Deutsch

FUJITSU Software BS2000

CRYPT V2.0

Sicherheit mit Kryptographie

Benutzerhandbuch

Ausgabe Juli 2017

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2008 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Inhalt

1 Einleitung

CRYPT ist ein Subsystem, das kryptographische Funktionen und Schnittstellen im BS2000 zur Verfügung stellt.

1.1 Zielsetzung und Zielgruppen des Handbuchs

Das vorliegende Handbuch wendet sich an Systemadministratoren und Anwender der kryptographischen Schnittstelle im BS2000.

Für eine kryptographisch sichere Anwendungsprogrammierung wird das Verständnis der eingesetzten kryptographischen Funktionen vorausgesetzt.

1.2 Sicherheitshinweis

HAFTUNGSAUSSCHLUSS!

Wenn Sie im Rahmen eines Projektes eine Sicherheitsarchitektur implementieren, dann konfigurieren und benutzen Sie die auswählbaren Funktionen und Mechanismen gemäß ihren tatsächlichen Anforderungen.
Fujitsu Technology Solutions übernimmt keinerlei Verantwortung für jegliche Art von Zwischenfällen oder Schäden, die aufgrund von falsch gesetzten Parameterwerten oder fehlerhaft benutzten Mechanismen oder Funktionen der CRYPT-Schnittstellen eintreten.

1.3 Konzept des Handbuchs

Das vorliegende Handbuch beschreibt die Architektur von CRYPT und die kryptographischen Funktionen und Schnittstellen im BS2000.

Es basiert auf der Beschreibung des Standards „PKCS#11 V2.20: Cryptographic Token Interface Standard“ der RSA Laboratories von Dezember 1999. Diese Spezifikation ist frei erhältlich im Internet unter *http://germany.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm* und ist für den richtigen Einsatz der Parameter erforderlich. An allen Stellen im Handbuch, wo noch zusätzliche Informationen aus dem Standard hilfreich sind, finden Sie Verweise auf die entsprechenden Kapitel oder Abschnitte der PKCS#11-Spezifikation.

Kapitel „Beschreibung der Assembler-Makroaufrufe“ auf Seite 27 erläutert ausführlich die Assembler-Makroaufrufe anhand von Syntaxdiagrammen und detaillierten Operandenbeschreibungen. Ein Programmbeispiel rundet das Kapitel thematisch ab. Die PKCS#11-Spezifikation liefert vertiefende Zusatzinformationen.

Kapitel „Beschreibung der Funktionen in C“ auf Seite 93 dient als Orientierungshilfe für den Anwender und zum schnellen Auffinden der detaillierten Beschreibung in der PKCS#11-Spezifikation. Das Kapitel gibt einen Überblick über diejenigen Funktionen des Standards, die in der CRYPT-Schnittstelle in der Sprache C implementiert wurden, und verweist jeweils auf die zugehörigen Kapitel oder Abschnitte in der PKCS#11-Spezifikation. Außerdem enthält Kapitel 8 ein zusammenhängendes Programmbeispiel.

Kapitel „Diagnose-Unterlagen erstellen“ auf Seite 103 erläutert die Erstellung von Diagnose-Unterlagen.

Für die Arbeit mit der C-Schnittstelle ist die PKCS#11-Spezifikation unbedingt erforderlich.

Readme-Datei

Funktionelle Änderungen der aktuellen Produktversion und Nachträge zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter *http://manuals.ts.fujitsu.com* zur Verfügung. Alternativ finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen unter BS2000

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie im BS2000-System die folgende Datei:

`SYSRME.<product>.<version>.<lang>`

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen.
Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemitteilung. Solche Freigabemitteilungen finden Sie online unter *http://manuals.ts.fujitsu.com*.

1.4 Änderungen gegenüber dem Vorgänger-Handbuch

Das vorliegende Handbuch beschreibt die Funktionalität von CRYPT V2.0A. Gegenüber der letzten Ausgabe dieses Handbuchs ergeben sich folgende Änderungen:

Zur Ausführung der Funktionen von CRYPT wird keine Hardware mehr verwendet. Die Funktionen werden im BS2000 ohne Konfiguration des Subsystems ausgeführt.

1.5 Darstellungsmittel

Im vorliegenden Handbuch werden die folgenden typografischen Gestaltungsmittel verwendet:

zur Kennzeichnung von Hinweistexten

VORSICHT!

zur Kennzeichnung von Sicherheits- und Warnhinweisen

`dicktengleiche Schrift` Programmtext in Beispielen

kursive Schrift Namen von kryptographischen Funktionen und Operationen (in C), Funktionsparametern und Dateien im beschreibenden Text; Syntaxvariable

Die Gestaltungsmittel für die Beschreibung der Assembler-Makroaufrufe werden in den Abschnitten „Metasyntax für Makros“ auf Seite 28 und „Makrosyntax für Formatoperanden“ auf Seite 30 vorgestellt.

Verweise innerhalb des Handbuchs geben die betreffende Seite im Handbuch und je nach Bedarf auch den Abschnitt bzw. das Kapitel an. Verweise auf Themen, die in einem anderen Handbuch beschrieben sind, enthalten den Kurztitel des betreffenden Handbuchs. Den vollständigen Titel finden Sie im Literaturverzeichnis.

2 Verschlüsselung im BS2000

2.1 PKCS#11-Standard

RSA Laboratories haben in Zusammenarbeit mit Entwicklern von Sicherheitssystemen aus Industrie, Hochschulen und Regierung Spezifikationen entwickelt, mit dem Ziel die Entwicklung von Ver- und Entschlüsselung über öffentliche Schlüssel zu beschleunigen.
Diese Spezifikationen sind die sogenannten Public-Key Cryptography Standards, kurz PKCS. Beiträge aus dieser PKCS-Reihe sind mittlerweile Bestandteil von vielen formalen und De-facto-Standards wie ANSI X9 Dokumenten, PKIX, SET, S/MIME und SSL.

PKCS#11 ist der Cryptographic Token Interface Standard. Das Produkt CRYPT basiert auf der Version 2.10 von PKCS#11.
Darüber hinaus umfasst CRYPT Funktionen, wie z.B. die AES-Unterstützung, die auf der neueren Version 2.11 von PKCS#11 basieren.

Dieser De-facto-Standard spezifiziert eine Programm-Schnittstelle (API) zu Geräten, die kryptographische Informationen speichern und kryptographische Funktionen ausführen. Die Kurzbezeichnung für diese Schnittstelle – Cryptographic Token Interface – lautet Cryptoki.

Die Spezifikation zum PKCS#11-Standard finden Sie im Internet z.B. unter:

http://germany.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

3 PKCS#11-Funktionen und entsprechende Schnittstellen im BS2000

Die nachfolgende Tabelle gibt einen Überblick darüber, welchen BS2000-Schnittstellen die einzelnen PKCS#11-Funktionen zugeordnet sind.

Erläuterungen zur Tabelle

- Das Präfix C_ bei den PKCS#11-Funktionen steht für Funktion.
- „nein“ in der Spalte „Funktionalität verfügbar“:
  Die Schnittstellen sind aus Portabilitätsgründen im BS2000 implementiert, die Makroaufrufe sind jedoch funktionslos. Da für diese Funktion lediglich eine Schnittstelle ohne die zugehörige Funktionalität zur Verfügung steht, wird ein Aufruf mit MAINCODE = LINKAGE_ERROR und SUBCODE1 = FCT_NOT_AVAILABLE beantwortet.

