Interface specification

for securing files with

FLAM®

Hybrid method with a random key per FLAMFILE® and a static symmetric transport key protected by a hardware security module (HSM)

Version 1.3

06.05.2015

Table of content

1 Introduction....1
2 System model....2
3 Specification FLAM Key Management CONTEXT (FKMC)....4
3.1 Input requirements of FLAM®....4
3.2 The data structure in version 001....4
3.2.1 Explanation of the data fields....5
3.2.1.1 Field 1: Info data....5
3.2.1.2 Field 2: Generation and version of the FMKY....6
3.2.1.3 Field 3: KTV for the FMKY....6
3.2.1.4 Field 4: Creation time of the FLAMFILE®....6
3.2.1.5 Field 5: Random number for dynamization....7
3.2.1.6 Field 6: Encrypted FKEY....7
3.2.1.7 Field 7: Hash of field 4, field 5 and the clear FKEY....7
4 Specification FLAM® Key Management EXIT (FKME)....8
4.1 The function interface....8
4.1.1 Explanation of the parameters....9
4.1.1.1 Para 1: Function code....9
4.1.1.2 Para 2: Return code....9
4.1.1.3 Para 3: Length of input parameters....9
4.1.1.4 Para 4: Input parameters in version 001....10
4.1.1.5 Para 5: Length of context data (FKMC) in version 001....10
4.1.1.6 Para 6: Context data (FKMC) in version 001....11
4.1.1.7 Para 7: Key (FKEY) length in version 001....11
4.1.1.8 Para 8: Key (FKEY) in version 001....11
4.1.1.9 Para 9: Message length....11
4.1.1.10 Para 10: Message....11
4.2 Proceedings in version 001....11
4.2.1 Encryption of a FLAMFILE®....11
4.2.2 Rekeying of a FLAMFILE®....12
4.2.3 Decryption of a FLAMFILE®....14

5 Appendix 15
5.1 FKME interface 15
5.1.1 FKME for mainframe in Assembler 15
5.1.2 FKME for other platforms in ANSI C 16
5.2 FLAM interface 17
5.2.1 FLAM(UP) for mainframe 17
5.2.1.1 Calling FLAMUP with ‚MSGFILE=filename, PARFILE=filename' as parameter 20
5.2.1.2 Example for the FLAMUP call in COBOL: 20
5.2.1.3 Example for the FLAMUP call in ASSEMBLER: 21
5.2.1.4 Example for the FLAMUP call in C++: 22
5.2.2 FLAM(UP) for other platforms 23
5.2.2.1 The parameters for the FLAM-Key-Management-Exit (FKME) 25
5.2.2.2 Example call of FLAMUP 25
5.3 FLAM implementation recommendations 26
5.3.1 Handling of generation and version 27
5.3.2 Passing the input parameters to the EXIT via FLAM / FLAMUP 27
5.3.3 The last 10 bytes of the info field of the FKMC 28
5.3.4 EBCDIC and ASCII conversion 28
5.3.5 Result messages 28
5.3.5.1 Error messages of the Exit 28
5.3.5.2 OK messages from the Exit 30
5.3.5.3 Information about the Exit itself 30
6 List of abbreviations 31

Table of figures

Figure 1 Overview 1

Table of tables

Table 1 FLAM® Key Management CONTEXT (FKMC) 4

Table 2 FLAM® Key Management EXIT (FKME)....8

Table 3 Parameters for encryption....11

Table 4 Parameters for rekeying....13

Table 5 Parameters for decryption....14

Version history

Version	Status	Date	Editor	Changes
00.10 00.30	In Progress	07.11.2005 09.11.2005	F. Reichbott	- Document creation
00.40	In Progress	16.11.2005	F. Reichbott	- Added version history
00.50	In Progress	05.12.2005	F. Reichbott	- Change of hash for key test from MDC2 to RipeMD160 - Version of FLAM® with FKME
00.60	In Progress	06.12.2005	F. Reichbott	- Added implementation recommendations section to appendix - Added FLAMUP interface to appendix
00.70	In Progress	09.12.2005	F.Reichbott	- minor fixes - MSGFILE needs to be specified first - Call of FLAM(UP) for securing files
00.80	In Progress	27.12.2005	F. Reichbott	- Change of hash for key test from RipeMD160 to SHA1
01.00	Approved	23.01.2006	F. Reichbott	- no significant changes
01.10	Approved	21.02.2006	F. Reichbott	- Minor changes to error messages - Timestamp defined as GMT - Changed generation and version from POV to BIN - Defined message upon successful execution
1.20	Approved	13.03.2006	F. Reichbott	- Fixed grammar of error messages - Fixed mistakes in example on page 3
1.30	Approved	06.05.2015	F. Reichbott T. Eckert	- Review and English translation

1 Introduction

This document describes how the cryptographic protection of files for storing with an entity (archive, logs, ...) and for transmission between entities (file transfer) is supposed to work through the FLAM® Key Management EXIT (FKME). For this purpose, a hybrid technique is specified which allows to generate a random key (FKEY) per FLAMFILE® using a Hardware Security Module (HSM) and to securely exchange this file-specific key using a FLAM® master key (FMKY) from the HSM. To ensure the secure transmission of the FLAM® key, the FLAM® Key Management CONTEXT (FKMC) in the user header of every FLAMFILE® is used. Established methods of key management are used for the provision of the FMKY. The following figure illustrates the relationships.

