Schnittstellenspezifikation

für die Absicherung von Dateien mit

FLAM®

Hybrides Verfahren mit einem klaren zufälligen Schlüssel pro FLAMFILE® und einem statischen symmetrischen Transportschlüssel geschützt durch ein Hardware-Sicherheits-Modul (HSM)

Version 1.2

13.03.2006

Inhalt

1 Einleitung 1
2 Systemmodell 2
3 Spezifikation FLAM Key Management CONTEXT (FKMC) 4
3.1 Vorgaben durch FLAM® 4
3.2 Aufbau der Datenstruktur in der Version 001 4
3.2.1 Erläuterung der Datenfelder 5
3.2.1.1 Feld 1: Infodaten 5
3.2.1.2 Feld 2: Generation und Version des FMKY 6
3.2.1.3 Feld 3: KTV für den FMKY 6
3.2.1.4 Feld 4: Zeitpunkt der Erzeugung der FLAMFILE® 6
3.2.1.5 Feld 5: Zufallszahl zur Dynamisierung 7
3.2.1.6 Feld 6: Verschlüsselte FKEY 7
3.2.1.7 Feld 7: Hash über Feld 4 und Feld 5 sowie den klaren FKEY 7
4 Spezifikation FLAM® Key Management EXIT (FKME) 8
4.1 Aufbau der Funktionsschnittstelle 8
4.1.1 Erläuterung der Parameter 9
4.1.1.1 Para 1: Funktionscode 9
4.1.1.2 Para 2: Returncode 9
4.1.1.3 Para 3: Länge der Inputparameter 10
4.1.1.4 Para 4: Inputparameter in der Version 001 10
4.1.1.5 Para 5: Länge der Kontextdaten (FKMC) in der Version 001 11
4.1.1.6 Para 6: Kontextdaten (FKMC) in der Version 001 11
4.1.1.7 Para 7: Länge des Schlüssels (FKEY) in der Version 001 11
4.1.1.8 Para 8: Schlüssel (FKEY) in der Version 001 11
4.1.1.9 Para 9: Länge der Message 11
4.1.1.10 Para 10: Message 11
4.2 Abläufe in der Version 001 11
4.2.1 Verschlüsseln einer FLAMFILE® 11

4.2.2 Umschlüsseln einer FLAMFILE® 13
4.2.3 Entschlüsseln einer FLAMFILE® 14
5 Anhang 16
5.1 FKME- Schnittstelle 16
5.1.1 FKME für die Mainframe in Assembler 16
5.1.2 FKME für die anderen Pattformen in ANSI C 17
5.2 FLAM- Schnittstelle 18
5.2.1 FLAM(UP) für die Mainframe 19
5.2.1.1 Aufruf FLAMUP mit ‚MSGFILE=dateiname, PARFILE=dateiname' als Parameter. 22
5.2.1.2 Beispiel für den Aufruf von FLAMUP in COBOL: 22
5.2.1.3 Beispiel für den Aufruf von FLAMUP in ASSEMBLER: 23
5.2.1.4 Beispiel für den Aufruf von FLAMUP in C ++: 24
5.2.2 FLAM(UP) für die anderen Plattformen 24
5.2.2.1 Die Parameter für den FLAM-Keymanagement-Exit (FKME) 26
5.2.2.2 Beispielaufruf für das FLAMUP 26
5.3 FLAM Implementierungsempfehlung 28
5.3.1 Behandlung von Generation und Version 28
5.3.2 Übergabe der Input-Parameter über FLAM bzw. FLAMUP an den EXIT 28
5.3.3 Die letzten 10 Byte im Info-Feld des FKMC 29
5.3.4 EBCDIC- und ASCII- Wandlung 29
5.3.5 Nutzung von Messages 30
5.3.5.1 Fehlermeldung des Exits 30
5.3.5.2 Gutmeldung des Exits 31
5.3.5.3 Selbstauskunft des Exits 31
6 Abkürzungsverzeichnis 32

Abbildungen

Abbildung 1 Überblick 1

Tabellen

Tabelle 1 FLAM® Key Management CONTEXT (FKMC) 4

Tabelle 2 FLAM® Key Management EXIT (FKME) 8

Tabelle 3 Parameter beim Verschlüsseln 12

Tabelle 4 Parameter beim Umschlüsseln 13

Tabelle 5 Parameter beim Entschlüsseln 14

Versionsführung

Version	Status	Datum	Durch	Änderung
00.10 00.30	In Arbeit	07.11.2005 09.11.2005	F. Reichbott	- Erstellung des Dokuments
00.40	In Arbeit	16.11.2005	F. Reichbott	- Versionsführung ergänzt
00.50	In Arbeit	05.12.2005	F. Reichbott	- Änderung des Hash für Key-Test von MDC2 auf RipeMD160 - Version von FLAM® mit FKME
00.60	In Arbeit	06.12.2005	F. Reichbott	- Anhang zu Implementierungsempfehlungen ergänzt - Anhang um Schnittstelle zum FLAMUP ergänzt
00.70	In Arbeit	09.12.2005	F.Reichbott	- kleine Korrekturen - MSGFILE muss als erstes angegeben werden - Aufruf von FLAM(UP) für die Dateiabsicherung
00.80	In Arbeit	27.12.2005	F. Reichbott	- Änderung des Hash für Key-Test von RMD160 auf SHA1
01.00	Freigegeben	23.01.2006	F. Reichbott	- keine grundlegenden Änderungen
01.10	Freigegeben	21.02.2006	F. Reichbott	- Fehlermeldungen leicht angepasst - Timestemp als GMT festgelegt - Generation und Version von POV auf BIN geändert - Meldungen bei erfolgreicher Ausführung festgelegt
1.20	Freigegeben	13.03.2006	F. Reichbott	- Bei den Fehlermeldungen Grammatik korrigiert

				- Fehler im Beispiel auf Seite 3 korrigiert

1 Einleitung

In diesem Dokument wird beschrieben, wie über den FLAM® Key Management EXIT (FKME), die kryptographische Absicherung von Dateien zur Speicherung bei einer Instanz (Archiv, Logs oder ähnliches) und zum Austausch zwischen verschiedenen Kommunikationspartnern (Filetransfer) geschehen soll. Hierzu wird ein hybrides Verfahren spezifiziert, welches es ermöglicht, über verschiedene HSM (Infrastruktur des Financial PIN Supports) einen zufälligen Schlüssel (FKEY) pro FLAMFILE® zu erzeugen und diesen Einmalschlüssel über einen FLAM®-Master-Key (FMKY) des HSM sicher auszutauschen. Für die sichere Übermittlung des FLAM® Keys (FKEY) wird der FLAM® Key Management CONTEXT (FKMC) im User-Header einer jeden FLAMFILE® benutzt. Für die Bereitstellung des FMKY werden die etablierten Methoden zum Schlüsselmanagement herangezogen. Das folgende Bild veranschaulicht die Zusammenhänge.