PKCS#11-Funktion	**BS2000-Schnittstelle (SVC/ISL)**	**Funktionalität verfügbar**
Allgemeine Funktionen		
C_Initialize	CGENRAL	ja
C_Finalize	CGENRAL	ja
C_GetInfo	CGENRAL	ja
C_GetFunctionList	CGENRAL	ja, über C-Schnittstelle
Funktionen für die Slot- und Token-Verwaltung		
C_GetSlotList	CGTSTMI	ja
C_GetSlotInfo	CGTSTMI	ja
C_GetTokenInfo	CGTSTMI	ja
C_WaitForSlotEvent	CWTFSLE	nein
C_GetMechanismList	CGTSTMI	ja
C_GetMechanismInfo	CGTSTMI	ja

PKCS#11-Funktionen und entsprechende Schnittstellen im BS2000 (Teil 1 von 4)

PKCS#11-Funktion	BS2000-Schnittstelle (SVC/ISL)	Funktionalität verfügbar
C_InitToken	CINITTK	nein
C_InitPIN	CPIN	nein
C_SetPIN	CPIN	nein
Funktionen für die Sitzungsverwaltung		
C_OpenSession	CSESION	ja
C_CloseSession	CSESION	ja
C_CloseAllSessions	CSESION	ja
C_GetSessionInfo	CSESION	nein
C_GetOperationState	COPSTAT	nein
C_SetOperationState	COPSTAT	nein
C_Login	CLOG	ja
C_Logout	CLOG	ja
Funktionen für die Objektverwaltung		
C_CreateObject	COBJMGT	ja
C_CopyObject	COBJMGT	ja
C_DestroyObject	COBJMGT	ja
C_GetObjectSize	COBJMGT	nein
C_GetAttributeValue	COBJMGT	ja
C_SetAttributeValue	COBJMGT	ja
C_FindObjectsInit	COBJMGT	ja
C_FindObjects	COBJMGT	ja
C_FindObjectsFinal	COBJMGT	ja
Verschlüsselungsfunktionen		
C_EncryptInit	CCRYINI	ja
C_Encrypt	CCRY	ja
C_EncryptUpdate	CCRY	ja
C_EncryptFinal	CCRYFIN	ja
Entschlüsselungsfunktionen		
C_DecryptInit	CCRYINI	ja

PKCS#11-Funktionen und entsprechende Schnittstellen im BS2000 (Teil 2 von 4)

PKCS#11-Funktion	BS2000-Schnittstelle (SVC/ISL)	Funktionalität verfügbar
C_Decrypt	CCRY	ja
C_DecryptUpdate	CCRY	ja
C_DecryptFinal	CCRYFIN	ja
Funktionen für den Message-Digest		
C_DigestInit	CCRYINI	ja
C_Digest	CCRY	ja
C_DigestUpdate	CCRY	ja
C_DigestKey	CCRYINI	ja
C_DigestFinal	CCRYFIN	ja
Funktionen zum Signieren und MACing		
C_SignInit	CCRYINI	ja
C_Sign	CCRY	ja
C_SignUpdate	CCRY	ja
C_SignFinal	CCRYFIN	ja
C_SignRecoverInit	CCRYINI	nein
C_SignRecover	CCRY	nein
Funktionen zum Überprüfen von Signaturen und MACs		
C_VerifyInit	CCRYINI	ja
C_Verify	CCRY	ja
C_VerifyUpdate	CCRY	ja
C_VerifyFinal	CCRYFIN	ja
C_VerifyRecoverInit	CCRYINI	ja
C_VerifyRecover	CCRY	ja
Kryptographische Funktionen mit Doppelfunktion		
C_DigestEncryptUpdate	CCRY	nein
C_DecryptDigestUpdate	CCRY	nein
C_SignEncryptUpdate	CCRY	nein
C_DecryptVerifyUpdate	CCRY	nein

PKCS#11-Funktionen und entsprechende Schnittstellen im BS2000 (Teil 3 von 4)

PKCS#11-Funktion	BS2000-Schnittstelle (SVC/ISL)	Funktionalität verfügbar
Funktionen für die Schlüsselverwaltung		
C_GenerateKey	CGENKEY	ja
C_GenerateKeyPair	CGENKPR	ja
C_WrapKey	CWRPKEY	ja
C_UnwrapKey	CUNWKEY	ja
C_DeriveKey	CDRVKEY	ja
Funktionen zur Generierung von Zufallszahlen		
C_SeedRandom	CRANDOM	ja
C_GenerateRandom	CRANDOM	ja
Funktionen zur parallelen Funktionsverwaltung		
C_GetFunctionStatus	–	nein
C_CancelFunction	–	nein
Rückruf-Funktionen		
Surrender callbacks: Callbacks übergeben	–	nein
Vendor-defined callbacks: Anwenderdefinierte Call-backs	–	nein

PKCS#11-Funktionen und entsprechende Schnittstellen im BS2000 (Teil 4 von 4)

4 PKCS#11-Mechanismen in CRYPT

Ein Mechanismus ist ein Prozess zur Implementierung von kryptographischen Operationen.

Das Subsystem CRYPT umfasst die im Folgenden aufgelisteten Mechanismen des PKCS#11-Standards.

Die Erläuterungen der einzelnen Mechanismen finden Sie im Kapitel „Fachwörter“ auf Seite 113 sowie im Kapitel 12 „Mechanisms“ der PKCS#11-Spezifikation.
Das Präfix CKM_ in der Spezifikation steht für Mechanismustyp.

Symmetrische Algorithmen

- Blockchiffren
  - DES
  - DES3
  - SD2 (entspricht RC2)
  - AES
- Betriebsarten
  - ECB
  - CBC
  - OFB
- Stromchiffren
  - SD4 (entspricht RC4)
- Schlüsselerzeugung für die unterstützten Algorithmen

Hash-Algorithmen und Integritätscodes

MD2, MD5, SHA-1, RIPEMD160, HMAC-MD5, HMAC-SHA

Public-Key-Verfahren

- Chiffren/Schlüsselaustauschverfahren
  - RSA (PKCS#1)
  - RSA (pure)
  - Diffie-Hellman
- Signaturverfahren
  - RSA (PKCS#1)
  - RSA (pure)
  - DSA
- Schlüsselerzeugung für die unterstützten Algorithmen

Zufallsgeneratoren

Pseudo-Zufallsgeneratoren DES-, DES3-OFB

5 PKCS#11 - Implementierung im BS2000

Dieses Kapitel beschreibt alle Besonderheiten der CRYPT-Schnittstelle im BS2000, die von den Vorgaben des PKCS#11-Standards abweichen.

In der PKCS#11-Spezifikation werden zahlreiche Präfixe verwendet. Die nachfolgende Tabelle listet die Präfixe mit ihrer Bedeutung auf, die in diesem Kapitel vorkommen. Darüber hinaus finden Sie die Erläuterung weiterer Präfixe in der PKCS#11-Spezifikation im Kapitel 5 „Symbols and Abbreviations“.

Präfix	**Bedeutung**
C_	Funktion
CKA_	Attribute
CKM_	Mechanismustyp
CKR_	Returncode

5.1 Mechanismen

Ein Mechanismus spezifiziert, wie ein bestimmter kryptographischer Prozess auszuführen ist.