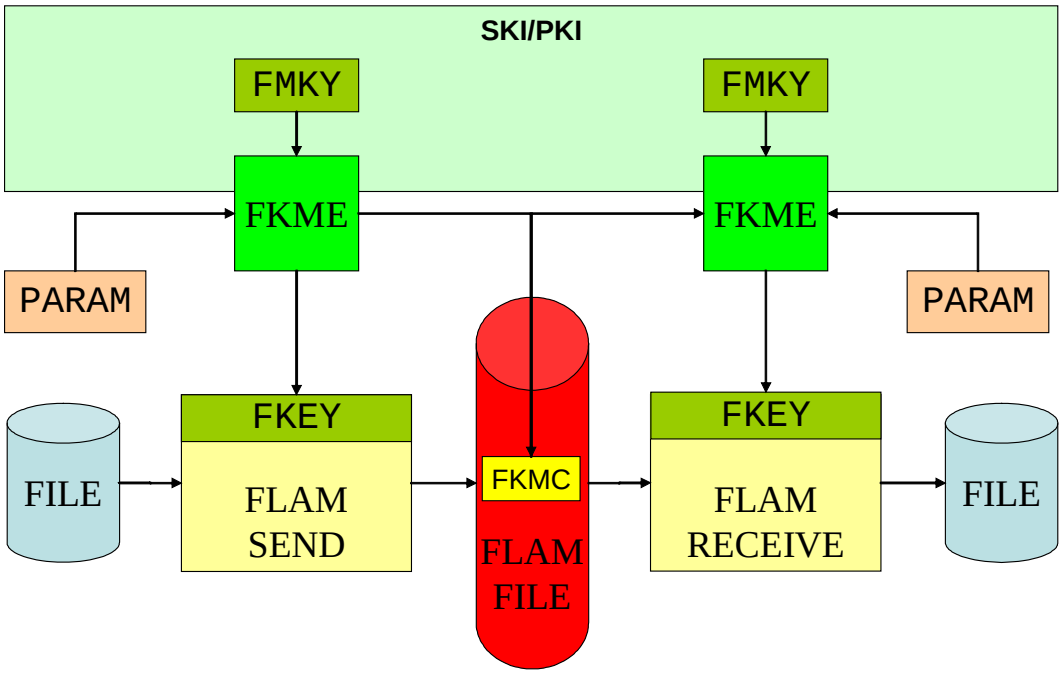

Figure 1 Overview

This specification is intended to fulfill, among other things, the requirements of the "Payment Card Industry Data Security Standard" (PCI DSS) and other requirements catalogs through FLAM®. In this respect, secure storage and archiving of files, the exchange of these files between parties as well as securing logs, dumps, database backups and other files that occur in the transaction processing plays a role.

Furthermore, this infrastructure should be usable for the secure exchange of payment data, like it has been possible with FLAM® in the past. There are also other applications where confidentiality and integrity of files must be ensured.

2 System model

FLAM® with AES (as of version 4.0) realizes integrity protection and confidentiality through a 64-byte key that is used for the derivation of four 128-bit AES keys on each of the three logical layers of a FLAMFILE® using a one-way hash function. They are used for the encryption of the user data (segment) and the calculation of Message Authentication Codes (MACs) for the respective components on each level (segment, member, file) of the FLAMFILE®.

In order to use the cleartext FLAM® key (FKEY) securely within a HSM-based cryptographic infrastructure, two requirements must be met:

1. The cleartext FKEY may exist in system memory (RAM) only for a brief period of time.

2. The FKEY may be used only for one single FLAMFILE® so that extracting its value from system memory by an attacker does not affect another FLAMFILE®.

In consequence, the FKEY only exists where the file's cleartext data also exists which is all that an attacker can get by obtaining the respective FKEY. Especially, he would not get access to other FLAMFILEs®. The provision of the FKEY occurs through the FLAM® Key Management EXIT (FKME), which is supported **since version 4.1 of FLAM®** for the integration of various cryptographic infrastructures.

The FKEY must consist of 64 bytes of randomness which must be generated by the respective HSM. After that, the HSM provides a structure protected by the hardware (FLAM Key Management CONTEXT (FKMC)) for the user header of the FLAMFILE®. This allows the respective personalized HSM to restore the FKEY in its cleartext form for reading.

To protect the FKMC, a static, symmetrical master key is used for FLAM® (FMKY), which is used by the sender to encrypt the FKEY and used by the receiver to decrypt it. In order to not risk the HSMs security, the cleartext FKEY looks like 64 bytes of normal data from the HSM's point of view.

For the different use cases (ENCIPHER and DECIPHER), directed key relationships can be created through established Static Key Management Systems (SKMS) which conform to all requirements of PCI DSS. These allow to prove the origin of a FLAMFILE® distinctively. Intermediaries, "Kopfstellen" (central payment gateways) and other ownership transitions can be conducted safely through rekeying of the data (IDATXLAT and ODATXLAT). For this, only the FKEY needs to be rekeyed by the HSM and the user header of the FLAMFILE® must be exchanged. The actual data remains encrypted during this process at all times. The FKEY remains protected by the HSM throughout. This process for secure rekeying of a FLAMFILE® makes sure that the data needs to exist in cleartext only on its creation and extraction. This circumstance is illustrated in the following example of a generating office (GS) for card production and a personalization system of the card producer:

- Secure storage of the generated data within the generating office
  `FLAM(COMP, FMKY.GS2GS.ENCIPHER.ggvv)`

- Transfer of the card data to the producer
  `FLAM(CHNG, FMKY.GS2GS.IDATXLAT.ggvv, FMKY.GS2PS.ODATXLAT.ggvv)`

- Reception and secure storage at the producer
  `FLAM(CHNG, FMKY.GS2PS.IDATXLAT.ggvv, FMKY.PS2PS.ODATXLAT.ggvv)`

- Decryption of the stored data for production
  `FLAM(DECO, FMKY.PS2PS.DECIPHER.ggvv)`

Furthermore, rekeying allows to transition from one generation and version to another of the same or a different FMKY. It is also possible to migrate between different cipher suites if this migration path is supported by the EXIT.

The generation and exchange of the FMKY between entities should be carried out according to the established procedures for static key management (at least two-tiered (2 or 3 clear key components + FMKY as cryptogram)). This is not part of this interface specification. The requirements applying to static key management in the respective environment (VISA, ZKA, BSI, …) must be met. This document assumes that the static FMKY is available through the HSM. The keys, like any static key, should be replaced frequently and should be replaceable ad-hoc. Therefore, for every key, there is a generation (GG) and a version (VV).

3 Specification FLAM Key Management CONTEXT (FKMC)

3.1 Input requirements of FLAM®

The context field (FKMC) must not exceed 512 bytes on all platforms and must have a constant length for the successful rekeying of FLAMFILES®. There are no requirements regarding the contents, except that the first 50 bytes are logged as information.

Note: If more than 512 bytes are needed, a new version of FLAM with larger buffers is required.

