VeriFone VeriSmart EC-Card Subsystem

Custom Security Builder Hardware for VeriFone

Part 2: Security Policy

Chapter 1: Introduction

The EC Card is a PCI-bus card that carries out cryptographic operations in a tamper-resistant metal shroud. Upon removal of the shroud, all the keying information is actively grounded to zero.

The deletion of keying material via the appropriate function call causes the storage area to be overwritten by zeroes.

There is access control to the cryptographic functions by means of User Identification and Password. There are two roles: Cryptographic Officer and User. See Section 2 for further information.

1.1 Scope

The Security Policy specifies the security rules under which the EC Card operates. The document assumes familiarity with [1].

1.2 Definitions and Acronyms

API	Application Programmer's Interface - the only access to the functionality of the EC Card. The API is documented in [1].
EC Card	A PCI-bus card built by Certicom to satisfy the requirements for a secure housing in which to carry out cryptographic functions.

Chapter 2: Summary of Roles and Services

The summaries of the roles and services may found in the sections indicated below.

2.1 Roles.

2.2 Services.

2.1 Roles

There are two distinct operator roles: User role and Cryptographic Officer role. The separation of roles is enforced using operator authentication. An operator must select a role and then log-in to the Card using the appropriate access control password for said role. At the end of each session, the operator must log-out.

The two roles are delineated as follows. A complete list of which services are available to which role may be found in 2.2.

1. The Cryptographic Officer role (CO):

   An authorized operator acting as CO is allowed to generate public-key pairs, inject DES keys into the Card, and change passwords. No cryptographic activities beyond these are permitted for the CO.

2. The User role:

   An authorized operator acting as User has access to all cryptographic activities except those restricted to the CO as specified above.

2.2 Services

The table below specifies which services are permitted by which role. Services are implemented via function calls. The table lists all the functions available through the API with one column per role specifying whether the function is available or denied by said role. Functions marked with the em-dash (—) are denied to the role; functions marked with the word “yes” are available to the role. Further information may be found in the sections indicated.

API Function Permissions

		Roles		
Class	***Function***	**CO**	**User**	***Section***
Access Control	`sb_login()`	yes	yes	5.1
	`sb_logout()`	yes	yes	
	`sb_changeCryptoOfficerPwd()`	yes	—	
	`sb_setUserPwd()`	yes	—	
	`sb_setUserId()`	yes	—	
	`sb_getRemainingLoginAttempts()`	yes	yes	
Cryptosession	`sb_dataSize()`	yes	yes	5.2
	`sb_initialize()`	yes	yes	
	`sb_end()`	yes	yes	
	`sb_clearAllSessionKeys`	yes	yes	
Backup/Restore	`sb_cardGetKeyShadow()`	yes	—	5.3
	`sb_cardBackupDataBase()`	yes	—	
	`sb_cardDesBackupDataBase()`	yes	—	
	`sb_cardSendKeyShadow()`	yes	—	
	`sb_cardRestoreDataBase()`	yes	—	
Generate Keys	`sb_genKeyPair()`	yes	—	5.4
	`sb_keyDeleteKeyPair()`	yes	—	
	`sb_keyGetPublicKey()`	yes	yes	
MAC Keys	`sb_keyCalculateMAC()`	—	yes	5.5
	`sb_keyDeleteMAC()`	yes	yes	
DES Keys	`sb_desDeleteKey()`	yes	yes	5.6
	`sb_desDeleteAcquirerKey`	yes	yes	
	`sb_desDeleteKeyEncryptingKey()`	yes	—	
	`sb_desStoreKEK()`	yes	—	
	`sb_desDeleteKEK()`	yes	—	
	`sb_desStoreKey()`	yes	—	
	`sb_desStoreAquirerKey`	yes	yes	
	`sb_desStoreEncryptedKey()`	yes	yes	
	`sb_desReEncrypt()`	—	yes	

API Function Permissions

		Roles		
Class	***Function***	**CO**	**User**	***Section***
Key-exchange	`sb_dhGenerateValues()`	yes	—	5.7
	`sb_dhSharedSecretWithAddInfo()`	—	yes	
	`sb_dhSharedSecretPATM()`	—	yes	
Random-Numbers	`sb_rngInitialSeed()`	yes	—	5.8
	`sb_rngSeedFIPS186()`	—	yes	
	`sb_rngFIPS186Session()`	—	yes	
	`sb_rngRandomBytes()`		yes	
ActivCard Functions	`sb_storeEncrypedMTMK()`	yes	—	5.9
	`sb_storeMTMK()`	yes	—	
	`sb_deleteMTMKActiveCard()`	yes	—	
	`sb_genSessionKey()`	—	yes	
	`sb_storeEncryptedSessionKey`	—	yes	
	`sb_genSessionMAC()`	—	yes	
	`sb_updateTMK()`	yes	—	
Miscellaneous	`sb_cardKeyUsage()`	yes	yes	5.10
	`sb_cardKeyStatus(0`	yes	yes	
	`sb_retrieveErrorCodes()`	yes	yes	
	`sb_cardProductVersion()`	yes	yes	
	`Re-program Firmware`	yes	—	