Die folgende Tabelle zeigt, welche cryptoki-Mechanismen von welchen verschiedenen kryptographischen Operationen unterstützt werden. Sie ersetzt die Tabelle am Beginn des Kapitels 12 „Mechanisms“ der Spezifikation PKCS#11 V2.20.

XX: Die Funktion(en) wird/werden unterstützt.
X: Einteilige Operationen werden unterstützt.
Wrap und Unwrap sind nur auf geheimen Schlüsseln möglich.

Mechanismen	Funktionen						
	Encrypt & Decrypt	**Sign & Verify**	**Sign- / Verify-Recover**	**Digest**	**Genera-te Key/ KeyPair**	**Wrap & Unwrap**	**Derive**
CKM_RSA_PKCS_KEY_PAIR_ GEN					xx		
CKM_RSA_9796	x	x	x			xx	
CKM_RSA_PKCS	x	x	x			xx	
CKM_RSA_X_509	x	x	x			xx	
CKM_MD2_RSA_PKCS		xx					
CKM_MD5_RSA_PKCS		xx					
CKM_SHA1_RSA_PKCS		xx					
CKM_RIPEMD160_RSA_PKCS		xx					
CKM_DSA_KEY_PAIR_GEN					xx		
CKM_DSA		x					
CKM_DSA_SHA1		xx					
CKM_DH_PKCS_KEY_PAIR_GEN					xx		
CKM_DH_PKCS_DERIVE							xx
CKM_RC2_KEY_GEN					xx		
CKM_RC2_ECB	xx					xx	
CKM_RC2_CBC	xx					xx	
CKM_RC2_CBC_PAD	xx					xx	
CKM_RC2_MAC_GENERAL		xx					
CKM_RC2_MAC		xx					

Mechanismen und Funktionen, die diese Mechanismen unterstützen (Teil 1 von 3)

Mechanismen	Funktionen						
	Encrypt & Decrypt	**Sign & Verify**	**Sign- / Verify-Recover**	**Digest**	**Genera-te Key/ KeyPair**	**Wrap & Unwrap**	**Derive**
CKM_RC4_KEY_GEN					xx		
CKM_RC4	xx						
CKM_DES_KEY_GEN					xx		
CKM_DES_ECB	xx					xx	
CKM_DES_CBC	xx					xx	
CKM_DES_CBC_PAD	xx					xx	
CKM_DES_MAC_GENERAL		xx					
CKM_DES_MAC		xx					
CKM_DES2_KEY_GEN					xx		
CKM_DES3_KEY_GEN					xx		
CKM_DES3_ECB	xx					xx	
CKM_DES3_CBC	xx					xx	
CKM_DES3_CBC_PAD	xx					xx	
CKM_DES3_MAC_GENERAL		xx					
CKM_DES3_MAC		xx					
CKM_MD2				xx			
CKM_MD2_HMAC_GENERAL		xx					
CKM_MD2_HMAC		xx					
CKM_MD5				xx			
CKM_MD5_HMAC_GENERAL		xx					
CKM_MD5_HMAC		xx					
CKM_SHA_1				xx			
CKM_SHA_1_HMAC_GENERAL		xx					
CKM_SHA_1_HMAC		xx					
CKM_RIPEMD160				xx			
CKM_RIPEMD160_HMAC_ GENERAL		xx					
CKM_RIPEMD160_HMAC		xx					
CKM_AES_KEY_GEN					xx		
CKM_AES_ECB	xx					xx	

Mechanismen und Funktionen, die diese Mechanismen unterstützen (Teil 2 von 3)

Mechanismen	Funktionen						
	Encrypt & Decrypt	**Sign & Verify**	**Sign- / Verify-Recover**	**Digest**	**Genera-te Key/ KeyPair**	**Wrap & Unwrap**	**Derive**
CKM_AES_CBC	xx					xx	
CKM_AES_CBC_PAD	xx					xx	
CKM_AES_MAC_GENERAL		xx					
CKM_AES_MAC		xx					

Mechanismen und Funktionen, die diese Mechanismen unterstützen (Teil 3 von 3)

5.2 Allgemeine Datentypen

slotId

In CRYPT V2.0 ist die SlotId bedeutungslos. In den Funktionen, die eine SlotId verlangen, braucht diese nicht versorgt zu werden.

5.3 Objekte

Attribute bei Erzeugung von Objekten

Die Applikation hat dafür Sorge zu tragen, dass die übergebenen Templates korrekt und vollständig sind. Ein fehlerhaftes oder unvollständiges Attribut-Template bei Erzeugen eines Objekts wird nicht in jedem Fall erkannt und ergänzt (siehe auch Spezifikation PKCS#11 V2.20: Kapitel 10).

Dies betrifft folgende Funktionen:
- C_CreateObject
- C_CopyObject
- C_GenerateKey
- C_GenerateKeyPair
- C_UnwrapKey
- C_DeriveKey

Attribute CKA_SENSITIVE, CKA_EXTRACTABLE, CKA_LOCAL, CKA_TOKEN

Die Hinweise in diesem Abschnitt beziehen sich auf den Abschnitt 10.4 „Storage objects“ in der Spezifikation PKCS#11 V2.20.

Die persistente Speicherung von Schlüsseln und anderen Objekten wird nicht unterstützt. Das Attribut CKA_TOKEN muss daher immer auf FALSE gesetzt sein.

VORSICHT!

Kein durchgängiger Schutz der geheimen Daten eines Objekts.
Zwar verhindern die Attribute CKA_SENSITIVE und CKA_EXTRACTABLE das Auslesen der geheimen Daten mit *C_GetAttributeValue* bzw. den Export dieser Daten mit *C_WrapKey*. Jedoch werden die diesbezüglich gesetzten Flags von den anderen Cryptoki-Funktionen ignoriert, sodass sich die Schutzfunktion umgehen lässt. Auch das Attribut CKA_LOCAL ist nicht immer korrekt gesetzt.

5.4 Funktionen

Eine Operation ist eine Abfolge von mehreren Funktionen.

Die Hinweise in diesem Abschnitt beziehen sich auf Kapitel 11 „Functions“ der Spezifikation PKCS#11 V2.20.

- Im BS2000 können die meisten Funktionen ab der Version 1.1 über die BS2000-spezifischen Assembler-Schnittstellen nicht nur synchron, sondern auch asynchron ausgeführt werden.

  Detailliertere Information dazu entnehmen Sie dem Abschnitt „Asynchrone Ausführung“ auf Seite 32.

- Folgende allgemeine Funktionen sind im BS2000 nicht erforderlich:

  C_InitToken

  C_Login

  C_Logout

- Die maximale Ausgabe-Datenlänge bei den Funktionen *encryptFinal*, *decryptFinal*, *digestFinal*, *signFinal*, *verifyFinal*, *wrapKey* und *generateRandom* beträgt 2048 byte.

- Lassen Sie sich bei bestimmten Funktionen **nicht** zunächst die Größe des Ausgabebereichs ermitteln. Dies geht zu Lasten der Performance. Vergleichen Sie dazu den Abschnitt 11.2 „Conventions for functions returning output in a variable-length buffer“ in der Spezifikation PKCS#11 V2.20.