3.2 The data structure in version 001

The following table is an overview of the data structure. Used abbreviations and their meaning:

Lg = Length in bytes

CHR = Character in the character set of the creating machine
POV = BCD-encoded, right-aligned and padded with 0
BIN = Binary

Field	Description	Content	Lg	Format
1	Info data	FKMC V002 L144 AES3 KL32 EZ04 ICBC SHA2	50	CHR
2	Generation and version of the FMKY	GGVV	2	BIN
3	KTV for the FMKY	RRRRRRRR	4	BIN
4*	Creation time (GMT) of the FLAMFILE®	YYYYMMDDHHMMSSss	8	POV
5*	Random number for dynamization (if needed: prevention of replay attacks)	RRRRRRRRRRRRRRR	8	BIN
6**	Encrypted FKEY	RRR…RRR	64	BIN
7*	HASH of 4, 5 and the cleartext FKEY	RRRRRRRRRRRRRRR	8	BIN

Table 1 FLAM® Key Management CONTEXT (FKMC)

*) The fields are set at creation time of the FLAMFILE(*) and not changed later.

**) The cleartext FKEY is also generated at creation time and remains the same. Only its ciphered version changes depending on the FMKY.

3.2.1 Explanation of the data fields

3.2.1.1 Field 1: Info data

The info data is contained within the first 50 bytes of the context field, providing information about the EXIT itself. The following mandatory details must be provided in uppercase and separated by whitespace characters.

1. **Identifier for the FKMC** = FKMC
   4 bytes long constant for checking and detecting the character set (EBCDIC or ASCII).

2. **Version of the EXIT** = V001
   The version of the EXIT is defined as the constant V001. This information is used to differentiate between later implementations.

3. **Length of the EXIT** = L144
   The length of the context field of this EXIT is 144 bytes. The length may vary depending the version and is used for plausibility checking.

4. **Algorithm for the FMKY** = TDES
   This field defines the algorithm used to encrypt the FKEY using the FMKY. In version 001, this is Triple DES.

5. **Key length of the FMKY** = KL16
   The length of the FMKY is defined by the fifth field which is 112 bits. This corresponds to a double length TDES Key with 16 bytes.

6. **Verification algorithm for the FMKY** = EZ04
   Defines the algorithm used for calculation of the Key-Test-Values (KTV) for the FMKY, which is realized by the method Enc-Zeros. Only the four high-order bytes are stored, however, which matches the VISA Key-Test-Procedure.

7. **Cipher mode for the FKEY** = ICBC
   The seventh field specifies the mode of encryption of the FKEY. ICBC stands for Cipher-Block-Chaining (CBC) with an initialization vector.

8. **One-way hash algorithm** = SHA1
   The algorithm used to hash the timestamp, random number and the cleartext FLAM® key is SHA-1.

9. **Further information**
   Further application-specific information (e.g. the implementation of the EXIT (IBMCCA, IBMDKMS, PKCS11, ATALLA, THALES, …)) may be defined. If no information is present, the remaining 10 bytes are padded with whitespace.

To verify the implementation of the EXIT against the info data, in version 001, the first 40 bytes are compared with the following constant (note the whitespace at the end):

„FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 “

It is important that the character set used for the comparison must not play a role. The determination of the character set should be performed on the first byte of info data, which must be ‚C6’hex in EBCDIC and ‚46’hex in ASCII for a capital ‚F’.

3.2.1.2 Field 2: Generation and version of the FMKY

This field is set to the generation (gg) and version (vv) of the FMKY when sending. It is used to select the correct FLAM master key (FMKY) when receiving the FLAMFILE®. Determination of generation and version when sending and their usage when receiving depends on the implementation of the EXIT. In general, a fully-qualified label for the key and the matching generation and version should be passed to the EXIT when sending. When receiving, it is then sufficient if the label contains placeholders for generation and version.

3.2.1.3 Field 3: KTV for the FMKY

The Key-Test-Value (KTV) for the FMKY is the result of encrypting 8 bytes of zeros with TDES (EDE) using the FMKY. However, only the 4 high-order (leftmost) bytes are stored. This algorithm was chosen because it is supported by most HSMs, matches the security requirements, can also be generated by the sender by encrypting 8 data bytes and can be implemented using TDES as base algorithm.

3.2.1.4 Field 4: Creation time of the FLAMFILE®

Based on the requirements, the point in time of creation of the FLAMFILE® must be recorded cryptographically secure. This happens by including it in the HASH calculation and the generation of the IV for CBC mode encryption. As indicated in the table above, the format consists of:

- YYYY - Year
- MM - Month
- DD - Day
- HH - Hour
- MM - Minute
- SS - Second
- ss - Millisecond

It is desirable to use a reliable time source for these 8 bytes. The time zone is Greenwich Mean Time (GMT) in order to allow international conversions.

3.2.1.5 Field 5: Random number for dynamization

The 8 bytes long random number serves as attribute for dynamization of a FLAMFILE® and is used for HASH calculation and IV generation.

3.2.1.6 Field 6: Encrypted FKEY

In version 001, the 64 bytes long random FKEY (8 blocks) is TDES-encrypted in CBC mode using the FMKY. The IV is the result of XORing fields 4 and 5. The result is a ciphertext for the FKEY of 64 bytes which is put into the context structure in binary form by the EXIT.

3.2.1.7 Field 7: Hash of field 4, field 5 and the clear FKEY

Calculation of the hash value occurs by using the SHA-1 algorithm on the following 80 bytes:

- 8 bytes of field 4: Creation time of the FLAMFILE®
- 8 bytes of field 5: Random number for dynamization
- 64 bytes random FKEY in cleartext

Only the 8 high-order (leftmost) bytes of the hash value are stored in the context structure. It is calculated when the key is generated. The verification should only happen when the key is used. In other words, verification is only done when the cleartext key is necessary. This is not the case when a FLAMFILE® is rekeyed.

4 Specification FLAM® Key Management EXIT (FKME)

The FLAM® Key Management EXIT (FKME) is a function that is called by FLAM® when corresponding parameters (PARAM) for an EXIT are passed to FLAM®. It allows the inclusion of an arbitrary cryptographic infrastructure to enable implementing session key procedures with FLAM®.

