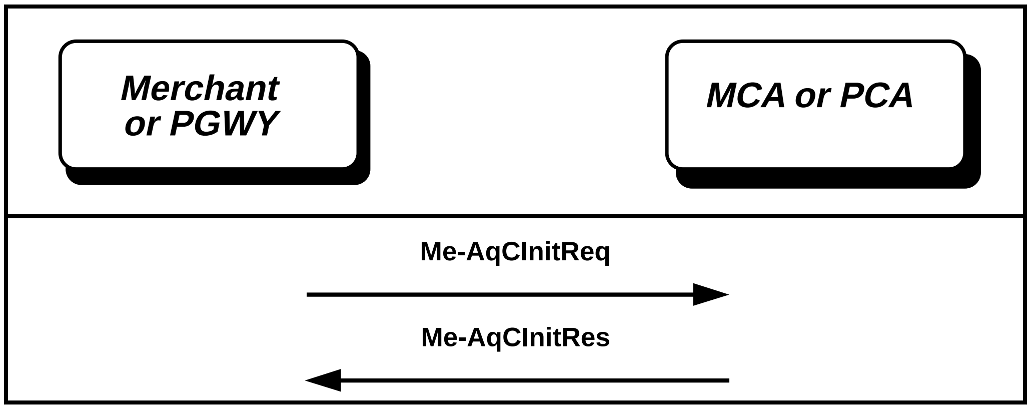

Figure 6: Merchant/Payment Gateway Certificate Initiation Processing

Merchant/Payment Gateway Generates Me-AqCInitReq

Create Me-AqCInitReq

Step	Action	
1	Construct *IDData*:	
	• If the EE is a Merchant, construct *MerchantAcquirerID*:	
	merchantBIN	the value provided to the merchant by the acquirer
	merchantID	the value provided to the merchant by the acquirer
	• If the EE is a Payment Gateway, construct *AcquirerID*:	
	acquirerBIN	the value provided to the acquirer by the brand
	acquirerBusinessID	the value provided to the acquirer by the brand (optional) Note: This differentiates the financial institution when a BIN is shared by multiple institutions.
2	Construct *Me-AqCInitReq*:	
	rrpid	a fresh statistically unique **RRPID**
	lid-EE	a unique local identifier
	chall-EE	a fresh random challenge
	requestType	Populate according to Table 10 on page 253.
	idData	the result of Step 1
	brandID	the **BrandID** that is stored or that was received in the initiation message
	language	the user's preferred language
	thumbs	the result of "Create **Thumbs**" on page **Error! Bookmark not defined.** (optional)
3	Store the result of Step 2 in the message database.	
4	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input :	
	recip	the CA
	msg	the result of Step 2
	ext	any message extension(s) required to support additional business functions (optional)
	rrpid	*rrpid* from Step 2
	lid-M	*lid-EE* from Step 2

Continued on next page

Merchant/Payment Gateway Generates Me-AqCInitReq, continued

Me-AqCInitReq data

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 page 253.*
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 BCI currently held by EE.*

Table 8: Me-AqCInitReq Data

IDData data

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 9: IDData Data

Continued on next page

Merchant/Payment Gateway Generates Me-AqCInitReq, continued

RequestType values (Merchant and Payment Gateway)

The following values are defined for **RequestType** for Merchant and Payment Gateway certificate requests. (Values for Cardholder certificate requests are shown on page 239.)

RequestType	Signature Certificate only	Encryption Certificate only	Both Certificates
Merchant Initial	**4**	**5**	**6**
Payment Gateway Initial	**7**	**8**	**9**
Merchant Renewal	**13**	**14**	**15**
Payment Gateway Renewal	**16**	**17**	**18**

Table 10: Enumerated Values for RequestType (Merchant/Payment Gateway)

Additional RequestType 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 **RequestType** values in Table 10.

RequestType Value	Request Type	Restrictions
5, 8	Initial encryption certificate	The end entity shall have a valid signature certificate and shall use the corresponding private key to sign the request.
13, 15, 16, 18	Renewal of signature certificate	The renewal request shall be signed with both: • the private key corresponding to the signature certificate being renewed, and • the private key of the new signature certificate.
14, 15, 17, 18	Renewal of encryption certificate	The end entity shall have a valid signature certificate and shall use the corresponding private key to sign the request. The Subject Name of the signature certificate used to sign the request shall match the Subject Name of the prior encryption certificate.

CA Processes Me-AqCInitReq

Process Me-AqCInitReq

Step	Action
1	Receive as input: **hdr** — an instance of *MessageHeader* **msg** — an instance of *Me-AqCInitReq* ***ext*** — any message extension(s) required to support additional business functions (optional)
2	Validate the following contents of ***msg***: *rrpid* — ***hdr*.rrpid** *lid-EE* — ***hdr*.lid-M** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — ~~*unknownRRPID*~~ *wrapperMsgMismatch*
3	If the CA does not process requests (or make referrals) for ***msg*.brandID**, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *unsupportedBrand*
4	Invoke "Create **Me-AqCInitRes**" on page 255 with the following input: ***req*** — ***msg*** ***ext*** — ***ext***

CA Generates Me-AqCInitRes

Create Me-AqCInitRes

Step	Action
1	Receive as input: **_req_** — an instance of *Me-AqCInitReq* **_ext_** — any message extension(s) required to support additional business functions (optional)
2	Locate a registration form (including the account data field if the CA authenticates the Merchant or Payment Gateway via the *AcctData*) that corresponds to ***req*.idData**, ***req*.language** and ***req*.requestType**. If not found, continue with Step 5. Note: If **req.requestType** is any variety of renewal and if this Acquirer (as indicated by ***p*.iddata**) does not require a registration form, continue with Step 3.
3	Construct *RegTemplate*: *regFormID* — the ID of the registration form identified in Step 2 *brandLogoURL* — the URL of the brand logo (optional) *cardLogoURL* — the URL of the financial institution logo (optional) *regFieldSeq* — the fields of the registration form identified in Step 2 (optional for renewal) See Appendix F: Logo Display during Certificate Registration for additional information on logos. See "Registration Form Creation" on page 302 for additional information on creating the fields of the registration form.
4	Construct *RegFormData*: *regTemplate* — the result of Step 3 *policyText* — the policy that corresponds to the registration form identified in Step 2 Continue with Step 7.
5	Identify one or more URLs of alternate CAs that can probably process requests for ***req*.idData**. If not found, identify a URL where the user can obtain more information concerning the service denial. Note: The URL should correspond to the transport mechanism of the request (electronic mail or World Wide Web).

Continued on next page

CA Generates Me-AqCInitRes, continued

Create Me-AqCInitRes (continued)

Step	Action	
6	Construct *ReferralData*:	
	reason	the service denial information that will be displayed to the user (optional)
	referralURLSeq	the result of Step 5 (if found)
	Note: Either *reason* or *referralURLSeq* or both must be included.	
7	Construct *Me-AqCInitResTBS*:	
	rrpid	**_req_.rrpid**
	lid-EE	**_req_.lid-EE**
	chall-EE	**_req_.chall-EE**
	lid-CA	a unique local identifier (optional)
	chall-CA	a fresh random challenge
	requestType	**_req_.requestType**
	regFormOrReferral	the result of Step 4 or Step 6 (as appropriate)
	acctDataField	an instance of *RegField* identified in Step 2 (optional)
	caeThumb	the Thumbprint of the CA key encryption certificate
	brandCRLIdentifier	the current **BrandCRLIdentifier** (if not specified in **_req_.thumbs**)
	thumbs	**_req_.thumbs**
8	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input:	
	s	the CA signature certificate Note: If the CA only makes referrals for **_req_.brandID**, select any valid signature certificate.
	t	the result of Step 7
	type	*id-set-content-Me-AqCInitResTBS*
	certs	the CA key-encryption certificate
9	Optionally store the result of Step 7 in the message database. Note: If any portion of *formOrReferral* is generated specifically for this request, the CA must retain the information for subsequent messages.	

Continued on next page

CA Generates Me-AqCInitRes, continued

Create Me-AqCInitRes (continued)

Step	Action
10	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input:

recip	the merchant or payment gateway
msg	the result of Step 8
ext	any message extension(s) required to support additional business functions (optional)
rrpid	***req*.rrpid**
lid-M	***req*.lid-EE**

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. See "**RegFormOrReferral** Data" on page 246.

Me-AqCInitRes data

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 253.*
RegFormOrReferral	*See page 245.*
AcctDataField	**RegField** *(see "RegFormOrReferral data" on page 245); 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 350.*
Thumbs	*Copied from* **Me-AqCInitReq**.

Table 11: Me-AqCInitRes Data

Merchant/Acquirer Processes Me-AqCInitRes

Process Me-AqCInitRes

Step	Action
1	Receive as input: **_hdr_** — an instance of *MessageHeader* **_msg_** — an instance of *SignedData* **_ext_** — any message extension(s) required to support additional business functions (optional)
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: **_d_** — **_msg_** **_type_** — *id-set-content-Me-AqCInitResTBS* Designate the value of **_t_** returned as **_res_**.
3	Validate the following contents of **_res_**: *rrpid* — **_hdr_.rrpid** *lid-EE* — **_hdr_.lid-M** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — ~~*unknownRRPID*~~ *wrapperMsgMismatch*
4	From the message database, retrieve the instance of *Me-AqCInitReq* whose **RRPID** matches **_msg_.rrpid**. If found, designate it as **_req_**; if not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: **_errorCode_** — *unknownRRPID*

Continued on next page

Merchant/Acquirer Processes Me-AqCInitRes, continued

Process Me-AqCInitRes (continued)

Step	Action
5	Validate the following contents of ***res***: <table><tr><td>*lid-EE*</td><td>***req*.lid-EE**</td></tr><tr><td>*chall-EE*</td><td>***req*.chall-EE**</td></tr><tr><td>*requestType*</td><td>***req*.requestType**</td></tr><tr><td>*thumbs*</td><td>***req*.thumbs**</td></tr></table> If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed: <table><tr><td>***errorCode***</td><td>*lid-EE*</td><td>*unknownLID*</td></tr><tr><td></td><td>*chall-EE*</td><td>*challengeMismatch*</td></tr><tr><td></td><td>*requestType*</td><td>*requestTypeMismatch*</td></tr><tr><td></td><td>*thumbs*</td><td>*thumbsMismatch*</td></tr></table>

Continued on next page

Merchant/Acquirer Processes Me-AqCInitRes, continued

Process Me-AqCInitRes (continued)

Step	Action
6	If ***res*.formOrReferral** contains **RegFormData**, continue with Step 7; otherwise, process the referral by displaying a message to the user that contains: • if ***res*.referralData.reason** is provided, the reason text; • if ***res*.referralData.referralURLSeq** is provided, the alternate URL(s). and allow the user to abort processing or to select an alternate URL. If the user chooses an alternate URL, restart processing with the new CA using "Create **Me-AqCInitReq**" on page 251.
7	Search the trusted certificate cache for a certificate whose Thumbprint matches ***res*.caeThumb**. If found, continue with Step 8. Otherwise, search the untrusted certificate cache for it. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** | *missingCertificateCRLorBCI*
8	Invoke "Verify certificate" on page **Error! Bookmark not defined.** with the following input: ***cert*** | the result of Step 7 Continue with Step 10.
9	~~If ***res*.caeThumb** is specified, select the result of Step 7; otherwise,~~ Search the trusted certificate cache for a certificate whose Thumbprint matches ***initRes*.caeThumb**.
10	~~If a field is required and invisible and the application cannot populate the field, leave the field empty and populate the remainder of the registration form and transmit in the **CertReq** as specified.~~ Determine whether ***res*.regFormData.regTemplate.regFieldSeq** contains any fields that are invisible and required. If not, continue with Step 11. For each invisible and required field, determine if the application is capable of generating the data. If not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** | *unrecognizedField*
11	Invoke "Create **CertReq**" on page 265 with the following input: ***r*** | the result of Step 9 ***idData*** | ***req*.idData** ***res*** | ***res***

Section 5
Certificate Request/Response 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** 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 resubmit the corrected registration form in a new **CertReq**. If the registration form cannot be validated immediately, the status will indicate that a **CertInqReq** will be necessary to obtain the final result.

