Open Protocol for Access Control Identification and Ticketing with privacY

Specifications

Version 3.7 | July 15th, 2011

Document Control

Document Contributor(s)

Name	Department
Eric Le Saint	CTO Office
Dom Fedronic	CTO Office
Steven Liu	Contractor

Document Revision(s)

Date	Author	Version	Revision Description
April 7 2010	ELS	2.7	Added persistent binding to OPACITY ZKM
April 9 2010	ELS	2.8	Added optimizations from EngTeam feedback
April 12 2010	ELS	2.8.1	Added corrected P1/P2 values in Opacity ZKM
April 22 2010	ELS	2.8.2	Align KDF with 800-56A : Added Key Confirmation session key.
May 12 2010	ELS	3.0	Removed OPACITY-BASIC and OPACITY-FP and OPACITY-ZKM-FS. Updated Opacity-ZKM APDU interface
May 27 2010	ELS	3.3	Added SAM specifications. Described activation mechanisms for SAM. Added Illustrations
May 28 2010	ELS	3.4	Minor corrections
June 2 2010	ELS	3.4.1	Minor corrections + unwrap command
June 7 2010	ELS	3.4.2	unwrap command (cont.)
August 11, 2010	ELS	3.4.3	Corrected typos in protocol diagrams – Editorial review
August 12, 2010	DFE	3.4.4	Various definition edits, CVC definition edits including GUID removal replaced by SN in the CardHolderName of the CVC would include a cardSN instead of the GUID for FS and nothing for ZKM.
August 13, 2010	DFE/ELS	3.4.5	Review Domain concept and SAM activation. 2.2.11.3. introduce PIN based activation, add annex C on scalable persistent binding and multiple domains.
August 24, 2010	ELS	3.5	OPACITY-ZKM optionally returns a unique 16-byte globally unique identification code protected for confidentiality. Alternatively the ICC CVC optionally does not include an identifier tag 5F20.
August 31, 2010	ELS	3.5.1	OPACITY-ZKM optionally derives a single secure messaging session key instead of 3. Aligned the Control Byte (CB) bit field values.
September 15, 2010	ELS	3.6	OPACITY-ZKM optionally delivers an encrypted GUID.
July 15th, 2011	ELS	3.7	Renamed Forward Secrecy ➔ Full Secrecy. Updated FS Mode to comply with 800-56A with a sequence of two C(1,1) steps. Updated FS mode to allow the establishment of long

			session keys. Clarified in ZKM mode that CVC signature includes GUID, although the GUID is returned encrypted and outside the CVC in the Opacity ZKM response.

Related Document(s)

Reference	Author	Version	Revision Description
[1] SP 800-56A	NIST	Rev1 (March 2007	
[2] SP 800-57-1	NIST	March 2007	
[3] SP 800-108	NIST	2009	

Table of Contents

1.0 Overview

1.1 Introduction

OPACITY is a suite of generic and high performance authentication and key agreement protocols for securing transactions using contact or contactless ICCs for physical access, logical access or ticketing applications.

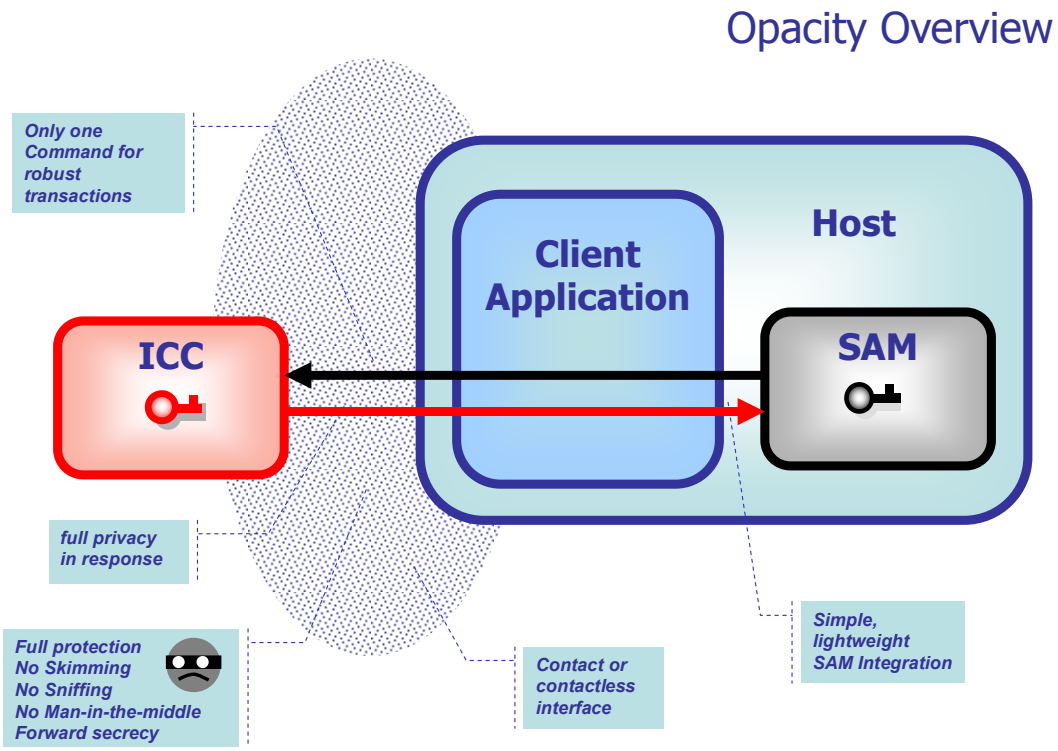


Figure 1. Opacity Overview

The protocol suite is designed with the following requirements:

- Standard cryptography
  - Compliance with NIST cryptographic mandates SP 800-57 part1, NIST SP 800-56A, FIPS 140-2, and ability to fulfill NSA recommendation on the choice of cryptography (SUITE-B)
- Superior security.
  - No identity leaks. Privacy protection is maximum, ie, both Personally Identifiable Information and Unique Identifiers cannot be accessed by unauthorized parties,
  - Forward secrecy: i.e. previously transmitted confidential information cannot be revealed in the clear even if the static host authentication key has been compromised. This feature relies on the ability to use ephemeral elliptic curve keys, and is an advantage over symmetric keys or RSA based system.
  - Mutual authentication. Host authentication to the card is either implicit or explicit. In *implicit* mode, only the host with a trusted private key can decrypt the card output, so the card receives assurance that only an authentic and authorized host can use the card output. In *explicit* mode the host applies secure messaging on card commands, so the card receives assurance that an authentic and authorized host has effectively received the card output.

  - Confidentiality and integrity: The protocol establishes session keys allowing secure messaging for transporting application data or key exchange with confidentiality.
  - Protection against Man-In-The-Middle attacks with proof of possession of static private keys.

- Exceptional performance
  - No overhead - One short APDU command + one short APDU response.
  - Elimination of unecessary CPU consuming artifacts, use of fixed length data elements for optimal processing
  - Robust to tearing / power off in any situation. Capability to resume broken contexts.
  - Extremely fast and scalable session key establishment when the protocol is executed a second time between a card and a host.
- Simple integration
  - The specification covers both SAM and ICC interfaces.
  - From the client application perspective, the integration is the simplest:
    - A single ICC command with a SAM-generated public key and identification data. Direct forwarding of ICC response to host SAM for processing. SAM returns authenticated ICC identification data or ICC privilege information. The session keys are established on both sides.
  - Simple PKI key management and limited infrastructure costs.

1.2 OPACITY Use cases

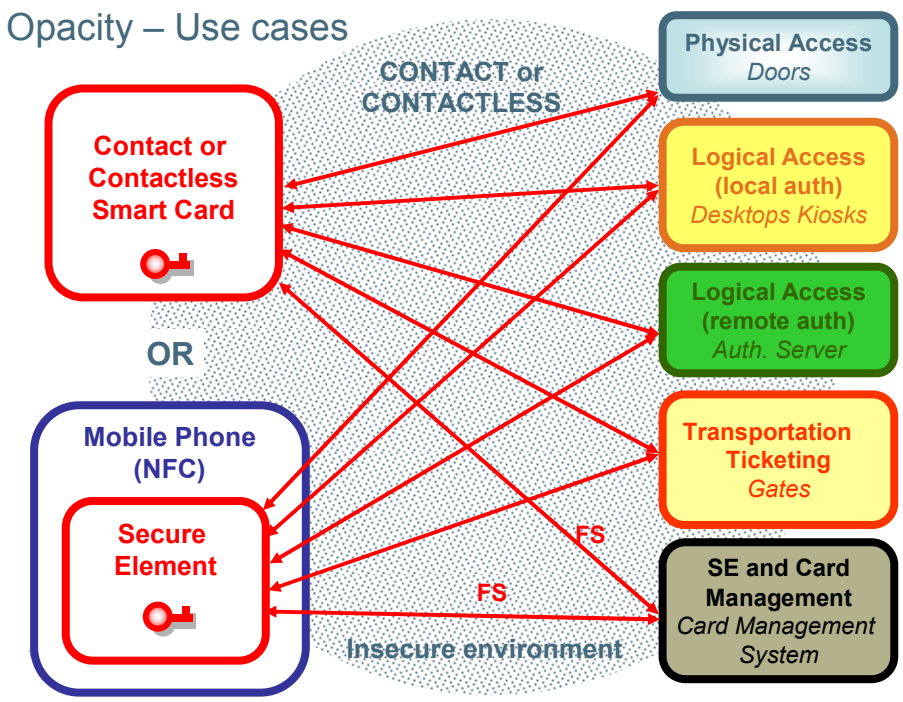


Figure 2. Opacity Use Cases

Opacity is a suite of compact, flexible secure and fast authentication protocols with secure messaging capability. It therefore covers most cases requiring card only authentication or mutual authentication or secure messaging today. It also extends the use offered by current protocols by offering additional protection mechanisms that prevent the risks of usage in contactless or unprotected environments.

SE Management:

- End-to-end post issuance management of smart card application or secure element in contact or contactless environments

SE Applications:

- PKI contactless Authentication to the door or door controller for physical access. Use of end-to-end secure messaging for contactless mutil-factor authentication, to confidentially transport a PIN or biometrics to a Secure Element or PACS credential from a Secure Element via a contactless communication.
- Contact or contactless PKI Authentication to Desktops, Laptops and Kiosks for logical access. Use of secure messaging to provide an end-to-end protected path for document or transaction decryption and signature using a secure element or a smart card.
- PKI-based contactless authentication for ticketing and mass-transit applications.
- PKI-based authentication to access to any remote service across insecure environments (i.e secure access to the cloud). Use of end-to-end secure messaging to confidentially transport a PIN or biometrics or PACS credential via a contact, contactless or other wireless communication.

1.3 Acronyms

- APDU: Application Protocol Data Unit

- CVC: Card Verifiable Credential
- EC: Elliptic Curve
- FS: Full Secrecy mode
- GICS: Generic Identity Command Set
- GUID: Globally Unique IDentifier
- ICC: Integrated Circuit Chip.
- PACS: Physical Access Control System
- SAM: Secure Authentication Module
- SM: Secure messaging

1.4 References

[1] NIST SP 800-56A – March 2007 - Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography

[2] NIST SP 800-57-1 - March 2007 – Recommendation for key management

2.0 Features

The suite includes two modes of different strengths that can be applied to various use cases or eco-systems:

2.1 OPACITY Modes

Two modes are proposed to cover a very large range of use cases requiring secure transactions with secure elements.

1. The Opacity Full Secrecy mode (**OPACITY-FS**) is optimized for contactless authentication transactions between a Secure Element and a remote entity, when mutual authentication is necessary or when it is necessary that the identity of the Secure Element or card holder is never revealed to unauthorized parties. An external third-party without privilege should not be able to associate the identity of the card holder to a transaction made with the card. Under this mode the protocol protects end-to-end sensitive information that needs to be transported to or from the Secure Element, and which remains sensitive and valuable after the transaction or the session is completed. For instance in key management use cases, when the key material that is communicated must remain protected for confidentiality at least during the life time of the key.
2. The Zero Key Management mode (**OPACITY-ZKM**) is a lightweight option best suited to protect contact or contactless transactions when terminals are not capable of protecting static secrets or supporting a hardware security module..the protocol is also best suited to support very fast PKI authentication transactions as it only requires a single Elliptic-Curve Diffie_Hellman operation on the chip.

2.2 OPACITY Common Features

These characteristics apply to all modes of the protocol:

2.2.1 Simplified Key management

OPACITY Key management is simple. Each Opacity enabled ICC or Secure Element includes:

- Zero or one persistent EC private Key and its corresponding Card Verifiable Certificate (CVC).

- A list of CVC verification public EC keys for each domain. These keys are used for verifying the CVC signatures of other opacity-enabled devices willing to communicate.

Since the EC key pairs may be generated on card there is no need to distribute secrets, key ceremonies or involving master keys.