4.1 The function interface

The following table provides an abstract overview of the parameters of the interface. Used abbreviations:

INT = 32 Bit (4 Byte) in two's complement
STR = Array of bytes

Para	Name	Comment	Direction	Type	Length
1	Fuco	Function code	INPUT	INT	4 bytes
2	RetCo	Return code	OUTPUT	INT	4 bytes
3	ParLen	Length of input parameters	INPUT	INT	4 bytes
4	Param	Input parameters	INPUT	STR	variable
5	DatLen	Length of the data (FKMC)	COMP:INPUT/OUTPUT DECO:INPUT/OUTPUT CHNG:INPUT=OUTPUT	INT	4 bytes
6	Data	Data (FKMC)	COMP:OUTPUT DECO:INPUT CHNG:INPUT/OUTPUT	STR	variable
7	KeyLen	Length of the key (FKEY)	INPUT/OUTPUT CHNG: not used	INT	4 bytes
8	Key	Key (FKEY)	OUTPUT CHNG: not used	STR	variable
9	MsgLen	Length of the message	INPUT/OUTPUT	INT	4 bytes
10	Message	Message	OUTPUT	STR	variable

Table 2 FLAM® Key Management EXIT (FKME)

All parameters are passed as pointers (Call by Reference). FLAM® loads the FKME dynamically at runtime as module with a single function. The name of the module and (if applicable) the name of the function can be passed to FLAM® as parameters. The input lengths always specify the length of the data passed in the next parameter. The output lengths will be set to the length of the required memory after function execution.

4.1.1 Explanation of the parameters

4.1.1.1 Para 1: Function code

The function code is used to select the method for the EXIT. The following methods are defined:

- **0 – Decompression/Decryption**
  The input parameters and the context field (FKMC) from the user header are passed to the EXIT which returns the cleartext key (FKEY) to FLAM.

- **1 – Compression/Encryption**
  The input parameters are passed to the EXIT which then randomly generates the 64 bytes FKEY and returns it together with the context field (FKMC). FLAM stores the context field inside the user header and uses the generated key for creating the FLAMFILE®.

- **2 – Change/Rekeying**
  The input parameters and the context field (FKMC) from the user header are passed to the EXIT which calculates and returns the new context field (FKMC) to the program. The program then creates a new FLAMFILE® with a modified user header. The memory block must be large enough to be able to hold the newly generated context field.

- **0xFFFFFFFF – Information**
  Prompts the EXIT to output information about itself into the message field.

4.1.1.2 Para 2: Return code

The return code is used for program control of FLAM. If the return code equals 0 after the call to the EXIT, the call was successful, and FLAM continues execution as expected. If it equals 4, the amount of memory provided to the EXIT was insufficient. This leads to another call with sufficient memory for platforms with dynamic memory management. On other platforms, this error, like all other error codes larger than 0, result in an abort of FLAM. The reason for such an abort can be communicated to the outside by the EXIT only through the message field. It is always output when the length of the message is not equal to 0. This allows the EXIT to communicate warnings even if the return code is 0. If everything was ok, this fact, the function (COMP, CHNG, DECO, RENW), the timestamp and the random number are logged through the message field.

4.1.1.3 Para 3: Length of input parameters

The length of the input parameters depends on the method and the implementation of the EXIT. It is determined by FLAM® when called and passed to the EXIT.

4.1.1.4 Para 4: Input parameters in version 001

The input parameters depend on the method and the implementation of the EXIT. Basically, they should include the following information depending on the method:

- **Encryption**
    - If necessary, identification data and authentication data for the HSM
    - Referencing data for the FLAM-Master-Key (FMKY)
        - Key label, generation and version *or*
        - Key label and key label template
- **Decryption**
    - If necessary, identification data and authentication data for the HSM
    - Referencing data for the FLAM-Master-Key (FMKY)
        - Key label template
- **Rekeying**
    - If necessary, identification data and authentication data for the HSM
    - Referencing data for the FLAM-Master-Key (FMKY) incoming
        - Key label template
    - Referencing data for the FLAM-Master-Key (FMKY) outgoing
        - Key label, generation and version *or*
        - Key label and key label template

When calling FLAM, the maximum buffer size for input parameters currently is 256 bytes.

4.1.1.5 Para 5: Length of context data (FKMC) in version 001

The length of the context data (FKMC) is 144 bytes. The size of the buffer is lower than the 512 bytes that are provided by FLAM. Hence, there should always be enough memory available for the context field. On the output side, the EXIT must set this parameter to 144 and the field must be checked to be equal to 144 if a context field is passed by FLAM® (decryption or rekeying).

4.1.1.6 Para 6: Context data (FKMC) in version 001

The context field (FKMC) is provided or accepted by the EXIT through this field. Its specification can be found in section 3.2.

4.1.1.7 Para 7: Key (FKEY) length in version 001

If returned by the EXIT (encryption or decryption), the length of the random cleartext key is always 64 bytes, which are provided by FLAM® on the input side.

4.1.1.8 Para 8: Key (FKEY) in version 001

The EXIT provides the 64 bytes long key (random number) to FLAM® for encryption and decryption through this parameter.

4.1.1.9 Para 9: Message length

FLAM® provides a message buffer of size 128 bytes. It can be used by the EXIT to let FLAM® log error messages or other information.

4.1.1.10 Para 10: Message

Contains the message which is output if the length does not equal 0. If the function code is ‚FFFFFFFFFF'hex, the EXIT returns its info data through this parameter.

4.2 Proceedings in version 001

4.2.1 Encryption of a FLAMFILE®

FLAM® calls the EXIT with the following parameters and the EXIT returns the following values upon success:

Para	Input values	Output values
Fuco	1	=
RetCo	0	0
ParLen	>0	=
Param	Parameter for the EXIT	=
DatLen	512	144
Data	undefined	FKMC as per 3.2
KeyLen	64	64
Key	undefined	64 bytes of random data
MsgLen	128	Length of the OK message
Message	undefined	OK message

Table 3 Parameters for encryption

Below, the encryption procedure (Fuco=1) is set out in bullet points.

- Check of buffer lengths
- Determination of identification and authentication data for the HSM from the parameter field
- Determination of the label, generation and version for the key from the parameter field
- Initialization of the context structure with the information data
- Setting the generation and version in the context field
- Calculation of the KTV for the FMKY based on the label and entering it into the context field
- Determination of the timestamp and setting it in the context field
- Generation of the random number and setting it in the context field
- Construction of the IV from timestamp and random number
- Generation of the 64 bytes long key as return value to FLAM®
- Calculation of the SHA-256 over the timestamp, the random number and the cleartext key and setting it in the context field
- CBC encryption of the key using the IV and the FMKY, putting the result into the context field
- Setting the buffer lengths, ok message and the return code
- Return from the EXIT

4.2.2 Rekeying of a FLAMFILE®

The program calls the EXIT with the following parameters. On successful execution, the EXIT returns the following values:

Para	Input values	Output values
Fuco	2	=
RetCo	0	0
ParLen	>0	=
Param	Parameter for the EXIT	=
DatLen	144	144
Data	FKMC as per 3.2	FKMC as per 3.2
KeyLen	undefined	=
Key	undefined	=
MsgLen	128	Length of the OK message
Message	undefined	OK message

Table 4 Parameters for rekeying

Below, the rekeying procedure (Fuco=2) is set out in bullet points.