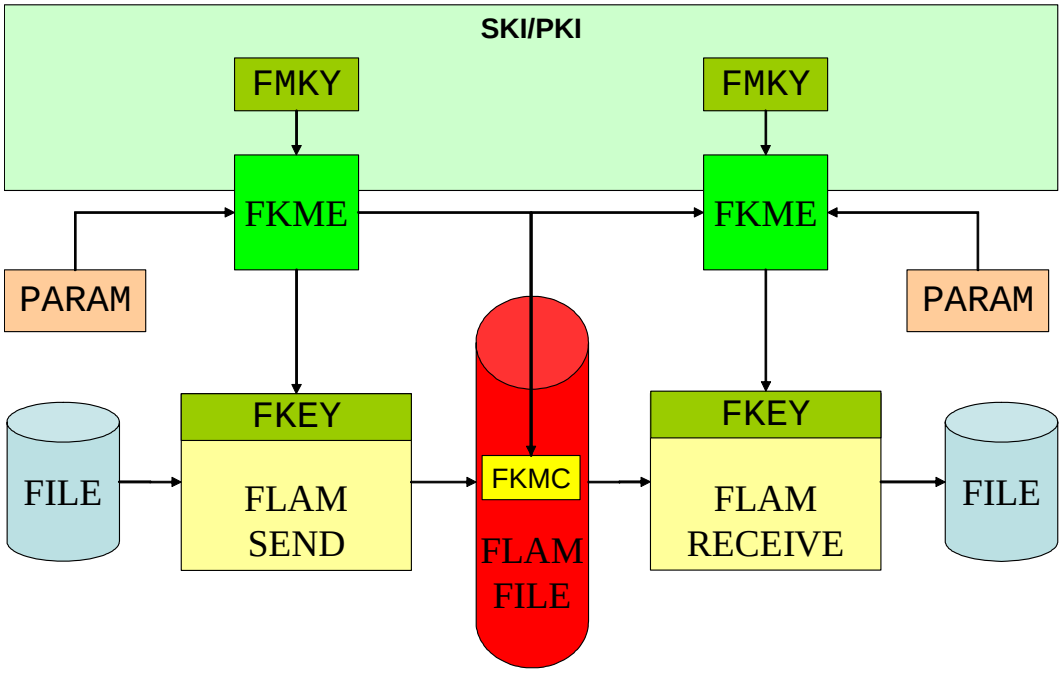

Abbildung 1 Überblick

Diese Spezifikation soll es unter anderem ermöglichen, den Anforderungen des „Payment Card Industry Data Security Standard“ (PCI DSS) mit FLAM® gerecht zu werden. Hierbei spielt die sichere Speicherung und Archivierung der Dateien für die Kartenproduktion, der Austausch dieser Dateien zwischen den beteiligten Parteien (Issuer, Kartenproduzenten, ..), als auch die Absicherung von Logs und anderen Dateien im Transaktionsbetrieb eine Rolle.

Des Weiteren sollte diese Infrastruktur, wie mit FLAM® bisher auch, für den sicheren Austausch von Zahlungsverkehrsdateien verwendbar sein. Hinzu kommen mögliche andere Anwendungen, wo die Vertraulichkeit und Integrität von Dateien sichergestellt werden soll.

2 Systemmodell

FLAM® mit AES (ab Version 4.0) realisiert den Integritätsschutz und die Vertraulichkeit über einen 64 Byte langen Schlüssel, welcher zur Ableitung von jeweils 4 AES 128 Bit Schlüsseln verwendet wird. Diese 4 Schlüssel werden über alle drei Ebenen der FLAMFILE® mit Hilfe einer Einweghashfunktion abgeleitet und dienen der Verschlüsselung der Nettodaten (Segment) und der Bildung von Message Authentication Codes (MAC) in jeder Ebene über die jeweiligen Bestandteile (Segment, Member, File) der FLAMFILE®.

Damit der klare Schlüssel für FLAM® (FKEY) innerhalb einer HSM- basierten kryptographischen Infrastruktur sicher verwendet werden kann, müssen zwei Anforderungen erfüllt werden:

1. Der FKEY darf nur dort kurzzeitig in klar im Arbeitsspeicher (RAM) des Rechners existieren, wo auch die Daten in klar vorkommen müssen.

2. Der FKEY darf nur einmalig pro FLAMFILE® verwendet werden, damit ein mögliches Ausspähen des Wertes im Speicher des Rechners für eine andere FLAMFILE® keine Rolle spielt.

Damit existiert der FKEY nur dort, wo auch gleich die klaren Daten ausgespäht werden können, und an mehr als diese Daten kann ein Angreifer durch den jeweiligen FKEY nicht gelangen. Die Bereitstellung des FKEY erfolgt über den FLAM® Key Management EXIT (FKME), welcher **ab Version 4.1 von FLAM®** für die Anbindung von verschiedenen kryptographischen Infrastrukturen unterstützt wird.

Damit muss der FKEY aus 64 Byte nicht vorhersehbaren Zufall bestehen, welcher von dem jeweiligen HSM in der entsprechenden Qualität generiert werden muss. Am Anschluss stellt das HSM eine entsprechend durch die Hardware geschützte Struktur (FLAM Key Management CONTEXT (FKMC)) für den User-Header der FLAMFILE® bereit. Dies gestattet es einem entsprechend personalisierten HSM, den FKEY beim Lesen in seiner klaren Form wieder herzustellen.

Zum Schutz des FKMC wird ein statischer symmetrischer Master Key für FLAM® (FMKY) verwendet, welcher auf der Seite des Senders zum Verschlüsseln und beim Empfänger zum Entschlüsseln des FKEY genutzt wird. Damit durch dieses hybride Verfahren die Sicherheit der HSMs nicht gefährdet wird, entspricht der klare FKEY aus der Sicht des HSM 64 Byte normalen Daten.

Über die verschiedenen Verwendungszwecke (ENCIPHER und DECIPHER) können über die etablierten statischen Key-Management-Systeme (SKMS), welche alle Anforderunen des PCI DSS gerecht werden, gerichtete Schlüsselbeziehungen aufgebaut werden. Diese ermöglichen es, die Urheberschaft für eine FLAMFILE® eindeutig nachzuweisen. Über die verschiedenen Verwendungszwecke für das Umschlüsseln von Daten (IDATXLAT und ODATXLAT) können Vermittler, Kopfstellen und andere Besitzstandsübergänge sicher abgebildet werden. Hierbei wird nur der FKEY vom HSM umgeschlüsselt und der FKMC im

User-Header der FLAMFILE® ausgetauscht. Die eigentlichen Daten bleiben bei diesem Vorgang unter dem jeweiligen FKEY verschlüsselt. Der FKEY selbst bleibt hierbei durch das HSM durchgehend geschützt. Diese Möglichkeit zum sicheren Umschlüsseln einer FLAMFILE® stellt sicher, dass die Daten nur bei der Erzeugung und bei ihrer Verarbeitung in klar existieren müssen. Dieser Sachverhalt soll durch das folgende Beispiel zwischen Generierungsstelle (GS) für die Kartenproduktion und dem Personalisierungssystem (PS) beim Kartenproduzenten deutlich gemacht werden.

- Sicheres Speichern der generierten Daten innerhalb der Generierungsstelle
  `FLAM(COMP, FMKY.GS2GS.ENCIPHER.ggvv)`
- Transferieren der Kartendaten zum Produzenten
  `FLAM(CHNG, FMKY.GS2GS.IDATXLAT.ggvv, FMKY.GS2PS.ODATXLAT.ggvv)`
- Empfang und sichere Speicherung beim Produzenten
  `FLAM(CHNG, FMKY.GS2PS.IDATXLAT.ggvv, FMKY.PS2PS.ODATXLAT.ggvv)`
- Entschlüsselung der gespeicherten Daten zur Produktion
  `FLAM(DECO, FMKY.PS2PS.DECIPHER.ggvv)`

Des Weiteren ist es über die Umschlüsselung möglich, von einer Generation und Version auf eine andere des gleichen oder eines anderen FMKY überzugehen. Hinzu kommt noch, wenn dieser Migrationspfad von der jeweiligen Implementierung des EXIT unterstützt wird, der Wechsel zwischen verschiedenen Ciphersuites.

Die Generierung und der Austausch der FMKY zwischen den verschiedenen Entitäten sollten nach den etablierten und anerkannten Verfahren zum statischen Key Management (mindestens zweistufig (2 oder 3 klare Schlüsselkomponenten + FMKY als Kryptogramm)) erfolgen. Dies ist nicht Bestandteil dieser Schnittstellenspezifikation. Hier müssen einfach die geltenden Anforderungen zur statischen Schlüsselverwaltung im jeweiligen Umfeld (VISA, ZKA, …) eingehalten werden. In diesem Dokument wird davon ausgegangen, dass die statischen FMKY für ihren entsprechenden Verwendungszweck über das HSM zur Verfügung stehen. Die Schlüssel sollten, wie alle statischen Schlüssel, regelmäßig gewechselt und adhoc ausgetauscht werden können. Hierfür werden für jeden der Schlüssel eine Generation (GG) und eine Version (VV) vorgesehen.