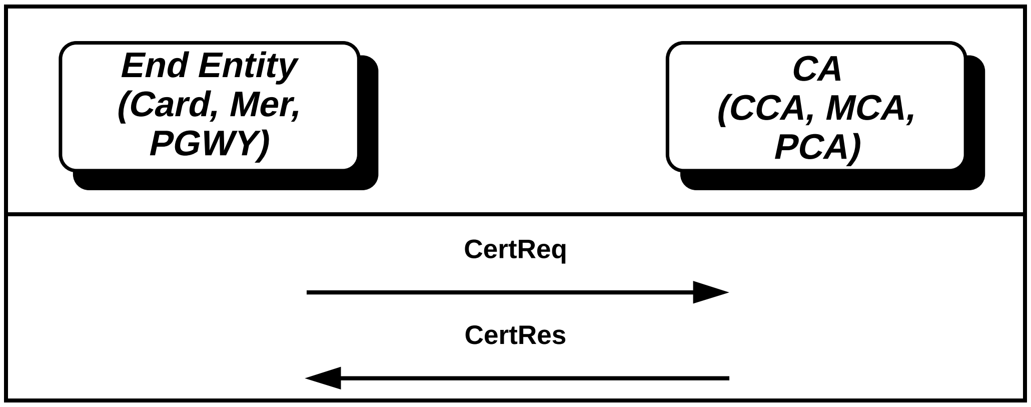

Figure 7: Certificate Request and Generation Processing

End Entity Generates CertReq

End entity input

The end entity:	enters requested information in the registration form:
Cardholder	• expiration date (if not obtained earlier), and • other information requested by the CCA.
Merchant	• Merchant authentication data (if any), and • other information requested by the MCA.
Payment Gateway	• Payment Gateway authentication data (if any), and • other information requested by the PCA.

CertReq contents

The certificate request (**CertReq**) may contain:

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

CertReq handling

The end entity signs the message using:

- the private key corresponding to an existing signature certificate (if one exists), and/or
- the new signature private key (if any).

The signed data and the signatures are then encrypted using a symmetric algorithm. OAEP is applied to the symmetric key used for this encryption along with the end entity account information, if present, and the result is encrypted using a public-key algorithm.

CertReq resubmission

If the CA finds errors in the submitted registration form, the errors are indicated in the **CertRes** and the end entity may submit a corrected registration form in a new **CertReq**.

Continued on next page

End Entity Generates CertReq, continued

Creating 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 whether **AcctInfo** is required in the AcctInfoField. *EncX* is only used if **AcctInfo** is present.~~

If end entity is:	then **AcctInfo**:	and encryption is by means of:
Cardholder	contains **PAN**	*EncX*
Merchant or Payment Gateway	contains authentication data if required by the CA (if **Me-AqCInitRes. AcctDataField** is present)	*EncX*
	is omitted (if **Me-AqCInitRes. AcctDataField** is omitted)	*Enc*

If the **CertReq** is being resubmitted with a corrected registration form, a new value for **Chall-EE3** and a new statistically unique value for **RRPID** shall be included.

Continued on next page

End Entity Generates CertReq, continued

Create CertReq

Step	Action
1	Receive as input: ***r*** — the key-exchange certificate of the CA ***p*** — an instance of *PANOnly* (optional) ***idData*** — an instance of *IDData* (optional) ***res*** — an instance of *RegFormRes* or *Me-AqCInitRes* Note: If the certification request is invoked as the first message in the exchange with the CA, no input is received. In this case, the application must have access to ***r*** (and the certificates necessary to authenticate it), either ***p*** or ***idData***, as well as the data corresponding to the following fields of ***res***: • **requestType** • **regFormOrReferral.regFormData** • **acctDataField** (optional)
2	If ***res*.regFormData** is not specified, continue with Step 7; otherwise, process the registration form: Display ***res*.regFormData.policy** and require user acknowledgement. Construct the registration form: • If ***res*.regFormData.regTemplate.brandLogoURL** is specified, retrieve and include the logo. • If ***res*.regFormData.regTemplate.cardLogoURL** is specified, retrieve and include the logo. • If ***res*.acctDataField** is specified, include ***res*.acctDataField.fieldName** and an input field for the user response. • For each field in ***res*.regFormData.regTemplate.regFieldSeq** that is visible, include **regField.fieldName** and an input field for the user response. Note: If **brandLogoURL** or **cardLogoURL** cannot be retrieved, display the registration form without the corresponding logo.
3	Optionally fill the registration form with known information: For each field that includes **fieldId**, if the application has retained information from a prior registration form with an identical **fieldId**, insert the previous answer into the form.

Continued on next page

End Entity Generates CertReq, continued

Create CertReq (continued)

Step	Action
4	Display the form created in Step 2 and allow the user to input values; take appropriate actions based on user input as described in the table below in a manner consistent with the user interface design of the application.

The form should allow the user to:	**at which time the application shall:**
indicate completion of the form and acceptance of the policy	proceed to Step 5
cancel the request	abort processing
request additional information for a field; included only if:	if a message is available for the field containing the cursor, display the corresponding text:
at least one visible field provides **fieldDesc**	**fieldDesc**
at least one visible field provides **fieldId** and the application can provide help	the text corresponding to **fieldId** (supplied by the application)

Step	Action
5	Validate the user's input on the registration form by checking each field as follows:

If the field:	**take the following action:**
is required, but the user has not filled it in	move the cursor to the field and display a message to the user indicating that the field must be provided
contains more characters than allowed by **fieldLen**	move the cursor to the field and display a message to the user indicating the maximum length of the field
includes **fieldId** and the application is capable of performing an edit based on the object identifier	edit the user's input based on the indicated field type; if the edit fails, move the cursor to the field and display a message suggesting a corrective action; give the user the option to override the application edit

For additional information on **fieldId** editing, see Appendix L: Object Identifiers for Registration Form Fields.

If the action on any field requires additional user input, restore the user's input and continue with Step 4.

Continued on next page

End Entity Generates CertReq, continued

Create CertReq (continued)

Step	Action
6	Construct *RegForm*: For each visible field that the user provided an answer, populate an item in *RegFormItems*: *fieldName*: the corresponding field in ***res*.regFormData.regField.fieldName** *fieldValue*: the user's input For each invisible field that the application is capable of populating: *fieldName*: the corresponding field in ***res*.regFormData.regField.fieldName** *fieldValue*: a value provided by the application (Step 14 addresses **acctDataField**, the only data from the registration form that is not part of *RegForm*.)
7	Based on ***res*.requestType**, generate private/public key pairs: • if the request is for *signature certificate only* or *both certificates*, generate a signature key pair; • if the request is for *encryption certificate only* or *both certificates*, generate an encryption key pair. (The end entity populated **requestType** based on either Table 5 on page 239 or Table 10 on page 253.) Note: The application may generate the keys directly or obtain them from hardware cryptographic modules. See "Architecture" on page **Error! Bookmark not defined.** for additional information on requirements for the use of hardware cryptographic modules. The private key(s) shall be retained in secure data storage. See "Secure Data Storage" on page **Error! Bookmark not defined.**.
8	If this is a cardholder application, select a common preferred encryption algorithm from the **Tunneling** private extension of ***r*** for the CA to use to encrypt the **CertRes**. Construct *BackKeyData*: *backAlgID* / *algorithm*: the selected algorithm *backAlgID* / *parameters*: generate an eight-byte DES-CBC initialization vector *backKey*: a fresh symmetric DES key The following algorithm identifiers are supported: • id-desCBC • id-desCDMF

Continued on next page

End Entity Generates CertReq, continued

Create CertReq (continued)

Step	Action
9	Construct *CertReqData*:

rrpid	a fresh statistically unique **RRPID**
lid-EE	***res*.lid-EE** if ***res*** is available*; otherwise, generate a unique local identifier
chall-EE3	a fresh random challenge
lid-CA	***res*.lid-CA** if ***res*** is available* Note: If this is a subsequent submission of *CertReq* resulting from *regFormAnswerMalformed* (see page 289), use **CertResData.lid-CA**.
chall-CA	***res*.chall-CA** if ***res*** is available*
requestType	***res*.requestType** if ***res*** is available*; otherwise, populate according to Table 5 on page 239 or Table 10 on page 253
requestDate	the current date and time
idData	***idData*** (if specified)
regFormID	***res*.regFormData.regTemplate.regFormID**
regForm	the result of Step 6
caBackKeyData	for cardholder applications only, the result of Step 8
publicKeySorE	the result of Step 7
eeThumb	if the ***res*.requestType** is for the renewal of an encryption certificate, the Thumbprint of the certificate being renewed.
thumbs	the result of "Create **Thumbs**" on page **Error! Bookmark not defined.** (optional)

* that is, if the certification request is not the first message to be exchanged with the CA and a response message is available

Continued on next page

End Entity Generates CertReq, continued

Create CertReq (continued)

Step	Action
10	If not a Cardholder application, continue with Step 13. Otherwise generate **CardSecret**, a 160-bit random number. Note: This value is a shared secret between the cardholder application and the cardholder's financial institution. The application shall retain this value in the secure data storage. See "Secure Data Storage" on page **Error! Bookmark not defined.**.
11	Construct the following components of *PANData0*: *pan* — ***p*.pan** *cardExpiry* — the expiration date of *pan*. Note: The expiration date of the card is usually obtained during "Create **RegFormReq**" processing. *CardSecret* — the result of Step 10 The application shall retain these values in the secure data storage. See "Secure Data Storage" on page **Error! Bookmark not defined.**.
12	If ***res*.caeThumb** is not specified, retrieve the instance of *CardCInitResTBS* corresponding to ***res*.lid-EE** from the message database. Continue with Step 15.
13	If ***res*.acctDataField** is specified, continue with Step 15. Otherwise, invoke "Compose *Enc*" on page **Error! Bookmark not defined.** with the following input: (see table below)
14	Append the result of Step 13 to the tag [1]. Continue with Step 18.

Step 11 components:

pan	***p*.pan**
cardExpiry	the expiration date of *pan* Note: The expiration date of the card is usually obtained during "Create **RegFormReq**" processing.
CardSecret	the result of Step 10

Step 13 input:

s	if ***res*.requestType** is for *signature certificate only* or *both certificates*, the signature key pair generated in Step 7; otherwise, the signature certificate of the end entity
s2	if ***res*.requestType** indicates a renewal for *signature certificate only* or *both certificates*, the existing signature certificate of the end entity
r	a certificate from the trusted certificate cache whose Thumbprint matches ***res*.caeThumb**
t	the result of Step 9
type-t	*id-set-content-CertReqTBE*
type-s	*id-set-content-CertReqData*

Continued on next page

End Entity Generates CertReq, continued

Create CertReq (continued)

Step	Action
15	Construct *AcctData*: *acctIdentification*: the user's input to the field corresponding to ***res*.acctDataField** ~~(if a required field)~~
16	Invoke "Compose *EncX*" on page **Error! Bookmark not defined.** with the following input: ***s***: if ***res*.requestType** is for *signature certificate only* or *both certificates* the signature key pair generated in Step 7; otherwise, the signature certificate of the end entity ***s2***: if ***res*.requestType** indicates a renewal for *signature certificate only* or *both certificates*, the existing signature certificate of the end entity ***r***: a certificate from the trusted certificate cache whose Thumbprint matches ***res*.caeThumb** (or **caeThumb** from *CardCInitRes* if ***res*.caeThumb** is not specified) ***t***: the result of Step 9 ***p***: the result of Step 11 or Step 15 ***type-t***: *id-set-content-CertReqTBEX* ***type-s***: *id-set-content-CertReqTBS* ***type-p***: *id-set-content-AcctInfo*
17	Append the result of Step 16 to the tag [0].
18	Store the results of Steps 2, 4 and 9 in the message database.
19	Invoke "Send Message" on page **Error! Bookmark not defined.** with the following input: ***recip***: the CA ***msg***: the result of Step 14 or Step 17 ***ext***: any message extension(s) required to support additional business functions (optional) ***rrpid***: *rrpid* from Step 9 ***lid-C***: if a Cardholder, *lid-EE* from Step 9 ***lid-M***: if a Merchant or Payment Gateway, *lid-EE* from Step 9

Continued on next page

End Entity Generates CertReq, continued

CertReq data

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 pages 239 and 253.*
RequestDate	*Date of certificate request.*
IDData	*See page 252. Omit if EE is Cardholder.*

Table 12: CertReq Data

Continued on next page

End Entity Generates CertReq, continued

CertReq data (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 BCI 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 12: CertReq Data, continued

Continued on next page

End Entity Generates CertReq, continued

PANData0 data

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 "**Error! Reference source not found.**" on page **Error! Bookmark not defined.**).*
EXNonce	*A fresh nonce to foil dictionary attacks on* **PANData0**.