Chapter 3: Security Relevant Data Items

3.1 Definition Access Rights

Access rights to SRDIs are defined below. (The list of SRDIs and access rights may be found in x3.2.)

read access	A role has read access to an SRDI if said SRDI may be read within the Card and the value output from the Card in the clear.
write access	A role has write access to an SRDI if the current value of said SRDI may overwritten by another value input into the Card by the operator in said role in the clear.
read by proxy	A role has read access by proxy to an SRDI if said SRDI may be read on behalf of the operator in said role without the value being output from the Card in the clear or said value being overwritten. An example of the such is the calculation of the shared secret: The private key of the Card is read and used in the calculation of the shared secret by the Card on behalf of the User but the User may not modify or view the private key.
write by proxy	A role has write access by proxy to an SRDI if said SRDI may be written on behalf of the operator in said role without the value being input into the Card in the clear.

3.2 SRDI Definitions

The Security Relevant Data Items (SRDIs) are defined in the table below. Access rights to the SRDIs are indicated via the following symbols: r for read access, w for write access, p for read access by proxy, x for write access by proxy, and - denotes denial of access.

All SRDIs possess status marking them as valid or invalid. Invalid SRDIs may not be used. With the exceptions listed in Section 3.3, the initial status of each SRDI is marked as invalid.

Security Relevant Data Items (SRDIs)			
	Access Rights		
SRDI Name	***CO***	***User***	***Definition***
User Identification	`-w--`	`--p-`	A string of characters by which the operator in the User role is known and must be input to the Card before logging in to the Card.
CO's ID	`-wp-`	`----`	The fixed string of characters Crypto Officer by which the operator in the CO role is known and must be input to the Card before logging in to the Card.
CO's Password	`-wp-`	`----`	An array of bytes that must be submitted to the Card to enter the CO role.
User's Password	`-w--`	`--p-`	An array of bytes that must be submitted to the Card to enter the User role.
Initial Seed	`-wpx`	`--px`	See Note 1 (below)
Sessional Seed	`-wpx`	`-wpx`	See Note 2 (below)
Key Shadow	`rwp-`	`----`	One of the three shadows that comprise the private key with which the card's data-base is encrypted.
Key Encrypting Key	`-wpx`	`--p-`	A single-length DES key which is externally generated and injected into the Card in the clear. The key-encrypting key is stored in a distinguished location separate from the other DES keys.
PIN DES Key	`---x`	`--px`	A DES key arising from the key-exchange with the P-ATM and is used to encrypt data - specifically, the piece of data known as the PIN - from the P-ATM. The PIN DES keys are stored in distinguished locations separate from the other DES keys.
MAC Key	`---x`	`--px`	A secret key arising from the key-exchange with the P-ATM and is used to prived data-origin authentication. The MAC keys are stored in distinguished locations separate from the DES keys.
Public Key	`r--x`	`r---`	The public portion of an ECC key-pair.

Security Relevant Data Items (SRDIs)

SRDI Name	***Access Rights*** ***CO***	***User***	***Definition***
Private Key	`---x`	`--p-`	The private portion of an ECC key-pair.
Remote Key	`----`	`-w--`	The remote portion of the ECC Diffie-Hellman key-exchange carried out with the P-ATM. The remote portion is combined with the internally stored private key to generate the shared secret for the session. The first eight bytes of said shared secret forms the session DES key and the second eight bytes is the session MAC key.
Message	`----`	`-w--`	The message to be signed, and whose signature is verified, is data input to the card through the API in the clear.
PIN	`----`	`--p-`	An array of 8bytes that shall not appear in the clear outside of the card. The array is expected to hold the Personal Identification Number that is transmitted from the P-ATM to the Acquirer. See Appendix A (p. 19).
Acquirer DES Key	`---x`	`--px`	A single-length DES key originating from the acquirer and input into the Card encrypted under the KEK. These keys are stored in distinguished locations separate from the other DES keys.
ActivCard MTMK DES Key	`-wpx`	`--px`	The ActivCard Mother Terminal Master Keys (MTMKs) are triple DES keys and input into the Card encrypted under a KEK or in the clear. These keys are stored in distinguished locations separate from the other DES keys.
DefnTMK	`--px`	`--p-`	The ActivCard Terminal Master Keys (TMKs) are generated from the MTMK and a supplied seed. The TMKs are not stored on the card.