3 Spezifikation FLAM Key Management CONTEXT (FKMC)

3.1 Vorgaben durch FLAM®

Das Kontextfeld (FKMC) darf über alle Plattformen hinweg derzeit nicht länger als 512 Byte werden und muss für die erfolgreiche Umschlüsselung von FLAMFILES® immer eine konstante Länge haben. Für den Inhalt werden keine Vorgaben gemacht, außer dass die ersten 50 Byte als Information protokolliert werden.

Hinweis: Wenn mehr als 512 Byte benötigt werden, ist dies kein Problem. Hierfür muss aber bei den Versionen für den Mainframe eine neue Version von FLAM mit größeren Puffern bereitgestellt werden.

3.2 Aufbau der Datenstruktur in der Version 001

Die folgende Tabelle gibt eine Übersicht über die Datenstruktur. Hierbei bedeutet:

Lg = Länge in Byte

CHR = Zeichen im jeweiligen Zeichensatz der erzeugenden Maschine
POV = BCD-kodiert, rechtsbündig mit führenden 0
BIN = Binär

Feld	Bezeichnung	Inhalt	Lg	Format
1	Infodaten	FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1	50	CHR
2	Generation und Version des FMKY	GGVV	2	BIN
3	KTV für den FMKY	RRRRRRRR	4	BIN
4*	Zeitpunkt (GMT) der Erzeugung der FLAMFILE®	JJJJMMTTHHMMSSss	8	POV
5*	Zufallszahl zur Dynamisierung (ggf. Wiedereinreichungskontrolle)	RRRRRRRRRRRRRRR	8	BIN
6**	Verschlüsselter FKEY	RRR…RRR	64	BIN
7*	HASH über 4 und 5 sowie den klaren FKEY	RRRRRRRRRRRRRRR	8	BIN

Tabelle 1 FLAM® Key Management CONTEXT (FKMC)

*) Die Felder werden beim Erzeugen der FLAMFILE(*) belegt und dann nicht mehr verändert.

**) Der klare FKEY wird ebenfalls nur bei der Erzeugung generiert und dann beibehalten, nur dass sich sein Kryptogramm in Abhängigkeit vom FMKY ändern kann.

3.2.1 Erläuterung der Datenfelder

3.2.1.1 Feld 1: Infodaten

Die Infodaten nutzen die ersten 50 Byte des Kontextfeldes zur Selbstauskunft über den EXIT und legen gleichzeitig dessen Implementierung fest. Hierbei müssen folgende Angaben in Großbuchstaben und getrennt durch ein Leerzeichen gemacht werden.

1. **Kennzeichen für den FKMC** = FKMC
   4 Byte lange Konstante für die Prüfung und Erkennung des Zeichensatzes (EBCDIC oder ASCII).

2. **Version des EXIT** = V001
   Die Version des EXIT wird mit V001 konstant festgelegt. Dieses Kennzeichen wird zur Unterscheidung späterer Implementierungen herangezogen.

3. **Länge des EXIT** = L144
   Die Länge für das Kontextfeld dieses EXIT beträgt 144 Byte. Die Länge kann in Abhängigkeit von der Version variieren und dient der Plausibilitätsprüfung.

4. **Algorithmus für den FMKY** = TDES
   Über das vierte Feld wird der Algorithmus für die Verschlüsselung des FKEY mit dem FMKY festgelegt. In der Version 001 ist dies Triple DES.

5. **Schlüssellänge des FMKY** = KL16
   Über das fünfte Feld wird die Länge des FMKY festgelegt, welche 112 Bit beträgt. Dies entspricht einem double length TDES Key mit 16 Byte.

6. **Verifikationsverfahren für den FMKY** = EZ04
   Über das sechste Feld wird das Verfahren zur Berechnung des Key-Test-Values (KTV) für den FMKY festgelegt, welches über die Methode Enc-Zeros realisiert wird. Hierbei werden aber nur die höherwertigen 4 Byte gespeichert, was auch dem gängigen VISA Key-Test-Verfahren entspricht.

7. **Ciphermode für den FKEY** = ICBC
   Das siebente Feld legt den Modus für die Verschlüsselung des FKEY fest. ICBC steht hierbei für den Cipher-Block-Chaining Mode mit Verwendung eines Initialwertes.

8. **Verfahren für die Einweghashfunktion** = SHA1
   Als Verfahren für den Hash über den Zeitstempel, die Zufallszahl und den klaren FLAM®- Schlüssel wird SHA1 als Algorithmus festgelegt.

9. **Weitere Informationen**
   Weitere Informationen (Implementierung des EXIT (IBMCCA, IBMDKMS, PKCS11, ATALLA, THALES, …)) sind der jeweiligen Implementierung freigestellt. Wenn keine Informationen vorliegen, werden die restlichen 10 Byte mit Leerzeichen aufgefüllt.

Zum Check der Implementierung des EXIT gegen die Infodaten werden die ersten 40 Byte in der Version 001 gegen die folgende Konstante verglichen (Achtung: Leerstelle am Ende):

„FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 “

Hierbei ist es wichtig, dass der Zeichensatz für den Vergleich keine Rolle spielen darf. Die Ermittlung des Zeichensatzes sollte über das erste Byte der Infodaten erfolgen, welches in EBCDIC ‚C6’hex und in ASCII ‚46’hex für ein großes ‚F’ betragen muss.

3.2.1.2 Feld 2: Generation und Version des FMKY

In diesem Feld wird die Generation (gg) und die Version (vv) des FMKY beim Senden eingestellt. Sie dient beim Empfangen der FLAMFILE® dazu, den richtigen FLAM-Master-Key (FMKY) zu verwenden. Die Ermittlung der Generation und Version beim Senden und ihre Verwendung beim Empfangen hängen von der Implementierung des EXIT ab. In der Regel sollte dem EXIT beim Senden ein voll qualifiziertes Label für den Schlüssel zusammen mit der dazugehörigen Generation und Version übergeben werden. Beim Empfangen ist es dann ausreichend, wenn in dem Label Platzhalter für die Generation und Version enthalten sind.

3.2.1.3 Feld 3: KTV für den FMKY

Das Key-Test-Value (KTV) für den FMKY ist das Ergebnis der TDES Verschlüsselung (EDE) von 8 Byte Nullen mit dem FMKY. Hierbei werden aber nur die höherwertigen (linken) 4 Byte in dem Feld gespeichert. Das Verfahren wurde gewählt, da es von den meisten HSMs direkt unterstützt wird, den gestellten Sicherheitsanforderungen gerecht wird, auf der Seite des Senders auch über die Verschlüsselung von 8 Byte Daten erzeugt werden kann, was bei Verzicht auf die gerichteten Schlüsselbeziehungen auch beim Empfänger möglich wird, und weil es mit Hilfe von TDES als Basisalgorithmus abbildbar ist.

3.2.1.4 Feld 4: Zeitpunkt der Erzeugung der FLAMFILE®

Der Zeitpunkt der Erzeugung der FLAMFILE® muss auf Basis der Anforderungen kryptographisch sicher protokolliert werden. Dies geschieht zum einen durch die Einbeziehung in die HASH- Berechnung und zum anderen in die Bildung des IV für die CBC-Mode-Verschlüsselung. Das Format besteht, wie oben in der Tabelle angedeutet, aus:

- JJJJ - Jahr
- MM - Monat
- TT - Tag
- HH - Stunde
- MM - Minute

- SS - Sekunde
- ss - Millisekunde

Es ist wünschenswert, einen verlässlichen Zeitgeber für die Abbildung der 8 Byte zu nutzen. Für den Zeitpunkt wird die Greenwich Mean Time (GMT) festgelegt, damit ein Weltweiter vergleich möglich wird.

3.2.1.5 Feld 5: Zufallszahl zur Dynamisierung

Die 8 Byte Zufallszahl zur Dynamisierung dient einfach als eineindeutiges Kennzeichen für eine FLAMFILE® und wird in die HASH- Berechnung sowie die IV-Bildung mit einbezogen.