- Alle mit *...Init* eingeleiteten Operationen werden durch Folgeaufrufe, die den Returncode CKR_SESSION_HANDLE_INVALID oder CKR_ARGUMENTS_BAD liefern, **nicht** beendet. Auch die Returncodes CKR_KEY_HANDLE_INVALID, CKR_MECHANISM_INVALID, CKR_ATTRIBUTE_VALUE_INVALID beenden in der Regel **nicht** die aktive Operation. Vergleichen Sie dazu auch Abschnitt 11.4 „General-purpose functions“ in der Spezifikation PKCS#11 V2.20.

- Die Anzahl der Sessions, die ein BS2000-Nutzer gleichzeitig eröffnen kann, ist auf 999 begrenzt. Diese Obergrenze mit dem Kommando ADD-USER, bzw. MODIFY-USER-ATTRIBUTES durch den Operanden CRYPTO-SESSION-LIMIT auch auf einen niedrigeren Wert festlegen. Weitere Informationen zu diesen Kommandos finden Sie im Benutzerhandbuch „BS2000 OSD/BC Kommandos“ [2].

  Das Überschreiten der Maximalzahl paralleler Sessions wird mit dem Returncode session_count angezeigt.

C_Initialize, C_Finalize:

- C_Initialize:
  Zusätzlich zu der in der PKCS#11-Spezifikation beschriebenen Funktionalität wird im BS2000 mit C_Initialize gesteuert, ob das Programm synchron oder asynchron mit CRYPT arbeiten will:

  Wenn Sie synchrone Funktionsausführung wählen, ist ein C_Initialize im BS2000 nicht erforderlich.
  pInitArgs muss ein NULL_PTR sein.
- C_Finalize:
  C_Finalize ist im BS2000 wirkungslos.

Vergleichen Sie hierzu Abschnitt 11.4 „General-purpose functions“ in der Spezifikation PKCS#11 V2.20.

C_GetMechanismInfo:

In den Mechanismus-Infodaten der Mechanismen CKM_RSA_PKCS und CKM_RSA_X_509 sind die Flags für die Operationen *Sign* und *Verify* nicht gesetzt. Die entsprechenden Operationen werden dennoch unterstützt.
Vergleichen Sie dazu Abschnitt 11.5 „Slot and token management functions“ in der Spezifikation PKCS#11 V2.20.

C_CopyObject:

Die Flags CKA_SENSITIVE und CKA_EXTRACTABLE, die die sicherheitsrelevanten Daten eines Schlüssels gegen Auslesen schützen bzw. das Einpacken verbieten, können in beide Richtungen geändert werden.
Vergleichen Sie dazu Abschnitt 11.7 „Object management functions“ in der Spezifikation PKCS#11 V2.20.

C_SetAttributeValue:

VORSICHT!

Keine vollständige Prüfung der übergebenen Werte.
Möglicherweise treten inkonsistente Zustände auf, da es möglich ist, fehlerhafte Attribut-Werte zu setzen, und Attribute zu ändern, deren Modifikation laut Spezifikation verboten ist.
Die aufrufende Applikation muss sicherstellen, dass solche Zustände nicht auftreten.

C_GenerateKeyPair:

Für die Generierung von RSA-Schlüsseln ist die Angabe des Attributs CKA_PUBLIC_EXPONENT nicht erforderlich. Vergleichen Sie dazu den Abschnitt 11.14 „Key management functions“ in der Spezifikation PKCS#11 V2.20.

C_Encrypt, C_Decrypt, C_Digest, C_Sign, C_Verify

Die kryptographischen einteiligen Operationen (*C_Encrypt*, *C_Decrypt*, *C_Digest*, *C_Sign*, *C_Verify*) entsprechen einer Update-Operation gefolgt von einer Final-Operation.

Daher können Sie beispielsweise eine Folge von *C_EncryptUpdate*-Aufrufen wahlweise durch *C_EncryptFinal* oder durch *C_Encrypt* abschließen.

Einteilige Operation	**entsprechende Update-Operation + Final-Operation**
C_Encrypt	C_EncryptUpdate + C_EncryptFinal
C_Decrypt	C_DecryptUpdate + C_DecryptFinal
C_Digest	C_DigestUpdate + C_DigestFinal
C_Sign	C_SignUpdate + C_SignFinal
C_Verify	C_VerifyUpdate + C_VerifyFinal

Die Eingabe-Datenlänge einer Update-Operation muss nicht unbedingt den in der Spezifikation PKCS#11 V2.20 in Kapitel 12ff beschriebenen Kriterien genügen. Diese Kriterien beziehen sich nur auf die Gesamtlänge.

Vergleichen Sie dazu die Abschnitte 11.8 bis 11.12 „Encryption / Decryption / Message digesting / Signing and MACing functions and functions for verifying signatures and MACs“ in der Spezifikation PKCS#11 V2.20.

6 Beschreibung der Assembler-Makroaufrufe

Der einleitende Teil dieses Kapitels gibt Informationen zu folgenden Themen:

- Metasyntax für Makros
- Makrosyntax für Formatoperanden

Darüber hinaus finden Sie im Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3] detaillierte Informationen zu

- Registerverwendung
- Rückinformation und Fehleranzeigen (Returncodes) und deren Übergabe im Standardheader
- Standardheader
- ereignisgesteuerter Verarbeitung (Eventing)
- Contingency-Prozessen

Im Anschluss an den einleitenden Teil sind die einzelnen Makroaufrufe einzeln jeweils mit Syntaxdiagramm und Operandenbeschreibungen für den Anwender dargestellt.
Zusätzlich finden Sie jeweils einen Verweis auf die zugehörige ausführliche Beschreibung in der PKCS#11-Spezifikation.

6.1 Metasyntax für Makros

Kennzeichnung	Bedeutung	Beispiel
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter oder Konstanten, die in dieser Form vom Benutzer angegeben werden müssen. Schlüsselwörter beginnen mit *.	ACTION=*INITIALIZE
Kleinbuchstaben	Kleinbuchstaben bezeichnen Typen der Werte oder Variablen, die bei der Eingabe vom Benutzer durch aktuelle Werte ersetzt werden müssen, d.h. ihr Inhalt kann von Fall zu Fall verschieden sein.	AUTKEY=<var: int:4>
=	Das Gleichheitszeichen verbindet einen Operanden mit dem dazugehörigen Operandenwert.	SLOTID =<var: int:4>
/	Der Schrägstrich trennt alternative Operandenwerte.	ACTION= *OPENSESSION / *CLOSESESSION
< >	Spitze Klammern kennzeichnen den Datentyp des Operanden.	<var: int:4>
Unterstreichung	Die Unterstreichung hebt den Standardwert hervor. Das ist der Wert, den das System einsetzt, wenn der Benutzer keine Angabe macht (=Voreinstellung); Standardwerte die das Zeichen „_“ enthalten, werden anstelle der Unterstreichung durch ein vorangestelltes „default:“ gekennzeichnet.	TEMPL = <var: pointer> / NULL Keine Angabe impliziert TEMPL=NULL

Metasyntax für Makros

Datentypen der Operandenwerte

Datentyp	Zeichenvorrat	Besonderheiten
integer	0..9,+,-	„+“ bzw. „-“ kann nur erstes Zeichen sein. Der Zusatz n..m beschreibt den zulässigen Wertebereich. Beispiel: im Syntaxdiagramm: SESSION=<integer 0 .. 2147483647> bei der Eingabe: SESSION=5
var:		Leitet eine variable Angabe ein. Nach dem Doppelpunkt folgt der Datentyp der Variablen. Beispiel: im Syntaxdiagramm: LEN=<var: int:4> bei der Eingabe: LEN=100

Datentypen der Variablen und Registerinhalte

integer (n)	Kennzeichnet eine Ganzzahl, die n byte belegt, wobei n<=4 gilt. Fehlt die Längenangabe, wird n=1 angenommen.
enum NAME(n)	Kennzeichnet eine Aufzählung, die n byte belegt, wobei n<=4 gilt. Fehlt die Längenangabe, wird n=1 angenommen.
pointer	Zeiger (die Adresse wird übergeben).