- Check of buffer lengths
- Determination of identification and authentication data for the HSM from the parameter field
- Determination of the input template for the input key from the parameter field
- Determination of the output label, output generation and output version for the output key from the parameter field
- Determination of the input version and input generation from the context field
- Determination of the fully qualified input label from template, generation and version
- Verification of the KTV through the input label
- Setting the output generation and version in the context field
- Calculation of the KTV for the output FMKY based on output label and setting it the context field
- Calculation of the IV as XOR of timestamp and random number
- Rekeying of the 64 bytes long key from the input FMKY to the output FMKY using the IV; replacement of the ciphered key in the context field
- Setting the buffer lengths, ok message and the return code
- Return from the EXIT

4.2.3 Decryption of a FLAMFILE®

FLAM® calls the EXIT with the following parameters. On successful execution, the EXIT returns the following values:

Para	Input values	Output values
Fuco	0	=
RetCo	0	0
ParLen	>0	=
Param	Parameters for the EXIT	=
DatLen	144	=
Data	FKMC as per 3.2	=
KeyLen	64	64
Key	undefined	64 bytes decrypted random data from FKMC
MsgLen	128	Length of the OK message
Message	undefined	OK message

Table 5 Parameters for decryption

Below, the decryption procedure (Fuco=0) is set out in bullet points.

- Check of buffer lengths
- Determination of identification and authentication data for the HSM from the parameter field
- Determination of the input template for the static key from the parameter field
- Determination of the version and generation from the context field
- Determination of the fully qualified label from template, generation and version
- Verification of the KTV through the label
- Calculation of the IV as XOR of timestamp and random number
- CBC decryption of the 64 bytes long key using the IV and the FMKY, returning the result to FLAM® through the key parameter
- Calculation of the SHA-1 over the timestamp, the random number and the cleartext key and comparison with the value from the context field
- Setting the buffer lengths, ok message and the return code
- Return from the EXIT

5 Appendix

5.1 FKME interface

Below are some excerpts from the manual / specification.

5.1.1 FKME for mainframe in Assembler

This user exit is an interface to a special (e.g. user written) key management system. This user routine's job is to provide a key for the encryption or decryption of a FLAMFILE. It can be used in FLAM as well as in FLAMUP. The exit is activated via the parameter KMEXIT=<name> and is then called for every FLAMFILE. When called by FLAM, the parameter KMPARM (up to 256 bytes) is passed to the EXIT. When encrypting, the EXIT may return a 512 bytes long data string which is stored in the FLAMFILE (as user header, see function FLMPUH). On decryption, this data is passed back to the EXIT by FLAM. The first 50 bytes of the data string are displayed in the log as comment (FLM0487 USER HEADER: ...) on both, encryption and decryption.

Name: free choice (max. 8 characters)

Register usage:

→ **R1:** Address of the parameter list
→ **R13:** Points to save area (18 words)
→ **R14:** Contains return address
→ **R15:** Contains call address

Parameter list:

1 → **FUCO** **F** Function code
= 0 Decryption
= 1 Encryption

2 ← **RETCO** **F** Return code
= 0 No error
= else Error(s) detected

3 → **PARMLEN** **F** Length of parameter (up to 256 byte)

4 → **PARAM** **XLn** Parameter

5 ↔ **DATALEN** **F** Length of data
Decryption:
→ Length of data
Encryption:
→ Buffer length of field DATA (512)
← Length of returned data (max. 512)

6 ↔ DATA **XLn** Data (of length of DATALEN)

7 ↔ CKYLEN **F** Key length
→ Size of key buffer (field CRYPTOKEY) (64)
← Length of key (up to 64)

8 ← CRYPTOKEY **XLn** Key (of length of CKYLEN)

9 ↔ MSGLEN **F** Message length
→ Size of message buffer (field MESSAGE) (128)
← Length of returned message (max. 128)

10 ← MESSAGE **CLn** Message (of length MSGLEN)

If a message is returned (MSGLEN > 0), it is sent to the protocol (FLM0445 ...). The data DATA is stored in the user header of the FLAMFILE as-is. If special security is required, the EXIT has to take care of it. Usage of this exit overrules the parameters COMMENT and CRYPTOKEY, if present.

The returned key is not sent to the protocol.

The EXIT is only called once for each FLAMFILE. So, if multiple files compressed into a Group-FLAMFILE (C,FLAMIN=user.*), the EXIT is called once at the beginning. When reading multiple FLAMFILEs (D,FLAMFILE=user.*.aes), the EXIT is called for each opened FLAMFILE. Concatenated FLAMFILEs are treated as one FLAMFILE!

Note: Look for an example in FLAM.SRCLIB(KMXSAMPL).