3.2.1.6 Feld 6: Verschlüsselte FKEY

Der 64 Byte lange zufällige FKEY (8 Blöcke) wird in der Version 001 im CBC- Mode mit dem FMKY TDES verschlüsselt. Als IV fließt das byteweise XOR- Produkt der Felder 4 und 5 in die Berechnung ein. Das Ergebnis ist ein Kryptogramm für den FKEY von 64 Byte, welches vom EXIT in binärer Form in die Kontextstruktur eingestellt wird.

3.2.1.7 Feld 7: Hash über Feld 4 und Feld 5 sowie den klaren FKEY

Die Berechnung des Hashwertes erfolgt nach dem SHA1 Verfahren über die folgenden 80 Byte:

- 8 Byte Feld 4: Zeitpunkt der Erzeugung der FLAMFILE®
- 8 Byte Feld 5: Zufallszahl zur Dynamisierung
- 64 Byte klarer zufälliger FKEY.

Es werden nur die höherwertigen (linken) 8 Byte des 20 Byte langen Hashwertes in der Kontextstruktur hinterlegt. Ihre Berechnung erfolgt bei der Generierung des Schlüssels, die Verifikation dieses Wertes sollte aber nur bei der Verwendung des Schlüssels erfolgen. Das heißt, dass die Verifikation nur vorgenommen wird, wenn der klare Schlüsselwert gebraucht wird. Dies ist bei der Umschlüsselung einer FLAMFILE® nicht der Fall.

4 Spezifikation FLAM® Key Management EXIT (FKME)

Der FLAM® Key Management EXIT (FKME) ist eine Funktion, welche von FLAM® gerufen wird, wenn entsprechende Parameter (PARAM) für einen EXIT an FLAM übergeben wurden. Er dient zur Anbindung beliebiger kryptographischer Infrastrukturen, um Session-Key-Verfahren mit FLAM® abbilden zu können.

4.1 Aufbau der Funktionsschnittstelle

Die folgende Tabelle gibt eine abstrakte Übersicht über die Parameter der Schnittstelle. Hierbei bedeutet:

INT = 32 Bit (4 Byte) im Zweierkomplement
STR = Array von Bytes

Para	Bez.	Kommentar	Richtung	Typ	Länge
1	Fuco	Funktionscode	INPUT	INT	4 Byte
2	RetCo	Returncode	OUTPUT	INT	4 Byte
3	ParLen	Länge der Inputparameter	INPUT	INT	4 Byte
4	Param	Inputparamter	INPUT	STR	Var
5	DatLen	Länge der Daten (FKMC)	COMP:INPUT/OUTPUT DECO:INPUT/OUTPUT CHNG:INPUT=OUTPUT	INT	4 Byte
6	Data	Daten (FKMC)	COMP:OUTPUT DECO:INPUT CHNG:INPUT/OUTPUT	STR	Var
7	KeyLen	Länge des Schlüssels (FKEY)	INPUT/OUTPUT CHNG: not used	INT	4 Byte
8	Key	Schlüssel (FKEY)	OUTPUT CHNG: not used	STR	Var
9	MsgLen	Länge der Nachricht	INPUT/OUTPUT	INT	4 Byte
10	Message	Nachricht	OUTPUT	STR	Var

Tabelle 2 FLAM® Key Management EXIT (FKME)

Alle Parameter werden als Zeiger (Call by Reference) übergeben. Der FKME wird als Modul mit einer Funktion zur Laufzeit von FLAM® dynamisch geladen. Hierbei sind der Name des Moduls und ggf. der Name der Funktion über FLAM® parametrisierbar. Die Inputlängen geben immer den bereitgestellten Speicher an wobei die Outputlängen den benötigten Speicher wieder zurückgeben müssen.

4.1.1 Erläuterung der Parameter

4.1.1.1 Para 1: Funktionscode

Über den Funktionscode wird die Methode für den EXIT ausgewählt. Folgende Verfahren sind definiert:

- **0 – Decompression/Entschlüsselung**
  In diesem Fall werden die Inputparameter zusammen mit dem Kontextfeld (FKMC) aus dem Userheader an den EXIT übergeben, welcher daraufhin den klaren Schlüssel (FKEY) an FLAM zurückgibt.

- **1 – Compression/Verschlüsselung**
  In diesem Fall werden die Inputparameter an den EXIT übergeben, welcher daraufhin 64 Byte Zufall (FKEY) generiert und diesen zusammen mit dem Kontextfeld (FKMC) zurückgibt. FLAM schreibt daraufhin das Kontextfeld in den Userheader und nutzt den klaren zufälligen Schlüssel für die Erzeugung der FLAMFILE®.

- **2 – Change/Umschlüsselung**
  In diesem Fall werden die Inputparamter zusammen mit dem Kontextfeld (FKMC) aus dem Userheader an den EXIT übergeben, welcher daraufhin das neue Kontextfeld (FKMC) berechnet und an das Programm zurückgibt. Dieses Programm erstellt nun eine neue FLAMFILE® mit angepasstem Userheader.

- **0xFFFFFFFF – Information**
  Mit dieser Methode wird der EXIT aufgefordert, eine Selbstauskunft in das Messagefeld einzustellen.

4.1.1.2 Para 2: Returncode

Der Returncode dient einzig und allein der Programmsteuerung von FLAM. Ist dieser nach dem Aufruf des EXIT 0, war alles in Ordnung, und FLAM arbeitet, wie erwartet, weiter. Wenn der Returncode 4 beträgt, wurde beim Aufruf des EXIT zu wenig Speicher bereitgestellt. Dies wird auf Plattformen mit dynamischer Speicherverwaltung zu einem erneuten Aufruf mit genügend Speicher führen. Auf den anderen Plattformen führt dieser Fehler wie alle anderen Fehler > 0 zu einem Abbruch von FLAM. Die Ursachen für einen solchen Abbruch können vom EXIT nur über das Messagefeld nach außen mitgeteilt werden. Dieses wird immer ausgegeben, wenn die Länge der Nachricht ungleich 0 ist. Damit ist es dem EXIT auch möglich, Warnungen bei einem Returncode = 0 nach Außen zu geben. Wenn alles in Ordnung war, dann wird dies zusammen mit der Funktion (COMP, CHNG, DECO), dem Zeitstempel und der Zufallszahl entsprechend über das Messagefeld protokolliert.

4.1.1.3 Para 3: Länge der Inputparameter

Die Länge der Inputparameter ist abhängig von der Methode und der Implementierung des EXITS. Sie wird von FLAM® beim Aufruf bestimmt und an den EXIT weitergegeben.

4.1.1.4 Para 4: Inputparameter in der Version 001

Die Inputparameter sind von der Methode und der Implementierung des EXITS abhängig. Grundsätzlich sollten sie in Abhängigkeit der Methode folgende Informationen beinhalten.

- **VERSCHLÜSSELN**
    - Ggf. Identifikationsdaten und Authentifikationsdaten für das HSM
    - Referenzierungsdaten für den FLAM-Master-Key (FMKY)
        - Key Label und Generation + Version oder
        - Key Label und Key Label Template
- **ENTSCHLÜSSELN**
    - Ggf. Identifikationsdaten und Authentifikationsdaten für das HSM
    - Referenzierungsdaten für den FLAM-Master-Key (FMKY)
        - Key Label Template
- **UMSCHLÜSSELN**
    - Ggf. Identifikationsdaten und Authentifikationsdaten für das HSM
    - Referenzierungsdaten für den FLAM-Master-Key (FMKY) reingehend
        - Key Label Template
    - Referenzierungsdaten für den FLAM-Master-Key (FMKY) rausgehend
        - Key Label und Generation + Version oder
        - Key Label und Key Label Template

Für die Inputparameter beim Aufruf von FLAM steht derzeit maximal ein Puffer von 256 Byte zur Verfügung.