Table 13: PANData0 Data

AcctData data

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 14: AcctData Data

CA Processes CertReq

Process CertReq

Step	Action	
1	Receive as input:	
	hdr	an instance of *MessageHeader*
	msg	a tag followed by an instance of *SignedData*
	ext	any message extension(s) required to support additional business functions (optional)
	This procedure uses the following internal variables:	
	status	an instance of *CertStatusCode*
	failedItems	an instance of *FailedItemSeq*
	certs	an instance of *Certificates*
2	Examine the tag at the beginning of ***msg*** to determine whether the sender used *Enc* or *EncX*. If the *EncX* version of **CertReq** is used, continue with Step 3; otherwise continue with Step 4.	
3	Invoke "Verify *EncX*" on page **Error! Bookmark not defined.** with the following input:	
	d	***msg*** (without the leading tag [0])
	type-t	*id-set-content-CertReqTBEX*
	type-s	*id-set-content-CertReqTBS*
	type-p	*id-set-content-AcctInfo*
	unauthOK	TRUE
	Designate the value of ***t*** returned as ***req***. Continue with Step 5.	
4	Invoke "Verify *Enc*" on page **Error! Bookmark not defined.** with the following input:	
	d	***msg*** (without the leading tag [1])
	type-t	*id-set-content-CertReqTBE*
	type-s	*id-set-content-CertReqData*
	unauthOK	TRUE
	Designate the value of ***t*** returned as ***req***.	

Continued on next page

CA Processes CertReq, continued

Process CertReq (continued)

Step	Action
5	If ***si*** was not returned in Step 3 or Step 4, continue with Step 7. If ***req*.publicKeySorE.publicKeyS** is not specified, set ***status*** to *sigValidationFailure* and continue with Step 25.
6	Verify the signature in ***si*** using ***req*.publicKeySorE.publicKeyS**. If it does not verify, set ***status*** to *sigValidationFailure* and continue with Step 25.
7	Validate the following contents of ***msg***: *rrpid* **_hdr_.rrpid** *lid-EE* if ***req*.requestType** indicates a cardholder certificate request, ***hdr*.lid-C**, otherwise ***hdr*.lid-M** *date* **_hdr_.date** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** *~~unknownRRPID~~ wrapperMsgMismatch*
8	If ***req*.lid-CA** is not specified, continue with Step 10. From the message database, retrieve the corresponding response message (either *RegFormResTBS* or *Me-AqCInitResTBS*). If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** *unknownLID* Note: ***req*.lid-CA** must be present unless the **CertReq** is the first message sent from the end entity to the CA.
9	Validate the following contents of the response message identified in Step 8: *chall-CA* **_req_.chall-CA** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** *challengeMismatch*

Continued on next page

CA Processes CertReq, continued

Process CertReq (continued)

Step	Action
10	Validate that ***req*.publicKeySorE.publicKeyS** is consistent with ***req*.requestType**. If not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *missingData*
11	Initialize ***status*** to *requestComplete.* Initialize ***failedItems*** so that it contains zero entries. Initialize ***certs*** so that it contains zero entries. Note: The item number in ***failedItems*** corresponds to the order the fields appeared in **RegFormResTBS** or **Me-AqCInitResTBS**.
12	If the *Enc* version of **CertReq** is used, continue with Step 15.
13	If ***req*.requestType** indicates a Cardholder certificate request, validate the PAN and expiration date contained in ***p*** from the result of Step 3; otherwise, validate the account data contained in ***p*** from the result of Step 3. If the validation fails: • set ***status*** to *rejectedByIssuer*; • add an entry to ***failedItems*** with an item number of zero and an appropriate message; • continue with Step 25.
14	If ***req*.requestType** indicates a Merchant or Payment Gateway certificate request and the **Me-AqCInitResTBS** did not include an *acctDataField,* • set ***status*** to *rejectedByIssuer;* • add an entry to ***failedItems*** with an item number of zero and an appropriate message; • continue with Step 25.
15	If ***req*.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 the certificates have been renewed before, set ***status*** to *rejectedByCA* and continue with Step 25.
16	Verify that ***req*.regFormID** is consistent with ***req*.language** and ***req*.requestType**~~, and BIN or **PAN**~~. If not, set ***status*** to *rejectedByCA* and continue with Step 25.

Continued on next page

CA Processes CertReq, continued

Process CertReq (continued)

Step	Action
17	If ***req*.requestType** indicates a cardholder certificate request: • if ***req*.caBackKeyData** is not specified, set ***status*** to *unableToEncryptCertRes* and continue with Step 25; • if ***req*.caBackKeyData.backAlgID** indicates an algorithm that is not supported by the CA, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — unsupportedAlgorithm
18	Verify that each field value in an invisible field of ***req*.regForm** contains an acceptable response; if any field has a validation failure, set ***status*** to ~~*rejectedByIssuer*~~ *regFormAnswerMalformed*. For each field with a validation failure, add an entry to ***failedItems*** with the item number and an appropriate message.
19	~~If the above checks are successful,~~ Verify: • that the field value of each visible field in ***req*.regForm** contains an acceptable response (the length, format and character type are valid); • that each required field appears in ***req*.regForm**. If any field has a validation failure, set ***status*** to *regFormAnswerMalformed.* For each field with a validation failure, add an entry to ***failedItems*** with the item number and an appropriate message.
20	If ***status*** is *regFormAnswerMalformed* and financial institution policy forbids further processing, continue with Step 25.
21	If the system to perform financial institution authentication is available, continue with Step 22. If ***status*** is not *requestComplete*, continue with Step 23; otherwise, • set ***status*** to *requestPended;* • store ***req*** and ***status*** in the message database; and • continue with Step 25. Note: When ***status*** is *requestPended*, additional processing is required when the financial institution authentication is performed; see "Deferred processing" on page 279.

Continued on next page

CA Processes CertReq, continued

Process CertReq (continued)

Step	Action
22	Perform financial institution authentication on page 279. If any field has an authentication failure, set ***status*** to *regFormAnswerMalformed.* For each field with an authentication failure, add an entry to ***failedItems*** with the item number and an appropriate message.
23	Based on the policy of the financial institution, determine if the end entity will be permitted to correct any errors found in Steps 18 through 22. If not, • set the **status** to *rejectedByIssuer*. • initialize **failedItems** so that it contains zero entries; and • continue with Step 25.
24	If ***status*** is *requestComplete*, create the certificate(s) and add to ***certs***.
25	Invoke "Create **CertRes**" on page 281 with the following input:

req	***req***
status	***status***
failedItems	***failedItems***
certs	***certs***
ext	any message extension(s) required to support additional business functions (optional)

Financial Institution Authenticates Data

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 it is not accepted,~~ The status is returned to the CA for use in composing the **CertRes.CertStatusCode**.

Deferred processing

If the system to perform financial institution authentication is unavailable to perform authentication at the time the **CertReq** is processed, the following processing shall be performed when it becomes available.

Step	Action
1	This procedure uses the following internal variables: ***status*** — an instance of *CertStatusCode* ***failedItems*** — an instance of *FailedItemSeq* ***certs*** — an instance of *Certificates*
2	Initialize ***status*** to *requestComplete.* Initialize ***failedItems*** so that it contains zero entries. Initialize ***certs*** so that it contains zero entries.
3	From the message database, retrieve *CertReqData* (and designate it as ***req***) and *CertResData* (and designate it as ***res***).
4	Perform financial institution authentication. If any field has an authentication failure, set ***status*** to *regFormAnswerMalformed.* For each field with an authentication failure, add an entry to ***failedItems*** with the item number and an appropriate message.
5	Based on the policy of the financial institution, determine if the end entity will be permitted to correct any errors found in Step 4. If not, • set the **status** to *rejectedByIssuer.* • initialize **failedItems** so that it contains zero entries; and • continue with Step 7.
6	If ***status*** is *requestComplete*, create the certificate(s) and add to ***certs.***
7	Store ***status***, ***failedItems***, and ***certs*** in the message database using ***res*.lid-CA** as the key.

CA Generates CertRes

CertRes overview

The **CertRes** contains either the requested certificates or the status of the certificate request. The **CertRes** shall ~~will~~ be signed and optionally encrypted, depending on the data that is 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 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 "Certificate Format" beginning on page 312 for additional information about how the fields are populated.

Continued on next page

CA Generates CertRes, continued

Create CertRes

Step	Action		
1	Receive as input:		
		req	an instance of *CertReqData*
		status	an instance of *CertStatusCode*
		failedItems	an instance of *FailedItemSeq* (optional)
		certs	an instance of *Certificates* (optional)
		ext	any message extension(s) required to support additional business functions (optional)
2	If ***req*.requestType** indicates a request from a cardholder and ***status*** is *requestComplete*, construct *CAMsg*:		
		cardLogoURL	the URL for financial institution logo(s) (optional)
		brandLogoURL	the URL for brand logo(s) (optional)
		cardCurrency	the ISO 4217 value corresponding to the cardholder's billing currency (optional)
		cardholderMsg	a message from the CA (optional)
	Refer to Appendix F: Logo Display during Certificate Registration for additional information on logo URLs.		
3	Construct *CertStatus*:		
		certStatusCode	***status***
		nonceCCA	a fresh nonce (if ***req*.requestType** indicates a request from a cardholder and ***status*** is *requestComplete*)
		eeMessage	a message from the CA (optional; not included for cardholder requests where ***status*** is *requestComplete*)
		caMsg	the result of Step 2 (optional; only required if any of the fields are populated)
		failedItemSeq	***failedItemSeq***

Continued on next page

CA Generates CertRes, continued

Create CertRes (continued)

Step	Action	
4	Construct *CertResData*:	
	rrpid	***req*.rrpid**
	lid-EE	***req*.lid-EE**
	chall-EE3	***req*.chall-EE3**
	lid-CA	***req*.lid-CA** if present; otherwise generate a unique local identifier
	certStatus	the result of Step 3
	certThumbs	the Thumbprint(s) of ***certs*** (if specified)
	brandCRLIdentifier	the current **BrandCRLIdentifier** (if not specified in ***req*.thumbs**)
	thumbs	***req*.thumbs**
5	If *CAMsg* was generated in Step 2, continue with Step 7; otherwise, invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input:	
	s	the CA's signature certificate
	t	the result of Step 4
	type	*id-set-content-CertResData*
	certs	***certs***
6	Append the result of Step 5 to the tag [0]. Continue with Step 9.	
7	Invoke "Compose *EncK*" on page **Error! Bookmark not defined.** with the following input:	
	k	***req*.caBackKeyData.backKey**
	s	the CA's signature certificate
	t	the result of Step 4
	type-t	*id-set-content-CertResData*
	type-s	*id-set-content-CertResTBE*
	aid	***req*.caBackKeyData.backAlgID**
	certs	***certs***
8	Append the result of Step 7 to the tag [1].	
9	Store the result of Step 4. See "Store **CertRes**" on page 283.	