5.1.2 FKME for other platforms in ANSI C

```
#ifndef FlamKme_h
#define FlamKme_h

#ifndef STDCALL
  #ifdef WINDOWS
    #define STDCALL __stdcall
  #else
    #define STDCALL
  #endif
#endif

#ifndef DECLSPEC
  #define DECLSPEC __declspec( dllimport )
#endif

#ifndef U32
 #if ( sizeof(long) == 4 )
 typedef unsigned long  U32;
// typedef unsigned int  U32
 #endif
#endif

#ifndef I32
 #if ( sizeof(long) == 4 )
 typedef long  I32;
```

```
// typedef int  I32
  #endif
#endif

typedef unsigned char  U8;

#if ( sizeof( U32 ) != 4 ) || ( sizeof( I32 ) != 4 )
 #error wrong type U32/I32
#endif


const U32 FLAMKME_FUCO_DECO     = 0; // Decipher (Decompression)
const U32 FLAMKME_FUCO_COMP     = 1; // Encipher (Compression)
const U32 FLAMKME_FUCO_CHNG     = 2; // Translate  (for future use)
const U32 FLAMKME_FUCO_INFO     = 0xFFFFFFFF; // Information (Version, ...)

const I32 FLAMKME_ERROR_SUCCESS       = 0; // OK
const I32 FLAMKME_ERROR_SIZE          = 4; // Buffer to small
const I32 FLAMKME_ERROR_ABORT         = 8; // > 4 Ábort

#ifdef __cplusplus
extern "C" {
#endif
  DECLSPEC void STDCALL flamkme(
    const U32  * Fuco,        // Function code
    I32        * Retco,       // Return code
    const U32  * ParLen,      // Length of parameter list
    const U8   * Param,       // Parameter list  TOKENID:USERPIN:OBJECTNAME (PKCS11)
                              //                 USERID:PASSPHRASE:LABEL (CCA)
    U32        * DatLen,      // FLAMKME_FUCO_COMP: before call -> buffer length
                              //                    after call -> required bytes
                              // FLAMKME_FUCO_DECO: Amount of bytes in Data
                              // FLAMKME_FUCO_CHNG: Amount of bytes in Data (Input = Output)
    U8         * Data,        // Data for/from FLAM-Header (BLOB)
    U32        * Keylen,      // Length of key data: before call -> buffer length (64)
                              //                     after call -> required bytes (64)
    U8         * Key,         // FLAMKME_FUCO_DECO/COMP: Session Key (64 bytes random data)
                              // FLAMKME_FUCO_CHNG: KeyLen and Key is not used
    U32        * MsgLen,      // Length of the Message: before call -> buffer length
                              //                        after call -> required bytes
    U8         * Message      // Message
                             // if MsgLen!=0 then log msg
    );

#ifdef __cplusplus
}
#endif

#endif
```

5.2 FLAM interface

Below are some excerpts from the manual / specification. Further information can be found in the manual for FLAM®. The following sections relate to FLAM® and the FLAMUP, which support encryption and decryption of files. The functionality for Rekeying of FLAMFILES® is implemented by a new tool to allow differentiation between end nodes and intermediate nodes. This new tool will behave similar to FLAM(UP).

5.2.1 FLAM(UP) for mainframe

FLAMUP can be called as subprogram by other programs (like FLAM itself calls FLAMUP). FLAMUP is a file interface (in contrast to the record interface FLAMREC), i.e. whole files are processed, not just records.

The interface convention conforms to the Assembler / Cobol subprogram interface. In C, this is called OS convention.

Below, the interface is described using the ASSEMBLER language. The following table shows, how the different data types must be defined in COBOL and FORTRAN.

Assembler	Cobol	Fortran	Meaning
F	PIC S9 (8) COMP SYNC	INTEGER*4	Aligned Fullword
H	PIC S9 (4) COMP SYNC	INTEGER*2	Aligned Halfword
CL*n*	PIC X (*n*) USAGE DISPLAY	CHARACTER**n*	*n* printable characters
XL*n*	PIC X (*n*)	CHARACTER**n*	*n* binary characters

D

The arrows define the direction of data flow:

→	the field must be filled by the calling program
←	the field is filled by the called program
↔	the field is filled by the calling program as well as by the called program

Register usage:

→ R1: Address of the parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameters:

1 ← **FILEID** **F** Identification

2 ← **RETCO** **F** Return code
= 0 No error

Some common error codes

=	**1**	Records truncated
=	**10**	File is not a FLAMFILE
=	**11**	FLAMFILE format error
=	**12**	Record length error
=	**13**	File length error
=	**14**	Checksum error
=	**21**	Invalid size of matrix buffer
=	**22**	Invalid compression mode
=	**23**	Invalid code in FLAMFILE
=	**24**	Invalid MAXRECORDS parameter
=	**25**	Invalid record length MAXSIZE
=	**29**	Password error
=	**30**	FLAMFILE is empty
=	**31**	FLAMFILE is not assigned
=	**33**	Invalid file type
=	**34**	Invalid record format
=	**35**	Invalid record length
=	**36**	Invalid block length
=	**37**	Invalid key position (not 1)
=	**38**	Invalid key length
=	**39**	Invalid file name
=	**x'Exxxxxxx'**	I/O error for original file input on input
=	**x'Axxxxxxx'**	I/O error for original file input on output
=	**x'Fxxxxxxx'**	I/O error for compressed file
=	**x'Cxxxxxxx'**	I/O error for parameter file
=	**x'Dxxxxxxx'**	I/O error for message file
=	**x'xFxxxxxx'**	Error in data management (VSAM)
=	**40**	Module or table cannot be loaded
=	**41**	Module cannot be called
=	**42**	Module cannot be unloaded
=	**43-49**	Abort by user exit
=	**52**	Too many or invalid keys
=	**57 - 79**	FLAM error
=	**80**	Syntax error during parameter input
=	**81**	Unknown parameter (keyword)
=	**82**	Unknown parameter value
=	**83**	Parameter value not decimal
=	**84**	Parameter value too long
=	**96**	No file name found or error when determining file names
=	**98**	Not all files were processed
=	**999**	Error during memory request

3 →	**PARAM**	**CL*n***	Parameter area

4 →	**PARLEN**	**F**	Length of parameter area
	= **0**		No parameter
	> **0**		Parameter present

Hinweis: Parameters must be written in the same way as in the utility. Only upper case letters are allowed!

FLAMUP can also be called with the parameters

```
C,MO=ADC,CRYPTOMO=AES,KME=name,KMP=parameter
```

and the files can be specified via DD statement in the JCL. It is also possible to call FLAMUP without parameters and also put the control parameters into a file. By default, FLAM looks for the file with the DD name FLAMPAR, which also can be parameterized (PARDD=ddname or PARFILE=dsn). If needed, input and output files can also be specified as parameters (FLAMIN=.., FLAMOUT=.., FLAMFILE=..). By default, message output is written into a file with the DD name FLPRINT (can be changed with MSGDD=ddname or MSGFILE=dsn).

The EXIT must be located in accessible library, i.e. in the STEPLIB, JOBLIB or LINKLIST concatenation. This is IBM standard.

It is recommended to put the associated parameters into a file. This is easy, the code remains slim, is easy to maintain.