4.1.1.5 Para 5: Länge der Kontextdaten (FKMC) in der Version 001

Die Länge der Kontextdaten (FKMC) beträgt 144 Byte. Diese Größe des Puffers unterschreitet die 512 Byte, welche von FLAM zur Verfügung gestellt werden. Damit sollte für das Kontextfeld immer genügend Speicher bereit stehen. Dieser Parameter ist ausgangsseitig vom EXIT entsprechend auf 144 zu setzen, und es muss kontrolliert werden, wenn von FLAM® ein Kontextfeld übergeben wird (ENTSCHLÜSSELUNG oder UMSCHLÜSSELUNG), dass dieser Wert auf 144 steht.

4.1.1.6 Para 6: Kontextdaten (FKMC) in der Version 001

Über diesen Parameter wird das Kontextfeld (FKMC) vom EXIT bereitgestellt oder entgegengenommen. Seine Spezifikation ist dem Punkt 3.2 zu entnehmen.

4.1.1.7 Para 7: Länge des Schlüssels (FKEY) in der Version 001

Die Länge des klaren zufälligen Schlüssels beträgt, wenn dieser vom EXIT zurückgegeben wird (VER- oder ENTSCHLÜSSELN), immer 64 Byte, welche auch von FLAM® hierfür eingangsseitig bereitgestellt werden.

4.1.1.8 Para 8: Schlüssel (FKEY) in der Version 001

Über diesen Parameter werden FLAM® vom EXIT die 64 Byte Zufall zur Ver- oder Entschlüsselung zur Verfügung gestellt.

4.1.1.9 Para 9: Länge der Message

Für die Message hält FLAM® derzeit einen maximalen Puffer von 128 Byte bereit. Dieser kann vom EXIT genutzt werden, um Fehlerursachen oder andere Informationen von FLAM protokollieren zu lassen.

4.1.1.10 Para 10: Message

Beinhaltet die Message, welche bei Länge ungleich 0 ausgegeben wird. Wenn der Funktionscode ‚FFFFFFFFFF'hex ist, dann gibt der EXIT seine Infodaten zurück.

4.2 Abläufe in der Version 001

4.2.1 Verschlüsseln einer FLAMFILE®

FLAM® ruft den EXIT mit der folgenden Belegung der Parameter auf, und der EXIT gibt, wenn alles erfolgreich verläuft, die darauf folgenden Werte zurück:

Para	Inputwerte	Outputwerte
Fuco	1	=
RetCo	0	0
ParLen	>0	=
Param	Parameter für den EXIT	=
DatLen	512	144
Data	Undefiniert	FKMC laut 3.2
KeyLen	64	64
Key	Undefiniert	64 Byte generierter Zufall
MsgLen	128	Länge der Gutmeldung
Message	Undefiniert	Gutmeldung

Tabelle 3 Parameter beim Verschlüsseln

An dieser Stelle wird der Ablauf bei der Verschlüsselung (Fuco=1) in Stichpunkten dargelegt.

- Kontrolle der Pufferlängen
- Bestimmung der Identifikations- und Authentificationdaten für das HSM aus dem Parameterfeld
- Bestimmung des Labels und der Generation und Version für den Schlüssel aus dem Parameterfeld
- Initialisierung der Kontextstruktur mit den Infodaten
- Eintragen der Generation und Version in das Kontextfeld
- Berechnung des KTV für den FMKY auf Basis des Labels und Eintragung ins Kontextfeld
- Bestimmung des Zeitstempels und Eintragung ins Kontextfeld
- Generierung der Zufallszahl und Eintragung ins Kontextfeld
- Berechnung des XOR- Produktes aus Zeitstempel und Zufallszahl und Zwischenspeicherung als IV
- Generierung des 64 Byte langen Schlüsselwertes als Rückgabewert an FLAM
- Berechnung des SHA1 über den Zeitstempel, die Zufallszahl und den klaren Schlüsselwert und Eintragung in das Kontextfeld
- CBC Verschlüsselung des Schlüsselwertes mit dem IV und dem FMKY und Eintragung des Ergebnisses in das Kontextfeld

- Setzten der Pufferlängen, der Gutmeldung und des Returncodes
- Rücksprung aus dem EXIT

4.2.2 Umschlüsseln einer FLAMFILE®

Das Programm ruft den EXIT mit der folgenden Belegung der Parameter auf, und der EXIT gibt, wenn alles erfolgreich verläuft, die darauf folgenden Werte zurück:

Para	Inputwerte	Outputwerte
Fuco	2	=
RetCo	0	0
ParLen	>0	=
Param	Parameter für den EXIT	=
DatLen	144	144
Data	FKMC laut 3.2	FKMC laut 3.2
KeyLen	Undefiniert	=
Key	Undefiniert	=
MsgLen	128	Länge der Gutmeldung
Message	Undefiniert	Gutmeldung

Tabelle 4 Parameter beim Umschlüsseln

An dieser Stelle wird der Ablauf bei der Umschlüsselung (Fuco=2) in Stichpunkten dargelegt.

- Kontrolle der Pufferlängen
- Bestimmung der Identifikations- und Authentificationdaten für das HSM aus dem Parameterfeld
- Bestimmung des Inputtemplates für den Inputschlüssel aus dem Parameterfeld
- Bestimmung des Outputlabels und der Outputgeneration und -version für den Outputschlüssel aus dem Parameterfeld
- Bestimmung der Inputversion und –Generation aus dem Kontextfeld
- Bestimmung des voll qualifizierten Inputlabels aus Template, Generation und Version
- Verifikation des KTV über das Inputlabel
- Eintragen der Outputgeneration und -version in das Kontextfeld
- Berechnung des KTV für den Output- FMKY auf Basis des Outputlabels und Eintragung ins Kontextfeld

- Berechnung des XOR- Produktes aus Zeitstempel und Zufallszahl und Zwischenspeicherung als IV
- Umschlüsselung des 64 Byte langen Schlüsselwertes vom Input- FMKY zum Output-FMKY auf Basis des IV und Ersetzung des Kryptogramms im Kontextfeld
- Setzen der Pufferlängen, der Gutmeldung und des Returncodes
- Rücksprung aus dem EXIT

4.2.3 Entschlüsseln einer FLAMFILE®

FLAM® ruft den EXIT mit der folgenden Belegung der Parameter auf, und der EXIT gibt, wenn alles erfolgreich verläuft, die darauf folgenden Werte zurück:

Para	**Inputwerte**	**Outputwerte**
Fuco	0	=
RetCo	0	0
ParLen	>0	=
Param	Parameter für den EXIT	=
DatLen	144	=
Data	FKMC laut 3.2	=
KeyLen	64	64
Key	Undefiniert	64 Byte entschlüsselter Zufall aus FKMC
MsgLen	128	Länge der Gutmeldung
Message	Undefiniert	Gutmeldung

Tabelle 5 Parameter beim Entschlüsseln

An dieser Stelle wird der Ablauf bei der Entschlüsselung (Fuco=0) in Stichpunkten dargelegt.

- Kontrolle der Pufferlängen
- Bestimmung der Identifikations- und Authentificationdaten für das HSM aus dem Parameterfeld
- Bestimmung des Templates für den statischen Schlüssel aus dem Parameterfeld
- Bestimmung der Version und –Generation aus dem Kontextfeld
- Bestimmung des voll qualifizierten Labels aus Template, Generation und Version
- Verifikation des KTV über das Label das FMKY

- Berechnung des XOR- Produktes aus Zeitstempel und Zufallszahl und Zwischenspeicherung als IV
- CBC Entschlüsselung des 64 Byte langen Schlüsselwertes mit Hilfe des FMKY und des IV und Bereitstellung als Rückgabewert an FLAM®
- Berechnung des SHA1 über den Zeitstempel, die Zufallszahl und den klaren Schlüsselwert und Vergleich mit dem Wert im Kontextfeld
- Setzten der Pufferlängen, der Gutmeldung und des Returncodes
- Rücksprung aus dem EXIT

5 Anhang

5.1 FKME- Schnittstelle

An dieser Stelle werden die entsprechenden Auszüge aus den Handbüchern bzw. Spezifikationen zusammenfassend noch mal aufgeführt.