Security Relevant Data Items (SRDIs)			
	Access Rights		
SRDI Name	***CO***	***User***	***Definition***
ActivCard Key Encrypting Key	`----`	`--px`	An array of 16 random bytes that shall not appear in the clear outside of the card. The ActivCard key encrypting key is encrypted with the terminal master key using triple DES ECB.
DSA Parameters (P, Q and G)	`--p-`	`----`	The firmware on the Card must contain a valid DSA signature. When the card boots up, the kernel validates the signature as part of the self test procedure. The DSA parameters are hard coded in the kernel.
DSA Public Key	`--p-`	`----`	The DSA public key used to validate the signature of the firmware. Like the other DSA parameters, the public key is hard coded in the kernel.

Notes

1. The Initial and Sessional Seeds are designed to facilitate the use of a pure software FIPS-186 software RNG. The Initial Seed is set by the CO. To establish a crypto session, the function **`sb_initialize()`** is called with a Sessional Seed. This Sessional Seed is mixed (via XOR) with the Initial Seed before seeding the FIPS-186 RNG. Upon invocation of **`sb_end()`**, the Initial Seed is updated by mixing it (via XOR) with the current state of the RNG. The Initial Seed is maintained even when power is turned off. The values of these seeds are not important if the hardware RNG on the Card is used (either as a source of random numbers, or providing a seed to the FIPS-186 RNG).
2. Every crypto-session employs a sessional seed which is created at the start of the session (via the call **`sb_initialize()`**) and destroyed at the end of the session (via the call **`sb_end()`** { see Section 5.2) or upon log-out.

The following pictorial illustrates the storage locations for the keys given in the previous table. The MAC keys are labelled MACi , the PIN DES keys are labelled Ki , the acquirer DES keys are labelled AKi , and the key-encrypting key is labelled KEK. There are two public key-pairs; there are 100 each of MAC keys, PIN DES keys, and acquirer DES keys.

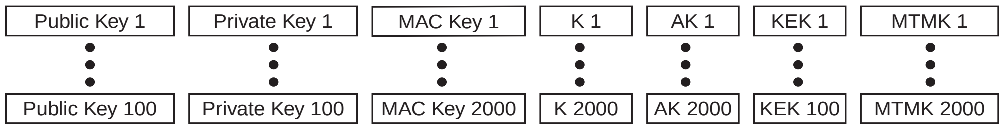

Illustration of key storage locations.

3.3 Default Values of SRDIs

The following SRDIs are not initially marked as being invalid but rather have the default values.

The CO’s Password is initially set to ECC_CARD (with an embedded space indicated by the underscore).

3.4 SRDI and Function Access

The following tables list the following:

- the list of SRDIs and the functions which act upon them.
- the functions permitted by the various roles and the SRDIs affected by the functions.

The following abbreviations are employed.

User ID is the User's Identity.

Shadow is a key shadow.

ALL refers to all SRDIs except the initial seed and the sessional seed.

KEK is the key-encrypting key.

API functions and Crypto-Officer Access Rights

Function	*SRDI*	*Rights*
`sb_login()`	CO’s Password	`--p-`
`sb_logout()`	None	
`sb_changeCryptoOfficerPwd()`	CO’s Password	`-wp-`
`sb_setUserPwd()`	User Pwd	`-w--`
`sb_setUserId()`	User ID	`-w--`
`sb_getRemainingLoginAttempts()`	None	
`sb_dataSize()`		

API functions and Crypto-Officer Access Rights

Function	*SRDI*	*Rights*
`sb_initialize()`	Initial Seed	`--px`
	Sessional Seed	`---x`
`sb_end()`	Sessional Seed	`---x`
`sb_clearAllSessionKeys()`	Session Keys	`-w--`
`sb_cardGetKeyShadow()`	Shadow	`r---`
`sb_cardBackupDataBase()`	ALL	`--p-`
	Initial Seed	`--px`
`sb_cardDesBackupDataBase()`	ALL	`--p-`
	Initial Seed	`--px`
`sb_cardSendKeyShadow()`	Shadow	`-w--`
`sb_cardRestoreDataBase()`	ALL	`---x`
`sb_genKeyPair()`	Initial Seed	`--px`
	Sessional Seed	`--px`
	Public Key	`---x`
	Private Key	`---x`
`sb_keyDeleteKeyPair()`	Public Key	`---x`
	Private Key	`---x`
`sb_keyGetPublicKey()`	Public Key	`r---`
`sb_keyDeleteMAC()`	MAC Key	`---x`
`sb_desDeleteKey()`	PIN DES Key	`---x`
`sb_desDeleteKeyEncryptingKey()`	DES KEK	`---x`
`sb_desDeleteAcquirerKey()`	Acquirer DES Key	`---x`
`sb_desDeleteKEK()`	KEK	`---x`
`sb_desStoreKey()`	KEK	`-w--`
`sb_desStoreEncryptedKey()`	KEK	`--p-`
	Acquirer’s DES Key	`---x`
`sb_desStoreKEK()`	KEK	`-w--`
`sb_desStoreAcquirerKey()`	KEK	`--p-`
	Acquirer’s DES Key	`---x`
`sb_dhGenerateValues()`	Sessional Seed	`--px`
	Private Key	`--p-`