5.2.1.1 Calling FLAMUP with ‚MSGFILE=filename, PARFILE=filename' as parameter

```
for encryption:

COMPRESS,MODE=ADC
CRYPTOMODE=AES
KMEXIT=...
KMPARM=C'...'
FLAMIN=originalfilename
FLAMFILE=flamfilename

for decryption:

DECOMPRESS
KMEXIT=...
KMPARM=C'...'
FLAMOUT=originalfilename
FLAMFILE=flamfilename
```

5.2.1.2 Example for the FLAMUP call in COBOL:

```
 IDENTIFICATION DIVISION.

 PROGRAM-ID. MUSTER.

*
```

```
* EXAMPLE FOR CALLING FLAMUP
*
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 FLAMID    PIC S9(8) COMP SYNC.
 77 RETCO     PIC S9(8) COMP SYNC.
 77 PARAM     PIC X(80) VALUE"C,MO=ADC".
 77 PARLEN    PIC S9(8) COMP SYNC VALUE 8.
*
 PROCEDURE DIVISION.
*
    CALL "FLAMUP" USING FLAMID, RETCO, PARAM, PARLEN.
*
     STOP RUN.
```

5.2.1.3 Example for the FLAMUP call in ASSEMBLER:

```
EXAMPLE   CSECT
*
*   CALLING FLAMUP
*
         LA   1,FLAMUPAR
         L    15,=V(FLAMUP)
         BALR 14,15
*
```

```
*   PARAMETER LIST FOR FLAMUP
*
FLAMUPAR DC  A(FLAMID)
         DC  A(RETCO)
         DC  A(PARAM)
         DC  A(X'80000000'+PARLEN)
*
*   PARAMETERS FOR FLAMUP
*
FLAMID   DS  F
RETCO    DS  F
PARAM    DC  C'C,MO=ADC'
PARLEN   DC  F'8'
*
*   SAVEAREA
*
SAVEAREA DS  18F
         END
```

5.2.1.4 Example for the FLAMUP call in C++:

```
// an example for calling flamup from C++
//
// set linkage convention
extern "OS" void FLAMUP(void **,long *,char *,long *);
int main()
```

```
{

 void *flamid;

 long retco;

 long parlen=8;

 char param[10]="C,MO=ADC";

 FLAMUP(&flamid,&retco,param,&parlen);

 return 0;

}
```

5.2.2 FLAM(UP) for other platforms

The subprogram **flamup** offers the same functionality as **flam4.exe:**

```
void flamup(
   void** Ptr,
   long*  Retco,
   char*  ParString,
   long*  ParStringLen);
```

Ptr: Pointer for internal use, may be NULL

Retco: Result of the call, same as return code of **flam4.exe**

ParString: Parameter string (see below)

ParStringLen: Length of the parameter string

The parameter string contains the parameters like when calling **flam4.exe.** The individual parameters are separated by comma.

Example:

The call *flam4 comp flamfile=filename flamin=inputfile mode=adc*

translates to the ParString: "*comp,flamfile=DateiName,flamin=EingabeDatei,mode=adc"*

If a parameter value contains commas, the parameter value must be enclosed in curly brackets.

Example of a comma-separated file list:

flam4 … flamin=file1,file2,file3 …

which, for flamup, translates to: *"…,flamin=(file1,file2,file3),.."*

In consequence, filenames must not contain commas.

In general: Every parameter value containing a comma must be enclosed in curly brackets. Within those brackets (not part of the parameter value), curly brackets that are part of the parameter value must be escaped by putting them twice.

5.2.2.1 The parameters for the FLAM-Key-Management-Exit (FKME)

The FLAM-Key-Management-EXIT (FKME) is located in library that is loaded dynamically. The name of the library and the name of the function can be set via the following parameters.

kmlib=*Name of the library*

kmexit=*Name of the function*

kmparam=*Parameter string that is passed to the function*

All three parameters can also be combined into one string:

kmexit=*functionname(libraryname)parameterstring*

and if using the default value for kmlib:

kmexit=*functionsname()parameterstring*

Defaults for WINDOWS:

kmlib=flamkme.dll

kmexit=flamkme

If the parameter string contains commas, it has to be enclosed in curly brackets when passing it to flamup (see above):

kmexit=**(***functionname((libraryname))parameterstring***)**

or

kmparam=**(***parameterstring***)**

In FLAM®, keywords can be abbreviated as long as they are unique:

kmlib, **kme**xit, **kmp**aram.

5.2.2.2 Example call of FLAMUP

```
void *Ptr = NULL;
long Retco;
long ParLen;
char Param[1024];

// Parameters for compression with default settings for
// the DLL "flamkme.dll" and function "flamkme"
```

```
strcpy( Param,
"logfile=LogFilename,mode=aes,comp,flamfile=FlamfileName.adc,flamin=InputFi
lename.dat,inrecformat=undef,kmparam=CustomExitSpecificString" ) ParLen =
strlen( Param );

flamup ( &Ptr, &Retco, Param, &ParLen );
if ( Retco != 0 ) {
 // Error Handling goes here }

For decompression:

strcpy( Param,
"logfile= LogFilename,deco,flamfile=FlamfileName.adc,flamout=
OutputFilename.dat,outrecformat=undef,kmparam= CustomExitSpecificString")
ParLen = strlen( Param );


INRECFORMAT and OUTRECFORMAT obviously depend on the data!

If a custom DLL / function should be used, the following parameters would
need to be added:

strcat( Param, "kmexit=MyFunction,kmlib=MyDll" );

If kmparam contains commas, the parameter string must be enclosed in curly
brackets:

The parameter string "One,Two,Three" becomes "(One,Two,Three)".
If it contains curly brackets, each bracket is doubled:
"One(of three),Two,Three(last one)" becomes "(One((of
three)),Two,Three((last one)))"

The alternative "kmexit=FunctionName(DllName)ParameterValue" works
accordingly:
"kmexit=(FunctionName((DllName))Parameter((values)))"
This option has been added for compatibility with earlier FLAM versions and
the FLAM Unix version.

It is recommended to use kmlib=..,kmexit=..,kmparam=...

The order of parameters is arbitrary, but it makes sense to specify the
"logfile=" parameter at the beginning, so that all errors can be logged,
even parameter ones!
```

5.3 FLAM implementation recommendations

The following subsections provide some recommendations for the implementation of EXITs which should be followed, as far as the utilized HSM architecture permits.

5.3.1 Handling of generation and version

When sending, the generation and version is determined from the key label by applying a template. For the template, the following wildcard characters are defined:

- Generation ,++'
- Version ,**'

Alternatively, they can be replaced by ,%' so that the position of generation and version in the label can be determined by the EXIT. All other characters must match with the corresponding label.