5.1.1 FKME für die Mainframe in Assembler

Dieser Benutzerausgang dient zum Anschluss an ein Schlüsselverwaltungssystem (Key Management). Die Aufgabe dieser Benutzerroutine ist es, zur Ver- / Entschlüsselung einer FLAMFILE einen Schlüssel zur Verfügung zu stellen. Er kann in FLAM und FLAMUP benutzt werden. Der EXIT wird über den Parameter KMEXIT=name aktiviert. Er wird dann für jede FLAMFILE aufgerufen. Beim Aufruf durch FLAM werden die Angaben des Parameters KMPARM (max. 256 Bytes) zur Verfügung gestellt. Der EXIT kann bei der Verschlüsselung einen Datenstring von max. 512 Bytes zurückgeben, der in der FLAMFILE gespeichert wird (als Userheader, vgl. Funktion FLMPUH). Zur Entschlüsselung werden diese Daten wieder von FLAM an den EXIT übergeben. Die ersten 50 Zeichen des Datenstrings werden bei der Dekomprimierung/Entschlüsselung als Kommentar im Protokoll angezeigt (FLM0487 USER HEADER: ...).

Name: frei wählbar (max. 8 Zeichen)

Registerbelegung:

→	**R1:**	Adresse der Parameterliste
→	**R13:**	zeigt auf Sicherstellungsbereich (18 Worte)
→	**R14:**	enthält die Rücksprungadresse
→	**R15:**	enthält die Aufrufadresse

Parameterliste:

1 →	**FUCO**		**F**	Funktionscode
	=	**0**		Entschlüsselung
	=	**1**		Verschlüsselung
2 ←	**RETCO**		**F**	Returncode
	=	**0**		kein Fehler
	=	**sonst**		Fehler
3 →	**PARMLEN**		**F**	Länge Parameter (max. 256)
4 →	**PARAM**		**XLn**	Parameter (in Länge PARMLEN)
5 ↔	**DATALEN**		**F**	Datenlänge
		Entschlüsselung:		
		→		Länge Daten

Verschlüsselung:
→ Größe Feld DATA (512)
← Länge Daten (max. 512)

6 ↔ DATA XLn Daten (in Länge DATALEN)

7 ↔ CKYLEN F Schlüssellänge
→ Größe Schlüsselpuffer (Feld CRYPTOKEY) (64)
← Länge Schlüssel (max. 64)

8 ← CRYPTOKEY XLn Schlüssel (in Länge CKYLEN)

9 ↔ MSGLEN F Nachrichtenlänge
→ Größe Nachrichtenpuffer (Feld MESSAGE) (128)
← Länge Nachricht (max. 128)

10 ← MESSAGE CLn Nachricht (in Länge MSGLEN)

Wird eine Nachrichtenlänge > 0 zurückgegeben, wird die Meldung MESSAGE im Protokoll ausgegeben (FLM0445 …). Die Daten DATA werden unverändert im Userheader der FLAMFILE gespeichert. Wird ein spezieller Schutz gewünscht, ist er vom EXIT selbst zu realisieren. Bei Verwendung dieses EXITs werden die FLAM- Parameter (z.B. der Kommandozeile) COMMENT und CRYPTOKEY überschrieben.

Der übergebene Schlüssel wird NICHT protokolliert.

Der EXIT wird pro FLAMFILE nur einmal aufgerufen. D.h. werden mehrere Dateien in eine Sammel-FLAMFILE komprimiert (C,FLAMIN=user.*), erfolgt der Aufruf nur einmal zu Beginn. Werden aber mehrere FLAMFILEs gelesen (D,FLAMFILE=user.*.aes), wird nach jedem Öffnen einer FLAMFILE der EXIT aufgerufen. Konkatinierte FLAMFILEs gelten als eine Datei!

Hinweis: Ein funktionsfähiges Beispiel ist in der ausgelieferten Bibliothek FLAM.SRCLIB(KMXSAMPL) enthalten.

5.1.2 FKME für die anderen Pattformen 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: befor 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,      // Lenght of key data: befor call -> buffer length (64)
                              //                     after call -> required bytes (64)
    U8         * Key,         // FLAMKME_FUCO_DECO/COMP: Session Key (64 Byte random data)
                              // FLAMKME_FUCO_CHNG: KeyLen and Key is not used
    U32        * MsgLen,      // Length of the Message: befor call -> buffer length
                              //                        after call -> required bytes
    U8         * Message      // Message
                             // if MsgLen!=0 then log msg
    );

#ifdef __cplusplus
}
#endif

#endif
```

5.2 FLAM- Schnittstelle

An dieser Stelle werden die entsprechenden Auszüge aus den Handbüchern bzw. Spezifikationen zusammenfassend noch mal aufgeführt. Alle weiteren Informationen müssen den Handbüchern zu FLAM® entnommen werden. Hierbei beziehen sich die folgenden Kapitel auf FLAM® und das FLAMUP, welche das Verschlüsseln und Entschlüsseln der Dateien unterstützt. Die Funktionalität zum Umschlüsseln von FLAMFILES®, wird von einem

neuen Tool realisiert, um die notwendigen Funktionalitäten zwischen End-Notes und Intermediate-Notes differenzieren zu können. Dieses neue Tool wird sich aber analog zu FLAM(UP) verhalten.

5.2.1 FLAM(UP) für die Mainframe

Mit FLAMUP können Dateien vollständig komprimiert oder dekomprimiert werden. Analog zum Dienstprogramm können Parameter übergeben werden. FLAMUP verwendet die gleichen Parameter wie das Dienstprogramm FLAM®.

FLAMUP kann von anderen Programmen als Unterprogramm aufgerufen werden (so ruft z.B. das Dienstprogramm FLAM selbst FLAMUP auf). FLAMUP ist eine Dateischnittstelle (im Gegensatz zur Satzschnittstelle FLAMREC), d.h. es werden ganze Dateien verarbeitet, nicht nur Datensätze.

Die Schnittstellenkonvention entspricht der Assembler- bzw. Cobol Unterprogramm Schnittstelle. In C spricht man von der OS-Konvention.

Im Folgenden werden die Schnittstellen in ASSEMBLER beschrieben. Die Tabelle zeigt, wie die verschiedenen Datentypen in COBOL und FORTRAN definiert werden müssen.

Assembler	Cobol	Fortran	Bedeutung
F	PIC S9 (8) COMP SYNC	INTEGER*4	ausgerichtetes Ganzwort
H	PIC S9 (4) COMP SYNC	INTEGER*2	ausgerichtetes Halbwort
CL*n* D i e	PIC X (*n*) USAGE DISPLAY	CHARACTER**n*	*n* abdruckbare Zeichen
XL*n* P f e	PIC X (*n*)	CHARACTER**n*	*n* binäre Zeichen

Die Pfeile bezeichnen die Richtung des Datenflusses:

→ das Feld ist vom rufenden Programm zu versorgen

← das Feld wird vom gerufenen Programm gefüllt

↔ sowohl rufendes als auch gerufenes Programm versorgen das Feld

Registerbelegung:

→ R1: Adresse der Parameterliste

→ R13: zeigt auf Sicherstellungsbereich (18 Worte)

→ R14: enthält die Rücksprungadresse

→ R15: enthält die Aufrufadresse

Parameter:

1 ←	**FILEID**	**F**	Kennung
2 ←	**RETCO**	**F**	Returncode
	= 0		Kein Fehler
			Einige gängige Fehlercodes (siehe auch Kapitel 8.1)
	= 1		Sätze verkürzt
	= 10		Datei ist keine FLAMFILE
	= 11		FLAMFILE Formatfehler
	= 12		Satzlängenfehler
	= 13		Dateilängenfehler
	= 14		Checksummenfehler
	= 21		Unzulässige Größe des Matrixpuffers
	= 22		Unzulässiges Kompressionsverfahren
	= 23		Unzulässiger Code in FLAMFILE
	= 24		Unzulässiger MAXRECORDS Parameter
	= 25		Unzulässige Satzlänge MAXSIZE
	= 29		Passwort-Fehler
	= 30		FLAMFILE ist leer
	= 31		FLAMFILE nicht zugeordnet
	= 33		Ungültiger Dateityp
	= 34		Ungültiges Satzformat
	= 35		Ungültige Satzlänge
	= 36		Ungültige Blocklänge
	= 37		Unzulässige Schlüsselposition (ungleich 1)
	= 38		Ungültige Schlüssellänge
	= 39		Ungültiger Dateiname
	= x'Exxxxxxx'		I/O-Fehler für Originaldatei bei Eingabe
	= x'Axxxxxxx'		I/O-Fehler für Originaldatei bei Ausgabe
	= x'Fxxxxxxx'		I/O-Fehler für Komprimatsdatei
	= x'Cxxxxxxx'		I/O-Fehler für Parameterdatei
	= x'Dxxxxxxx'		I/O-Fehler für Meldungsdatei

=	**x'xFxxxxxx'**	Fehler der Datenverwaltung (VSAM)
=	**40**	Modul oder Tabelle kann nicht geladen werden
=	**41**	Modul kann nicht aufgerufen werden
=	**42**	Modul kann nicht entladen werden
=	**43-49**	Fehlerabruch durch Exit
=	**52**	zu viele oder unzulässige Schlüssel
=	**57 - 79**	FLAM-Fehler
=	**80**	Syntaxfehler bei Parametereingabe
=	**81**	Unbekannter Parameter (Schlüsselwort)
=	**82**	Unbekannter Parameterwert
=	**83**	Parameterwert nicht dezimal
=	**84**	Parameterwert zu lang
=	**96**	Keinen Dateinamen gefunden, bzw. Fehler beim Ermitteln von Dateinamen
=	**98**	Nicht alle Dateien wurden bearbeitet
=	**999**	Fehler bei Speicheranforderung

3 →	**PARAM**	**CL*n***	Bereich mit Parametern

4 →	**PARLEN**	**F**	Länge des Parameterbereichs
	= **0**		keine Parameter vorhanden
	> **0**		Parameter vorhanden

Hinweis: Die Parameter müssen in der gleichen Weise wie beim Dienstprogramm geschrieben werden. Für Parameter sind nur Großbuchstaben zulässig.

Man kann FLAMUP mit den Parametern

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

aufrufen und die Dateien über DD-Statements in der JCL angeben. Es ist aber auch möglich, nur FLAMUP ohne Parameter aufzurufen und auch die Steuerungsparameter in eine Datei zu legen. Standardmäßig sucht FLAM nach der Datei mit DD-Namen FLAMPAR für Parameter. Aber auch das ist im Aufruf parametrisierbar (PARDD=ddname oder PARFILE=dsn). Ein- und Ausgabedateien lassen sich ggf. auch als Parameter angeben (FLAMIN=..,FLAMOUT=..,FLAMFILE=..). Die Meldungsausgabe erfolgt standardmäßig in eine Datei mit DD-Namen FLPRINT (Änderbar mit MSGDD=ddname oder MSGFILE=dsn).

Der EXIT muss in einer Bibliothek zu finden sein, die im Zugriff steht. D.h. in der STEPLIB-, JOBLIB oder LINKLIST-Verkettung. Dies ist IBM-Standard.

Empfehlenswert ist es, die zugehörigen Parameter dynamisch in eine Datei zu schreiben. Dies ist am einfachsten, das Coding bleibt überschaubar und ist hervorragend wartbar.

5.2.1.1 Aufruf FLAMUP mit ‚MSGFILE=dateiname, PARFILE=dateiname' als Parameter.