API functions and Crypto-Officer Access Rights

Function	***SRDI***	***Rights***
	Remote Key	`-w--`
`sb_rngInitialSeed()`	Initial Seed	`-w--`
`sb_storeEncryptedMTMK()`	KEK	`--p-`
	MTMK	`---x`
`sb_storeMTMK()`	MTMK	`-w--`
`sb_deleteMTMKActivCard()`	MTMK	`---x`
`sb_updateTMK()`	MTMK	`--p-`
	TMK	`--px`
	ActiveCard KEK	`--px`
`sb_cardKeyUsage()`	None	
`sb_cardKeyStatus()`	None	
`sb_retrieveErrorCodes()`	None	
`sb_cardProductVersion()`	None	
`Re-program Firmware`	CO's Password	

API functions and User Access Rights

Function	***SRDI***	***Rights***
`sb_login()`	User ID	`--p-`
	User Pwd	`--p-`
`sb_logout()`	None	
`sb_getRemainingLoginAttempts()`	None	
`sb_dataSize()`		
`sb_initialize()`	Initial Seed	`--px`
	Sessional Seed	`---x`
`sb_end()`	Sessional Seed	`---x`
`sb_clearAllSessionKeys()`	Session Keys	`---X`
`sb_keyGetPublicKey()`	Public Key	`r---`
`sb_keyCalculateMAC()`	MAC Key	`--p-`

API functions and User Access Rights

Function	*SRDI*	*Rights*
`sb_keyDeleteMAC()`	MAC Key	`---x`
`sb_desStoreEncryptedKey()`	KEK	`--p-`
	Acquirer's DES Key	`---x`
`sb_desStoreAcquirerKey()`	KEK	`--p-`
	Acquirer's DES Key	`---x`
`sb_desDeleteKey()`	PIN DES Key	`---x`
`sb_desDeleteAcquirerKey()`	Acquirer DES Key	`---x`
`sb_desReEncrypt()`	PIN DES Key	`--p-`
	Acquirer's DES Key	`--p-`
	PIN	`--px`
`sb_dhSharedSecretWithAddInfo()`	Private Key	`--p-`
	PIN DES Key	`---x`
	MAC Key	`---x`
`sb_dhSharedSecretPATM()`	Private Key	`--p-`
	PIN DES Key	`---x`
	MAC Key	`---x`
`sb_rngSeedFIPS186()`	Initial Seed	`--px`
	Sessional Seed	`--px`
`sb_rngFIPS186Session()`	Sessional Seed	`--px`
`sb_rngRandomBytes()`	None	
`sb_genSessionKey()`	MTMK	`--p-`
	TMK	`--p-`
	ActivCard KEK	`--px`
	PIN Encryption Key	`--px`
	MAC Session Key	`--px`
`sb_storeEncryptedSessionKey()`	MTMK	`--p-`
	TMK	`--p-`
	ActivCard KEK	`--p-`
	PIN Encryption Key	`---x`
	MAC Session Key	`---x`
`sb_genSessionMAC()`	MAC Session Key	`--p-`

API functions and User Access Rights

Function	***SRDI***	***Rights***
`sb_cardKeyUsage()`	None	
`sb_cardKeyStatus()`	None	
`sb_retrieveErrorCodes()`	None	
`sb_cardProductVersion()`	None	

Chapter 4: EC Card Security Rules

Security rules are either enforced by the EC Card or are required to be carried out by the CO to ensure secure operation of the EC Card.

4.1 Enforced Security Rules

The following security rules are enforced by the EC Card.

1. There is only one CO and one User per card.
2. The CO’s password must be at least 8 bytes in length; the User’s password must be at least 5 bytes in length.
3. After fifteen unsuccessful CO log-on attempts, all keying material is overwritten with zeroes and the IDs and passwords revert to the default.
4. After ten unsuccessful User log-on attempts, the User’s password is revoked, requiring the User to return to the CO to obtain a working password.
5. The following actions must be performed by the CO before the User is allowed access to the cryptographic functions.
   (a) Change the CO’s default password.
   (b) Set the User’s password and Identity.
6. An attempt to logon when an operator is already logged on will result in the return of a failure code without affecting the current crypto-session.
7. No cryptographic operation shall be allowed in the absence of an initial seed, except when the hardware random number generator is selected when a crypto-session is initialized.
8. Restoration of the card’s data-base shall invalidate the initial seed.
9. The DES key to encrypt the PIN shall not be used for any other purpose.
10. Weak DES keys shall not be accepted for input to the card.