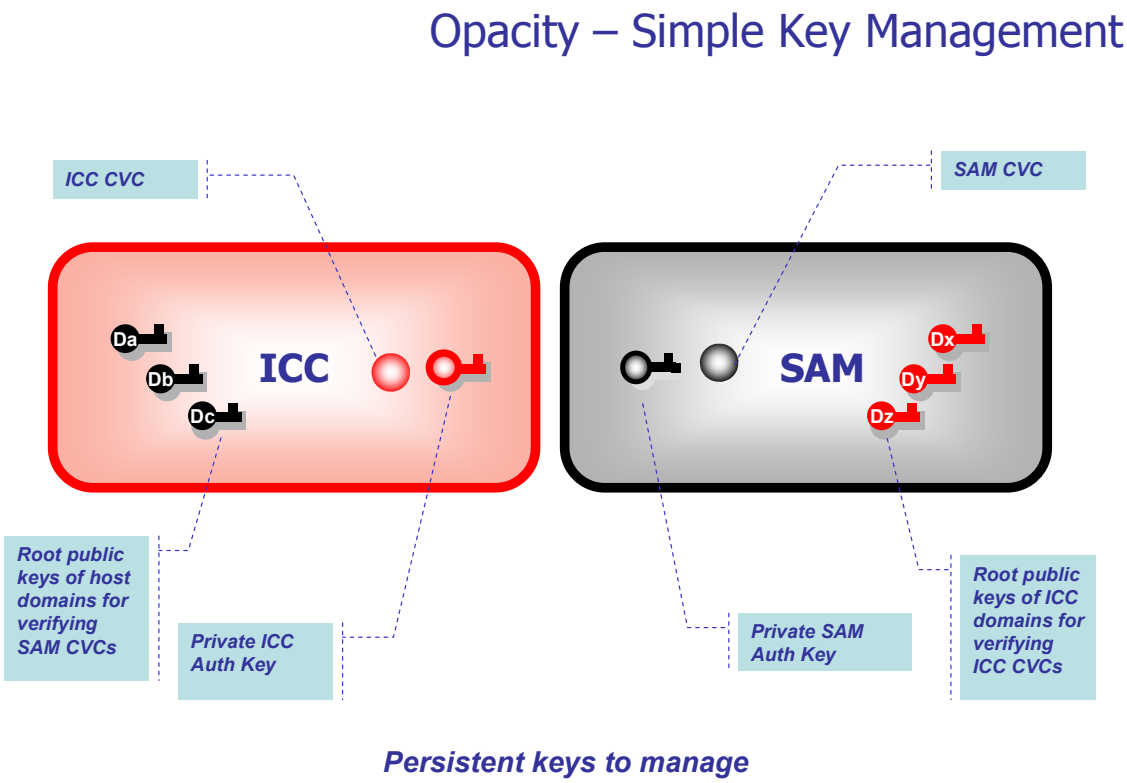


Figure 3 - Opacity Simple Key management

2.2.2 Support Multiple Standard Cipher Suites

The protocol can be configured to support the following cipher suites:

FIPS 140-2 modes	Fast ZKM only	FS/ZKM	Strong Key transport	Strong FS	Government Classified
Cipher Suite	**CS1**	**CS2**	**CS3**	**CS4**	**CS5**
Channel Strength (bits)	**112**	**128**	**192**	**192**	**192**
Encryption or MAC (Session keys)	AES128	AES128	AES 256	AES 192	AES256
ICC CVC Signature	ECDSA 224	ECDSA 256	ECDSA 256	ECDSA 384	ECDSA 384
Host CVC Signature	N/A	ECDSA 256	ECDSA 384	ECDSA 384	ECDSA 384
ICC Key Agreement	ECDH 224	ECDH 256	ECDH 256	ECDH 384	ECDH 384
Host Key Agreement	ECDH 224	ECDH 256	ECDH 384	ECDH 384	ECDH 384
Hashing	SHA 1	SHA 256	SHA 384	SHA 384	SHA 384
Nonces (per 800-56A)	16 bytes	16 bytes	24 bytes	24 bytes	32 bytes

2.2.3 Simple Command flow

All protocols of the Opacity Suite are optimized to be implementable as a single ICC command-response transaction between a client application host and a Secure Element. It is possible to authenticate the unique identitity of the Secure Element for instance a GUID or FASC-N, and establish session keys on both sides in a single command. This challenge response paradigm between the host and the ICC simplifies a lot the integration, reduces command processing and communication overhead, and improves the robustness of the transaction.

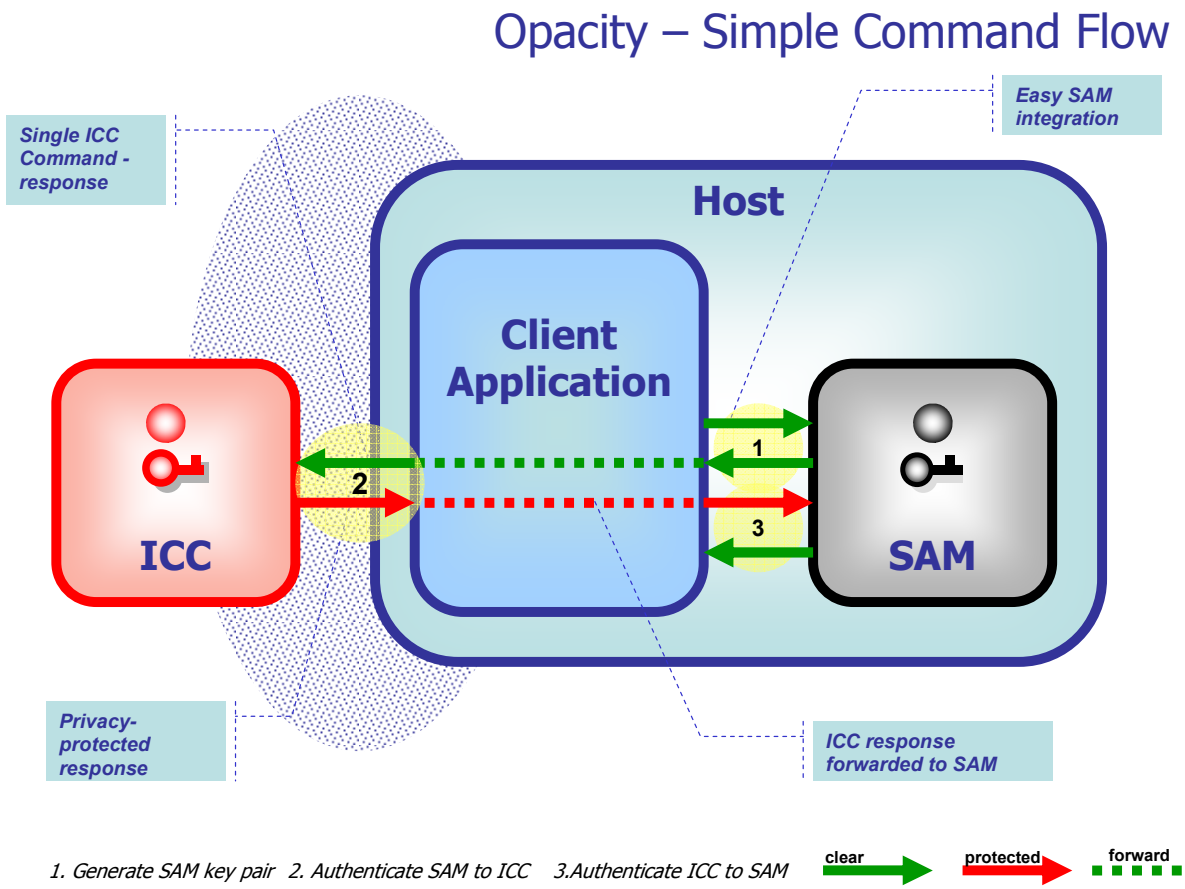


Figure 4 – Simple Command flow

2.2.4 Secure credential transfer

The OPACITY protocol can be used to securely transfer an ID credential from the Secure Element or ICC to the client application, as part of the confidentially protected response to the OPACITY command. The client application receives enough inforamtion to authenticate the credential for further use, such as for physical access. The credential is never exposed at the card edge in the clear, and appears as a random byte field while communicated, it cannot be used to identify the card or its holder.

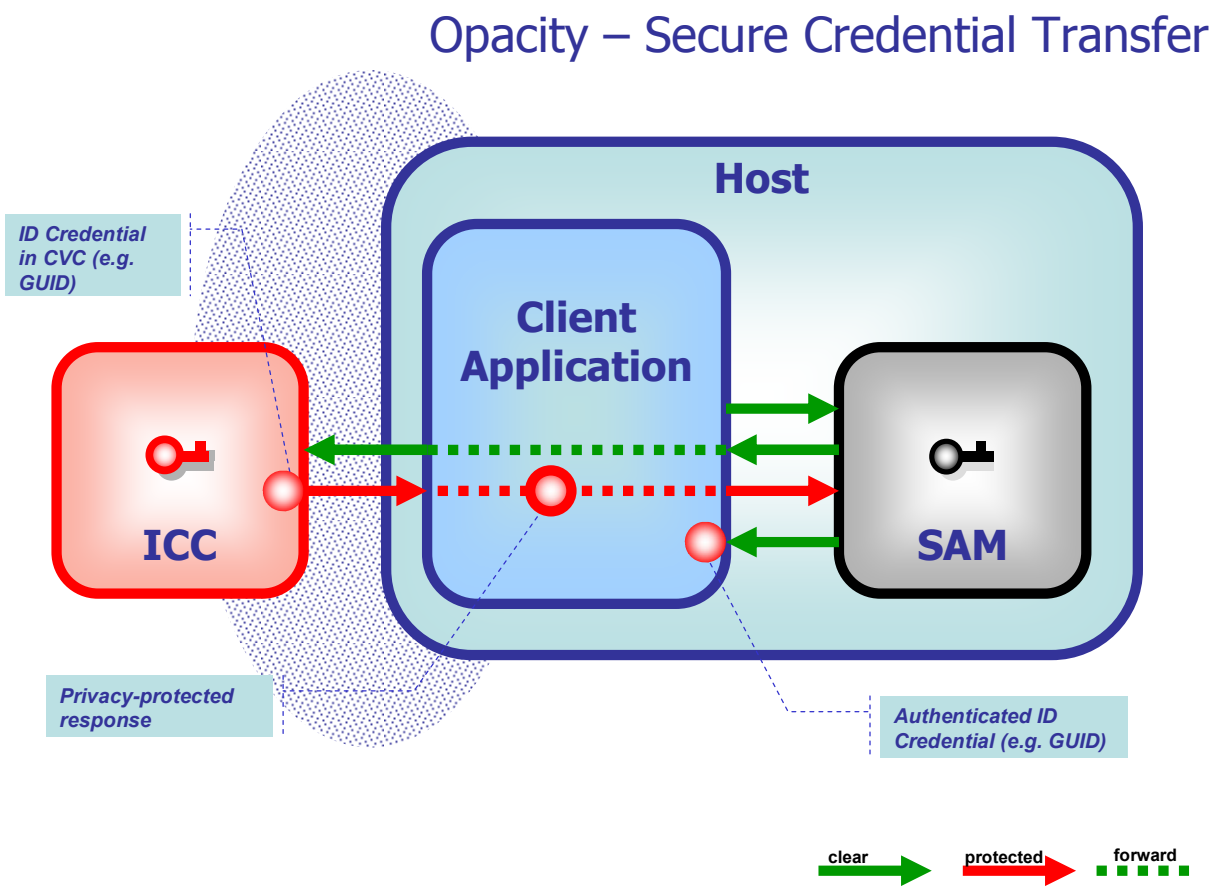


Figure 5. Secure Credential Transfer

2.2.5 Sessions Keys and Secure Messaging

The OPACITY protocol establishes either one or three AES session keys, available on both host and ICC sides. The session keys can be used to protect further commands and responses to and from the ICC using a secure messaging (SM) mechanism such as with ANSI GICS part-1. The session keys are meant to protect the ICC commands for confidentiality, integrity of the command and integrity of the response.

The secure messaging maintains the privacy protection and security obtained during the key establishment.

Interoperable implementations should follow the ISO 7816-4 Annex B secure messaging such as ISO 24727-4 or ANSI GICS-1 Secure Messaging.

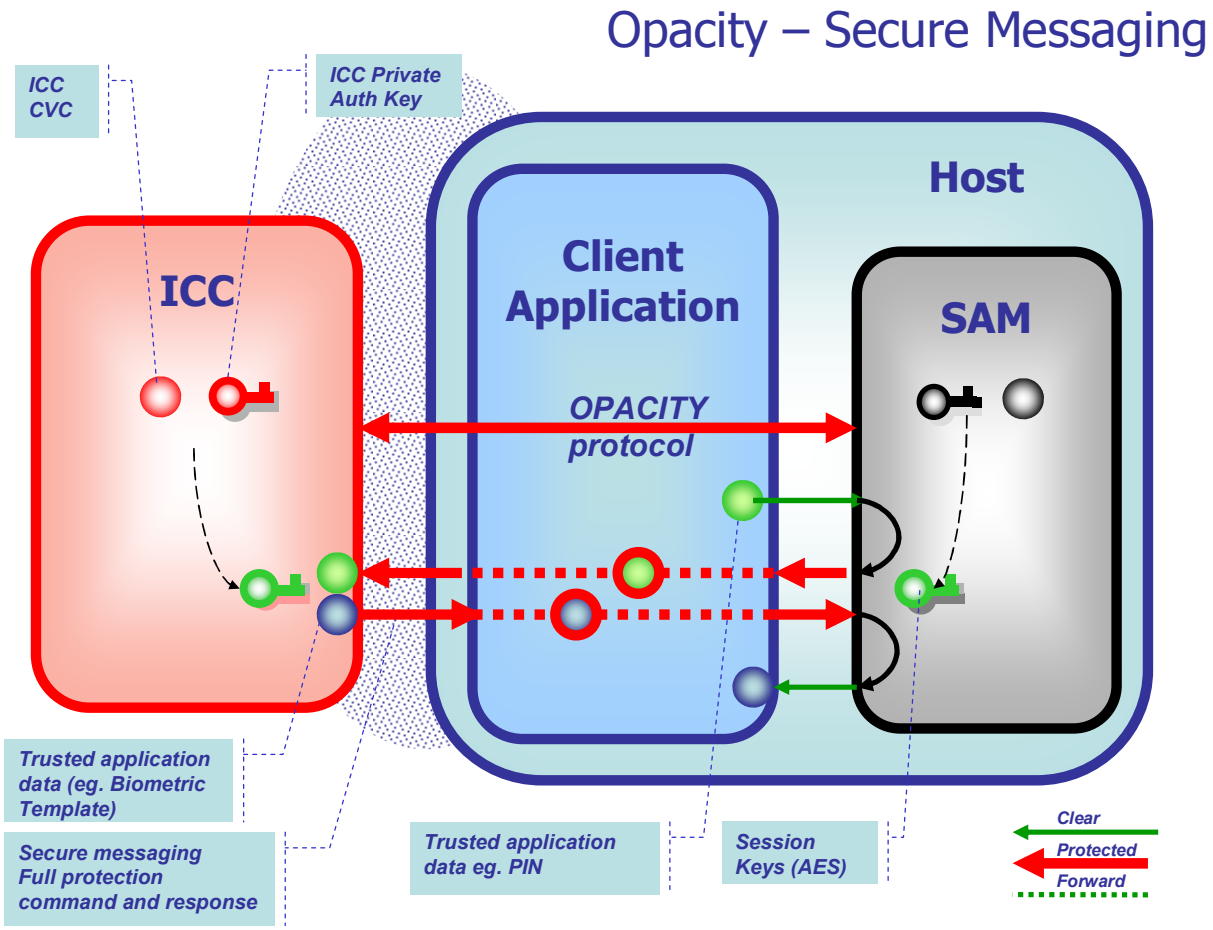


Figure 6 Opacity – Secure messaging

2.2.6 Resume Interruptions

OPACITY is designed to resume interruptions when the card leaves the communications field prematurely and then returns. The terminal issues a new request as if a new card is presented, i.e. with a new nonce and/or ephemeral public key. When the card recognizes that the host CVC is identical to the one of the previous failed connection, the ICC can resume the processing at the point where it failed.