Continued on next page

CA Generates CertRes, continued

Create CertRes (continued)

Step	Action
10	Invoke "*Send Message*" on page **Error! Bookmark not defined.** with the following input: ***recip*** — the end entity ***msg*** — the result of Step 6 or Step 8 ***ext*** — ***ext*** ***rrpid*** — ***req*.rrpid** ***lid-C*** — if a Cardholder, ***req*.lid-EE** ***lid-M*** — if a Merchant or Payment Gateway, ***req*.lid-EE**

recip	the end entity
msg	the result of Step 6 or Step 8
ext	***ext***
rrpid	***req*.rrpid**
lid-C	if a Cardholder, ***req*.lid-EE**
lid-M	if a Merchant or Payment Gateway, ***req*.lid-EE**

Store CertRes

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

Continued on next page

CA Generates CertRes, continued

CertRes data

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 351.*
Thumbs	*Copied from* **CertReq**.
CertStatusCode	*Enumerated code indicating the status of the certificate request. See page 286.*
Nonce-CCA	*If request is complete and from a cardholder, the other half of the ultimate shared secret between Cardholder and CCA. See* **PANData0** *on page 273. Present only if EE is Cardholder.*

Table 15: CertRes Data

Continued on next page

CA Generates CertRes, continued

CertRes data (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 15: CertRes Data, continued

Continued on next page

CA Generates CertRes, continued

CertStatusCode values

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

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 16: Enumerated Values for CertStatusCode

End Entity Processes CertRes

Process CertRes

Step	Action
1	Receive as input: ***hdr*** — an instance of *MessageHeader* ***msg*** — a tag followed by an instance of *SignedData* or *EncryptedData* ***ext*** — any message extension(s) required to support additional business functions (optional)
2	Examine the tag at the beginning of ***msg*** to determine whether the sender used *S* or *EncK*. If the *S* version of **CertRes** is used, continue with Step 3; otherwise continue with Step 4.
3	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***d*** — ***msg*** (without the leading tag [0]) ***type*** — *id-set-content-CertResData* Designate the value of ***t*** returned as ***res***. Continue with Step 5.
4	From the message database, retrieve the instance of *CertReq* whose **RRPID** matches ***res*.rrpid**. • If found, designate it as ***req***; if a Cardholder, retrieve the corresponding entry for ***PANData0***. • If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *unknownRRPID*
5	Invoke "Verify *EncK*" on page **Error! Bookmark not defined.** with the following input: ***k*** — ***req*.caBackKeyData.caKey** ***d*** — ***msg*** (without the leading tag [1]) ***type-t*** — *id-set-content-CertReqTBE* ***type-s*** — *id-set-content-CertReqData* Designate the value of ***t*** returned as ***res***.

Continued on next page

End Entity Processes CertRes, continued

Process CertRes (continued)

Step	Action
6	Validate the following contents of ***res***: | *rrpid* | ***hdr*.rrpid** | | *lid-EE* | if ***req*.requestType** indicates a cardholder certificate request, ***hdr*.lid-C**, otherwise ***hdr*.lid-M** | If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: | ***errorCode*** | *wrapperMsgMismatch* |
7	Validate the following contents of ***res***: | *lid-EE* | ***req*.lid-EE** | | *chall-EE3* | ***req*.chall-EE3** | | *thumbs* | ***req*.thumbs** | If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed: | ***errorCode*** | *lid-EE* | *unknownLID* | | | *chall-EE* | *challengeMismatch* | | | *thumbs* | *thumbsMismatch* |
8	| **If *res*.certStatus.certStatusCode is:** | **go to Step** | **on Page:** | | *regFormAnswerMalformed* | A | 289 | | *requestPended* | B | 289 | | *requestInProgress* | B | 289 | | *requestComplete* | C | 290 | | any other value | D | 291 |

Continued on next page

End Entity Processes CertRes, continued

Process CertRes (continued)

Step	Action
A	This processing applies when **certStatusCode** is *regFormAnswerMalformed.*
9	Retrieve the registration form created in Step 2 of "Create **CertReq**" on page 265 and restore the user's input.
10	For each visible item in ***res*.certStatus.failedItemSeq**, indicate the failure to the end entity and make the **itemReason** available in a manner consistent with the user interface design of the application.
11	Continue with Step 4 of "Create **CertReq**" on page 265.
B	This processing applies when **certStatusCode** is *requestPended* or *requestInProgress.*
12	If ***res*.certStatus.eeMessage** is included, display it to the user.
13	Perform any combination of the following actions: • If ***res*.certStatus.eeMessage** is not included, display a message indicating that the processing of the request is incomplete and the information must be retrieved from the CA at a later time. • Automatically perform the *CertInqReq* processing after waiting for a reasonable length of time (for example, one hour). • Allow the user to initiate *CertInqReq* processing; the application should wait a reasonable length of time for the *CertInqRes* before allowing the user to initiate an additional *CertInqReq.*
14	Store ***res*** in the message database. Note: This step completes the processing of the *CertRes* for *requestPended* or *requestInProgress.*

Continued on next page

End Entity Processes CertRes, continued

Process CertRes (continued)

Step	Action
C	This processing applies when **certStatusCode** is *requestComplete*.
15	For each thumbprint in ***res*.certThumbs,** search the untrusted certificate cache for a certificate matching it. If found, continue with Step 17; otherwise, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *missingCertificateCRLorBCI*
16	Invoke "Verify certificate" on page **Error! Bookmark not defined.** with the following input: ***cert*** — the result of Step 15
17	For each certificate retrieved in Step 15, verify that its public key appears in ***req*.publicKeySorE**; if not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *invalidCertificateCRLorBCI* Repeat Steps 15 through 17 if an additional thumbprint appears in ***res*.certThumbs**. ~~Verify that the CertThumbs received match those sent in the CertReq. If not, invoke "SET Error Processing" on page with the following input: *ErrorCode* thumbsMismatch~~
18	If the end entity is not a cardholder, continue with Step 24. If specified, retrieve the logos ***res*.certStatus.caMsg.cardLogoURL** and ***res*.certStatus.caMsg.cardLogoURL**; store the logos and ***res*.certStatus.caMsg.cardCurrency** in persistent storage for usage during purchase processing. If ***res*.certStatus.caMsg.cardholderMsg** is specified, display it now along with the logos.
19	If ***res*.certStatus.nonceCCA** is not specified, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *missingData*
20	Exclusive-or ***res*.certStatus.nonceCCA** with ***PANData0*.cardSecret** to obtain **PANSecret**.
21	Append ***PANData0*.cardExpiry** to ***PANData0*.pan**.

Continued on next page

End Entity Processes CertRes, continued

Process CertRes (continued)

Step	Action
22	Invoke "HMAC" on page **Error! Bookmark not defined.** with the following input: ***t*** — the result of Step 21 ***k*** — the result of Step 20
23	Base64-encode the results of Step 22. (For information about base64 encoding, see RFC 1521, listed in "Related documentation" in the Preface.)
24	For each certificate retrieved in Step 16, verify that the *commonName* component of the *Subject Name* matches the result of Step 23; if not, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *invalidCertificateCRLorBCI*
25	Store the certificate(s) retrieved in Step 15.
26	If ***res*.certStatus.eeMessage** is included, display it to the user; otherwise, display a message indicating that the processing of the request is complete. Note: This step completes the processing of the *CertRes* for *requestComplete*.
D	This processing applies for all other values of **certStatusCode** (that is, all values except *regFormAnswerMalformed*, *requestPended*, *requestInProgress*, and *requestComplete*).
27	If ***res*.certStatus.eeMessage** is included, display it to the user; otherwise, display a message indicating that the processing of the request cannot be completed. Note: This step completes the processing of the *CertRes* for error responses.

Section 6
Certificate Inquiry Request/Response Processing

Overview

Certificate inquiry protocol

If a **CertRes** is returned without a certificate, the end entity can request the status of the certificate request by sending a **CertInqReq** to the CA. The **CertInqRes** will return the certificate if it is ready~~ or will provide information as to when the certificate will be ready~~. This is the only way for the end entity to get the certificate if it was not returned in **CertRes**.

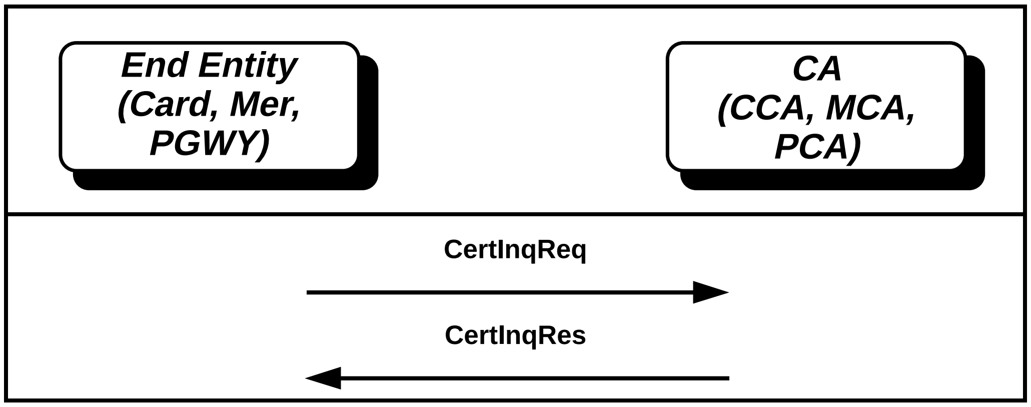

Figure 8: Certificate Inquiry Protocol

End Entity Generates CertInqReq

Create CertInqReq

Step	Action
1	From the message database, retrieve *CertReqData* (and designate it as ***req***) and *CertResData* (and designate it as ***res***).
2	Construct *CertInqReqTBS*: *rrpid*: a fresh statistically unique **RRPID** *lid-EE*: ~~a unique local identifier~~ ***res*****.lid-EE** *chall-EE3*: a fresh random challenge *lid-CA*: ***res*****.lid-CA**
3	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***s***: if ***req*****.requestType** is *signature certificate only* or *both certificates*, the signature key pair in ***req*****.publicKeySorE.publicKeyS**; otherwise, the signature certificate of the end entity ***s2***: if ***res*****.requestType** indicates a renewal, the existing signature certificate of the end entity ***t***: the result of Step 2 ***type***: *id-set-content-CertInqReqTBS*
4	Invoke "Send *Message*" on page **Error! Bookmark not defined.** with the following input: ***recip***: the CA ***msg***: the result of Step 3 ***ext***: any message extension(s) required to support additional business functions (optional) ***rrpid***: *rrpid* from Step 2 ***lid-C***: if a Cardholder, ***res*****.lid-EE** ***lid-M***: if a Merchant or Payment Gateway, ***res*****.lid-EE**

Continued on next page

End Entity Generates CertInqReq, continued

CertInqReq data

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 17: CertInqReq Data

CA Processes CertInqReq

Process CertInqReq

Step	Action
1	Receive as input: ***hdr*** — an instance of *MessageHeader* ***msg*** — an instance of *CertInqReq* ***ext*** — any message extension(s) required to support additional business functions (optional)
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***d*** — ***msg*** ***type*** — *id-set-content-CertInqReqTBS* ***unauthOK*** — TRUE Designate the value of ***t*** returned as ***req***.
3	From the message database, retrieve the instance of *CertReqData* whose **lid-CA** matches ***req*.lid-CA**. • If found, designate it as ***certReq*** and retrieve the corresponding entries for *CertStatusCode* (and designate it as ***status***), *FailedItemSeq* (and designate it as ***failedItems***), and *Certificates* (and designate it as ***certs***). • If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *unknownLID*
4	If ***si*** was not returned in Step 2, continue with Step 6. If ***certReq*.publicKeySorE.publicKeyS** is not specified, set ***status*** to *sigValidationFailure* and continue with Step 7.
5	Verify the signature in ***si*** using ***certReq*.publicKeySorE.publicKeyS**. If it does not verify: • set ***status*** to *sigValidationFailure*; • initialize ***failedItems*** so that it contains zero entries; • initialize ***certs*** so that it contains zero entries; and • continue with Step 7.