4.2 Other Security Rules

1. The CO and User shall keep their passwords confidential.
2. The three key shadows resulted from a database backup operation shall be stored at separate secure locations.

Chapter 5: EC Card Function Behaviour

The Card's Services are grouped as follows. With each function which employs keys may be found a description of the restrictions thereof.

x5.1 : Access Control.

x5.2 : Cryptosession.

x5.3 : Backup/Restore.

x5.4 : Generate Keys.

x5.5 : MAC Keys.

x5.6 : DES Keys.

x5.7 : Key Exchange.

x5.8 : Random Numbers.

x5.9 : ActivCard Functions

5.1 Access Control

The following Access Control functions are available.

`sb_login()` logs the operator (User or CO) in to the card.

`sb_logout()` logs the operator out from the card.

`sb_setUserId()` allows the CO to change the User's Identification to anything but the Identification that distinguishes the CO.

`sb_setUserPwd()` allows the CO to change the User's log-in password.

`sb_changeCryptoOfficerPwd()` allows the CO to change his (or her) log-in password.

`sb_getRemainingLoginAttempts()` allows the CO and the user to retrieve the number of remaining login attempts for the CO and user.

5.2 Cryptosession

The following functions are available.

`sb_dataSize()` is a utility function of no cryptographic significance. The function returns the amount of memory to be allocated on the host to hold crypto-session identifiers.

`sb_initialize()` instructs the Card to initialize a crypto-session. Initialization includes passing an initial seed to the random-number generator.

`sb_end()` terminates the crypto-session and frees any resources allocated on the Card. Said function also mixes the sessional seed into the initial seed (via xor) and overwrites all the sessional data with zeroes.

`sb_clearAllSessionKeys()` clears all the DES and MAC session keys.

5.3 Backup and Restore

The following Backup and Restore functions are available.

During the backup process, a DES or triple DES key is generated with which the information on the Card is encrypted. The Card's data-base is encrypted using DES or triple DES ECB mode and the encrypted data-base is output from the Card. A hash (computed via SHA-1) is stored within the encrypted data-base so as to determine whether the contents were corrupted. Three shadows (or shares) are computed from the key using Shamir's Threshold scheme ([4, x12.7.2]). The shadows are output from the Card.

During the restoration process, two out of the three shadows are input to the Card followed by the encrypted data-base. The shadows are combined to form the key with which the encrypted data-base is decrypted and then passed through SHA-1; the data-base will not be restored if the resulting hash does not agree with the hash found within the encrypted data-base after decryption.

`sb_cardGetKeyShadow()` retrieves a specified key shadow from the Card.

`sb_cardBackupDataBase()` retrieves the encrypted (triple DES CBC mode) data-base.

`sb_cardDesBackupDataBase()` retrieves the encrypted (single DES CBC mode) data-base.

`sb_cardSendKeyShadow()` sends a key shadow to the Card. At least two distinct shadows are required to decrypt and restore the data-base.

`sb_cardRestoreDataBase()` sends the encrypted data-base to the Card.

5.4 Generate Keys

The following functions are available.

`sb_genKeyPair()` generates a public-key key-pair. The generated private key is stored in the location reserved for private keys and no other. Similarly, the generated public key is stored in the location reserved for public keys and no other.

`sb_keyDeleteKeyPair()` deletes the public-key key-pair by overwriting the storage location with zeroes and marking the location as invalid.

`sb_keyGetPublicKey()` retrieves the public component of the key-pair from the Card.

5.5 MAC Keys

The following functions are available.

`sb_keyCalculateMAC()` calculates a MAC of a given message using the specified MAC key and returns said MAC. The function is restricted to employ only MAC keys.

`sb_keyDeleteMAC()` deletes the specified MAC key by overwriting the storage location with zeroes and marking the location as invalid. Only MAC keys may be deleted.

5.6 DES Keys

The following functions are available.

`sb_desDeleteKey()` deletes the specified session DES key (in the area reserved for acquirer's DES keys) by overwriting the storage location with zeroes and marks said location as invalid. Only a DES key may be deleted.

`sb_desStoreKey()` stores the key-encrypting DES key in the distinguished location reserved for the such and the location is marked as valid. Said key is input to the function in the clear.

`sb_desStoreKEK()` stores the key-encrypting DES key in the specified location in the area reserved for such and the location is marked as valid. Said key is input to the function in the clear. This function supports the splitting of the KEK into key parts, and the generation of and verification by check digits.