2.2.7 Extreme Performance and Persistent Binding

OPACITY is optimized to rapidly initiate transactions between an ICC and host SAM (or TPM, HSM, ..) that have already exchanged key material. Shared secrets and one-time card identifiers for use in the next transaction are calculated and stored on each side.

This feature is actually a device pairing or "*persistent binding*" between a host SAM and a ICC or Secure Element. It allows both sides to immediately derive new session keys for authentication or secure messaging from previously registered binding data related to that unique ICC-host relationship. This allows to skip all Elliptic Curve steps, and can be considered as a method to establish unique static symmetric keys in secue elements without master keys or key ceremonies..

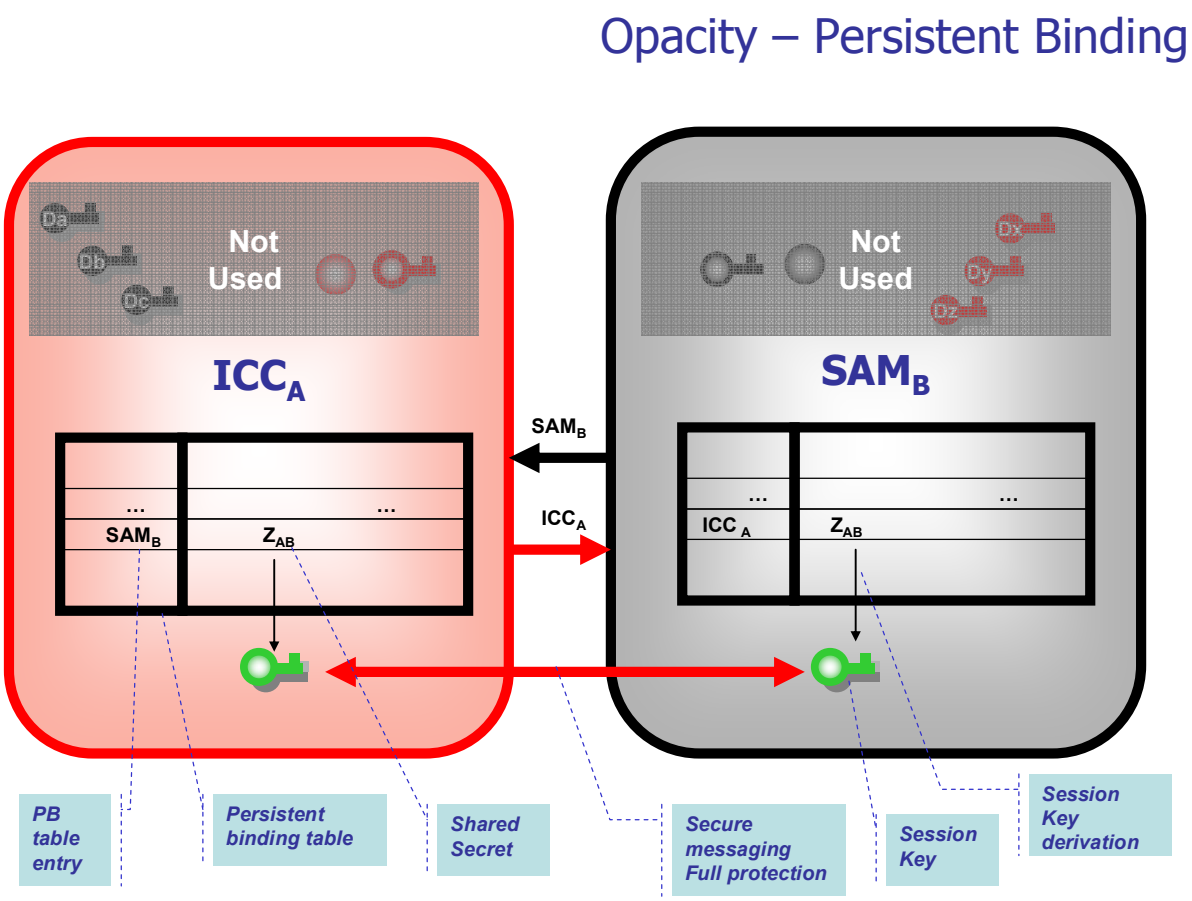


Figure 7 – Persistent Binding

2.2.8 Anti-tearing and Synchronization

De-synchronization of persistent binding records may occur when the ICC or SAM registry tables are unilaterally erased for security or other reasons. Another de-synchronization situation is when the OPACITY response from the ICC never reaches the host although the ICC has newly registered a shared secret. The OPACITY protocol allows the SAM and ICC to detect and recover from de-synchronization.

2.2.9 Simplified SAM-based integration with Terminal

The OPACITY protocol workflow is quite simple from an integrator's perspective (see figure 4). The Client application calls the SAM to generate an ephemeral EC key pair, sends an authentication command to the ICC including the public ephemeral key and the SAM ECC. Then it forwards directly the authentication response of the ICC as a second authentication command to the SAM. If successful, then session keys are established on both sides. The client application builds APDU commands, calls the SAM to wrap them and then sends the wrapped command to the ICC.

2.2.10 Cross-Domain Authentication

To enable the OPACITY protocol, each host is equipped with a SAM protecting a private host authentication key. The host authentication key has a companion credential that is a Card Verifiable Credential described in Annex B. The host shall be capable of transmitting the CVC.

The host CVC is signed with the “root domain private key” which provides some assurance that the binding between the host and the authentication key is approved by the domain authority. A “domain” is a set of hosts and underlying system managed under the same authority. OPACITY enabled cards must include the root domain public key to enforce host CVC verification and host authentication. It is assumed that an OPACITY enabled card may potentially interact with multiple domains, and therefore include multiple root domain public keys.

2.2.11 Key revocation

OPACITY role is to protect transactions and authenticate data transported to and from the ICC in any environment. Its role is not to enforce PKI authorization or revocation on behalf of access control systems.

We assumed that the OPACITY keys are unique authentication keys, and for instance are generated on the ICC. These keys are never shared. Obviously ICCs that have been issued may be stolen or operated in unauthorized contexts.

This proposal recommends that:

1) The life cycle of the Opacity authentication keys is managed via the ICC post-issuance system in coordination with the life of the card. In particular, the ICC management system shall be able of regenerating the OPACITY static key pair, installing a corresponding CVC, and updating the list of authorized root domain public key.
2) In case of theft, the security of the OPACITY-based system relies on the prompt declaration of loss or theft to the Issuer, service provider or local IT. This results in a) the actual revocation of the application credentials and associated rights (CHUID, PIV Auth Cert, etc.), which are not the OPACITY credentials, and b) results in updates the corresponding access control backend.
3) The revocation of Opacity CVCs is not expected in the current version of the specification. Note that OPACITY PKI role is to protect and authenticate the transported information rather than provide authorization or revocation information.
4) It is also recommended that the list of valid CVC validation public keys is regularly updated to phase out revoked cards.

2.2.12Scalability

When a large number of ICCs needs to be registered in a SAM Persistent binding table, the time needed to find the entry referencing the shared secret with the correct ICC index may be significant. The OPACITY specification allows the client application to quickly determine or discover the PB table entry indexes, so the SAM can immediately access the PB table entry contents.

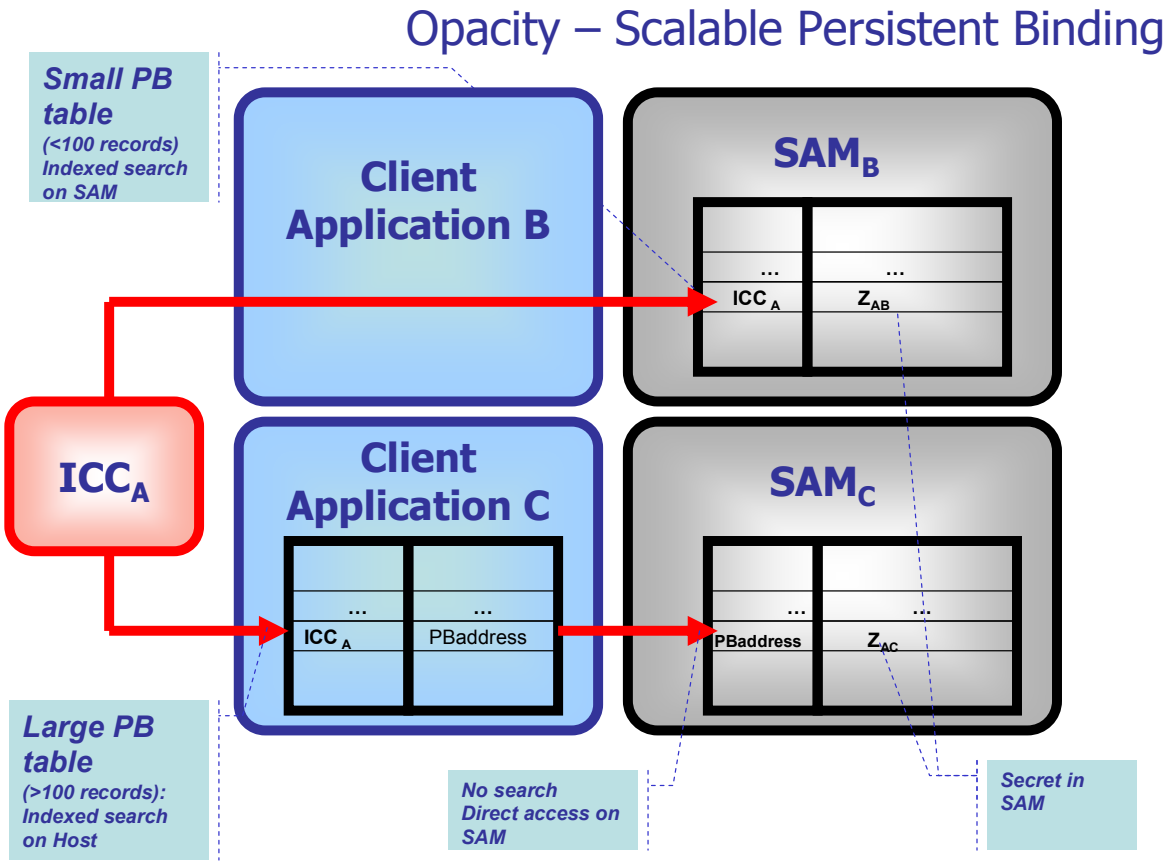


Figure 8. Opacity Scalable Persistent Binding

2.2.13 Anti-theft : SAM Activation and Deactivation

SAMs can only be active in an approved context of use. An attacker controlling a SAM may be able to capture sensitive identity information.

The proposed design includes a flexible and enforceable SAM activation and SAM deactivation mechanism.

The SAM activation mechanism consists of a successful OPACITY authentication with a trusted “administrator ICC” which CVC is signed with a domain administrator root private key. The corresponding domain administrator root public key is stored on the SAM.

An Administrator ICC can be either a personal card or secure element of a mobile phone which communicates to the SAM via the client application as a regular OPACITY enabled card.

An Administrator ICC can also be a secure element attached to an online connected service, allowing real-time control on SAMs.

A SAM which has been reset or powered on is in deactivated state. A deactivated SAM requires a successful OPACITY transaction with an administrator ICC prior to allow any other OPACITY transaction with other ICC types.

To deactivate a SAM, it must either be:

- powered off then on
- reset
- SAM application is deselected
- Internal OPACITY transaction counter reaches a maximum. The counter is reset upon activation.

2.3 OPACITY Full Secrecy Mode (OPACITY-FS)

2.3.1 Privacy

Not identifiable - The Opacity protocol does not divulge any identifier associated to a particular ICC or card holder during an Opacity session.

Not traceable - the Opacity protocol does not divulge any data that allows the correlation of two protocol executions with same ICC during an Opacity session.

2.3.2 Authentication

Opacity explicitly authenticates the ICC to the host using FIPS-approved EC-based authentication protocols described in NIST SP 800-56A. The ICC delivers a Card Verifiable Credential to the host, which allows the host to verify the binding between the ICC identifier and the ICC EC private key. The ICC identifier can be chosen to be the same value registered in the access control system (for instance, GUID or FASC-N).

Opacity may either implicitly or explicitly authenticate the host to the ICC, using FIPS-approved EC-based authentication protocols described in NIST SP 800-56A.

- Implicit authentication means that the ICC validates the CVC but does not expect a response from the host showing that the host actually owns the private key corresponding to that CVC. However, only an authentic host will be able to decipher the ICC response.

- Explicit authentication means that the ICC actually receives at least an additional command from the host that proves to the ICC that the host owning the private key is present...

2.3.3 Forward Secrecy

Forward secrecy provides assurance that the encrypted data that is transferred cannot be decrypted after the communication has ended and the session keys and the ephemeral EC key pairs are zeroized. This mode is particularly useful when Opacity is used for SM encryption. Specifically, with forward secrecy, the SM session keys cannot be reproduced and the transported data cannot be decrypted even if both the original persistent key material and the original communication data have been compromised or captured from the Host.

Note that the protocol does not implement “Perfect Forward Secrecy”, ie. if both unique key pairs on the ICC and SAM are known, the session keys can be reproduced.

2.4 OPACITY - Zero Key Management Mode (OPACITY-ZKM)

Opacity has an operating mode that extremely reduces the key management requirement for terminals with no Secure Access Modules available locally or remotely, as is the case with legacy Physical Access Control infrastructure.