Continued on next page

CA Processes CertInqReq, continued

Process CertInqReq (continued)

Step	Action
6	Validate the following contents of ***req***: *rrpid* | ***hdr*.rrpid** *lid-EE* | if ***certReq*.requestType** indicates a cardholder certificate request, ***hdr*.lid-C**, otherwise ***hdr*.lid-M** If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** | *~~unknownRRPID~~ wrapperMsgMismatch*
7	Invoke "Create **CertInqRes**" on page 297 with the following input: ***req*** | ***req*** ***certReq*** | ***certReq*** ***status*** | ***status*** ***failedItems*** | ***failedItems*** ***certs*** | ***certs*** ***ext*** | any message extension(s) required to support additional business functions (optional)

CA Generates CertInqRes

Create CertInqRes

Step	Action
1	Receive as input:

req	an instance of *CertInqReqTBS*
certReq	an instance of *CertReqData*
status	an instance of *CertStatusCode*
failedItems	an instance of *FailedItemSeq* (optional)
certs	an instance of *Certificates* (optional)
ext	any message extension(s) required to support additional business functions (optional)

Step	Action
2	If ***certReq*.requestType** indicates a request from a cardholder and ***status*** is *requestComplete*, construct *CAMsg*:

cardLogoURL	the URL for financial institution logo(s) (optional)
brandLogoURL	the URL for brand logo(s) (optional)
cardCurrency	the ISO 4217 value corresponding to the cardholder's billing currency (optional)
cardholderMsg	a message from the CA (optional)

Refer to Appendix F: Logo Display during Certificate Registration for additional information on logo URLs.

Step	Action
3	Construct *CertStatus*:

certStatusCode	***status***
nonceCCA	a fresh nonce (if ***certReq*.requestType** indicates a request from a cardholder and ***status*** is *requestComplete)*
eeMessage	a message from the CA (optional; not included for cardholder requests where ***status*** is *requestComplete*)
caMsg	the result of Step 2 (optional; only required if any of the fields are populated)
failedItemSeq	***failedItemSeq***

Continued on next page

CA Generates CertInqRes, continued

Create CertInqRes (continued)

Step	Action	
4	Construct *CertResData*:	
	rrpid	***req*.rrpid**
	lid-EE	***req*.lid-EE**
	chall-EE3	***req*.chall-EE3**
	lid-CA	***req*.lid-CA**
	certStatus	the result of Step 3
	certThumbs	the thumbprint(s) of ***certs*** (if specified)
	brandCRLIdentifier	the current **BrandCRLIdentifier** (if not specified in ***certReq*.thumbs**)
	thumbs	***certReq*.thumbs**
5	If *CAMsg* was generated in Step 2, continue with Step 7; otherwise, invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input:	
	s	the CA's signature certificate
	t	the result of Step 4
	type	*id-set-content-CertResData*
	certs	***certs***
6	Append the result of Step 5 to the tag [0]. Continue with Step 9.	
7	Invoke "Compose *EncK*" on page **Error! Bookmark not defined.** with the following input:	
	k	***certReq*.caBackKeyData.backKey**
	s	the CA's signature certificate
	t	the result of Step 4
	type-t	*id-set-content-CertResData*
	type-s	*id-set-content-CertResTBE*
	aid	***certReq*.caBackKeyData.backAlgID**
	certs	***certs***
8	Append the result of Step 7 to the tag [1].	
9	Store the result of Step 4 in the message database. See "Store **CertInqRes**" on page 300.	

Continued on next page

CA Generates CertInqRes, continued

Create CertInqRes (continued)

Step	Action
10	Invoke "Send *Message*" on page **Error! Bookmark not defined.** with the following input:

recip	the end entity
msg	the result of Step 6 or Step 8
ext	***ext***
rrpid	***req*.rrpid**
lid-C	if a Cardholder, ***req*.lid-EE**
lid-M	if a Merchant or Payment Gateway, ***req*.lid-EE**

Store CertInqRes

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

CertInqRes data

The **CertInqRes** contains the same data as the **CertRes**.

CertInqRes	*Identical to a* **CertRes**. *See page 284.*

Table 18: CertInqRes Data

End Entity Processes CertInqRes

Process CertInqRes

Step	Action
1	Receive as input: ***hdr*** — an instance of *MessageHeader* ***msg*** — a tag followed by an instance of *SignedData* or *EncryptedData* ***ext*** — any message extension(s) required to support additional business functions (optional)
2	Invoke "Process **CertRes**" on page 287 with the following input: ***hdr*** — ***hdr*** ***msg*** — ***msg*** ***ext*** — ***ext***

Section 7 Registration Form Creation

Overview

Registration form creation

Registration forms are typically created in advance of certificate processing. This section describes how to create the registration forms.

Create RegFields

Step	Action
1	For each field in the registration form, construct *RegField*:
	fieldID — object identifier for the content of the field (optional)
	fieldName — text that will be displayed to the user as a label for the input field
	fieldDesc — supplementary text that will be displayed to the user for assistance in completing the form (optional)
	fieldLen — the maximum length of the field (optional; default is 128)
	fieldRequired — Boolean value indicating if the field must be provided
	fieldInvisible — Boolean value indicating if the field is invisible (not displayed to the user) and therefore populated by the application
	For additional information on values for *fieldId*, see Appendix L: Object Identifiers for Registration Form Fields.
2	Assign a unique *RegFormID* to the results of Step 1.
3	Store the results of Steps 1 and 2 for use during certificate processing. Note: The method for selecting a particular registration form is determined by brand and financial institution policy.

Chapter 3
Certificate Revocation or Cancellation

Overview

Introduction

A certificate may need to be revoked or canceled 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.

This chapter describes the process of revoking or canceling a SET certificate.

revoke	A compromised certificate is *revoked* if it is placed on a Certificate Revocation List (CRL).
cancel	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:

- Purpose of Certificate Revocation
- Cardholder Certificate Cancellation
- Merchant Certificate Cancellation
- Payment Gateway Certificate Revocation
- Higher Level Compromise Recovery
- Brand CRL Identifier
- CRL Responsibilities

Assumptions

In defining SET certificate revocation/cancellation, the following assumptions were made:

- The process shall minimize change to Issuers' existing payment card systems and maximize the reuse of existing payment card infrastructures.
- Because a Cardholder certificate is bound to the payment card account:
  - When a payment card number is canceled, the associated certificate will be canceled.
  - When a payment card is lost or stolen or the account is terminated, the certificate is no longer usable.
- When a Merchant's certificate is canceled, only the Acquirer needs to know, because:
  - 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.
  - 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.

Purpose of Certificate Revocation

Cardholder protection

Payment Gateway certificates - 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 CRLs that are distributed to Cardholders. Cardholder applications shall ~~will~~ identify Payment Gateway certificates created using a revoked 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.

Merchant certificates - 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 identify Merchant certificates created using a revoked CA certificate.

Continued on next page

Purpose of Certificate Revocation, continued

Merchant protection

Payment Gateway certificates - 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 shall ~~will~~ identify Payment Gateway certificates created using a revoked 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.

Cardholder certificates - Merchants do not need to verify the validity of Cardholder certificates to protect payment information. The Merchant shall ~~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 protection

Cardholder certificates - The Payment Gateway shall:

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

Merchant certificates - The Payment Gateway shall verify that:

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

Cardholder Certificate Cancellation

Purpose

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

Issuer-based cancellation approach

Since payment requests will go to the Cardholder's Issuer for payment authorization, the Issuer can ~~maintain the Cardholder certificate canceled status in the context of~~ determine that a Cardholder certificate is canceled by determining that it is in the payment card exception files currently maintained. When the Issuer receives a payment authorization request from the Acquirer for an account with a canceled certificate, the Issuer will reject the request.

Issuer-based cancellation approach

Since payment requests will go to the Cardholder's Issuer for payment authorization, the Issuer can determine that a Cardholder certificate is canceled by determining that it is in the payment card exception files currently maintained. When the Issuer receives a payment authorization request from the Acquirer for an account with a canceled certificate, the Issuer will reject the request.

Merchant Certificate Cancellation

Purpose

A Merchant's certificates and associated private keys are used to provide and authenticate the Merchant's identity. 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

If a Merchant terminates its association with a specific Acquirer, the Acquirer has 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.

Payment Gateway Certificate Revocation

Background

A Payment Gateway has two certificates:

- a key encryption certificate, used for encrypting Payment Instructions, and
- a 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~~ shall be stored on hardware cryptographic modules with restricted physical access.

Revoking a certificate

In the event that any of the Payment Gateway's private keys is compromised (or suspected of compromise), the Acquirer should ~~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.

Distributing new certificate

Once new certificates are distributed to the Payment Gateway, Merchants will receive them 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).

Higher Level Compromise Recovery

CA compromise recovery

The likelihood of a successful attack against a CA is very low. However, if a successful attack does occur, a new CA certificate will ~~shall~~ be distributed and the old certificate will ~~shall~~ be revoked. The identity of any compromised CA certificate is included in a CA CRL and distributed to all entities in the system.

Brand CA compromise recovery

If the unlikely compromise of a Brand CA private key does occur, the Root CA will generate and distribute a CRL identifying the Brand CA certificate.

Root key compromise recovery

In the still more unlikely event of a Root key compromise, the Root certificate and all of the Brand CA certificates in its hierarchy will be revoked and a Root CRL generated. This CRL will be distributed via a secure mechanism to the brands. This will effectively shut down the Root and Brand CAs (and their hierarchies) until a new Root key pair can be generated and new Root and Brand CA certificates created and distributed.

Brand CRL Identifier

Overview

All current CRLs are listed in the Brand CRL Identifier (BCI).

The BCI is included in all response messages (unless Thumbprints in the corresponding request indicate that the end entity has the current BCI). Each SET entity uses the BCI to check that it holds all current CRLs.

Further detail

For more detail, see "CRL Responsibilities" on page 311, "Certificate Revocation List and Brand CRL Identifier" on page 345 and "CA to CA Messages" on page 354.

CRL Responsibilities

Brand rules

Each brand is responsible for managing CRLs within its own domain. The brand, therefore, will establish its own rules regarding CRLs, including:

- frequency;
- validity period;
- the circumstances under which an empty CRL may be required;
- the circumstances under which an empty CRL may be distributed.

Empty CRL

The brand may choose to use an empty CRL as the means by which it is advised that the CRL of a CA within its domain no longer has any revoked certificates whose validity period has not expired. (That is, the CRL for this CA previously listed revoked certificates. Now all those revoked certificates have expired and there are no other revoked certificates to list.)

Because the BCI indicates all CRLs of significance, even if a brand chooses to receive empty CRLs, it will probably choose not to list them on the BCI under normal circumstances.

Under certain circumstances, the brand may find it convenient to list an empty CRL. For example, if the previous CRL issued by a CA was populated, listing the empty CRL can serve as an indicator that all the certificates on that previous CRL have expired and need no longer be considered.

Brand CA

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.

Geopolitical CA

The Geopolitical CA will generate, maintain, and distribute CRLs for compromised certificates that it generated and signed.

Cardholder CA

While the CCA will not generate and maintain a CRL, it will be responsible for distributing CRLs created by Root, Brand, Geopolitical, and Payment Gateway CAs. (Revoked cardholder certificates do not require a CRL, as the Issuer will decline transactions for those accounts.)

Merchant CA

While the MCA will not generate and maintain a CRL, it will be responsible for distributing CRLs created by Root, Brand, Geopolitical, and Payment Gateway CAs. (Revoked merchant certificates do not require a CRL, as the Acquirer will decline transactions that use them.)

Payment Gateway CA

The PCA is responsible for generating, maintaining, and distributing CRLs for compromised Payment Gateway certificates.

Chapter 4
Certificate Format

Overview

Introduction

This chapter describes 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.	313
2	X.509 Extensions	Describes all of the fields in the standard X.509 extensions used in SET.	321
3	SET Private Extensions	Describes all of the fields in the SET-specific extensions.	332
4	Certificate Profiles	Lists the certificates required for each SET entity and the possible extensions for each.	340

Section 1
X.509 Certificate Definition

Certificate Data

Restrictions

Table 19 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~~ 2.
serialNumber	INTEGER	Unique serial number assigned by the CA that issued the certificate.
signature .AlgorithmIdentifier	OBJECT IDENTIFIER	Defines the algorithm used to sign the certificate.
issuer	Name (see page 315)	Contains the Distinguished Name (DN) of the CA that issued the certificate.
validity .notBefore	UTCTime	Specifies when the certificate becomes active.
validity .notAfter	UTCTime	Specifies when the certificate expires. For a Cardholder certificate, the validity period shall not extend beyond the card's expiration date.
subject	Name (see page 315)	Contains the DN of the entity owning the key.
subjectPublicKeyInfo .algorithm .AlgorithmIdentifier	OBJECT IDENTIFIER	Specifies which algorithms can be used with this key. In a SET certificate, *id*-*rsaEncryption* is the only permissible value ~~for subjectPublicKeyInfo. AlgorithmIdentifier~~.
subjectPublicKeyInfo .subjectPublicKey	BIT STRING	Contains the public key provided in the certificate request.