`sb_desDeleteKeyEncryptingKey()` deletes the key-encrypting DES key (in the area reserved for the key-encrypting DES key) by overwriting the storage location with zeroes and marks said location as invalid. Only the key-encrypting DES key may be deleted.

`sb_desDeleteKEK()` deletes the key-encrypting DES key in the specified location (in the area reserved for the key-encrypting DES keys) by overwriting the storage location with zeroes and marks said location as invalid.

`sb_desStoreEncryptedKey()` stores an acquirer's DES key in the specified location in the area reserved for acquirer's DES keys. The incoming key is assumed to be encrypted via DES in ECB mode with the key-encrypting key (KEK). If the KEK has not been stored on the card previously via **`sb_desStoreKey()`**, decryption will not occur. If decryption occurs, the resulting key overwrites the specified location and the location is marked as valid.

`sb_desStoreAcquirerKey()` stores an acquirer's DES key in the specified location in the area reserved for acquirer's DES keys. The incoming key is assumed to be encrypted via DES in ECB mode with the key-encrypting key in the specified slot. If the KEK has not been stored on the card previously via **`sb_desStoreKey()`**, decryption will not occur. If decryption occurs, the resulting key overwrites the specified location and the location is marked as valid.

`sb_desDeleteAcquirerKey()` deletes the specified acquirer's DES key (in the area reserved for acquirer's DES keys) by overwriting the storage location with zeroes and marks the location as invalid. Only an acquirer's DES key may be deleted.

`sb_desReEncrypt()` decrypts the input encrypted data with the specified session DES key, encrypts the resulting plain text with the acquirer's DES key, and returns the resulting cypher text. Decryption only occurs if the specified session DES key is valid. After encryption with the acquirer's DES key, the memory location holding the plaintext is overwritten with zeroes.

5.7 Key Exchange

The following functions are available.

`sb_dhGenerateValues()` generates a local and remote value in preparation for the ECC Diffie-Hellman key-exchange. The local and remote values are stored in the locations reserved for their use and the locations are marked as valid.

`sb_dhSharedSecretWithAddInfo()` generates a shared secret employing the local value on the Card, the remote value input by the function, and additional information passed following [5, x7.2.1]. The first eight bytes of the shared secret are stored as a sessional DES key and may be stored only in the area reserved for sessional DES keys. The second eight bytes of the shared secret are stored as a MAC key and may be stored only in the area reserved for MAC keys. The locations are specified in the function's parameters. After storage, the respective key locations are marked as valid.

`sb_dhSharedSecretPATM()` generates a shared secret employing the local value on the Card, the remote value input by the function, and additional information passed following [5, x7.2.1]. The first eight bytes of the shared secret are stored as a sessional DES key and may be stored only in the area reserved for sessional DES keys. The second eight bytes of the shared secret are stored as a MAC key and may be stored only in the area reserved for MAC keys. The locations are specified in the function's parameters. After storage, the respective key locations are marked as valid.

5.8 Random Numbers

The following functions are available.

`sb_rngInitialSeed()` inputs the initial seed to the card. Said seed persists between power cycles.

`sb_rngSeedFIPS186()` seeds the FIPS-based random number generator.

`sb_rngFIPS186Session()` retrieves a byte stream of random numbers generated by the FIPS-based session-key random number generator.

`sb_rngRandomBytes()` retrieves a byte stream of random numbers generated by the hardware random number generator.

5.9 ActivCard Functions

The following functions are available.

`sb_storeEncryptedMTMK()` stores an MTMK triple DES key in the specified location in the area reserved for acquirer's DES keys. The incoming key is assumed to be encrypted via DES in with a key-encrypting key (KEK). If the KEK has not been stored on the card previously via **`sb_desStoreKey()`**, decryption will not occur. If decryption occurs, the resulting key overwrites the specified location.

`sb_storeMTMK()` stores an MTMK triple DES key in the specified location in the area reserved for acquirer's DES keys. The incoming key is passed in the clear.

`sb_deleteMTMKActivCard()` deletes the MTMK in the specified location (in the area reserved for MTMKs) by overwriting the storage with zeros and marking the location as invalid.

`sb_genSessionKey()` generates a MAC session key or a PIN encryption key and an ActivCard key encrypting key. Each session key is encrypted by an ActivCard key encrypting key which in turn is encrypted by the terminal master key or TMK.

`sb_storeEncryptedSessionKey()` stores a MAC session key or a PIN encryption key. A TMK is generated using the input seed and the MTMK in the specified location. The TMK is used to decrypt the input ActiveCard KEK, which is then used to decrypt the input session key. The session key is stored in the specified location in the area reserved for MAC session keys or PIN session keys.

`sb_genSessionMAC()` generates a 4-byte MAC value of the input message using a specified MAC session key.