OPACITY ZKM – Fast Contactless Transactions

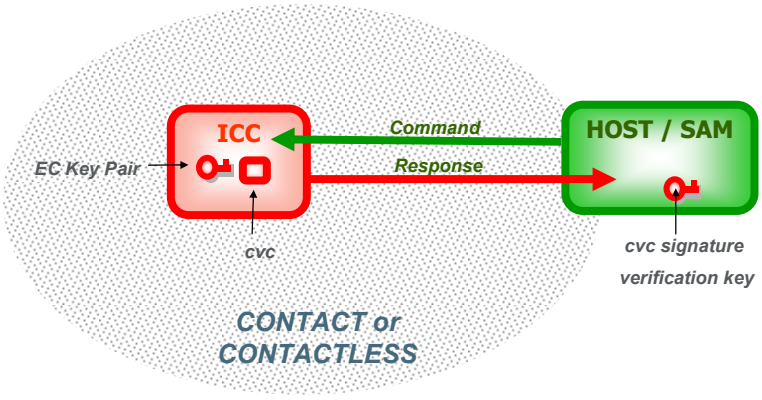


Figure 10 – Opacity-ZKM managed Keys

This mode does not require static terminal keys except for the root Public Key to verify the Card Verifiable Certificate signature. It provides Card Authentication but not Terminal Authentication, and is to be used in environments where terminals are known and trusted.

In this mode, the protection against identity leaks is achieved by encrypting the identification data in the response.

The basic ZKM authentication protocol is an ICC internal authentication protocol and key agreement using EC cryptography. The initiator generates an ephemeral key pair but has no static key pair; the responder has only a static key pair

3.0 Common Specifications

This section is applicable to OPACITY FS and ZKM modes.

3.1 Glossary

Name	Comment	Format	Max Size (in bytes)
$OTID_{ICC}$	ICC anonymous identifier, valid one time	Binary	8 bytes
ID_{sICC}	Static, non anonymous ICC identifier. Truncated Hash of C_{ICC}	Binary	8 bytes
GUID	Unique Cardholder or device identifier.	Binary	16 bytes
C_{ICC}	Card Verifiable Credential authenticating Q_{sICC}	CVC - see Annex B	See Annex B
C_{ICC}*	Confidential Card Verifiable Credential for privacy, derived from C_{ICC} as follows: - The GUID TLV data element of C_{ICC} has been replaced with TL data element (T=0x5f20 [GUID Tag], L=0) All other data elements and their order are identical to those in C_{ICC}	CVC – see Annex B	See Annex B
$Q_{rootICC}$	Root public Key to verify the ECDSA signature of ICC authentication credential C_{ICC}		P-224: 57 P-256: 65 P-384: 97
Q_{sICC} $[d_{sICC}]$,	ICC authentication public key, matched with the corresponding private key: d_{sICC}		P-224: 57 P-256: 65 P-384: 97
, Q_{eICC} $[d_{eICC}]$	ICC ephemeral public key, matched with the corresponding ephemeral private key: d_{eICC}		P-224: 57 P-256: 65 P-384: 97

Name	Comment	Format	Max Size (in bytes)
ID_{sH}	Terminal or host identifier. Subject nameof C_H	Binary	8 bytes
C_H	Host Card Verifiable Credential (CVC) binding Q_{sH} and ID_{sH}.	CVC	See Annex B
Q_{rootH}	root public key for verifying host CVCs on the ICC	Binary	P-224: 57 bytes P-256: 65 bytes P-384: 97 bytes
Q_{eH} , $[d_{eH}]$,	Host ephemeral public key, matched with the corresponding ephemeral private key: d_{eH}		P-224: 57 bytes P-256: 65 bytes P-384: 97 bytes
Q_{sH} $[d_{sH}]$,	static host ECC public key, matched with the corresponding host static private key: d_{sH}	Binary	P-224: 57 bytes P-256: 65 bytes P-384: 97 bytes
N_H	Client Application nonce	Binary	16 bytes per 800-56A (Section 5.4)
N_{ICC}	ICC nonce	Binary	16 bytes per 800-56A (Section 5.4)
SK_{MAC} , SK_{ENC}, SK_{CFRM}	Secure Messaging Session Keys		According to algorithm: AES 128: 16 bytes AES 192: 24 bytes AES 256: 32 bytes
$T_8(Data)$	Leftmost 8-byte Truncation Method	Binary	Extract the leftmost 8 bytes of the argument “Data”.
$T_{16}(Data)$	Leftmost 16-byte Truncation Method	Binary	Extract the leftmost 16 bytes of the argument “Data”.

Name	Comment	Format	Max Size (in bytes)
ENCRYPT(Algo, Key, Data)	Encrypt Data with Key using Algorithm 'Algo'.	N/A	N/A
KDF (Z, len, info)	Key Derivation Function Specified in Annex A	N/A	N/A
EC_DH	ECC based key agreement functions, specified in 800-56A	N/A	N/A
K1, K2	Intermediate Shared secrets (AES 128)	Binary	16 bytes
Info	According to 800-56A specifications. See Annex A – KDF Specification	N/A	N/A
len	According to 800-56A specifications. See Annex A – KDF Specification Value: (2+x)*y Where: x=1 or 3 (nb authentication keys) y: length of chosen AES session keys in bytes	Binary	1 byte
KDFHashAlgorithm	SHA-256 or SHA-224 or SHA-384		
C-MAC	NIST 800-38B AES-128 based MAC algorithm		
CB_{ICC}	Protocol Control Byte returned by ICC. See section 0 - PB: Host was found in ICC PB registry - NO_PB: Persistent binding has not been used for this transaction - PB_INIT: ICC has set a new Persistent binding entry for the host (re-computed ECDH) - ENC_GUID: the encrypted GUID is present in the response	Binary	1 byte

Name	Comment	Format	Max Size (in bytes)
	- CLR_GUID. The GUID is returned as part of the CVC - ONE_SK: ICC has established only one session key for Secure messaging, - THREE_SK: ICC has established Three session keys for Secure Messaging.		
CB_H	Protocol Control Byte sent by host: - PB: ICC may use Persistent binding if supported - NO_PB: No persistent binding supported by this host: ICC may not use persistent binding - PB_INIT: ICC shall reset Persistent binding entry for the host (re-compute ECDH) - RET_GUID: ICC includes the encrypted GUID in the response - ONE_SK: Request ICC to establish only one session key THREE_SK: ICC has established Three session keys for Secure Messaging	Binary	1 byte
PBaddress	This value can be set by the client application. When non-null, this value indicates where the Persistent Table entry record [OTID, Z] or [IDsicc, Z] need to be located on the SAM, thus avoiding any search on the SAM.	Integer	4bytes.

3.2 Compliance with 800-56A

It is expected that implementations of OPACITY comply with 800-56A:

3.2.1 Key Agreement

- Opacity FS follows a sequence of two C(1,1) steps.
- Opacity ZKM corresponds to one C(1,1) steps
- All key establishment prerequisites must be executed on both host and ICC side prior to executing the protocol. Each party shall have an authentic copy of the same set of domain parameters, *D*. And obtain assurance of the validity of these parameters.
- For OPACITY-FS:
    - the prerequisites for the C(1,1) ECC CDH mode ([1], section 6.2.2 – Prerequisites) must be applied.
    - the requirements of 6.2.2.2 Full Unified Model, C(1,1, ECC CDH) must be applied.
- For OPACITY-ZKM
    - the prerequisites for the C(1,1) ECC CDH mode ([1], section 6.2.2 – Prerequisites) must be applied.
    - the requirements of 6.2.2.2 One Pass Diffie-Hellman, C(1,1, ECC CDH) must be applied.

3.2.2 Key Derivation

- The Key Derivation Function is the concatenation KDF as specified in NIST SP800-56A (§ 5.8.1).

3.2.3 Key Confirmation

- The Key Confirmation Follows section 8.4.9 “C(0, 2) Scheme with Unilateral Key Confirmation Provided by V to U” in NIST SP 800-56A

3.3 Host Initial State

Opacity needs to be enabled on the host side as follows:

Setup a client application executing on the host $\boldsymbol{ID_{sH}}$, equipped with a HSM, SAM or TPM to implement the cryptographic functions, and to communicate with a smart card or secure element $\boldsymbol{ID_{sICC}}$.

The client application has access to a unique static private EC authentication key $\boldsymbol{d_{sH}}$. The corresponding Card Verifiable Credential $\boldsymbol{C_H}$ includes $\boldsymbol{ID_{sH}}$ and the matching public key $\boldsymbol{Q_{sH}}$

According to 800-56A Figure 1 [1], the host must execute all key establishment preparations prior to launching the protocol.

The client application has access to a registry of Host-ICC pairing records needed to support the persistent binding capability. Each record includes:

- $\boldsymbol{OTID_{ICC,}}$ the ICC record identifier.
- **Z**, a one-time shared secret, valid for the next communication.

The registry is accessed with $\boldsymbol{OTID_{ICC}}$

If the ICC authentication settings are variable, the client application ensures that the ICC authentication method and parameters are set:

The client application has access to the root ICC public key ($\boldsymbol{Q_{rootICC}}$) to validate ICC authentication credentials, $\boldsymbol{C_{ICC}}$ and card authentication public key ($\boldsymbol{Q_{sICC}}$)

3.4 ICC Initial State

Opacity needs to be enabled on the ICC side as follows:

- $\boldsymbol{Q_{rootH}}$, CVC root public key for on card verification of the client CVC. $\boldsymbol{Q_{rootH}}$ may be determined with the client CVC Issuer Identification Number

The ICC has access to a registry of host pairing records needed for the persistent binding capability. Each record includes:

- $\boldsymbol{ID_{sH}}$ the host identifier and index.
- $\boldsymbol{OTID_{ICC,}}$ the one-time ICC identifier that was used during the last session.
- **Z**, a one-time shared secret, only valid for the next communication.

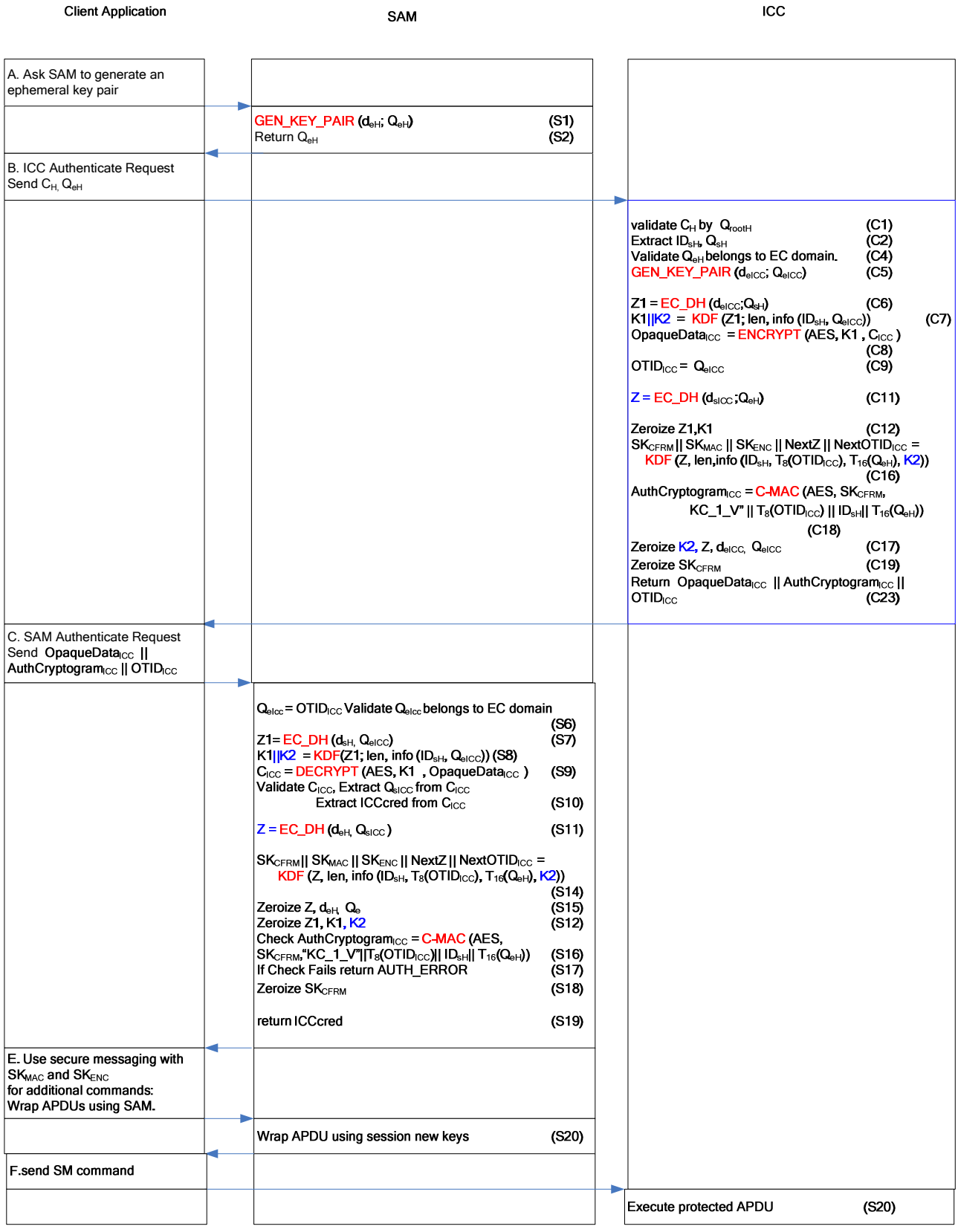
The card application has access to a card authentication key ($\boldsymbol{d_{sICC}}$) and a Card CVC ($\boldsymbol{C_{ICC}}$) including $\boldsymbol{Q_{sICC}}$.

According to 800-56A Figure 1 [1], the ICC must execute all key establishment preparations prior to launching the protocol.

4.0 Base Protocol Specifications

This section describes the protocol for educating purpose. All optimizations are defined in section 5, Optimized Protocol Specifications.

4.1 OPACITY with Full Secrecy



4.1.1 Client Application Protocol Steps

Step #	Description	Comment
A.	Request to SAM to generate an ephemeral EC key pair	This is the initial step of the protocol. It always occurs.
B.	Authentication Request to ICC Send $C_{H,}$ Q_{eH}	The client application challenges the ICC to authenticate.
C.	Authentication Request to SAM Send $OpaqueData_{ICC}$ || $AuthCryptogram_{ICC}$ || $OTID_{ICC}$	The client application forwards the ICC response to the SAM to authenticate the ICC. Only an authentic SAM can decipher the ICC response.
E.	Use secure messaging with SK_{MAC} and SK_{ENC} for additional commands	The Client application coordinates the exchange of session key - protected commands between the SAM and the ICC. The SAM is used to wrap commands.