6.2 Makrosyntax für Formatoperanden

Die Makro-Operanden können in zwei Gruppen eingeteilt werden:

Steueroperanden	Operanden, die die Form und die Generierung des Makros festlegen.
Funktionsoperanden	schnittstellen-spezifische Operanden

Steueroperanden

MF	steuert die Code-Generierung („Makro-Form“)
PREFIX	steuert die Generierung der Namen (erster Buchstabe)
MACID	steuert die Generierung der Namen (zweiter bis vierter Buchstabe)
PARAM	steuert die Adressierung des Parameterbereichs
XPAND	steuert die Expansion einzelner Datenstrukturen

Makroformen

Der Operand MF legt die Form des Makros fest, er kann folgende Werte annehmen:

MF = C | D | L | E | M

MF = C Das Layout der Datenstruktur (in der Regel der Parameterbereich) wird erzeugt, wobei jedes Feld und jedes Equate benannt sind. Diese Datenstruktur wird Teil des aktuellen Programmabschnitts (CSECT/DSECT).

Die Funktionsoperanden des Makros werden nicht ausgewertet.

PREFIX

Der Operand PREFIX dient zur Generierung der zu erzeugenden Namen. PREFIX, der genau ein Buchstabe ist, wird als erster Buchstabe aller Namen verwendet. Default-Wert ist der Kennbuchstabe der Funktionseinheit, dem der Makro angehört. Um Namensgleichheit zu vermeiden, ist PREFIX zu verwenden, wenn dieselbe Datenstruktur mehrfach innerhalb eines Moduls verwendet wird.

MACID

Der Operand MACID dient zur Generierung der zu erzeugenden Namen, und bestimmt das zweite bis vierte Zeichen des Namens. Default-Wert: zwei Zeichen als Entwicklungsgruppenkennzeichen und ein Zeichen als makrospezifisches Kennzeichen. Der Default-Wert garantiert die Kollisionsfreiheit der Namen innerhalb der Komponentengruppe.

MF = D wie bei MF = C; zusätzlich wird ein DSECT-Statement erzeugt. Der MACID-Operand wird ignoriert, d.h. der Default-Wert wird angenommen.

MF = L Erzeugt eine Instanz des Parameterbereichs unter Auswertung der Funktionsoperanden. Diese Makroform erzeugt keine Feldnamen; die Label-Angabe wird zur Benennung der erzeugten Konstanten verwendet.

MF = E Erzeugt die zum Aufruf der Funktion notwendigen Befehle. Die Funktionsoperanden werden ignoriert. Die Adressierung des Parameterbereichs muss mit dem Steueroperanden PARAM sichergestellt werden:

PARAM

PARAM=<adresse>

Adresse des Parameterbereichs als Name

PARAM=(<reg>)

Adresse des Parameterbereichs, im Register mit dem Namen <reg> enthalten.

MF = M Modifiziert einen durch Kopieren einer MF=L-Form vorher initialisierten Parameterbereich unter Auswertung der angegebenen Funktionsoperanden. Nicht angegebene Operanden bleiben im ursprünglichen Zustand.

Die Konsistenz des Parameterbereichs liegt in der Verantwortung des Makro-Aufrufers.

MF=M setzt voraus, dass die MF=D-Form oder MF=C-Form mit denselben Werten der PREFIX- und MACID-Operanden aufgerufen wurde, und dass zur Adressierung der DSECT (MF=D-Form) ein USING-Statement abgesetzt worden ist.

6.3 Asynchrone Ausführung

Im BS2000 können die meisten Funktionen ab der Version 1.1 nicht nur synchron, sondern auch asynchron ausgeführt werden. Möglich ist dies über die BS2000-spezifischen Schnittstellen.

Synchrone Ausführung

Bei der synchronen Ausführung erhält das Programm die Steuerung erst zurück, wenn die Funktion ausgeführt ist. Dabei kann es, z.B. bei der Generierung von Schlüsseln, zu Wartezeiten kommen. Der Vorteil der synchronen Ausführung ist die einfache Handhabung.

Asynchrone Ausführung

Bei der asynchronen Ausführung erhält das Programm die Steuerung zurück, bevor die Funktion vollständig ausgeführt ist. Das Programm kann so Wartezeiten ausnutzen.

Für eine asynchrone Ausführung gehen Sie folgendermaßen vor:

1. Definieren Sie mit dem Aufruf ENAEI ein Ereigniskennzeichen.
2. Wenn beim Eintreten eines Ereignisses eine Unterbrechungsroutine gestartet werden soll, definieren Sie diese Routine mit dem Aufruf ENACO.
3. Legen Sie mit dem Aufruf CGENRAL ACTION=*INITIALIZE, EXEC=*ASYNCHRON fest, dass die Task asynchron mit CRYPT arbeitet.
4. Geben Sie das Ereigniskennzeichen bei CRYPT-Makroaufrufen mit dem Operanden BOID=... an.

   Geben Sie darüber hinaus einen Postcode-Bereich (RPOSTAD=, RPOSTL=) an.

   CRYPT verwendet den Postcode1, der folgendermaßen aufgebaut ist:

ETC	SC1	MC2	MC1

ETC Event Code Type
lautet für CRYPT ’23’
Der Makroaufruf CPKC11T enthält hierfür einen symbolischen Wert.

SC1 Subcode1
entspricht dem Wert im Standardheader

MC2 Maincode2
entspricht dem Wert im Standardheader

MC1 Maincode1
entspricht dem Wert im Standardheader

Der Maincode X’8000’ – Parameterbereich nicht zugreifbar – kann nur im Postcode1 erscheinen: CRYPT konnte zum Zeitpunkt der Funktionsausführung nicht auf den Parameterbereich zugreifen.

Verwenden Sie den Postcode2, insbesondere bei Verwendung einer Contingency, für eine Beziehung zum zugehörigen Funktionsaufruf (siehe Abschnitt „Asynchrone Ausführung – Beispiel“ auf Seite 83).

Der Parameterbereich sowie die darin eventuell angegebenen Datenbereiche müssen schreibend zugreifbar sein, bis das Ende der Funktionsbearbeitung signalisiert ist.

Die Session bleibt für weitere Funktionsaufrufe gesperrt, bis das Ende der Funktionsbearbeitung signalisiert ist.

5. Für das Ereigniskennzeichen sendet CRYPT ein Signal, sobald die Ausführung beendet ist. Mit dem Aufruf SOLSIG kann das Programm das Signal zur Verarbeitung anfordern.

Weitere Einzelheiten zur ereignisgesteuerten Verarbeitung und den dazu benötigten Makroaufrufen entnehmen Sie bitte dem Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3].