Table 19: X.509 Certificate Data Definitions

Continued on next page

Certificate Data, continued

Restrictions (continued)

Name	Format and Value Restrictions	Description
issuerUniqueID		Not used in SET.
subjectUniqueID		Not used in SET.
extensions .extnID	OBJECT IDENTIFIER	Contains the extension's object identifier as defined by X.509 or SET.
Extensions .critical	BOOLEAN	Each extension description states how this field will be set.
Extensions .extnValue		The extension data.

Table 19: X.509 Certificate Data Definitions, continued

Extensions

Additional information about the encoding of the X.509 extensions supported in SET can be found in Section 2 beginning on page 313. Additional information about SET private certificate extensions can be found in Section 3 beginning on page 332.

Subject Public Key

The BIT STRING of *subjectPublicKeyInfo.subjectPublicKey* contains the DER-encoded representation of the following:

```
RSASubjectPublicKey ::= SEQUENCE {
   modulus          INTEGER,
   publicExponent   INTEGER
}
```

Certificate Name Format

Name fields

Certificates and certificate extensions include Issuer and Subject Name fields. Each is constructed from a combination of the following components:

- countryName
- organizationName
- organizationalUnitName
- commonName

Table 20 through Table 21 on pages 315 through 316 define the components of the Name fields in increasing detail.

Name OIDs

The object identifiers of the Name fields are as follows:

countryName	{2 5 4 6}	id-at 6
organizationName	{2 5 4 10}	id-at 10
organizationalUnitName	{2 5 4 11}	id-at 11
commonName	{2 5 4 3}	id-at 3

Table 20: Name Object Identifiers

Continued on next page

Certificate Name Format, continued

Name components

Table 21 defines the components of the Name fields used to identify each SET entity.

entity identified	field	definition
Cardholder		
	countryName	country of issuing financial institution
	organizationName	BrandID
	organizationalUnitName[1]	Descriptive Name of issuing financial institution
	organizationalUnitName	optional - promotional card name
	commonName	unique cardholder ID
Merchant		
	countryName	country of acquiring financial institution
	organizationName	BrandID
	organizationalUnitName	Descriptive Name of acquiring financial institution
	commonName	Descriptive Name of merchant ~~as printed on cardholder statement~~
Payment Gateway		
	countryName	country of acquiring financial institution
	organizationName	BrandID
	organizationalUnitName	Descriptive Name of acquiring financial institution
	commonName	unique Payment Gateway ID

Table 21: Certificate Name Components

Continued on next page

[1] To distinguish between the two organizationalUnitNames, the organizationalUnitName representing the "Descriptive Name of issuing financial institution" shall appear first in the generated Cardholder certificate.

Certificate Name Format, continued

Name components (continued)

entity identified	field	definition
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

Table 21: Certificate Name Components, continued

Continued on next page

Certificate Name Format, continued

Name components (continued)

entity identified	field	definition
Geopolitical Certificate Authority		
	countryName	country of geopolitical 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

Table 21: Certificate Name Components, continued

Continued on next page

Certificate Name Format, continued

Definitions

Table 22 further explains the definitions of the Name fields listed in Table 21 on page 316.

Name Field	Definition	
BrandID	<Brand Name>[:<Product>]	
	Brand Name	The brand of payment card.
	Product~~Type~~	Optional: the type of product within the brand.
country	The 2-character ISO 3166 country code.	
Descriptive Name	A descriptive name of a SET 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 merchant 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 programs, affinity programs, etc.	
unique cardholder ID	The unique Cardholder ID in the Cardholder's certificate is the keyed-hashed account number. See detailed description on page 320.	
unique ID	a unique identifier assigned to a Certificate Authority system to differentiate it from other systems of the same financial institution	
unique Payment Gateway ID	<BIN:Serial Number>	
	BIN	Bank Identification Number
	Acquirer- or brand-assigned serial number	Uniquely identifies each Payment Gateway associated with the same Acquirer. Multiple certificates may exist for a BIN within a brand.

Table 22: Definitions of Certificate Name Fields

Continued on next page

Certificate Name Format, 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**, which is comprised of a Cardholder secret value (**CardSecret**) and a CA secret value (**Nonce-CCA**).

Unique Cardholder ID	HMAC (**Text**, **PANSecret**)

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))`

PANSecret	a 20 byte value computed by computing the exclusive-or of **CardSecret** (the Cardholder nonce) and **Nonce-CCA** (the CA nonce)

Following the HMAC computation, the resulting digest is base64-encoded prior to being placed in the certificate commonName field. (For information about base64 encoding, see RFC 1521, listed in "Related documentation" in the Preface.)

Section 2 X.509 Extensions

Overview

Purpose

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

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

AuthorityKeyIdentifier Extension

Overview

The **AuthorityKeyIdentifier** extension identifies which CA certificate shall be used to verify the current certificate's signature. This extension contains the following fields:

- Key Identifier (keyIdentifier)
- Certificate Issuer (authorityCertIssuer)
- Certificate Serial Number (authorityCertSerialNumber)

~~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 inherited from the signing CA's certificate are used to populate authorityCertIssuer and authorityCertSerialNumber.~~

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
authorityKeyIdentifier .AuthorityKeyIdentifier .keyIdentifier		Not used in SET.
authorityKeyIdentifier .AuthorityKeyIdentifier .authorityCertIssuer	Name	Contains the Issuer DN of the issuing CA's certificate.
authorityKeyIdentifier .AuthorityKeyIdentifier .authorityCertSerialNumber	Positive INTEGER	Contains the serial number of the issuing CA's certificate.

Table 23: AuthorityKeyIdentifier Extension Restrictions

KeyUsage Extension

Overview

The **KeyUsage** extension indicates how the public key in the certificate may be used.

Key usage in SET

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

usage in SET:	X.509 terminology:
message signing	digital signature
certificate signing	certificate signature
CRL and BCI signing	CRL signature
message signing and key encryption	data encryption and key encryption
certificate, CRL, and BCI signing	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 (see next page)	Indicates whether the public key contained in the certificate may be used for signature verification, encryption, etc.

Table 24: KeyUsage Extension Restrictions

Continued on next page

KeyUsage Extension, continued

KeyUsage and BasicConstraints

The values set in the **KeyUsage** and **BasicConstraints** extensions shall not conflict. Table 25 indicates the **KeyUsage** and **BasicConstraints.cA** values for each EE or CA certificate. (For more information on **BasicConstraints**, see page 330.)

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

Table 25: KeyUsage and BasicConstraints Values

PrivateKeyUsagePeriod Extension

Overview

The **PrivateKeyUsagePeriod** extension delimits the period of time that the private key corresponding to the certificate is valid. This extension is used only 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	Generalized Time (see page **Error! Bookmark not defined.**)	The start date and time of the private key's validity period.
privateKeyUsagePeriod .PrivateKeyUsagePeriod .notAfter	Generalized Time	The end date and time of the private key's validity period.

Table 26: PrivateKeyUsagePeriod Extension Restrictions

CertificatePolicies Extension

Overview

The **CertificatePolicies** extension contains a list of one or more certificate policies. A certificate policy is a set of rules defining the use of the certificate in SET. Each certificate policy is denoted by a globally unique Object Identifier (OID) and may optionally contain corresponding qualifiers.

Each SET certificate shall contain at least one policy 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.

Qualifiers

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 their qualifiers

Root policy qualifier

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

- **policyURL**
- **policyEmail**
- **policyDigest**
- **terseStatement**

Each of these qualifiers is optional. The **policyURL** and **policyEmail** contain a URL and an electronic mail address where a copy of the Root policy statement can be obtained. A hash of the policy may be included in **policyDigest** 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 subordinate CA may add one qualifier to the Root policy in a subordinate certificate. The additional qualifier identifies a policy statement for that CA. Like the Root policy, it is indicated by an OID and may be qualified using **policyURL**, **policyEmail**, **policyDigest**, and/or **terseStatement**.

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 OIDs in a SET end entity certificate, belonging to:

- the Root CA,
- the Brand CA,
- the Geopolitical CA, and
- the Cardholder, Merchant, or Payment Gateway CA.

Certificate generation

A generated certificate shall inherit all of the relevant policy information of the CA signing certificate. The subordinate certificate may omit the policyQualifier component of the AdditionalPolicy sequence.

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
certificatePolicies .PolicyInformation .policyIdentifier	OBJECT IDENTIFIER	The OID that points to the Root policy statement. The policy may be obtained from the URL or electronic mail address provided in the qualifiers.
certificatePolicies .PolicyQualifierInfo. .policyQualifierId	OBJECT IDENTIFIER	Set to *id-set-setQualifier*.
certificatePolicies .PolicyQualifierInfo. .qualifier	SETQualifier	May contain: • optional qualifiers to the Root policy • up to three additional optional qualifying policies and their qualifiers

Table 27: CertificatePolicies Extension Restrictions

Continued on next page

CertificatePolicies Extension, continued

Restrictions (continued)

Name	Format and Value Restrictions	Description
setPolicyQualifier .additionalPolicies .policyOID	OBJECT IDENTIFIER	The OID that points to the CA's policy statement. The policy may be obtained from the URL or electronic mail address provided in the associated qualifiers.
setPolicyQualifier .additionalPolicies .policyAddedBy	Certificate Type	Indicates the CA that added the policy to the generated certificate and to which the policy corresponds.
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 .policyURL	IA5String	URL where a copy of the policy statement may be found.
SETQualifier .policyEmail	IA5String	Electronic mail address where a copy of the policy statement may be found.

Table 27: CertificatePolicies Extension Restrictions, continued

SubjectAltName Extension

Overview

The **SubjectAltName** 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 .GeneralName	Name	One or more alternate names for the Distinguished Name (DN) in the certificate; the alternate name may be an electronic mail address, a URL, etc.

Table 28: SubjectAltName Extension Restrictions

BasicConstraints Extension

Overview

The **BasicConstraints** extension indicates whether the certified subject may act as a CA or an end entity. 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 .BasicConstraintsSyntax .cA	BOOLEAN	True for all CAs and subordinate CAs; false for end entities.
basicConstraints .BasicConstraintsSyntax .pathLenConstraint	INTEGER	Indicates the number of levels of CAs for which this certificate may sign certificates. For example, a zero in this field means that the CA certificate may only be used to sign EE certificates.

Table 29: BasicConstraints Extension Restrictions

Usage

BasicConstraints.cA shall be set to CA (TRUE) 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**.

IssuerAltName Extension

Overview

The **IssuerAltName** 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 (DN) in the certificate; the alternate name may be an electronic mail address, a URL, etc.

Table 30: IssuerAltName Extension Restrictions

Section 3
SET Private Extensions

Overview

Purpose

This section defines the following private extensions for SET certificates:

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

If a SET application receives a certificate with a non-critical extension that it is unable to recognize, it shall ignore the extension. This provision is made so that future versions of SET can define new certificate extensions.

No brand, vendor, or national market is permitted to define new certificate extensions for use within SET.

HashedRootKey Private Extension

Overview

The **HashedRootKey** private 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	OBJECT IDENTIFIER	*id-sha1*
HashedRootKey .DigestedData .digestAlgorithm .parameters		Set to NULL.
hashedRootKey .DigestedData .contentInfo .contentType	OBJECT IDENTIFIER	Set to *id-set-rootKeyThumb*.
hashedRootKey .DigestedData .contentInfo .content		Omitted.
hashedRootKey .DigestedData .digest	OCTET STRING	The hash of the DER-encoded subjectPublicKeyInfo.

Table 31: HashedRootKey Private Extension Restrictions

CertificateType Private Extension

Overview

The **CertificateType** private extension is used to identify the entity in the SET CA hierarchy. It is independent of the **cA** indicator in the **BasicConstraints** extension which indicates whether the certificate may be used to verify certificate signatures.

This extension is included in every SET certificate.