4.1.2SAM Protocol Steps

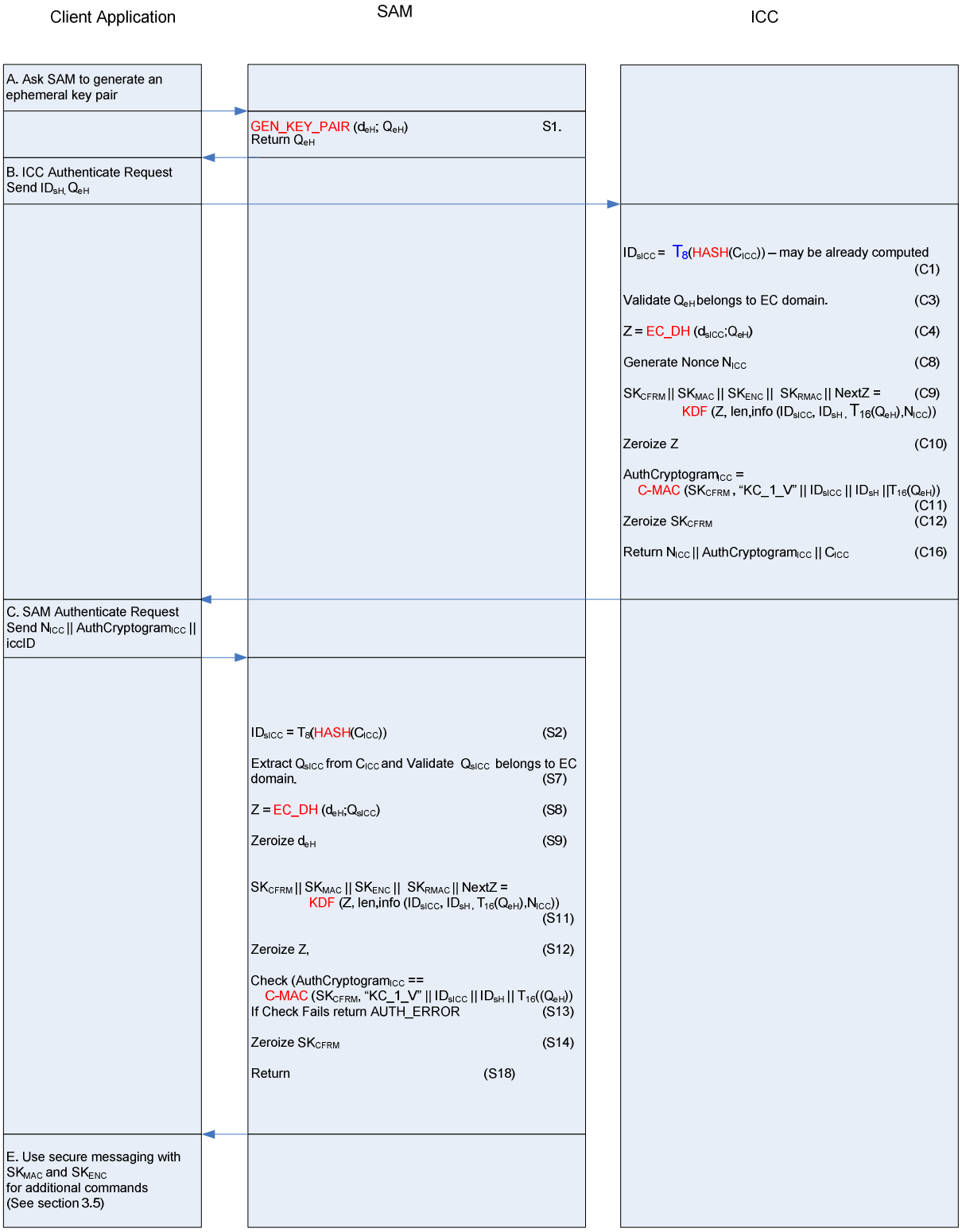
Step #	Description	Comment
S1	GEN_KEY_PAIR (d_{eH}; Q_{eH}) Return Q_{eH}	
S6	Q_{eIcc} = $OTID_{ICC}$ Validate Q_{eIcc} belongs to EC domain	then $OTID_{ICC}$ is Q_{eIcc} verify that Q_{eICC} corresponds to the elliptical domain parameters
S7	Z1= EC_DH ($d_{sH,}$ Q_{eICC})	Compute the ECDH shared secret Z1
S8	K1||K2 = KDF(Z1; len, info (ID_{sH}, Q_{eICC}))	Generate K1
S9	C_{ICC} = DECRYPT (AES, K1 , $OpaqueData_{ICC}$)	
S10	Validate C_{ICC}, Extract Q_{sICC} and ICCcred from C_{ICC}	
S11	Z = EC_DH ($d_{eH,}$ Q_{sICC})	Concatenation per 800-56A C(2,2)
S12	Zeroize Z1, K1	
S14	SK_{CFRM} || SK_{MAC} || SK_{ENC} || NextZ || $NextOTID_{ICC}$ = KDF (Z, len, info (ID_{sH}, $T_8(OTID_{ICC})$, $T_{16}(Q_{eH})$, K2))	Compute the session keys, as well as the shared secret for the next PB session
S15	Zeroize Z, K2, $d_{eH,}$ $Q_{eH,}$	
S16	Check $AuthCryptogram_{ICC}$ = C-MAC (AES, SK_{CFRM},"KC_1_V"||$T_8(OTID_{ICC})$|| ID_{sH}|| $T_{16}(Q_{eH})$)	Authenticate ICC by verifying it also possesses the session key
S17	If Check Fails return AUTH_ERROR	
S18	Zeroize SK_{CFRM}	Per 800-56A Section 5.8
S19	Return AUTH_OK	The authentication status is returned to the

Step #	Description	Comment
		client application

4.1.3 ICC Protocol Steps

Step #	Description	Comment
C1	Validate C_H by Q_{rootH}	Using (Q_{rootH}), The card application verifies the Client CVC (C_H), Also verify the EC curve parameters per 800-56A
C2	Extract ID_{sH}, Q_{sH}	assess the binding between the unique Host ID (ID_{sH}) and the CVC public key (Q_{sH}).
C3	Look for ID_{sH} in PB registry	The ICC scans for ID_{sH} in the registry.
C4	Validate Q_{eH} belongs to EC domain.	verify the EC curve parameters per 800-56A
C5	GEN_KEY_PAIR (d_{eICC}; Q_{eICC})	The ICC generates ephemeral ECC key pair (d_{eICC};Q_{eICC})
C6	Z1 = EC_DH (d_{eICC};Q_{sH})	Generate the shared secret
C7	K1 || K2 = KDF (Z1; len, info (ID_{sH}, Q_{eICC}))	
C8	$OpaqueData_{ICC}$ = ENCRYPT (AES, K1 , C_{ICC})	
C9	$OTID_{ICC}$ = Q_{eICC}	
C11	Z = EC_DH (d_{sICC} ;Q_{eH})	Concatenation per 800-56A C(2,2)
C12	Zeroize Z1,K1	
C16	SK_{CFRM} || SK_{MAC} || SK_{ENC} || NextZ || $NextOTID_{ICC}$ = KDF (Z, len,info (ID_{sH}, $T_8(OTID_{ICC})$, $T_{16}(Q_{eH})$, K2))	Compute the new session keys, the shared secret and the one-time identifier for the next session
C17	Zeroize Z, K2, $d_{eICC,}$ $Q_{eICC,}$	
C18	$AuthCryptogram_{ICC}$ = C-MAC (AES, SK_{CFRM}, KC_1_V" || $T_8(OTID_{ICC})$ || ID_{sH}|| $T_{16}(Q_{eH})$)	Compute return cryptogram allowing ICC authentication, showing it owns the session key to the client application
C19	Zeroize SK_{CFRM}	Per 800-56A Section 5.8
C23	Return $OpaqueData_{ICC}$ || $AuthCryptogram_{ICC}$ || $OTID_{ICC}$	

4.2 OPACITY with Zero Key Management (ZKM)



.

4.2.1 Client Application Protocol Steps

Step #	Description	Comment
A.	Ask SAM to generate an ephemeral key pair	This is the initial step of the protocol. It always occurs.
B.	ICC Authenticate Request Send ID_{sH}, Q_{eH},	The client application challenges the ICC to authenticate.
C.	SAM Authenticate Request Send N_{ICC} || $AuthCryptogram_{ICC}$ || iccID	The client application forwards the ICC response to the SAM to authenticate the ICC. Only an authentic SAM can decipher the ICC response.
OPACITY-ZKM complete.		
E.	Use secure messaging with SK_{MAC} and SK_{ENC} for additional commands	The Client application coordinates the exchange of session key - protected commands between the SAM and the ICC. The SAM is used to wrap commands.

4.2.2SAM Protocol Steps

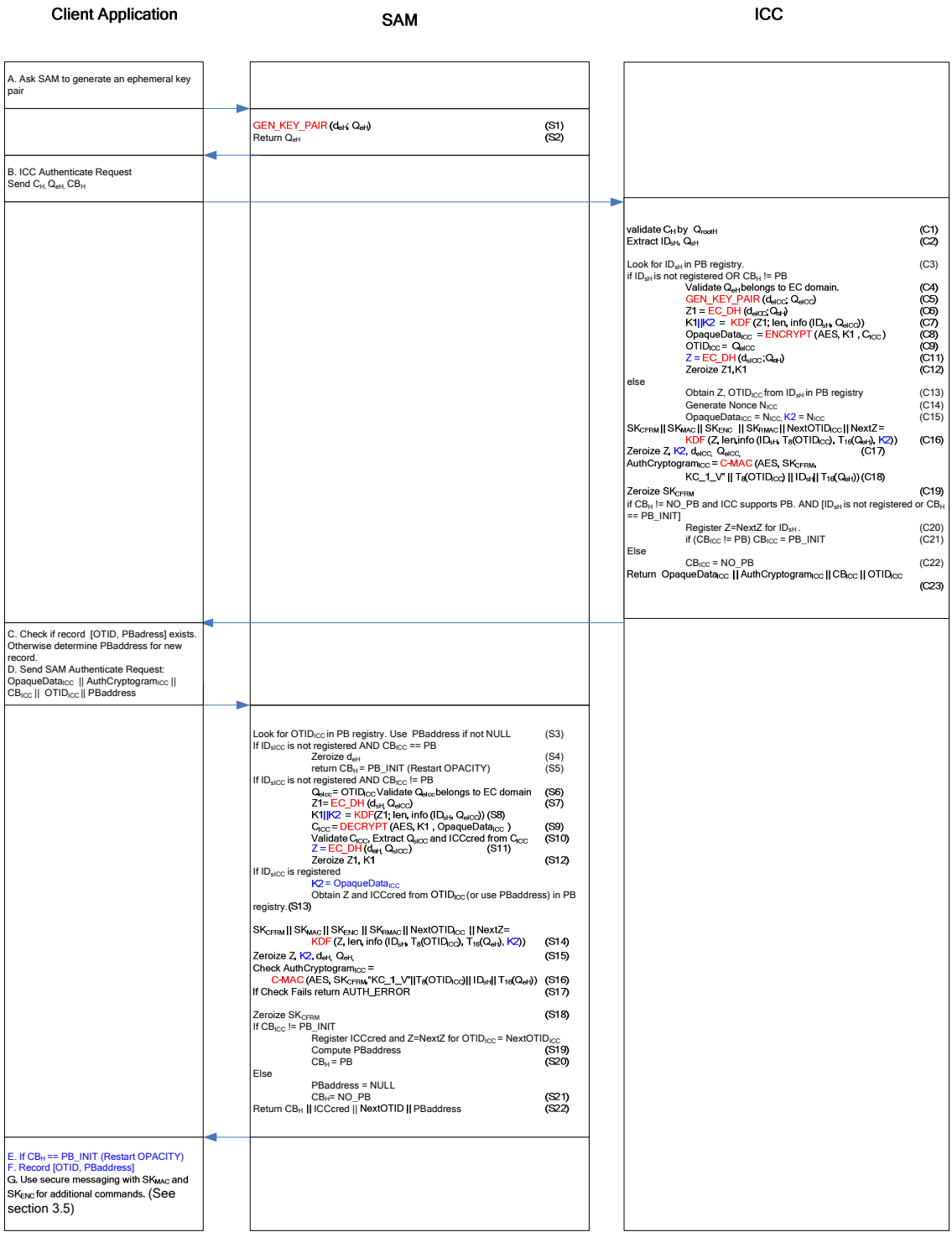
Step #	Description	Comment
S1	GEN_KEY_PAIR (d_{eH}; Q_{eH}) Return Q_{eH}	The SAM generates an EC Key Pair.
S2.	$ID_{sICC} = T_8(HASH(C_{ICC}))$	The ICC returns its CVC . The actual ICC persistent binding table entry to use is $T_8(HASH(C_{ICC}))$
S7.	Extract Q_{sICC} from C_{ICC} and Validate Q_{sICC} belongs to EC domain.	Extract the static ICC public key from the CVC
S8.	$Z = EC\_DH(d_{eH}; Q_{sICC})$	Compute the shared secret.
S9.	Zeroize d_{eH}	Destroy ephemeral host private key.
S11.	SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH}, $T_{16}(Q_{eH})$,N_{ICC}))	Compute the session keys and Next Z. NextZ length is 16bytes,
S12.	Zeroize Z,	Destroy Z.
S13.	Check ($AuthCryptogram_{ICC}$ == C-MAC (SK_{CFRM}, "KC_1_V" || ID_{sICC} || ID_{sH} || $T_{16}((Q_{eH}))$ If Check Fails return AUTH_ERROR	Check key confirmation cryptogram. Return authentication error if verification fails.
S14.	Zeroize SK_{CFRM}	Destroy session key used for key confirmation
S18	Return AUTH_OK	ICC Authentication succeeded..