`sb_updateTMK()` outputs a new TMK and the associated ActivCard KEK. The TMK is encrypted by the ActivCard KEK, which is in turn encrypted by the old TMK. The old and new TMKs are generated using the input MTMKs and seed values.

5.10 Miscellaneous Functions

The following functions are available.

`sb_cardKeyUsage()` returns the usage of the specified key type.

`sb_cardKeyStatus()` indicates if the specified location of the specified key type contains a valid key.

`sb_retrieveErrorCodes()` retrieves errors that have occurred in previous operations.

`sb_cardProductVersion()` retrieves the version strings from various components of the EC Card software and hardware.

The **`Re-program Firmware`** function re-programs the firmware on the EC-Card. Therefore, the firmware can be upgraded without openning the shroud of the EC-Card. The CO's password is verified before this function can begin. The firmware must contain a valid digital signature (DSA) for this operation to be successful. This function is not considered an operator function, so the interface is not documented in user manuals. A separate program is provided to re-program the EC-Card firmware.

Chapter 6: EC Card Cryptographic Algorithms

The cryptographic algorithms used in each API function are listed in the following table.

API Function and Cryptographic Algorithms

Class	***Function***	***FIPS-Approved***	***Others***
Access Control	`sb_login()`	—	—
	`sb_logout()`	—	—
	`sb_changeCryptoOfficerPwd()`	—	—
	`sb_setUserPwd()`	—	—
	`sb_setUserId()`	—	—
	`sb_getRemainingLoginAttempts()`	—	—
Cryptosession	`sb_dataSize()`	—	—
	`sb_initialize()`	—	—
	`sb_end()`	—	—
	`sb_clearAllSessionKeys`	—	—
Backup/Restore	`sb_cardGetKeyShadow()`	—	—
	`sb_cardBackupDataBase()`	SHA-1	TDES CBC
	`sb_cardDesBackupDataBase()`	DES CBC SHA-1	—
	`sb_cardSendKeyShadow()`	—	—
	`sb_cardRestoreDataBase()`	DES CBC SHA-1	TDES CBC
Generate Keys	`sb_genKeyPair()`	—	EC Key Pair Generation
	`sb_keyDeleteKeyPair()`	—	—
	`sb_keyGetPublicKey()`	—	—
MAC Keys	`sb_keyCalculateMAC()`	SHA-1	—
	`sb_keyDeleteMAC()`	—	—

API Function and Cryptographic Algorithms			
Class	***Function***	***FIPS-Approved***	***Others***
DES Keys	`sb_desDeleteKey()`	—	—
	`sb_desDeleteAcquirerKey`	—	—
	`sb_desDeleteKeyEncryptingKey()`	—	—
	`sb_desStoreKEK()`	DES ECB	TDES ECB
	`sb_desDeleteKEK()`	—	—
	`sb_desStoreKey()`	—	—
	`sb_desStoreAquirerKey`	DES ECB	—
	`sb_desStoreEncryptedKey()`	DES ECB	TDES ECB
	`sb_desReEncrypt()`	DES ECB	TDES ECB
Key-exchange	`sb_dhGenerateValues()`	—	EC Key Generation
	`sb_dhSharedSecretWithAddInfo()`	—	EC Diffie-Hellman
	`sb_dhSharedSecretPATM()`	—	EC Diffie-Hellman
Random-Numbers	`sb_rngInitialSeed()`	—	—
	`sb_rngSeedFIPS186()`	FIPS-186	—
	`sb_rngFIPS186Session()`	FIPS-186	—
	`sb_rngRandomBytes()`	—	Hardware RNG
ActivCard Functions	`sb_storeEncrypedMTMK()`	DES ECB	TDES ECB
	`sb_storeMTMK()`	—	—
	`sb_deleteMTMKActiveCard()`	—	—
	`sb_genSessionKey()`	DES ECB	TDES ECB
	`sb_storeEncryptedSessionKey`	DES ECB	TDES ECB
	`sb_genSessionMAC()`	FIPS-113	—
	`sb_updateTMK()`	—	TDES ECB
Miscellaneous	`sb_cardKeyUsage()`	—	—
	`sb_cardKeyStatus(0`	—	—
	`sb_retrieveErrorCodes()`	—	—
	`sb_cardProductVersion()`	—	—
	`Re-program Firmware`	DSS Verification	—

6.1 Random Number Generator

The Card offers three options for RNG

1. **`SB_FIPS_RNG`** - Use the FIPS-186 RNG with user supplied seed. The user supplied seed is mixed with the Initial Seed.
2. **`SB_FIPS_RNG_HW_SEED`** - Use the FIPS-186 RNG with a seed supplied by the hardware RNG on the Card. The hardware seed is mixed with the Initial Seed.
3. **`SB_EXT_RNG`** - Use the hardware RNG on the Card.