```
für Verschlüsselung:

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

analog für Entschlüsselung:

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

5.2.1.2 Beispiel für den Aufruf von FLAMUP in COBOL:

```
 IDENTIFICATION DIVISION.

 PROGRAM-ID. MUSTER.

*

* MUSTER FÜR DEN AUFRUF VON 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 Beispiel für den Aufruf von FLAMUP in ASSEMBLER:

```
MUSTER   CSECT
*
*   FLAMUP AUFRUFEN
*
         LA   1,FLAMUPAR
         L    15,=V(FLAMUP)
         BALR 14,15
*
*   PARAMETERLEISTE FÜR FLAMUP
*
FLAMUPAR DC  A(FLAMID)
         DC  A(RETCO)
         DC  A(PARAM)
         DC  A(X'80000000'+PARLEN)
*
*   PARAMETER FÜR 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 Beispiel für den Aufruf von FLAMUP 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) für die anderen Plattformen

Das Unterprogramm **flamup** hat die gleiche Funktionalität wie **flam4.exe:**

```
void flamup(
   void** Ptr,
```

```
   long*  Retco,
   char*  ParString,
   long*  ParStringLen);
```

Ptr: Anker für den internen Gebrauch, kann NULL sein.

Retco: Ergebnis des Aufrufs, entspricht dem Return-Code von **flam4.exe**

ParString: Parameter-String (siehe unten)

ParStringLen: Länge des Parameter-Strings

Der Parameterstring enthält die Parameter analog dem Aufruf von **flam4.exe**, wobei die einzelnen Parameter durch Kommata getrennt werden.

Beispiel:

Für den Aufruf *flam4 comp flamfile=DateiName flamin=EingabeDatei mode=adc*

lautet der ParStr für flamup: "*comp,flamfile=DateiName,flamin=EingabeDatei,mode=adc*"

Enthält ein Parameterwert Komata, so muss der Parameterwert in runden Klammern eingeschlossen werden.

Beispiel für eine durch Kommata getrennte Dateiliste:

flam4 … flamin=Datei1,Datei2,Datei3 …

wird für flamup zu: *"…,flamin=(Datei1,Datei2,Datei3),.."*

Daraus ergibt sich, dass Dateinamen keine Kommata enthalten dürfen.

Allgemein gilt: Jeder Parameterwert, der ein Komma enthält, muss in runden Klammern eingeschlossen werden. Innerhalb dieser äußeren Klammern (diese sind nicht Bestandteil des Parameterwertes!) dürfen keine unpaarigen runden Klammern auftreten. Das heißt, dass eine runde Klammer im String durch zwei (paarig) Klammern realisiert werden muss.

5.2.2.1 Die Parameter für den FLAM-Keymanagement-Exit (FKME)

Der FLAM-Keymanagement-EXIT (FKME) befindet sich in einer Library, die dynamisch nachgeladen wird, wobei der Name der Library und der Name der Funktion über Parameter angegeben werden können.

kmlib=*Name der Library*

kmexit=*Name der Funktion*

kmparam=*Parameter-String, der der Funktion übergeben wird*

Alle drei Parameter können zu einem einzigen String zusammengefasst werden:

kmexit=*Funktionsname(Libraryname)Parameterstring*

und bei Verwendung der Voreinstellung für kmlib:

kmexit=*Funktionsname()Parameterstring*

Voreinstellungen für WINDOWS:

kmlib=flamkme.dll

kmexit=flamkme

Enthält der Parameterstring Kommata, so ist er bei der Übergabe an flamup (s.o.) in runden Klammern einzuschließen:

kmexit=**(***Funktionsname((Libraryname))Parameterstring***)**

bzw.

kmparam=**(***Parameter-String***)**

in FLAM® können die Schlüsselworte abgekürzt werden, solange sie eindeutig bleiben:

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

5.2.2.2 Beispielaufruf für das FLAMUP

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

```
// Parameter für Kompression unter Verwendung der Defaulteinstellung für
// die DLL "flamkme.dll" und Funktion "flamkme"

strcpy( Param,
"logfile=MeldungsDatei,mode=aes,comp,flamfile=FlamfileName.adc,flamin=NameD
erDaten.dat,inrecformat=undef,kmparam=DasWeisNurDerExitHerstellerWasHierSte
henSoll" ) ParLen = strlen( Param );

flamup ( &Kennung, &Retco, Param, &ParLen );
if ( Retco != 0 ) {
 // Hier Fehler behandeln }

Für die Dekompression:

strcpy( Param,
"logfile=MeldungsDatei,deco,flamfile=FlamfileName.adc,flamout=NameDerDaten.
dat,outrecformat=undef,kmparam=DasWeisNurDerExitHerstellerWasHierStehenSoll
") ParLen = strlen( Param );


INRECFORMAT und OUTRECFORMAT sind natürlich von den verwendeten Daten
abhängig!

Soll eine spezielle DLL und oder Funktion verwendet werden, kann man obige
Parameter ergänzen:

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

Enthält kmparam Kommata, so muss der eigentliche Parameterstring in runde
Klammern gesetzt werden:

Der Parameter-String "EinWert,ZweiWerte,Drei" wird:
"(EinWert,ZweiWerte,Drei)"
Enthält er runde Klammern, so wird jede Klammer verdoppelt:
"EinWert(von Drei),ZweiWert,Drei(und Letzter)" wird zu "(EinWert((von
Drei)),ZweiWert,Drei((und Letzter)))"

Die Alternative "kmexit=Funkname(DllName)ParameterWerte" verhält sich
analog:
"kmexit=(Funkname((DllName))Parameter ((Werte)))"
Diese Möglichkeit ist aus Kompatibilitätsgründen zu früheren Parametern und
zu FLAM-UINIX aufgenommen worden.

Empfehlenswert ist die Version kmlib=..,kmexit=..,kmparam=...

Die Reihenfolge der Parameter ist beliebig, wobei "logfile=" als erster
Parameter Sinn macht, da dann alle Meldungen (auch Parameterfehler) sofort
in die Meldungsdatei protokolliert werden!
```

5.3 FLAM Implementierungsempfehlung

Damit die Implementierungen der verschiedenen EXIT von der Verwendung her sich ähnlich gestallten, werden im Folgenden einige Empfehlungen für die Implementierung gegeben, welche soweit es die jeweilige HSM Architektur zulässt, eingehalten werden sollten.

5.3.1 Behandlung von Generation und Version

Die Generation und Version beim Senden wird über ein Template aus dem Keylabel ermittelt. Hierfür werden die folgenden Ersetzungszeichen für das Template definiert:

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

Alle anderen Zeichen können durch ein ,%' ersetzt werden, damit die Position der Generation und Version im Label vom EXIT ermittelt werden kann. Alle anderen Zeichen müssen mit dem korrespondierenden Label übereinstimmen.

Beispiel: ,TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**' beim Senden.

Das Template fürs Empfangen darf kein ‚%' beinhalten, damit der Name für den Schlüssel vervollständigt werden kann.

Beispiel: ,TFMKY.BV000000.GUD00000.DAT0++**' beim Empfangen.

Beim Erzeugen einer FLAMFILE® wir somit immer ein vollständiges Label und ein Template an FLAM® für den EXIT übergeben. Wenn später auf eine FLAMFILE® zugegriffen werden muss, wird nur ein Template übergeben, wo ausschließlich die Generation und Version offen bleibt.

5.3.2 Übergabe der Input-Parameter über FLAM bzw. FLAMUP an den EXIT

Die Inputparameter für den EXIT werden als ein Parameter in der Parameterliste für FLAM® bzw. FLAMUP übergeben. Wie dies geschieht ist für die verschiedenen Plattformen unter 5.2 beschrieben. Hierbei gilt, dass die Parameter als ein String zusammengefasst werden müssen, welcher keine runden Klammern bzw. einfach Anführungszeichen (kann, wenn notwendig, durch eine weiter Klammer oder ein weiteres einfaches Anführungszeichen kenntlich gemacht werden) beinhalten darf. Hierbei sollten zuerst das ggf. erforderliche KeyLabel gefolgt von den KeyTemplates angegeben werden. Danach können optional die Identifikationsdaten und Authentifikationsdaten folgen. Alle Werte werden einfach durch ein Freizeichen getrennt angegeben. In Anlehnung an Kapitel 4.1.1.4 folgen nun in Abhängigkeit des Anwendungsfalls jeweils ein Beispiel für den Parameterstring ohne Anmeldeinformationen für das HSM.

- **VERSCHLÜSSELN**
  TFMKY.BV000000.GUD00000.DAT00601
  TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**

- **UMSCHLÜSSELN**
  TFMKY.BV000000.GUD00000.DAT0++**
  TFMKY.GUD00000.GUD00000.DAT00601
  TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**

- **ENTSCHLÜSSELN**
  TFMKY.GUD00000.GUD00000.DAT0++**

5.3.3 Die letzten 10 Byte im Info-Feld des FKMC

Die ersten 40 Byte der 50 Byte des Infofeldes sind unter 3.2.1.1 fest spezifiziert wurden. Die Restlichen 10 Byte stehen dem EXIT frei zur Verfügung. Hier ist es Empfehlenswert, wenn der EXIT ein Kennzeichen für seine Implementierung hinterlegt. Folgende Kennzeichen werden festgelegt:

- FLAMSTDxx
  Standardimplementierung in Software der LIMES Datentechnik
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 FLAMSTD01 “

- IBMCCCAxx
  IBM Implementierung gegen die SAPI der CCA
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 IBMCCCA01 “

- IBMDKMSxx
  IBM Implementierung gegen das DKMS General Purpose API
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 IBMDKMS01 “

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

- BVUTIMAxx
  UTIMACO Implementierung des Bankverlages
  „FKMC V001 L144 TDES KL16 EZ04 ICBC SHA1 BVUTIMA01 “

xx – Steht hierbei für die Version der Implementierung und sollte, wie in den Beispielen, mit 01 beginnen.

5.3.4 EBCDIC- und ASCII- Wandlung

Vom Zeichensatz ist nur das Infofeld (3.2.1.1) abhängig. Beim Erzeugen der FLAMFILE® wird das Infofeld in Abhängigkeit der Plattform des Senders gefüllt. Damit muss der

Empfänger prüfen, in welchen Zeichensatz die Infodaten eingestellt wurden, und ggf. eine entsprechende Wandlung vornehmen.

5.3.5 Nutzung von Messages

5.3.5.1 Fehlermeldung des Exits

Die folgenden Meldungen sollten beim Auftreten des entsprechenden Fehlers herangezogen werden.

- 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

Alle weiteren Fehler müssen mit ‘FKME - ‘ beginnen und können andere Meldungen in englischer Sprache gefolgt von entsprechenden Fehlerinformationen der jeweiligen Subsysteme oder des EXITS beinhalten.

5.3.5.2 Gutmeldung des Exits

Wenn kein Fehler bei der Ausführung des jeweiligen Funktionscodes aufgetreten ist, wird immer eine so genannte Gutmeldung erzeugt, welche der Protokollierung dient. Sie setzt sich zusammen aus dem Funktionscode, dem Zeitstempel und der Zufallszahl.

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

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

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

Durch die Protokollierung des Zeitstempels und der Zufallszahl werden neben den entsprechend möglichen Kontrollen und Quittierungen auch die Vorgaben von VISA und MasterCard erfüllt.

5.3.5.3 Selbstauskunft des Exits

Die ersten 50 Byte sollten dem Infofeld des FKMC entsprechen. Danach sollten weitere nützliche Informationen über die jeweilige Implementierung des EXIT folgen.

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

6 Abkürzungsverzeichnis

AES = Advanced Encryption Standard

BIN = Binär

CBC = Cipher Block Chaining

CHNG = Change

CHR = Character

COMP = Compression

DAT = Daten

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 = Funktionscode

GG = Generation

GS = Generierungsstelle

HSM = Hardware Security Module

ICBC = CBC mit IV

INT = Integer

IV = Initial Vector

KL = Key Length

KTV = Key Test Value

LG = Länge

MAC = Message Authentification Code

MDC = Message Digest Cipher

MSG = Message

PAR = Parameter

PARAM = Parameter

PCI DSS = Payment Crad Industry Data Security Standard

PIN = Personal Identification number

POV = BCD Kodierung

PS = Personalisierungsstelle

RAM = Random Access Memory

RETCO = Returncode

SKMS = Static Key Management System

STR = String

TDES = Triple DES

VV = Version

ZKA = Zentraler Kredit Ausschuss