Example: ,TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**' when sending

The template for receiving must not contain a ‚%' in order to enable completion of the name for the key.

Example: ,TFMKY.BV000000.GUD00000.DAT0++**' when receiving

When creating a FLAMFILE®, a complete label and a template for the EXIT is always passed to FLAM®. If a FLAMFILE® needs to be accessed later on, only a template is passed where only the generation and the version remain variable.

5.3.2 Passing the input parameters to the EXIT via FLAM / FLAMUP

The input parameters for the EXIT are passed as a parameter in the parameter list for FLAM® or FLAMUP. The parameters must be combined into a string. If it contains rounded brackets or single quotation marks, those must be escaped by doubling them. The necessary key label (if applicable) followed by the key templates should be supplied first. Then the optional identification data and authentication data may follow. All values a separated by a dot. Below, you can find one example for every use case without authentication information for the HSM:

- **Encryption**
  TFMKY.BV000000.GUD00000.DAT00601
  TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**
- **Rekeying**
  TFMKY.BV000000.GUD00000.DAT0++**
  TFMKY.GUD00000.GUD00000.DAT00601
  TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**
- **Decryption**
  TFMKY.GUD00000.GUD00000.DAT0++**

5.3.3 The last 10 bytes of the info field of the FKMC

The first 40 bytes of the 50 bytes of the info field are specified in **Fehler! Verweisquelle konnte nicht gefunden werden.**. The remaining 10 bytes are freely available to the EXIT. It is recommended that the EXIT stores an identifier for its implementation. The following identifiers are defined:

- FLAMSTDxx
  Standard software implementation by Limes Datentechnik
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 FLAMSTD01 “

- IBMCCCAxx
  IBM implementation against the SAPI of the CCA
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 IBMCCCA01 “

- IBMDKMSxx
  IBM implementation against the DKMS General Purpose API
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 IBMDKMS01 “

- GUDPKCSxx
  PKCS11 implementation of G+D
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 GUDPKCS01 “

- BVUTIMAxx
  UTIMACO implementation of the Bankverlag
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 BVUTIMA01 “

xx – A placeholder for the version of the implementation; should be 02, like in the examples.

5.3.4 EBCDIC and ASCII conversion

Only the info field depends on character set. When creating a FLAMFILE®, the info field is filled depending on the platform of the sender. Hence, the recipient has to check which character set the info data is stored in and need to convert it, if necessary.

5.3.5 Result messages

5.3.5.1 Error messages of the Exit

The following messages should be consulted when the corresponding error occurs.

- FKME – The function code is not supported + additional error info

- FKME – The input parameter length is not correct + additional error info

- FMKE – The input parameter is not formatted correctly + additional error info

- FKME – The length of the data field is too short + additional error info
- FKME – The length of the data field is not correct + additional error info
- FKME – The data field is not formatted correctly + additional error info
- FKME – The length of the key field is too short + additional error info
- FKME – The length of the key field is not correct + additional error info
- FKME – The key field is not formatted correctly + additional error info
- FKME – The authentication failed + additional error info
- FKME – The cipher suite not supported (The FKMC info field is not correct) + additional error info
- FKME – The determination of generation and version failed + additional error info
- FKME – The generation and version is not formatted correctly + additional error info
- FKME – The determination of the label for the FMKY failed + additional error info
- FKME – FMKY not found + additional error info
- FKME – The calculation of the key test pattern for FMKY failed + additional error info
- FKME – The verification of the key test pattern for FMKY failed + additional error info
- FKME – The determination of the time stamp failed + additional error info
- FKME – The verification of the time stamp failed + additional error info
- FKME – The generation of the random numbers failed + additional error info
- FKME – The calculation of the hash value (FKEY) failed + additional error info
- FKME – The verification of the hash value (FKEY) failed + additional error info
- FKME – The encryption of FKEY failed + additional error info
- FKME – The decryption of FKEY failed + additional error info
- FKME – The Translate of FKEY failed + additional error info

All other message must start with ‘FKME - ‘ and may consist of other messages in English followed by corresponding error information from the respective subsystems or EXITs.

5.3.5.2 OK messages from the Exit

If execution of a function code does not result in an error, a so called OK message is generated for the purpose of logging. It consists of the function code, the timestamp and the random number.

„FKME – COMP successful + TSP(YYYYMMDDHHMMSSss) RND(RRRRRRRRRRRRRRR)“

„FKME – CHNG successful + TSP(YYYYMMDDHHMMSSss) RND(RRRRRRRRRRRRRRR)“

„FKME – DECO successful + TSP(YYYYMMDDHHMMSSss) RND(RRRRRRRRRRRRRRR)“

By logging timestamp and random number, monitoring and inspection are possible. Additionally, the standards set by VISA and MasterCard are met.

5.3.5.3 Information about the Exit itself

The first 50 bytes should match the info field of the FKMC, followed by other useful information about the respective implementation of the EXIT.

„FKME - FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 FLAMSTD01 + additional info“

6 List of abbreviations

AES = Advanced Encryption Standard

BIN = Binary

CBC = Cipher Block Chaining

CHNG = Change

CHR = Character

COMP = Compression

DAT = Data

DECO = Decompression

DED = Decryption Encryption Decryption

DES = Data Encryption Standard

EDE = Encryption Decryption Encryption

EZ = Encrypted Zeros

FKEY = FLAM® Key

FKMC = FLAM® Key Management CONTEXT

FKME = FLAM€ Key Management EXIT

FLAM® = Frankenstein Limes Access Method

FMKY = FLAM® Master Key

FUCO = Function code

GG = Generation

GS = Generating Entity

HSM = Hardware Security Module

ICBC = CBC with IV

INT = Integer

IV = Initialization Vector

KL = Key Length

KTV = Key Test Value

LG = Length

MAC = Message Authentication Code

MDC = Message Digest Cipher

MSG = Message

PAR = Parameter

PARAM = Parameter

PCI DSS = Payment Card Industry Data Security Standard

PIN = Personal Identification number

POV = BCD Encoding

PS = Personalization System

RAM = Random Access Memory

RETCO = Return code

SKMS = Static Key Management System

STR = String

TDES = Triple DES

VV = Version

ZKA = Zentraler Kredit Ausschuss (Central Credit Committee)