Single certificate type

For the following end entity or CA types, a certificate can have only one type:

- Cardholder
- Merchant
- Payment Gateway
- Geopolitical Certificate Authority
- Brand Certificate Authority
- Root Certificate Authority

Multiple certificate types

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

- Cardholder Certificate Authority
- Merchant Certificate Authority
- Payment Certificate Authority

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.

Table 32: CertificateType Private Extension Restrictions

MerchantData Private Extension

Overview

The **MerchantData** private extension contains all of the data needed by the Payment Gateway about the merchant. This data is obtained from the merchant in the certificate request processing (in the registration form).

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 preference. 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
*	merNameSeq .language	Character string; optional	RFC 1766 definition of language
*	merNameSeq .name	Character string; required	The name by which the merchant is known to its customers
*	merNameSeq .city	Character string; required	The name of the city where the merchant is located
*	merNameSeq .stateProvince	Character string; optional	The state or province where the merchant is located
*	merNameSeq .postalCode	Character string; optional	The postal code for the merchant's location
*	merNameSeq .countryName	Character string; required	The name of the country (corresponds to merCountry)
	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 to accommodate non-SET business processing of transactions.

* The flagged items may appear more than once to carry information about the merchant in multiple character sets or translated into multiple languages.

Table 33: MerchantData Private Extension Restrictions

CardCertRequired Private Extension

Overview

The **CardCertRequired** private extension indicates whether the Payment Gateway supports exchanges with Cardholders that do not have a certificate.

Criticality

This extension is non-critical.

Restrictions

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

Table 34: CardCertRequired Private Extension Restrictions

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 supported by the Payment Gateway or the CA. The list is in order of the CA's or Payment Gateway'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 supported by the CA or Payment Gateway.

Table 35: Tunneling Private Extension Restrictions

SETExtensions Private Extension

Overview

The **SETExtensions** private extension lists the SET message extensions for payment instructions supported by the Payment Gateway. The Cardholder checks the Payment Gateway certificate prior to including critical message extensions in the Payment Instruction. Message extensions are indicated by Object Identifiers. This extension occurs in the Payment Gateway key encryption certificate only and is required, even if empty.

Criticality

This extension is non-critical.

Restrictions

Name	Format and Value Restrictions	Description
setExtensions .SETExtensionsSyntax	SEQUENCE OF OBJECT IDENTIFIER	List of Object Identifiers pointing to the message extensions supported by the Payment Gateway.

Table 36: SETExtensions Private Extension Restrictions

Populating for a Payment Gateway certificate

To populate **SETExtensions** for a Payment Gateway certificate, the PCA can include a field with the following characteristics on the registration form:

```
    RegField.fieldId = { id-set-field data(13) setExtensions(0) }
    RegField.fieldName = "any arbitrary name"
    RegField.fieldDesc = "any arbitrary description"
    RegField.fieldLen = 1024 -- a suitably large integer
    RegField.fieldRequired = FALSE
    RegField.fieldInvisible = TRUE
```

If the payment gateway supports any PI (Payment Instruction) extensions, it will populate the certificate request as follows:

```
    RegFormItems.fieldName = "any arbitrary name" -- matches request
    RegFormItems.fieldValue.octetString = SETExtensionsSyntax
```

Note: The data is a SEQUENCE OF OBJECT IDENTIFIER inside of an OCTET STRING.

Section 4
Certificate Profiles

Certificate Types

Summary

Table 37 lists all certificates needed in SET and the entities that require them.

Entity	Message Signing	Key Encryption	Certificate Signing	CRL Signing
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
Geopolitical Certificate Authority	X		X	X
Brand Certificate Authority			X	X
Root Certificate Authority			X	X

Table 37: 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 *keyCertSign* and *crlSign*, or
- both *keyEncipherment* and *dataEncipherment*.

No other functions shall ~~can~~ be combined into one certificate.

End Entity Certificate Extensions

	Cardholder Certificate	Merchant Certificate		Payment Gateway Certificate	
X.509 Extension	signature	signature	key/data encryption	signature	key/data encryption
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 38: End Entity Certificate Extensions

CA Certificate Extensions

CA certificate extensions

Several tables summarize the permissible and required extensions for various CA certificates.

• Root, Brand, and Geopolitical CAs	Table 39
• Payment Gateway CAs	Table 40 on page 343
• Merchant and Cardholder CAs	Table 41 on page 344

	Root CA	Brand CA	Geopolitical CA		
X.509 Extension	Certificate, CRL, and BCI Signing	Certificate, CRL, and BCI Signing	Message Signing	Certificate Signing	CRL and BCI Signing
AuthorityKeyIdentifier		X	X	X	X
KeyUsage	X	X	X	X	X
PrivateKeyUsagePeriod	X	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					
SETExtensions					

X = Required
O = Optional

Table 39: CA Certificate Extensions, Part I

Continued on next page

CA Certificate Extensions, continued

	Payment Gateway CA			
X.509 Extension	Message Signing	Key Encryption	Certificate Signing	CRL and BCI Signing
AuthorityKeyIdentifier	X	X	X	X
KeyUsage	X	X	X	X
PrivateKeyUsagePeriod	X		X	X
CertificatePolicies	X	X	X	X
SubjectAltName	O	O	O	O
BasicConstraints	X	X	X	X
IssuerAltName	O	O	O	O
Private Extension				
HashedRootKey				
CertificateType	X	X	X	X
MerchantData				
CardCertRequired				
Tunneling				
SETExtensions				

X = Required
O = Optional

Table 40: CA Certificate Extensions, Part II

Continued on next page

CA Certificate Extensions, continued

	Merchant CA			Cardholder CA		
X.509 Extension	Message Signing	Key Encryption	Certificate Signing	Message Signing	Key Encryption	Certificate Signing
AuthorityKeyIdentifier	**X**	**X**	**X**	**X**	**X**	**X**
KeyUsage	**X**	**X**	**X**	**X**	**X**	**X**
PrivateKeyUsagePeriod	**X**		**X**	**X**		**X**
CertificatePolicies	**X**	**X**	**X**	**X**	**X**	**X**
SubjectAltName	**O**	**O**	**O**	**O**	**O**	**O**
BasicConstraints	**X**	**X**	**X**	**X**	**X**	**X**
IssuerAltName	**O**	**O**	**O**	**O**	**O**	**O**
Private Extension						
HashedRootKey						
CertificateType	**X**	**X**	**X**	**X**	**X**	**X**
MerchantData						
CardCertRequired						
Tunneling					**X**	
SETExtensions						

X = Required
O = Optional

Table 41: CA Certificate Extensions, Part III

Chapter 5 Certificate Revocation List and Brand CRL Identifier

Overview

Introduction

This chapter describes the use of the Certificate Revocation List (CRL) and the Brand CRL Identifier (BCI) 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) shall ~~will~~ maintain a CRL as specified by brand rules. All CAs shall ~~will~~ distribute CRLs.

The 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 response 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:

Section	Topic	Contents	Page
1	X.509 CRL Data Definitions	Defines the data contained in X.509 CRLs.	346
2	CRL Extensions	Defines the use and content of the CRL extensions.	349
3	Brand CRL Identifier	Explains the concept and use of BCIs.	350

Section 1
X.509 CRL Data Definitions

Overview

CA responsibilities

A CA is responsible for revoking compromised certificates that it generated and signed. The CA shall ~~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.

Error! Reference source not found. on page **Error! Bookmark not defined.** lists the CAs which are responsible for maintaining and distributing CRLs in SET.

Organization

This section discusses:

- CRL Contents
- CRL Maintenance and Distribution

CRL Contents

Restrictions

Name	Format and Value Restrictions	Description
CRL .version	INTEGER	Indicates the CRL version. Always set to ~~2~~ 1
CRL .signature .AlgorithmIdentifier	OBJECT IDENTIFIER	Defines the algorithm used to sign the CRL.
CRL .issuer	Name	Contains the Subject Distinguished Name (DN) of the CA that issued the revoked certificate.
CRL .thisUpdate	UTCTime (see page **Error! Bookmark not defined.**)	Specifies when the CRL was generated.
CRL .nextUpdate	UTCTime	Specifies when the CRL expires. A new CRL may be generated before the previous one expires, if the list of revoked certificates changes.
CRL .revokedCertificates .CertificateSerialNumber	INTEGER	The serial numbers of the revoked certificates.
CRL .revokedCertificates .revocationDate	UTCTime	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**. (See page 349.)

Table 42: X.509 CRL Data Definitions

CRL Maintenance and Distribution

CRL update

A new CRL ~~is~~ may be created whenever a certificate is revoked and must be created when the current CRL expires ~~the list shall be updated~~. When the new CRL is created, any certificates on the list that have expired may be removed. The updated CRL will contain the complete list of all unexpired, revoked certificates that the CA issued.

CRL distribution

CRLs are distributed to CAs (including the Brand CA) and Payment Gateways using the distribution message discussed on page 361.

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 holds by putting the Thumbprints in the first request message. The recipient checks the Thumbprints and includes any missing CRLs in its response message.

Section 2
CRL Extensions

CRL extensions

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

X.509 Extension	Certificate Authority			
	PCA	Geopolitical CA	Brand CA	Root CA
CRLNumber	**X**	**X**	**X**	**X**
AuthorityKeyIdentifier	**X**	**X**	**X**	**X**

Table 43: Required CRL Extensions

AuthorityKeyIdentifier

The **AuthorityKeyIdentifier** extension is used the same way for CRLs as for certificates. See page 322.

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.

Table 44: CRLNumber CRL Extension Restrictions

Section 3
Brand CRL Identifier

Overview

Overview

The Brand CRL Identifier (BCI) is a structure defined by SET to identify all current CRLs for a given brand.

Each Brand CA maintains a single BCI, which contains a list of CRL numbers. The BCI is distributed in all 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 hierarchy updates a CRL.

Organization

This section includes the following topics:

- BCI Definition
- BCI Distribution and Usage

BCI Definition

Restrictions

Name	Format and Value Restrictions	Description
BrandCRLIdentifier .version	INTEGER	Indicates the BCI version. Always ~~1~~ 0.
BrandCRLIdentifier .sequenceNum	INTEGER	Increasing sequence number. The higher the sequence number, the more recent the BCI.
BrandCRLIdentifier .brandID	Name	The name of the brand that is issuing the BCI.
BrandCRLIdentifier .notBefore	Generalized Time (see page **Error! Bookmark not defined.**)	Specifies when the BCI becomes valid.
BrandCRLIdentifier .notAfter	Generalized Time	Specifies when the BCI expires.
BrandCRLIdentifier .crlIdentifierSeq .issuerName	Name	The Issuer Distinguished Name (DN) of a CRL that needs to be used in signature validations. The following entities may appear: • Root CA • Brand CA • Geopolitical CA • Payment Gateway CA
BrandCRLIdentifier .crlIdentifierSeq .crlNumber	INTEGER	The value of the **CRLNumber** extension of the CRL.
BrandCRLIdentifier .Extensions	Extensions	The only extension used in the BCI is **AuthorityKeyIdentifier**, which includes the Issuer DN and serial number of the Brand CA certificate that was used to sign this BCI. The same restrictions are applied as in its use in CRLs and certificates. See page 322.

Table 45: BrandCRLIdentifier Restrictions

Continued on next page

BCI Definition, continued

Restrictions (continued)

Name	Format and Value Restrictions	Description
AlgorithmIdentifier .algorithm	OBJECT IDENTIFIER	
AlgorithmIdentifier .parameters	NULL	

Table 45: BrandCRLIdentifier Restrictions, continued

Signature

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

Appropriate contents

See "Empty CRL" on page 311 for information about when to list a CRL with no entries.

BCI Distribution and Usage

BCI distribution to Cardholders and Merchants

BCIs are distributed to Cardholders and Merchants within response messages. An entity in the SET protocol indicates which BCI it is holding by putting its Thumbprint in a request message. The recipient compares the Thumbprint to that of the latest BCI. If the requesting entity is not holding the latest BCI, the responder includes the new BCI in its 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 366.

BCI updates

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

When to process BCIs

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

Chapter 6
CA to CA Messages

Overview

Organization

This chapter addresses the following topics:

Section 1
CA to CA Certificate Requests and Responses

Introduction

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* message.