4.2.3ICC Protocol Steps

Step #	Description	Comment
C1	$ID_{sICC} = T_8(HASH(C_{ICC}))$	Prepare ICC ID value– may be pre-computed
C3.	Validate Q_{eH} belongs to EC domain.	Check ephemeral public key validity
C4.	$Z = EC\_DH(d_{sICC}; Q_{eH})$	Compute shared secret
C8.	Generate Nonce N_{ICC}	
C9.	SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH}, $T_{16}(Q_{eH})$,N_{ICC}))	Compute session keys and shared secret for next session.
C10.	Zeroize Z	Destroy current shared secret.
C11.	$AuthCryptogram_{ICC}$ = C-MAC (SK_{CFRM}, "KC_1_V" || ID_{sICC} || ID_{sH} ||$T_{16}(Q_{eH})$)	Compute key confirmation cryptogram per 800-56A.
C12.	Zeroize SK_{CFRM}	Destroy session key sued to compute the KC cryptogram
C16.	Return N_{ICC} || $AuthCryptogram_{ICC}$ || CB_{ICC} || iccID	Return ICC response

5.0 Optimized Protocol Specifications

5.1 OPACITY with Full Secrecy (Optimized)



5.1.1 Client Application Protocol Steps

Step #	Description	Comment
A.	Ask SAM to generate an ephemeral key pair	This is the initial step of the protocol. It always occurs.
B.	Authenticate Request to ICC Send $C_{H,}$ $Q_{eH,}$ CB_H	The client application challenges the ICC to authenticate.
C	Determine PB offset based on OTID.	Search locally among [PBaddress, OTID] records. If OTID is not found, determine best PBaddress value (first free record slot). Feature can be disabled with PBaddress = NULL.
D	D. Send Authenticate Request to SAM $OpaqueData_{ICC}$ || $AuthCryptogram_{ICC}$ || CB_{ICC} || $OTID_{ICC}$ || PBaddress	The client application forwards the ICC response to the SAM to authenticate the ICC. Only an authentic SAM can decipher the ICC response.
E.	If CB_H == PB_INIT OR CB_H == PB and AUTH_ERROR (Restart OPACITY)	
F	Record [OTID, PBaddress] using the ICC output for faster search	The next time the same ICC is presented to the client application, the client application finds its[OTID, PBaddress] stored locally. PBaddress is then communicated to SAM which does not need to search OTID in the PB table.
G.	Use secure messaging with SK_{MAC} and SK_{ENC} for additional commands	The Client application coordinates the exchange of session key - protected commands between the SAM and the ICC. The SAM is used to wrap commands.

5.1.2SAM Protocol Steps

Step #	Description	Comment
S1	GEN_KEY_PAIR (d_{eH}; Q_{eH}) Return Q_{eH}	
S3	Look for $OTID_{ICC}$ in PB registry. Use PBaddress if not NULL	The client application scans for $OTID_{ICC}$ in the registry in PBaddress is NULL. Otherwise the PB entry van be found at PBaddress * sizeof (PB entry)
If ID_{sICC} is not registered AND CB_{ICC} == PB		If $OTID_{ICC}$ is not found,
S4	Zeroize d_{eH}	

Step #	Description	Comment
S5	return CB_H = PB_INIT (Restart OPACITY) (S5)	
If ID_{sICC} is not registered AND CB_{ICC} != PB		
S6	Q_{eIcc} = $OTID_{ICC}$ Validate Q_{eIcc} belongs to EC domain	then $OTID_{ICC}$ is Q_{eIcc} verify that Q_{eICC} corresponds to the elliptical domain parameters
S7	Z1= EC_DH ($d_{sH,}$ Q_{eICC})	Compute the ECDH shared secret Z1
S8	K1 || K2 = KDF(Z1; len, info (ID_{sH}, $T_{16}(Q_{eICC})$))	Generate K1
S9	C_{ICC} = DECRYPT (AES, K1 , $OpaqueData_{ICC}$)	
S10	Validate C_{ICC}, Extract Q_{sICC} from C_{ICC}	
S11	Z = EC_DH ($d_{eH,}$ Q_{sICC})	
S12	Zeroize Z1, K1	
If ID_{sICC} is registered		$OTID_{ICC}$ has been found in the registry
	K2 = $OpaqueData_{ICC}$	
S13	Obtain Z from $OTID_{ICC}$ in PB registry.	Z can be found using $OTID_{ICC}$ as Index.
Continue		
S14	SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || $NextOTID_{ICC}$ || NextZ = KDF (Z, len, info (ID_{sH}, $T_8(OTID_{ICC})$, $T_{16}(Q_{eH})$, K2))	Compute the session keys, as well as the shared secret for the next PB session Note that NextZ is only used when the condition (S19): CB_{ICC} == PB_INIT is TRUE. Next Z does not need to be computed otherwise.
S15	Zeroize K2, Z, $d_{eH,}$ $Q_{eH,}$	
S16	Check $AuthCryptogram_{ICC}$ = C-MAC (AES, SK_{CFRM},"KC_1_V"||$T_8(OTID_{ICC})$|| ID_{sH}|| $T_{16}(Q_{eH})$)	Authenticate ICC by verifying it also possesses the session key
S17	If Check Fails return AUTH_ERROR	
S18	Zeroize SK_{CFRM}	Per 800-56A Section 5.8
If CB_{ICC} == PB_INIT		
S19	Register Z=NextZ , ICCcred for $OTID_{ICC}$ = $NextOTID_{ICC}$ Use PBaddress to determine where the new PB entry needs to be stored.	The Client Application stores NextZ and ICCcred in the binding record storage, using $NextOTID_{ICC}$ as Index. It erases the previous entries (Z, $OTID_{ICC}$) The offset in SAM memory is PBaddress * sizeof (PB entry)
S20	CB_H = PB	
Else		
S21	CB_H= NO_PB	
S22	Return CB_H || OTID || PBaddress	Return information that the client application may use to determine the

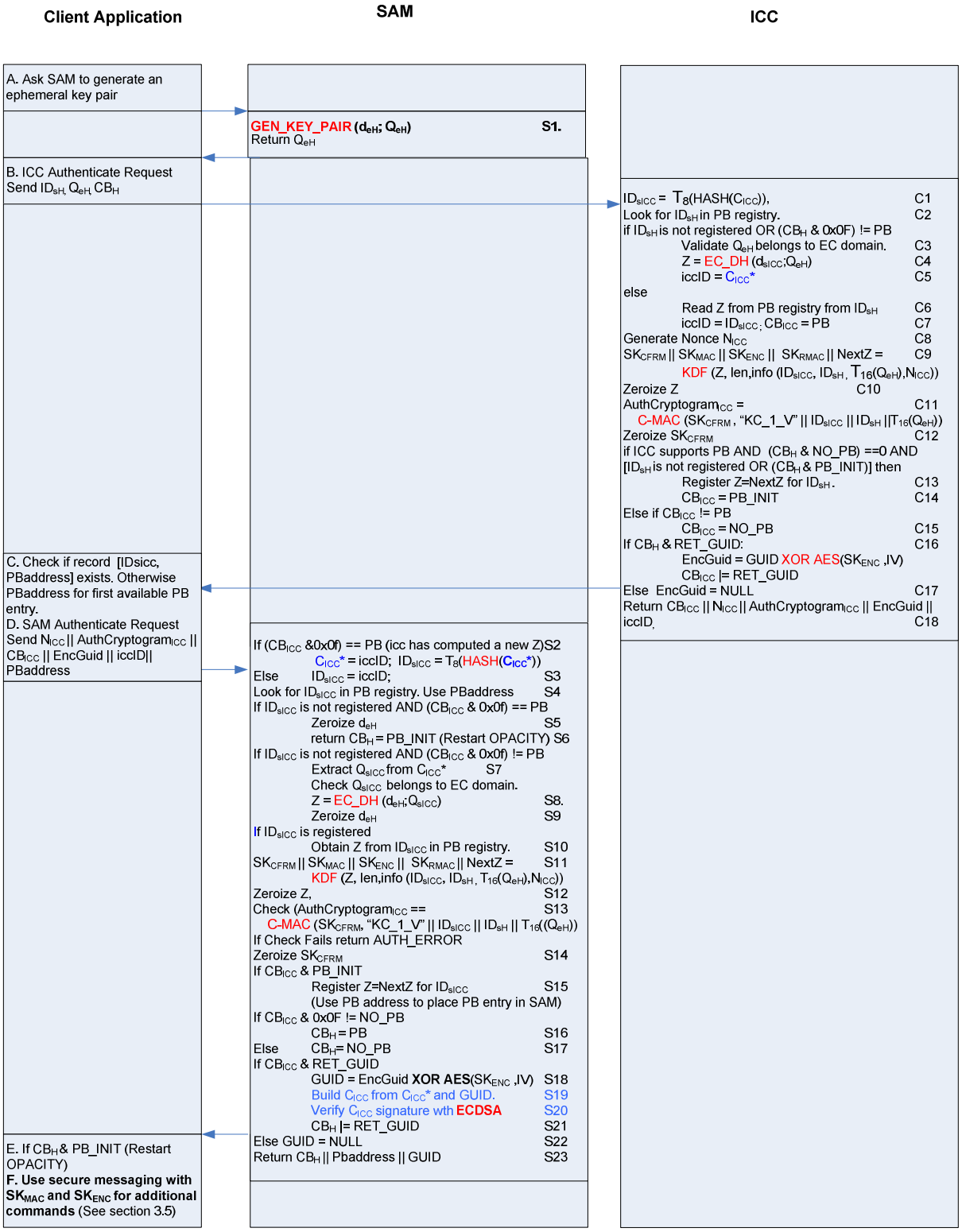
Step #	Description	Comment
		location of PB records based on OTID.

5.1.3ICC Protocol Steps

Step #	Description	Comment
C1	Validate C_H by Q_{rootH}	Using (Q_{rootH}), The card application verifies the Client CVC (C_H), Also verify the EC curve parameters per 800-56A
C2	Extract ID_{sH}, Q_{sH}	assess the binding between the unique Host ID (ID_{sH}) and the CVC public key (Q_{sH}).
C3	Look for ID_{sH} in PB registry	The ICC scans for ID_{sH} in the registry.
if ID_{sH} is not registered OR CB_H != PB		If ID_{sH} is not found in the PB table, execute a regular OPACITY-FS key agreement.
C4	Validate Q_{eH} belongs to EC domain.	verify the EC curve parameters per 800-56A
C5	GEN_KEY_PAIR (d_{eICC}; Q_{eICC})	The ICC generates ephemeral ECC key pair (d_{eICC};Q_{eICC})
C6	Z1 = EC_DH (d_{eICC};Q_{sH})	Generate the shared secret
C7	K1 || K2 = KDF (Z1; len, info (ID_{sH}, $T_{16}(Q_{eICC})$))	Use ICC entropy
C8	$OpaqueData_{ICC}$ = ENCRYPT (AES, K1 , C_{ICC})	
C9	$OTID_{ICC}$ = Q_{eICC}	
C10	N_{ICC} = NULL	
C11	Z = EC_DH (d_{sICC} ;Q_{eH})	
C12	Zeroize Z1,K1	
if ID_{sH} is registered AND CB_H == PB		If ID_{sH} is found in the PB table, use the registry to rcover the previously stored shared secret
C13	Obtain Z, $OTID_{ICC}$ from ID_{sH} in PB registry	Using ID_{sH} as index, obtain Z, $OTID_{ICC}$ from the registry
C14	Generate Nonce N_{ICC}	The card generates secret nonce N_{ICC}
C15	$OpaqueData_{ICC}$ = N_{ICC} , K2 = N_{ICC}	
Continue		
C16	SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || $NextOTID_{ICC}$ || NextZ = KDF (Z, len,info (ID_{sH}, $T_8(OTID_{ICC})$, $T_{16}(Q_{eH})$, K2))	Compute the new session keys, the shared secret and the one-time identifier for the next session Note that NextZ is only used when the condition (C19): if CB_H != NO_PB and ICC supports PB. AND [ID_{sH} is not registered or CB_H == PB_INIT] . Next Z does not need to be computed otherwise.
C17	Zeroize K2, Z, $d_{eICC,}$ $Q_{eICC,}$	
C18	$AuthCryptogram_{ICC}$ = C-MAC (AES, SK_{CFRM},	Compute return cryptogram allowing ICC authentication, showing it owns the session

Step #	Description	Comment
	KC_1_V" || $T_8(OTID_{ICC})$ || ID_{sH}|| $T_{16}(Q_{eH})$)	key to the client application
C19	Zeroize SK_{CFRM}	Per 800-56A Section 5.8
if CB_H != NO_PB and ICC supports PB. AND [ID_{sH} is not registered or CB_H == PB_INIT]		
C20	Register Z=NextZ and $OTID_{ICC}$ = $NextOTID_{ICC}$ for ID_{sH}	The ICC stores NextZ and $NextOTID_{ICC}$ in the binding record storage, using ID_{sH} as Index. It erases the previous entries (Z, $OTID_{ICC}$)
C21	if (CB_{ICC} != PB) CB_{ICC} = PB_INIT	
Else		
C22	CB_{ICC} = NO_PB	
C23	Return $OpaqueData_{ICC}$ || $AuthCryptogram_{ICC}$ || CB_{ICC} || $OTID_{ICC}$	

5.2 OPACITY with Zero Key Management (ZKM - Optimized)



This protocol is derived from the standard ISO/IEC 24727-3 "EC Key Agreement with ICC Authentication, Appendix A-1 section A.19".

This protocol is identified by the OID { iso(1) standard(0) iso24727(24727) part3(3) annex-a(0) protocol(0) EC-key-agreement-with-ICC-authentication(12) }.