6.4 Beschreibung der Makroaufrufe

Die einzelnen Makroaufrufe werden jeweils mit Syntaxdiagramm und Operandenbeschreibungen erläutert.
Für detailliertere Informationen finden Sie zusätzlich bei jedem Makroaufruf einen Verweis auf die zugehörige ausführliche Beschreibung in der PKCS#11-Spezifikation.
Das Präfix C_ bei den PKCS#11-Funktionen steht für Funktion.

Folgende Makroaufrufe sind funktionslos. Sie werden in CRYPT nicht unterstützt. Die Schnittstellen sind jedoch aus Portabilitätsgründen in BS2000 implementiert.

- CWTFSLE
- CINITTK
- CPIN
- COPSTAT

CGENRAL – Allgemeine Funktionen

Der Makro CGENRAL umfasst folgende allgemeine Funktionen:

- Cryptoki-Bibliothek initialisieren
- eine Anwendung mit der Cryptoki-Bibliothek beenden
- allgemeine Informationen über Cryptoki ausgeben
- Funktionsliste der Cryptoki-Bibliothek ausgeben

Alle Funktionen werden immer synchron ausgeführt

Eine detaillierte Beschreibung zu den Funktionen des Makros CGENRAL finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.4 „General-purpose functions“.

Makro	**Operanden**	
CGENRAL	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*INITIALIZE / *FINALIZE / *GETINFO / *GETFUNCTIONLIST / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,EXEC=	*SYNCHRON / *ASYNCHRON / <var: enum-of _exec_set:1> / default: _exec_set.synchron
	,INFO=	<var: pointer> / NULL
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*INITIALIZE entspricht der PKCS#11-Funktion *C_Initialize*;
initialisiert die Cryptoki-Bibliothek.

=*FINALIZE entspricht der PKCS#11-Funktion *C_Finalize*;
zeigt an, dass eine Anwendung mit der Cryptoki-Bibliothek ausgeführt wurde.

=*GETINFO entspricht der PKCS#11-Funktion *C_GetInfo*;
gibt allgemeine Information über Cryptoki aus.

=*GETFUNCTIONLIST
entspricht der PKCS#11-Funktion *C_GetFunctionList*;
gibt die Funktionsliste aus.

i Diese Funktion wird nicht unterstützt.

EXEC spezifiziert den Ausführungsmodus der folgenden Funktionen

=*SYNCHRON
gibt die Steuerung erst an den Aufrufenden zurück, nachdem die Funktion ausgeführt wurde.

=*ASYNCHRON
gibt die Steuerung an den Aufrufenden zurück, nachdem die Funktion an das Token gesandt wurde.

INFO
Art der ausgegebenen Information in Abhängigkeit von der Aktion:
- *INITIALIZE: INFO zeigt auf eine _INITIALIZE_ARGS-Struktur
- *FINALIZE: INFO sollte auf NULL_PTR gesetzt werden.
- *GETINFO: INFO zeigt auf den Speicherbereich, der die Information empfängt.
- *GETFUNCTIONLIST: INFO zeigt auf einen Wert, der einen Zeiger auf die _FUNCTION_LIST-Struktur der Bibliothek empfängt.

BOID
Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt
- bei asynchroner Ausführung:
  Ereigniskennung, die das Programm über die Terminierung von CRYPT informiert.

RPOSTAD
Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL
Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung:
  gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CGTSTMI – Information anzeigen

Der Makro CGTSTMI umfasst folgende Funktionen der Slot- und Token-Verwaltung:

- Liste der Slots im System ausgeben
- Information über einen einzelnen Slot im System ausgeben
- Information über ein einzelnes Token im System ausgeben
- Liste der Typen von Mechanismen ausgeben, die von einem Token unterstützt werden
- Information über einen einzelnen Mechanismus ausgeben, der von einem Token unterstützt wird

Die Funktionen C_GetMechanismList und C_GetMechanismInfo werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Die Funktionen C_GetSlotList, C_GetSlotInfo und C_GetTokenInfo werden immer synchron ausgeführt.

Eine detaillierte Beschreibung zu den Funktionen des Makros CGTSTMI finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.5 „Slot and token management functions“ unter „C_GetSlotList“, „C_GetSlotInfo“, „C_Get TokenInfo“, „C_-GetMechanismList“, „C_GetMechanismInfo“.

Makro	**Operanden**	
CGTSTMI	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*GETSLOTLIST / *GETSLOTINFO / *GETTOKENINFO / *GETMECHANISMLIST / *GETMECHANISMINFO / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,TOKNPRS=	<var: int:1> / *TRUE / *FALSE / 0
	,SLOTID=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,TYPE=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,INFO=	<var: pointer> / NULL
	,COUNT=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*GETSLOTLIST
entspricht der PKCS#11-Funktion *C_GetSlotList*;
gibt die Liste der Slots im System aus.

=*GETSLOTINFO
entspricht der PKCS#11-Funktion *C_GetSlotInfo*;
gibt Information über einen einzelnen Slot im System aus.

=*GETTOKENINFO
entspricht der PKCS#11-Funktion *C_GetTokenInfo*;
gibt Information über ein einzelnes Token im System aus.

=*GETMECHANISMLIST
entspricht der PKCS#11-Funktion *C_GetMechanismList*;
gibt die Liste der Typen von Mechanismen aus, die von einem Token unterstützt werden.

=*GETMECHANISMINFO
entspricht der PKCS#11-Funktion *C_GetMechanismInfo*;
gibt Information über einen einzelnen Mechanismus aus, der von einem Token unterstützt wird.

TOKNPRS Information über aktuelle Token;
ist nur für die Aktion *GETSLOTLIST relevant.

=*TRUE Ein Token muss vorhanden sein.

=*FALSE Es spielt keine Rolle, ob ein Token vorhanden ist oder nicht.

SLOTID ID des Slot

TYPE

Mechanismus-Typ: Wert von _mechanism_set (siehe auch Abschnitt „CPKC11T – Allgemeine Datentypen“ auf Seite 106);
ist nur für die Aktion *GETMECHANISMINFO relevant.

INFO

Art der ausgegebenen Information in Abhängigkeit von der Aktion:
- *GETSLOTLIST: NULL_PTR oder Pointer auf einen Speicherbereich, der die Slot-Liste empfängt.
- *GETSLOTINFO: INFO zeigt auf den Speicherbereich, der die Slot-Information empfängt.
- *GETTOKENINFO: INFO zeigt auf den Speicherbereich, der die Token-Information empfängt.
- *GETMECHANISMLIST: NULL_PTR oder Pointer auf einen Speicherbereich, der die Mechanismen-Liste empfängt.
- *GETMECHANISMINFO: INFO zeigt auf den Speicherbereich, der die Information über den Mechanismus empfängt.

COUNT

Größe des Speichers in Abhängigkeit von der Aktion:
- *GETSLOTLIST:
  INFO = NULL_PTR: Die Anzahl der Slots wird zurückgegeben;
  INFO <> NULL_PTR: muss die Größe des Speichers, auf den INFO zeigt, enthalten. Die Speichergröße entspricht der Anzahl von _SLOT_ID-Elementen.
- *GETSLOTINFO: COUNT wird nicht ausgewertet.
- *GETTOKENINFO: COUNT wird nicht ausgewertet.
- *GETMECHANISMLIST:
  INFO = NULL_PTR: Die Anzahl der Slots wird zurückgegeben;
  INFO <> NULL_PTR: muss die Größe des Speichers, auf den INFO zeigt, enthalten. Die Speichergröße entspricht der Anzahl von _MECHANISM_TYPE-Elementen.
- *GETMECHANISMINFO: COUNT wird nicht ausgewertet.