CA certificate issuance

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.

CA certificate renewal

The protocol for CA certificate renewal is identical to that used for initial issuance.

Organization

This section includes the following topics:

- Subordinate CA Generates *CertificationRequest*
- Superior CA Processes *CertificationRequest*
- Superior CA Generates *CertificationResponse*
- Subordinate CA Processes *CertificationResponse*

Subordinate CA Generates CertificationRequest

Create CertificationRequest

Step	Action
1	Construct *RSASubjectPublicKey* (see "Subject Public Key" on page 314): *modulus*: the modulus shared by the public and private keys *publicExponent*: the public exponent of the key pair
2	Encode the result of Step 1 as a BIT STRING.
3	Construct *SubjectPublicKeyInfo*: *algorithm*: *id-rsaEncryption* *subjectPublicKey*: the result of Step 2
4	Construct *CertificationRequestInfo*: *version*: *0* *subject*: the subject Name as described in "Certificate Name Format" on page 315 *subjectPublicKeyInfo*: the result of Step 3 *attributes*: up to five attributes from Table 47 that will appear in the certificate
5	Sign the results of Step 4 using the private key corresponding to the result of Step 1. Note: Generating a signature with an encryption key is possible because RSA is the only supported algorithm.
6	Verify the result of Step 5. (The validation method is at the discretion of the application developer.) If the verification fails, abort processing. Note: In a fully debugged system, this is an indication that the signature generation process is under attack to try to determine the private key.
7	Construct *SIGNED*: *toBeSigned*: the result of Step 4 *algorithm*: *id-rsaEncryption* *signature*: the result of Step 5
8	Store the result of Step 4 in the message database.
9	Pass the result of Step 7 to the transport mechanism. Depending on the transport mechanism, the message may be wrapped (for example, with a MIME or HTTP header). Transport of the *CertificationRequest* from the subordinate CA to the superior CA shall be coordinated out-of-band.

Continued on next page

Subordinate CA Generates CertificationRequest, continued

Restrictions

Table 46 defines the format and value restrictions for each field in the **CertificationRequest** message.

Name	Format and Value Restrictions	Description
version	INTEGER	Indicates the version of the *CertificationRequest*. Always set to *0*.
subject	Name (see page 315)	Contains the subject Distinguished Name of the entity that owns the key.
subjectPublicKeyInfo .algorithm .AlgorithmIdentifier	OBJECT IDENTIFIER	Specifies which algorithms can be used with this key. In a SET certificate, *id-rsaEncryption* is the only permissible value.
subjectPublicKeyInfo .subjectPublicKey	BIT STRING	Contains the public key provided in the *CertificationRequest*.
attributes		Up to five attributes from Table 47 that will appear in the certificate.

Table 46: CertificationRequest Data Definitions

Attributes

	CCA, MCA, or PCA				Geopolitical or Brand CA
SET Attribute	Message Signing	Certificate Signing	Key Encryption	CRL and BCI Signing	Certificate, CRL, and BCI signing
KeyUsage	**X**	**X**	**X**	**X**	**X**
PrivateKeyUsagePeriod	**X**	**X**		**X**	**X**
SubjectAltName	**O**	**O**	**O**	**O**	**O**
CertificateType	**X**	**X**	**X**	**X**	**X**
Tunneling			**X**		
~~**AdditionalPolicy**~~	~~**O**~~	~~**O**~~	~~**O**~~	~~**O**~~	~~**O**~~

X = Required
O = Optional

Table 47: Required CertificationRequest Attributes

Superior CA Processes CertificationRequest

Process CertificationRequest

Step	Action
1	Receive a *SIGNED CertificationRequestInfo* from the transport mechanism.
2	Verify the signature using the public key identified in *CertificationRequestInfo.subjectPublicKeyInfo*. If the signature verification fails, stop processing the request.
3	Validate the following contents of *CertificationRequestInfo*:

version	0	
subject	complies with the "Certificate Name Format" specified on page 315	
subjectPublicKeyInfo	*algorithm*	id-rsaEncryption
attributes	as required for the certificate type and key usage attributes according to Table 47 on page 357 Note: the certificate type must be determined out-of-band to SET.	

If the validation fails, stop processing the request.

Step	Action
4	Verify the authenticity of the request using the brand-specified procedure. If the verification fails, stop processing the request.
5	Invoke "Create **CertificationResponse**" on page 359.

Validation errors

If errors are encountered during the validation process, the certificate shall not be generated and the failure shall be communicated out-of-band.

Superior CA Generates CertificationResponse

Create CertificationResponse

Step	Action
1	If validation is successful, generate the certificate using the attributes included in the request.
2	Construct *SignedData*: *sdVersion*: 2 *contentInfo*: *contentType* – *data* *certificates*: the result of Step 1 plus any certificates the recipient will need to authenticate it *crls*: CRLs that the recipient will require to process the message
3	Pass the result of Step 2 to the transport mechanism. Depending on the transport mechanism, the message may be wrapped (for example, with a MIME or HTTP header). Transport of the message from the superior CA to the subordinate CA shall be coordinated out-of-band.

Step 2 nested table (*SignedData*):

sdVersion	2	
contentInfo	*contentType*	*data*
certificates	the result of Step 1 plus any certificates the recipient will need to authenticate it	
crls	CRLs that the recipient will require to process the message	

Subordinate CA Processes CertificationResponse

Process CertificationResponse

Step	Action		
1	Receive as input:		
	msg	an instance of *SignedData*	
2	Validate the following contents of ***msg***:		
	sdVersion	*2*	
	If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed:		
	errorCode	*sdVersion*	*decodingFailure*
3	Retrieve the corresponding instance of *CertificationRequestInfo* from the message database.		
4	Locate a certificate in ***msg*.certificates** with a public key that matches *CertificationRequestInfo.subjectPublicKeyInfo*. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input:		
	errorCode	*missingCertificateCRLorBCI*	
5	Invoke "Verify Certificate" on page **Error! Bookmark not defined.** with the following input:		
	cert	the result of Step 4	
6	Store the received certificate.		

Section 2
CRL Distribution

CRL Distribution

Overview

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 Brand CAs. Whenever a subordinate CA updates its CRL, it shall distribute the CRL to the Brand CA, which shall transmit it to the brand's designated CA for BCI distribution. The Brand CA shall provide an agreed transport mechanism through which the related CAs can send CRL update messages.

Organization

This section includes the following topics:

- CA Generates *CRLNotification*
- CA Processes *CRLNotification*
- CA Generates *CRLNotificationRes*
- CA Processes *CRLNotificationRes*

CA Generates CRLNotification

Create CRLNotification

Step	Action
1	Receive as input: ***crl*** — an instance of *CRL*
2	Compute the SHA-1 hash of ***crl*.toBeSigned** (the *EncodedCRL*).
3	Construct *CRLNotificationTBS*: *date* — current date *crlThumbprint* — the result of Step 2
4	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***s*** — the notifying CA's digital signature certificate ***t*** — result of Step 1 ***type*** — *id-set-content-CRLNotificationTBS* ***certs*** — all of the certificates necessary to verify all of the CRLs ***crls*** — ***crl***
5	Store *CRLNotificationTBS*, the result of Step 3, for use in processing *CRLNotificationRes*.
6	Pass the result of Step 4 to the transport mechanism for transmission to the brand's designated CA. Depending on the transport mechanism, the message may be wrapped (for example, with a MIME or HTTP header). The transport of the *CRLNotification* is outside of the scope of SET.

CRLNotification data

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.*

Table 48: CRLNotification Data

BCA Processes CRLNotification

Process CRLNotification

Step	Action
1	Receive as input: ***msg*** — an instance of *SignedData*
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***d*** — ***msg*** ***type*** — *id-set-content-CRLNotificationTBS* Designate the value of ***t*** returned as ***req***.
3	Validate the following contents of ***req***: *date* — later than the date of any previous CRL received from this CA If errors are encountered during the validation process, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *messageTooOld*
4	Search the trusted certificate cache for a certificate whose Thumbprint matches ***req*.cRLThumbprint**. If found, continue with Step 6. Otherwise, search the untrusted certificate cache for it. If not found, invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input: ***errorCode*** — *missingCertificateCRLorBCI*
5	Invoke "Verify CRL" on page **Error! Bookmark not defined.** with the following input: ***newCrl*** — the result of Step 4
6	Store the CRL identified in Step 4 for inclusion with a subsequent **BCIDistribution** message.
7	Invoke "Create *CRLNotificationRes*" on page 364 with the following input: ***msg*** — ***req***

BCA Generates CRLNotificationRes

Create CRLNotificationRes

Step	Action
1	Receive as input: ***msg*** | an instance of *CRLNotificationTBS*
2	Construct *CRLNotificationResTBS*: *date* | ***msg*.date** *crlThumbprint* | ***msg*.crlThumbprint**
3	Invoke "Compose *SignedData (S)"* on page **Error! Bookmark not defined.** with the following input: ***s*** | the notifying CA's digital signature certificate ***t*** | the result of Step 2 ***type*** | *id-set-content-CRLNotificationResTBS*
4	Pass the result of Step 3 to the transport mechanism for transmission to the notifying CA. Depending on the transport mechanism, the message may be wrapped (for example, with a MIME or HTTP header). The transport of the *CRLNotificationRes* is outside of the scope of SET.

CRLNotificationRes data

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

Table 49: CRLNotificationRes Data

CA Processes CRLNotificationRes

Process CRLNotificationRes

Step	Action
1	Receive as input: ***msg*** | an instance of *SignedData*
2	Invoke "Verify *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***d*** | ***msg*** ***type*** | *id-set-content-CRLNotificationResTBS* Designate the value of ***t*** returned as ***res***.
3	Retrieve the corresponding *CRLNotificationTBS* and designate it as ***req***. Validate the following contents of ***res***: *date* | ***req*.date** *crlThumbprint* | ***req*.crlThumbprint** If errors are encountered during the validation process, repost the CRL in a new *CRLNotification* and invoke "Create **Error** Message" on page **Error! Bookmark not defined.** with the following input based on the field that failed: ***errorCode*** | *date* | *requestResponseMismatch* | *crlThumbprint* | *thumbsMismatch*
4	Repost the CRL in a new *CRLNotification*.

Section 3
BCI Retrieval

Overview

BCI host

Each SET brand shall maintain an up-to-date version of the BCI and all CRLs referenced by the BCI in a **BCIDistribution** message. The BCA may perform this function itself or may designate a GCA or PCA to manage the BCI.

The brand's designated CA shall generate a new **BCIDistribution** message daily and shall provide one or more mechanisms whereby the **BCIDistribution** message can be downloaded by the supported CAs and the Payment Gateways.

Note: While the distribution message is generated daily, the BCI is usually generated on a less frequent basis.

BCI recipients

Each CA and Payment Gateway shall retrieve the **BCIDistribution** message of each brand daily and shall include the up-to-date BCIs and the associated CRLs in their response messages.

CA Generates BCIDistribution Message

Create BCIDistribution

Step	Action
1	Construct *BCIDistributionTBS*: *date* — current date *brandCRLIdentifier* — the current BrandCRLIdentifier
2	Invoke "Compose *SignedData (S)*" on page **Error! Bookmark not defined.** with the following input: ***s*** — the CA's digital signature certificate ***t*** — the result of Step 1 ***type*** — *id-set-content-BCIDistributionTBS* ***certs*** — all of the certificates necessary to verify all of the CRLs ***crls*** — all of the CRLs listed on the BCI
3	Encode the result of Step 2 in a form suitable for the agreed transport mechanism. Depending on the transport mechanism, the message may be wrapped (for example, with a MIME or HTTP header).
4	Post the result of Step 3 so that it is available for download by the CAs and Payment Gateways.

BCIDistribution contents

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.*

Table 50: BCIDistribution Message

CA or Payment Gateway Processes BCIDistribution Message

Process BCIDistribution

Step	Action
1	Receive as input: ***msg*** — an instance of *SignedData*
2	Invoke “Verify *SignedData (S)*” on page **Error! Bookmark not defined.** with the following input: ***d*** — ***msg*** ***type*** — *id-set-content-BCIDistributionTBS*