The RNG option is specified when the function **`sb_initialize()`** is called. Only options **`SB_FIPS_RNG`** and **`SB_FIPS_RNG_HW_SEED`** are FIPS-140 approved methods for generating random numbers for the crypto-session.

If **`SB_FIPS_RNG_HW_SEED`** is selected, the Card will periodically re-seed the FIPS-186 RNG with a hardware generated random number.

If the FIPS-186 RNG is selected, the function **`sb_rngFIPS186Session()`** can be used to retrieve a random stream. The function **`sb_rngSeedFIPS186()`** seeds the FIPS-186 RNG with the input value.

The function **`sb_rndRandomBytes()`** retrieves a random stream from the hardware RNG, no matter which RNG has been selected for the crypto-session.

6.2 Key Generation

The Card generates two types of keys:

1. EC Key Pairs - The functions **`sb_genKeyPart()`** and **`sb_dhGenerateValues()`** generate EC key pairs.
2. DES keys - The functions **`sb_cardBackupDataBase()`**, **`sb_cardDesBackupDataBase()`**, **`sb_genSessionKey()`** and **`sb_updateTMK()`** generates DES keys. The DES keys are random stream output by the RNG selected for the crypto-session. For FIPS-140 conformance, options **`SB_FIPS_RNG`** or **`SB_FIPS_RNG_HW_SEED`** must be selected when the crypto-session is initialized.

6.3 Key Agreement

The functions **`sb_dhSharedSecretWithAddInfo()`** and **`sb_dhSharedSecretPATM()`** generates shared secret using the EC Diffie-Hellman algorithm. The PIN encryption key and MAC key for the PATM appliance are derived from this shared secret. FIPS-140 does not specify any key agreement method.

6.4 Message Digest

The only message digest algorithm performed by the Card is SHA-1. SHA-1 is a FIPS-140 approved algorithm.

The function **sb_keyCalculateMAC()** computes a MAC by combining the input message with the MAC key, and computing the SHA-1 digest of the result.

The functions **sb_cardBackupDataBase()**, **sb_cardDesBackupDataBase()** and **sb_cardRestoreDataBase()** computes the SHA-1 digest of each block of backup data to ensure that the data have not been corrupted.

6.5 Encryption/ Decryption

The Card performs encryption in four modes: DES ECB, DES CBC, triple DES ECB and triple DES CBC. Of these modes, only DES ECB and DES CBC are FIPS validated. However, the triple DES modes are allowed for use within the US and Canadian governments in a FIPS mode of operation.

The function **sb_cardBackupDataBase()** encrypts the key database in triple DES ECB mode, while **sb_cardDesBackupDataBase()** encrypts it in DES ECB mode.

The function **sb_desStoreAcquirerKey()** performs decryption in DES ECB mode.

The functions **sb_desStoreKEK()**, **sb_desStoreEncryptedKey()**, **sb_desReEncrypt()**, **sb_storeEncryptedMTMK()**and **sb_genSessionMAC()** performs encryption/decryption in DES ECB or triple DES ECB mode, depending on the length of the input key.

The functions **sb_storeEncryptedSessionKey()**, **sb_genSessionKey()**,and **sb_updateTMK()** performs encryption and decryption in triple DES ECB mode.

6.6 Signature Verification

The firmware contains DSS signatures which are verified when the Card powers up. Signatures of new firmware are also verified before they are programmed into FLASH. DSS signature verification is a FIPS-140 approved algorithm. The Card does not generate any digital signatures.

Appendix A: System Overview

The contents of this chapter are informative only and describe the manner in which the EC Card may be employed.

	P-ATM		EC CARD		Acquirer
1.			Generate Remote Key		
2.		←	Inject key		
3.				←	Transmit KEK
4.			Store KEK into Card		
5.				←	Send encrypted key
6.			Store encrypted key		
7.	Generate Remote Key	→			
8.	Transmit key				
9.	Compute shared secret =DES+MAC		Compute shared secret =DES+MAC		
10.	Send encrypted PIN	→	Re-encrypt PIN	→	
11.	Calculate MAC				
12.	Transmit	→	Verify		

Bibliography

[1] *ECC Card Project API Definition*, Rev. 1.23, Document No. 037, Certicom Corp., 13 November 1997.

[2] Certicom Corp., *Security Builder Programmer's Reference Manual*, Release 1.1, Doc. No. SBPR-9706-0001, 1997.

[3] FIPS PUB 180-1, *Secure Hash Standard*, 17 April 1995.

[4] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, ISBN 0-8493-8523-7, 1997.

[5] IEEE P1363, *Standard Specifications for Public Key Cryptography*, Draft Standard, 22 August 1996.