BOID

Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CWTFSLE – Auf ein Slot-Ereignis warten

Der Makro CWTFSLE wartet auf ein Slot-Ereignis.

Dieser Makroaufruf ist funktionslos. Die Schnittstelle ist jedoch aus Portabilitätsgründen in BS2000 implementiert.

Eine detaillierte Beschreibung zur Funktion des Makros CWTFSLE finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.5 „Slot and token management functions“ unter „C_WaitForSlotEvent“.

Makro	**Operanden**	
CWTFSLE	MF=	C / D / L / M / E
	,DONTBLK=	<var: bit:1> / *TRUE / *FALSE
	,SLOTID=	<var: int:4> / <integer 0..2147483647> / 0

DONTBLK legt fest, ob der CWTFSLE-Aufruf blockiert oder nicht, wenn der Makro beispielsweise auf das Eintreten eines Slot-Ereignisses wartet.

SLOTID ID des Slots

CINITTK – Token initialisieren

Der Makro CINITTK initialisiert einen Token.

Dieser Makroaufruf ist funktionslos. Die Schnittstelle ist jedoch aus Portabilitätsgründen in BS2000 implementiert.

Eine detaillierte Beschreibung zur Funktion des Makros CINITTK finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.5 „Slot and token management functions“ unter „C_InitToken“.

Makro	**Operanden**	
CINITTK	MF=	C / D / L / M / E
	,SLOTID=	<var: int:4> / <integer 0..2147483647> / 0
	,PIN=	<var: pointer> / NULL
	,PINLEN=	<var: int:4> / <integer 0..2147483647> / 0
	,LABEL=	<var: pointer> / NULL

SLOTID — ID des Slots

PIN — zeigt auf den ersten PIN des Security Officer (SO)

PINLEN — Länge des PIN in Bytes

LABEL — 32 byte-Label des Token.
Das Label muss mit Leerzeichen aufgefüllt werden und darf nicht mit Null enden.

CPIN – Pin initialisieren oder verändern

Der Makro CPIN initialisiert oder verändert einen Pin.

i Dieser Makroaufruf ist funktionslos. Die Schnittstelle ist jedoch aus Portabilitätsgründen in BS2000 implementiert.

Eine detaillierte Beschreibung zu den Funktionen des Makros CPIN finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.5 „Slot and token management functions“ unter „C_InitPIN“ und „C_SetPIN“.

Makro	**Operanden**	
CPIN	MF=	C / D / L / M / E
	,ACTION=	*INITPIN / *SETPIN / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,OLDPIN=	<var: pointer> / NULL
	,OLDPINL=	<var: int:4> / <integer 0..2147483647> / 0
	,NEWPIN=	<var: pointer> / NULL
	,NEWPINL=	<var: int:4> / <integer 0..2147483647> / 0

ACTION — Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

- =*INITPIN — entspricht der PKCS#11-Funktion *C_InitPIN*; initialisiert einen PIN.
- =*SETPIN — entspricht der PKCS#11-Funktion *C_SetPIN*; verändert einen PIN.

SESSION — Identifier der Sitzung

OLDPIN — zeigt auf den alten PIN;
wird für die Aktion *INITPIN nicht genutzt.

OLDPINL — Länge des alten PIN in Bytes;
wird für die Aktion *INITPIN nicht genutzt.

NEWPIN — zeigt auf den neuen PIN

NEWPINL — Länge des neuen PIN in Bytes

CSESION – Sitzungsverwaltung

Der Makro CSESION umfasst folgende Funktionen der Sitzungsverwaltung:

- eine Sitzung zwischen einer Anwendung und einem Token in einem bestimmten Slot eröffnen
- eine Sitzung zwischen einer Anwendung und einem Token schließen
- alle Sitzungen einer Anwendung mit einem Token schließen
- Information über eine Sitzung ausgeben

Die Funktionen C_OpenSession, C_CloseSession und C_CloseAllSessions werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zu den Funktionen des Makros CSESION finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.6 „Session management functions“ unter „C_OpenSession“, „C_CloseSession“, „C_CloseAllSessions“ und „C_GetSessionInfo“.

Makro	**Operanden**	
CSESION	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*OPENSESSION / *CLOSESESSION / *CLOSEALLSESSIONS / *GETSESSIONINFO / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,SLOTID=	<var: int:4> / 0
	,RWSESS=	<var: bit:1> / *NO / *YES
	,SERIAL=	<var: bit:1> / *NO / *YES
	,INFO=	<var: pointer> / NULL
	,NOTIFY=	<var: pointer> / NULL
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0
	,GENKPR=	<var: bit:1> / *ALLOWED / *NOTALLOWED

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*OPENSESSION
entspricht der PKCS#11-Funktion *C_OpenSession*;
eröffnet eine Sitzung zwischen einer Anwendung und einem Token in einem bestimmten Slot.

=*CLOSESESSION
entspricht der PKCS#11-Funktion *C_CloseSession*;
schließt eine Sitzung zwischen einer Anwendung und einem Token.

=*CLOSEALLSESSIONS
entspricht der PKCS#11-Funktion *C_CloseAllSessions*;
schließt alle Sitzungen zwischen einer Anwendung und einem Token.

=*GETSESSIONINFO
entspricht der PKCS#11-Funktion *C_GetSessionInfo*;
gibt Information über eine Sitzung aus.

i Diese Funktion wird nicht unterstützt.

SESSION Identifier der Sitzung

SLOTID ID des Slots

RWSESS kennzeichnet Read-/Write- bzw. Read-only-Sitzungen

=*NO Read-only-Sitzung

=<u>*YES</u> Read-/Write-Sitzung

SERIAL — serielle Sitzung
- *OPENSESSION: SERIAL sollte immer auf *YES gesetzt werden.
- *GETSESSIONINFO / *CLOSESESSION / *CLOSEALLSESSIONS: SERIAL wird nicht genutzt.

INFO — Art der Information in Abhängigkeit von der Aktion:
- *OPENSESSION: INFO ist ein in der Anwendung definierter Pointer, der an den Notification-Callback übergeben wird; wird nicht unterstützt.
- *GETSESSIONINFO: INFO zeigt auf den Speicherbereich, der Sitzungsinformationen empfängt.
- *CLOSESESSION / *CLOSEALLSESSIONS: INFO wird nicht genutzt.

NOTIFY — Callback-Funktion
- *OPENSESSION: Adresse der Notification-Callback-Funktion; wird nicht unterstützt
- *CLOSESESSION / *CLOSEALLSESSIONS / *GETSESSIONINFO: Die Callback-Funktion wird nicht genutzt.

BOID — Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung: Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD — Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung: spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]). Länge des Postcodes: 4 oder 8 bytes

RPOSTL — Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

GENKPR — Dieser Operand ist wirkungslos.

COPSTAT – Operationszustand anzeigen/setzen

Der Makro COPSTAT umfasst folgende Funktionen der Sitzungsverwaltung:

- eine Kopie des Zustands der kryptographischen Operationen einer Sitzung ausgeben
- den Zustand der kryptographischen Operationen einer Sitzung wiederherstellen

i Dieser Makroaufruf ist funktionslos. Die Schnittstelle ist jedoch aus Portabilitätsgründen in BS2000 implementiert.