5.2.1 Client Application Protocol Steps

Step #	Description	Comment
A.	Ask SAM to generate an ephemeral key pair	
B.	ICC Authenticate Request Send $ID_{sH,}$ $Q_{eH,}$ CB_H	Send ICC Command. CB_H is the Persistent binding control byte for the host. It is a logical combination of : - PB: 0x01 Use Persistent binding - NO_PB: 0x00 No persistent binding for this host - PB_INIT 0x02: reset Persistent binding entry for the host (re-compute ECDH) - RET_GUID: 0x10 The host is requesting the encrypted GUID in the ICC response. - ONE_SK: 0x20 The host expects a single session key SK=SKmac = SKenc=SKrmac
Wait for ICC to respond Get ICC response: CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$ || EncGuid || iccID		CB_{ICC} is the Persistent binding control byte for the ICC. It is a logical combination of - PB: 0x01 Host was found in ICC PB registry - NO_PB: 0x00 No persistent binding for this ICC - PB_INIT: 0x02 ICC has set a new Persistent binding entry for the host (re-computed ECDH) - RET_GUID: 0x10 The ICC response includes the encrypted GUID. - ONE_SK: 0x20 The host expects a single session key SK=SKmac = SKenc=SKrmac iccID is either the truncated hash of Cicc (PB mode) or Cicc* otherwise.
C.	Determine PB offset based on IDsicc.	Search locally among [PBaddress, IDsicc] records. If IDsicc is not found, determine best PBaddress value (first free record slot). Feature can be disabled with PBaddress = NULL.
D.	SAM Authenticate Request Send CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$ || EncGuid || iccID || PBaddress	Send SAM Command. Forward ICC response.
Wait for SAM to respond Get SAM response: $CB_{H,}$ PBaddress, GUID (if CB_H & RET_GUID) != 0		
E.	If (CB_H & 0x0f) == PB_INIT (Restart OPACITY)	
OPACITY-ZKM complete.		
F.	Use secure messaging with SK_{MAC} and SK_{ENC} for additional commands	

5.2.2 SAM Protocol Steps

Step #	Description	Comment

Step #	Description		Comment
First ephemeral key generation command – may be executed in advance			
S1	**GEN_KEY_PAIR (d_{eH}; Q_{eH})** Return Q_{eH}		The SAM generates an EC Key Pair.
Wait for next session key establishment command: CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$ || EncGuid || iccID.			
	If (CB_{ICC} & 0x0f) == PB		If PB bit is not set in CB_{ICC} byte
S2.		**C_{ICC}* = iccID; $ID_{sICC} = T_8$(HASH(C_{ICC}*))**	The ICC returns C_{ICC}*. The actual ICC persistent binding table entry to use is T_8(HASH(C_{ICC}*)). The object (C_{ICC}*) returned in iccID is a transformation of the actual CVC, i.e. (C_{ICC}). The TLV data element (T=0x5F20, L=16, V=GUID) of (C_{ICC}) is replaced by TL, (T=0x5F20, L=0). The signature data element remains unmodified.
	Else		If (CB_{ICC} & 0x0f) != PB
S3.		ID_{sICC} = iccID;	Only the truncated hash of the CVC s returned
S4.		Look for ID_{sICC} in PB registry.	Check if the ICC is registered as a PB entry. Determine PBAddress.
`	If ID_{sICC} is not registered AND CB_{ICC} & 0x0f == PB		Registry Inconsistency. The host does not find the ICC in the PB registry (formerly erased) but ICC is expecting to be registered. This behavior is normal if the registry is full or is the policy requires that old registry records are erased.
S5.		Zeroize d_{eH}	Reset SAM status
S6		return CB_H = PB_INIT (Restart OPACITY)	Let client application know it needs to restart the protocol with CB_H = PB_INIT
	If ID_{sICC} is not registered AND (CB_{ICC} & PB) != 0		ICC is not expecting the host to have registered it. The SAM proceeds with a complete EC-based key agreement
S7.		**Extract Q_{sICC} from C_{ICC} and Validate Q_{sICC} belongs to EC domain.**	Extract the static ICC public key from the CVC
S8.		**Z = EC_DH (d_{eH};Q_{sICC})**	Compute the shared secret.
S9.		**Zeroize d_{eH}**	Destroy ephemeral host private key.
	If ID_{sICC} is registered		Each side has found the PB entry of the other side.
S10.		Obtain Z from ID_{sICC} in PB registry.	Read the PB entry indexed with ID_{sICC}
S11.	**SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH}, $T_{16}(Q_{eH})$,N_{ICC}))**		Compute the session keys and Next Z. NextZ length is 16bytes, Note that NextZ is only used when the condition (S15): CB_{ICC} == PB_INIT is TRUE. Next Z does not need to be computed otherwise. If CBicc & ONE_SK, the computation is as follows: **SK_{CFRM} || SK_{MAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH}, $T_{16}(Q_{eH})$,N_{ICC}))** And **SK_{ENC} = SK_{MAC}, SK_{RMAC} = SK_{MAC}**
S12.	**Zeroize Z,**		Destroy Z.
S13.	**Check ($AuthCryptogram_{ICC}$ == C-MAC (SK_{CFRM}, "KC_1_V" || ID_{sICC} || ID_{sH} || $T_{16}((Q_{eH}))$**		Check key confirmation cryptogram. Return authentication error if verification fails.

Step #	Description	Comment
	If Check Fails return AUTH_ERROR	
S14.	Zeroize SK_{CFRM}	Destroy session key used for key confirmation
	If CB_{ICC} == PB_INIT	
S15.	Register Z=NextZ for ID_{sICC}	Create a new PB registry entry. With the shared secret used f the next transaction. HINT: If the registry is full, then the oldest or less used records should be identified and replaced. The internal aging algorithms are independent from the card interface and outside the scope of this specification.
	If (CB_{ICC} & 0x0F) != NO_PB and (CB_H & 0x0F) != NO_PB	If ICC is maintaining a record for the host (i.e. CB_{ICC} == PB_INIT or CB_{ICC} == PB)
S16.	CB_H = PB	Prepare response to client application: CB_H = PB, i.e. Host has registered ICC in PB table.
	Else	ICC is not maintaining a record for the host: no ICC record shall be stored either.
S17.	CB_H= NO_PB	Prepare response to client application: CB_H = NO_PB, i.e. Host is not registering ICC in PB table
	If CB_{ICC} & RET_GUID	If the ICC has included an encrypted GUID in the response
S18	GUID = EncGuid **XOR AES**(SK_{ENC} ,IV) CB_H |= RET_GUID	Decrypt GUID. Indicate that SAM response includes the GUID. IV is a 16-byte constant known on both sides. "0x80 00 .. 00"
S19	Build C_{ICC} from C_{ICC}* and GUID. S19	Note: the object (C_{ICC}*) is a transformation of the actual CVC, i.e. (C_{ICC}). The TLV data element (T=**0x5F20,** L=16, V=GUID) of (C_{ICC}) is replaced by TL, (T=**0x5F20,** L=0). The signature data element remains unmodified.
S20	Verify C_{ICC} signature wth **ECDSA**	To verify the signature with ECDSA, the SAM must reconstruct (C_{ICC}) from (C_{ICC}*) the actual GUID TLV of the (C_{ICC}), L= len(GUID) and V=GUID.
S21	CB_H |= RET_GUID	
S22	Else GUID = NULL	
S23.	Return CB_H || PBaddress	Return PB control byte to client application.

5.2.3ICC Protocol Steps

Step #	Description	Comment
C1	**ID_{sICC} = T_8(HASH(C_{ICC}))**	**Prepare ICC ID value– may be pre-computed**
C2.	Look for ID_{sH} in PB registry.	Look in PB registry for host ID.
	if ID_{sH} is not registered OR (CB_H & 0x0F) != PB	If host not found in PB registry, or if host asks ICC to ignore registry, perform a full EC-based key agreement.
C3.	**Validate Q_{eH} belongs to EC domain.**	Check ephemeral public key validity
C4.	**Z = EC_DH (d_{sICC};Q_{eH})**	Compute shared secret
C5.	**iccID = C_{ICC}***	ICC identifier returned will be (C_{ICC}*), a transformation of Card Verifiable Credential (C_{ICC}) (C_{ICC}). The TLV data element (T=0x5F20, L=16, V=GUID) of (C_{ICC}) is replaced by TL, (T=0x5F20, L=0). The signature data element remains

Step #	Description		Comment
			unmodified.
	else		Host is found
C6.		Read Z from PB registry from ID_{sH}	Access shared secret
C7.		iccID = ID_{sICC} ; CB_{ICC} = PB	Return Truncated hash of CVC as identifier. Prepare ICC control byte response
C8.	**Generate Nonce N_{ICC}**		
C9.	**SK_{CFRM} || SK_{MAC} || SK_{ENC} || SK_{RMAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH} , $T_{16}(Q_{eH})$,N_{ICC}))**		Compute session keys and shared secret for next session. Note that NextZ is only used when the condition (C13): (ICC supports PB AND CB_H != NO_PB AND [ID_{sH} is not registered OR CB_H == PB_INIT]) is TRUE. Next Z does not need to be computed otherwise. If CBh & ONE_SK, the computation is as follows: **SK_{CFRM} || SK_{MAC} || NextZ = KDF (Z, len,info (ID_{sICC}, ID_{sH} , $T_{16}(Q_{eH})$,N_{ICC}))** And **SK_{ENC} = SK_{MAC}, SK_{RMAC} = SK_{MAC}** And CBicc |= ONE_SK
C10.	**Zeroize Z**		Destroy current shared secret.
C11.	**$AuthCryptogram_{ICC}$ = C-MAC (SK_{CFRM} , "KC_1_V" || ID_{sICC} || ID_{sH} ||$T_{16}(Q_{eH})$)**		Compute key confirmation cryptogram per 800-56A.
C12.	**Zeroize SK_{CFRM}**		Destroy session key sued to compute the KC cryptogram
	if ICC supports PB AND (CB_H & NO_PB) ==0 AND [ID_{sH} is not registered OR (CB_H & PB_INIT)] then		If both host and ICC supports PB, and no record for the host is present in the ICC registry, or if a new record is requested for that host.
C13.		Register Z=NextZ for ID_{sH} .	Register shared secret for next session.
C14.		CB_{ICC} = PB_INIT	Prepare ICC control byte response
	Else		If Either Host or ICC does not support PB
C15.		CB_{ICC} = NO_PB	Prepare ICC control byte response
	If CB_H & RET_GUID and GUID exists		If an encrypted GUID must be returned
C16		EncGuid = GUID **XOR AES**(SK_{ENC} ,IV) CB_{ICC} |= RET_GUID	Encrypt the GUID, i.e., the value V of the GUID data element 0x5F20 of (C_{ICC}). The encryption is performed with the session key for encryption. Indicate that the encrypted GUID is present in the response using CB_{ICC} IV is a 16-byte constant known on both sides. "0x80 00 .. 00"
	Else		
C17		EncGuid = NULL	No GUID in the response
C18	**Return CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$** || EncGuid || **iccID**		

6.0 ISO 7816-4 APDU Interface

The following is a 7816-4 card edge interface supporting the OPACITY protocol and recommended for interoperability. The APDU sequence is for ISO7816 T=0 card in contact mode.

6.1 Host Control Byte (CB_H)

CB_H encoding	FS mode	ZKM Mode	b7 (RFU)	b6	B5	b4	b3 (RFU)	B2	b1	b0
NO_PB	Y	Y	-	-	-	-	-	-	0	0
PB	Y	Y	-	-	-	-	-	-	0	1
PB_INIT	Y	Y	-	-	-	-	-	-	1	0
ENC_GUID	N	Y	-	-	-	1	-	-	-	-
CLR_GUID	N	Y	-	-	-	0	-	-	-	-
THREE_SK	Y	Y	-	-	0	-	-	-	-	-
ONE_SK	Y	Y	-	-	1	-	-	-	-	-
Opacity ZKM	N	Y		0						
Opacity Full Secrecy	Y	N	-	1	-	-	-	-	-	-

- PB: ICC may use Persistent binding if supported
- NO_PB: No persistent binding supported by this host: ICC may not use persistent binding
- PB_INIT: ICC shall reset Persistent binding entry for the host (re-compute ECDH)
- ENC_GUID: ICC includes the encrypted GUID in the response
- CLR_GUID: ICC includes the GUID in the CVC, not encrypted
- ONE_SK: Request ICC to establish only one session key
- THREE_SK: ICC has established Three session keys for Secure Messaging

6.2 ICC Control Byte (CB_{ICC})

CB_{ICC} encoding	FS Mode	ZKM mode	b7 (RFU)	b6	b5	b4	b3 (RFU)	B2	b1	b0
NO_PB	Y	Y	-	-	-	-	-	-	0	0
PB	Y	Y	-	-	-	-	-	-	0	1
PB_INIT	Y	Y	-	-	-	-	-	-	1	0
ENC_GUID	N	Y	-	-	-	1	-	-	-	-
CLR_GUID	N	Y	-	-	-	0	-	-	-	-
THREE_SK	Y	Y	-	-	0	-	-	-	-	-
ONE_SK	Y	Y	-	-	1	-	-	-	-	-
Opacity ZKM	N	Y	-	0	-	-	-	-	-	-
Opacity Full Secrecy	Y	N	-	1	-	-	-	-	-	-

- PB: Host was found in ICC PB registry
- NO_PB: Persistent binding has not been used for this transaction
- PB_INIT: ICC has set a new Persistent binding entry for the host (re-computed ECDH)
- ENC_GUID: the encrypted GUID is present in the response
- CLR_GUID. The GUID is returned as part of the CVC
- ONE_SK: ICC has established only one session key for Secure messaging,
- THREE_SK: ICC has established Three session keys for Secure Messaging.

6.3 Cipher Suites Encoding

The protocol can be configured to support the following cipher suites:

FIPS 140-2 modes	Fast ZKM only	FS/ZKM	Strong Key transport	Strong FS	Government Classified
Cipher Suite	**CS1**	**CS2**	**CS3**	**CS4**	**CS5**
Encoding (P1)	**0xE4**	**0xE8**	**0xE9**	**0xEA**	**0xEB**
Channel Strength (bits)	**112**	**128**	**192**	**192**	**192**
Encryption or MAC (Session keys)	AES128	AES128	AES 256	AES 192	AES256
ICC CVC Signature	ECDSA 224	ECDSA 256	ECDSA 256	ECDSA 384	ECDSA 384
Host CVC Signature	N/A	ECDSA 256	ECDSA 384	ECDSA 384	ECDSA 384
ICC Key Agreement	ECDH 224	ECDH 256	ECDH 256	ECDH 384	ECDH 384

Host Key Agreement	ECDH 224	ECDH 256	ECDH 384	ECDH 384	ECDH 384
Hashing	SHA 1	SHA 256	SHA 384	SHA 384	SHA 384
Nonces (per 800-56A)	16 bytes	16 bytes	24 bytes	24 bytes	32 bytes

6.4 OPACITY FS mode

Command	Response	Comments
Authenticate Host (SAM) to ICC		**Command sent to ICC**
00.86.P1.P2.LL { CB_H || C_H || Q_{eH} }		**GENERAL AUTHENTICATE** Where: P1 = cipher suite (see section 6.3) P2 = Opacity ICC Private Key d_{sICC} reference or 0x00 if implicit CB_H encoding: see section 6.1 - C_H is a self descriptive ICC CVC as defined in 7816-8 Annex B.2: {42 - L - Issuer identification number} - {'5F20' - L -ICCholder name} - {'5F49' - L - ICCholder public key} {'5F37' - L - Digital signature} - Q_{eH} is a public EC key (Q_{eH} is NULL for base Opacity)
	CB_{ICC} || $OpaqueData_{ICC}$ || $AuthCryptogram_{ICC}$ || $OTID_{ICC}$	Where if CB_{ICC} == NO_PB or PB_INIT: - $OpaqueData_{ICC}$ = CVC encrypted with K1 - $OTID_{ICC}$ = QeICC if CB_{ICC} == PB: - $OpaqueData_{ICC}$ = Nicc (16bytes) - $OTID_{ICC}$ = result of KDF. $AuthCryptogram_{ICC}$ result of Key Confirmation computation
Authenticate ICC to SAM (Host)		**Command sent to SAM**
00.86.P1.P2.LL { CB_{ICC} || $OpaqueData_{ICC}$ ||		

Command	Response	Comments
$AuthCryptogram_{ICC}$ || $OTID_{ICC}$ || PBaddress }		**GENERAL AUTHENTICATE** Where: P1 = cipher suite (see section 6.3) P2 = Opacity SAM Private Key d_{sH} reference or 0x00 if implicit CB_{ICC} encoding: See section 6.2 Where if CB_{ICC} == NO_PB or PB_INIT: - $OpaqueData_{ICC}$ = CVC encrypted with K1 - $OTID_{ICC}$ = QeICC if CB_{ICC} == PB: - $OpaqueData_{ICC}$ = Nicc (16bytes) - $OTID_{ICC}$ = result of ICC KDF. - $AuthCryptogram_{ICC}$ result of Key Confirmation computation
	CB_H || $OTID_{ICC}$ || PBaddress	CB_H encoding: see section 6.1
A secure channel with SKmac & SKenc is now available; The host may use this channel to perform additional transaction steps. For instance it may send an acknowledgement to the ICC that IccData has been processed successfully.A		
Wrap command APDU with SAM		**Command sent to SAM**
CLA.87.P1.P2.LL { 81. LL. Command-APDU-to-wrap }		CLA = '0x00' or '0x10' When command is chained or not. P1 = Command APDU **wrapping** cryptographic mechanism. P2: 0x00. key reference qualifier. The available session keys are used to wrap the APDU Command-APDU-to-wrap: GA Challenge Note the **wrapping** command shall maintain and update the single SM wrapping/unwrapping context associated to

Command	Response	Comments
		the P2 key reference. For instance, the wrapping command shall interpret and use the CLA byte of the command to determine if there are more chained commands to follow. It also analyzes the expected response length to support the subsequent unwrapping commands. Similarly, the unwrapping command shall interpret the SM context to determine if additional response bytes are expected. If any error or inconsistency occurs, the SM context shall be reset.
	82. LL. Wrapped-command APDU }	Wrapped-command-APDU: GA Response
Send wrapped APDU		**Command sent to ICC**
Wrapped APDU	Wrapped response	
Unwrap response APDU with SAM		**Command sent to SAM**
CLA.87.P1.P2.LL { 81. LL. Response-APDU-to-unwrap }		CLA = '0x00' or '0x10' When command is chained or not. P1: Response APDU **unwrapping** cryptographic mechanism (Secure Messaging). P2: 0x00. key reference qualifier. The available session keys are used to wrap the APDU Response-APDU-to-unwrap. GA Response. Note the **unwrapping** command shall maintain and update the single SM wrapping/unwrapping context associated to the P2 key reference. For instance, the wrapping command shall interpret and use the CLA byte of the command to determine if there are more chained commands to follow. It also analyzes the expected response length to support the subsequent unwrapping commands. Similarly, the unwrapping command shall interpret the SM context to determine if additional response bytes are expected. If any error or inconsistency occurs, the SM context shall be reset.
	81.LL.Unwrapped response	Unwrapped response: GA Response

6.5 OPACITY ZKM mode

Command	Response	Comments
Authenticate Host – ICC Command		
00.86. P1.P2 { CB_H || ID_{sH} || Q_{eH} }		**GENERAL AUTHENTICATE** Where: P1 = cipher suite (see section 6.3) P2 = Opacity ICC Private Key d_{sICC} reference or 0x00 if implicit CB_H encoding: see section 6.1 - ID_{sH} is a 8 bytes Host identifier. - Q_{eH} is a public EC key
`	CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$ || EncGuid || iccID	Where: - N_{ICC} is a 16-byte nonce - $AuthCryptogram_{ICC}$ is a 16 byte binary string. - CB_{ICC} indicates how the Persistent binding was executed on the card. See section 6.2 . - iccID is either: C_{ICC}* when CB_{ICC} & NO_PB or CB_{ICC} & PB_INIT: , i.e. a self descriptive ICC CVC as defined in 7816-8 Annex B.2, And in ANNEX B of this document. (T_8(HASH(C_{ICC})) when CB_{ICC} == PB:
Authenticate ICC to SAM (Host)		
00.86.P1.00.LL { CB_{ICC} || N_{ICC} || $AuthCryptogram_{ICC}$ || EncGuid || iccID || PBaddress }		GENERAL AUTHENTICATE Where: P1 = cipher suite (see section 6.3) CB_{ICC} encoding: See section 6.2

		. iccID = C_{ICC}* if CBICC == PB: Truncated Hash (leftmost 8 bytes of CVC hash) AuthCryptogramICC result of Key Confirmation computation - EncGuid is a 16 byte cryptogram, only present if RET_GUID is set in CB_{ICC}. Otherwise EncGuid is NULL. - PBaddress: table entry number (2 bytes). If PBaddress=0x0000, no entry is created. PBaddress=0x0001 first entry in table PBaddress=0x0002 second entry in table
	CB_H || PBaddress || GUID	CB_H encoding: - PB: 0x01 Use Persistent binding - NO_PB: 0x00 No persistent binding for this host - PB_INIT 0x02: reset Persistent binding entry for the host (re-compute ECDH) - RET_GUID: 0x10 The host is requesting the encrypted GUID in the ICC response. - ONE_SK: 0x20. Only one secure messaging session key is derived with KDF. PBaddress: table entry number (2 bytes). If PBaddress=0x0000, no entry is created. PBaddress=0x0001 first entry in table PBaddress=0x0002 second entry in table GUID: (16bytes)
Client application Verifies SAM Response		
		A secure channel with SKmac & SKenc is now available; The host may use this channel to perform additional transaction steps. For instance it may send an acknowledgement to the ICC that IccData has been processed successfully.
Wrap command APDU with SAM		**Command sent to SAM**
CLA.87.P1.P2.LL { 81. LL. Command-APDU-to-wrap }		CLA = '0x00' or '0x10'

		When command is chained or not. P1 = Command APDU **wrapping** cryptographic mechanism. P2: 0x00. key reference qualifier. The available session keys are used to wrap the APDU Command-APDU-to-wrap: GA Challenge Note the **wrapping** command shall maintain and update the single SM wrapping/unwrapping context associated to the P2 key reference. For instance, the wrapping command shall interpret and use the CLA byte of the command to determine if there are more chained commands to follow. It also analyzes the expected response length to support the subsequent unwrapping commands. Similarly, the unwrapping command shall interpret the SM context to determine if additional response bytes are expected. If any error or inconsistency occurs, the SM context shall be reset.
	82. LL. Wrapped-command APDU }	Wrapped-command-APDU: GA Response
Send wrapped APDU		**Command sent to ICC**
Wrapped APDU	Wrapped response	
Unwrap response APDU with SAM		**Command sent to SAM**
CLA.87.P1.P2.LL { 81. LL. Response-APDU-to-unwrap }		CLA = '0x00' or '0x10' When command is chained or not. P1: Response APDU **unwrapping** cryptographic mechanism (Secure Messaging). P2: 0x00. key reference qualifier. The available session keys are used to wrap the APDU Response-APDU-to-unwrap. GA Response. Note the **unwrapping** command shall maintain and update the single SM wrapping/unwrapping context associated to the P2 key reference. For instance, the wrapping command shall interpret and use the CLA byte of the

		command to determine if there are more chained commands to follow. It also analyzes the expected response length to support the subsequent unwrapping commands. Similarly, the unwrapping command shall interpret the SM context to determine if additional response bytes are expected. If any error or inconsistency occurs, the SM context shall be reset.
	81.LL.Unwrapped response	Unwrapped response: GA Response

7.0 ANNEX A - KDF Specifications

The Key Derivation Function is the concatenation KDF as specified in NIST SP800-56A (§ 5.8.1).

Note that SP800-108 KDF would have been an alternate choice for the successive KDFs following the first 800-56A KDF, and in particular of session keys from NextZ.

KDF takes as input a shared secret Z, total session key data length 'len' and supplementary data 'info', and produces session keys $\mathbf{SK_1 || .. || SK_p}$ as follows:

$\mathbf{SK_1 || .. || SK_p = KDF\ (Z, len, info(A_1, .. A_m))}$

Where:

$\mathbf{info\ (A_1, .. A_m) = AlgoID(SK_1) || .. || AlgoID(SK_p)\ \ || A_1 || .. || A_m}$

Algo ID	Value	Note
AES 128 key	0x09	800-78-1
AES 192 key	0x0B	800-78-1
AES 256 key	0x0D	800-78-1
OTID	0xF0	
NextZ	0xF1	

$\mathbf{len = \ Length(SK_1) + .. + Length(SK_p)}$

n = len / (hashLengthInBits (KDFHashAlgorithm))

DerivedKeyingMaterial =

KDFHashAlgorithm (0x00000001 || Z || info) ||

KDFHashAlgorithm (0x00000002 || Z || info) || ..

KDFHashAlgorithm (0x00000000 + n) || Z || info)

The resulting concatenated session keys ($\mathbf{SK_1}$ || .. || $\mathbf{SK_p}$) are the (len) left-most bits from DerivedKeyingMaterial.

8.0 ANNEX B - CVC Specifications

The Card Verifiable Credential Format used in this specification is as follows:

Tag	Length (BER-TLV format)	Name	Value and Description
0x5F29	0x01	Credential Profile Identifier	0x80 (for this version)
0x42	0x08	(Certificate) Issuer Identification Number	IIN (8bytes) : IssuerID (6 bytes) || IssuerKeyID (2 bytes) IssuerID: 6 bytes for Issuer id - e.g. 6 leftmost bytes of hash (SHA-256) of unique issuer name or fixed value chosen at random at setup time -. IssuerKeyID: 2 bytes for CVC signature verification public key number.
0x5F20	L1 (should not exceed 0x10)	Globally Unique IDentifier (GUID)	Application Specific, Identifies the card or cardholder.
0x7F49	L2 P-224: 2+5+2+57= 0x42 (66) P-256: 2+8+2+65= 0x4D (77) P-384: 2+8+2+97=0x6D (109)	CardHolderPublicKey	Data Object with the following tags: - 0x06: Object Identifier of the algorithm. Possible values are: ▪ 0x2B81040021 for secp224r1 {iso(1) identified-organization(3) certicom(132) curve(0) ansip224r1(33)} algorithm ▪ 0x2A8648CE3D030107 for Prime256v1 {iso(1) member-body(2) us(840) ansi-x962(10045) curves(3) prime(1) prime256v1(7)} algorithm ▪ 0x2B81040022 for ansiX9p384r1 ::= { iso(1) identified-organization(3) certicom(132) curve(0) 34 } - 0x86: Public Key coded as follows: 04 || X || Y, where X and Y are the coordinates of the point on the curve.

0x5F37	L3
	ECDSA with SHA-224:
	ECDSA with SHA-256: 0x37 (55)
	ECDSA with SHA-384:
		DigitalSignature	DigitalSignature ::= SEQUENCE {
			signatureAlgorithm AlgorithmIdentifier,
			signatureValue BIT STRING
			}
			Where
			– signatureAlgorithm is **ecdsa-with-SHA1(1)** (iso(1) member-body(2) us(840) ansi-x962(10045) signatures(4)ECDSA with SHA-224: 1.2.840.10045.4.3.1
			– ECDSA with SHA-256: 1.2.840.10045.4.3.2

			– ECDSA with SHA-384: 1.2.840.10045.4.3.3
			– signatureValue is a BITSTRING encoding of signature result **`ECDSA-Sig-Value`** defined below.
			ECDSA-Sig-Value ::= SEQUENCE {
			r INTEGER,
			s INTEGER
			}
			AlgorithmIdentifier ::= SEQUENCE {
			algorithm OBJECT IDENTIFIER,
			parameters ANY DEFINED BY algorithm OPTIONAL
			}
			The verification public key is referenced with the IssuerID Data Element.

0x5F4C	0x01	RoleIdentifier	<u>For root certificates:</u> `b8b7b6b5b4b3b2b1` `0 0 x 1 - - - -` Key assigned to card app. `0 0 1 x - - - -` Key assigned to client app. `- - - - 0 0 0 1` Verification key `- - - - 0 0 1 0` Authentication Key Card application root CVC role ID: 0x12 Client application root CVC role ID: 0x22 <u>For cardholder certificates:</u> ‘0x00’ : for card application key CVC ‘0x80’ : for card application administrator key CVC ’0x01’: for client application key CVC

.

9.0 ANNEX C – Advanced Opacity Implementation

This section describes how to implement the client application interface to a SAM.

9.1 Large PB registries for SAMs

When the persistent binding registry has many records (>100), the time necessary to locate the PB record address from the ICC identifier (***ID_{sICC}*** for ZKM and ***NextOTID*** for FS) can be prohibitive.
In such a situation, it is preferable to implement the indexation mechanism on the client application instead of the SAM.

The principle is as follows.
- During the first OPACITY transaction between a ICC and a SAM, the SAM registers the PB record assigned the ICC and returns the PB record address (***PBaddress***) and the next ICC identifier (***ID_{sICC}*** for ZKM and ***NextOTID*** for FS) to the client application.

- The client application maintains in a persistent local index table the association (next ICC identifier***, PBaddress).***

- During the following transactions between the same ICC and the same SAM, the client application first checks if the ICC (***ID_{sICC}*** for ZKM or ***OTID*** for FS) is already registered in the persistent local index table and in which case it obtains ***PBaddress***. ***PBaddress*** is then transmitted to the SAM for direct access to the PB record (shared secret Z, etc,) If there is no persistent binding record for the ICC, a Full OPACITY transaction is requested.

9.2 Multiple CVC signing keys

Multiple signers of CVCs are supported in the OPACITY specification.

The recommended method is to reserve the rightmost two bytes in the IIN data element (Tag: 0x42h, Length: 0x08h) of the CVC as a reference to a particular CVC verification public key.

When CVC verification public keys are registered in the SAM or ICCs they must be associated to those two bytes.