Eine detaillierte Beschreibung zu den Funktionen des Makros COPSTAT finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.6 „Session management functions“ unter „C_GetOperationState“ und „C_SetOperationState“.

Makro	**Operanden**	
COPSTAT	MF=	C / D / L / M / E
	,ACTION=	*GETOPERATIONSTATE / *SETOPERATIONSTATE / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,STATE=	<var: pointer> / NULL
	,STATEL=	<var: int:4> / <integer 0..2147483647> / 0
	,CRYKEY=	<var: int:4> / <integer 0..2147483647> / 0
	,AUTHKEY=	<var: int:4> / <integer 0..2147483647> / 0

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*GETOPERATIONSTATE
entspricht der PKCS#11-Funktion *C_GetOperationState*;
gibt eine Kopie des Zustands der kryptographischen Operationen einer Sitzung aus.

=*SETOPERATIONSTATE
entspricht der PKCS#11-Funktion *C_SetOperationState*;
stellt den Zustand der kryptographischen Operationen einer Sitzung wieder her.

SESSION Identifier der Sitzung

STATE	Zustand der kryptographischen Operation: – *GETOPERATIONSTATE: Speicherbereich, der den Zustand empfängt – *SETOPERATIONSTATE: Speicherbereich, an dem der gespeicherte Zustand abgelegt ist
STATEL	Länge des *...OPERATIONSTATE-Speicherbereichs
CRYKEY	Ver- und Entschlüsselungsschlüssel, nur bei *SETOPERATIONSTATE; *GETOPERATIONSTATE: CRYKEY wird nicht genutzt.
AUTHKEY	signiert/überprüft den Authentifikationsschlüssel, nur bei *SETOPERATIONSTATE; *GETOPERATIONSTATE: AUTHKEY wird nicht genutzt.

CLOG – Login/Logout

Der Makro CLOG umfasst folgende Funktionen der Sitzungsverwaltung:

- einen Anwender in ein Token einloggen
- einen Anwender aus einem Token ausloggen

Alle Funktionen werden immer synchron ausgeführt.

Eine detaillierte Beschreibung zu den Funktionen des Makros CLOG finden Sie in PK-CS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.6 „Session management functions“ unter „C_Login“ und „C_Logout“.

Makro	**Operanden**	
CLOG	MF=	C / D / L / M / E
	,ACTION=	*LOGIN / *LOGOUT / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,PIN=	<var: pointer> / NULL
	,PINL=	<var: int:4> / <integer 0..2147483647> / 0

ACTION — Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*LOGIN — entspricht der PKCS#11-Funktion *C_Login*;
loggt einen Anwender in ein Token ein.

=*LOGOUT — entspricht der PKCS#11-Funktion *C_Logout*;
loggt einen Anwender aus einem Token aus.

SESSION — Identifier der Sitzung

PIN — zeigt auf den PIN;
*LOGOUT: PIN wird nicht benutzt.

PINL — Länge des PIN in Bytes;
*LOGOUT: PINL wird nicht benutzt.

COBJMGT – Objektverwaltung

Der Makro COBJMGT umfasst folgende Funktionen der Objektverwaltung:

- ein neues Objekt erzeugen
- ein Objekt kopieren
- ein Objekt löschen
- die Größe eines Objekts in Bytes ausgeben
- den Wert eines oder mehrerer Attribute eines Objekts ausgeben
- den Wert eines oder mehrerer Attribute eines Objekts verändern
- eine Suche nach Token- und Sitzungs-Objekten, die einem Template entsprechen, initialisieren
- eine Suche nach Token- und Sitzungs-Objekten, die einem Template entsprechen, fortsetzen, wobei zusätzliche Objekt-Handles ausgegeben werden
- eine Suche nach Token- und Sitzungs-Objekten beenden

Die Funktionen C_FindObjectsInit und C_FindObjectsFinal werden immer synchron ausgeführt.

Alle anderen Funktionen werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zu den Funktionen des Makros COBJMGT finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.7 „Object management functions“.

Makro	**Operanden**	
COBJMGT	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*CREATEOBJECT / *COPYOBJECT / *DESTROYOBJECT / *GETOBJECTSIZE / *GETATTRIBUTEVALUE / *SETATTRIBUTEVALUE / *FINDOBJECTSINIT / *FINDOBJECTS / *FINDOBJECTSFINAL / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,OBJECT=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,TEMPLAT=	<var: pointer> / NULL
	,COUNT=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,OBJLIST=	<var: pointer> / NULL
	,OBJSIZE=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,MAXOBJ=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,OBJCNT=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*CREATEOBJECT
entspricht der PKCS#11-Funktion *C_CreateObject*;
erzeugt ein neues Objekt.

=*COPYOBJECT
entspricht der PKCS#11-Funktion *C_CopyObject*;
kopiert ein Objekt.

=*DESTROYOBJECT
entspricht der PKCS#11-Funktion *C_DestroyObject*;
löscht ein Objekt.

=*GETOBJECTSIZE
entspricht der PKCS#11-Funktion *C_GetObjectSize*;
gibt die Größe eines Objekts in Bytes aus.

Diese Funktion wird nicht unterstützt.

=*GETATTRIBUTEVALUE
entspricht der PKCS#11-Funktion *C_GetAttributeValue*;
gibt den Wert eines oder mehrerer Attribute eines Objekts aus.

=*SETATTRIBUTEVALUE
entspricht der PKCS#11-Funktion *C_SetAttributeValue*;
verändert den Wert eines oder mehrerer Attribute eines Objekts.

=*FINDOBJECTSINIT
entspricht der PKCS#11-Funktion *C_FindObjectsInit*;
initialisiert eine Suche nach Token- und Sitzungs-Objekten, die einem Template entsprechen.

=*FINDOBJECTS
entspricht der PKCS#11-Funktion *C_FindObjects*;
setzt eine Suche nach Token- und Sitzungs-Objekten fort, die einem Template entsprechen, wobei zusätzliche Objekt-Handles ausgegeben werden.

=*FINDOBJECTSFINAL
entspricht der PKCS#11-Funktion *C_FindObjectsFinal*;
beendet eine Suche nach Token- und Sitzungs-Objekten.

SESSION Identifier der Sitzung

OBJECT — Objekt-Handle
- *CREATEOBJECT: OBJECT empfängt den neuen Objekt-Handle
- *COPYOBJECT, *DESTROYOBJECT, *GETOBJECTSIZE, *GETATTRIBUTEVALUE,*SETATTRIBUTEVALUE: der Objekt-Handle
- *FINDOBJECTSINIT, *FINDOBJECTS, *FINDOBJECTSFINAL: Objekt-Handle wird nicht benutzt

TEMPLAT — Objekt-Template
- *CREATEOBJECT, *COPYOBJECT: das Template des Objektes
- *GETATTRIBUTEVALUE: TEMPLAT zeigt auf ein Template, das festlegt, welche Attributwerte ausgegeben werden müssen und das die Attributwerte empfängt.
- *SETATTRIBUTEVALUE: TEMPLAT zeigt auf ein Template, das festlegt, welche Attributwerte geändert werden müssen, und das deren neue Werte vorgibt.
- *FINDOBJECTSINIT: TEMPLAT zeigt auf ein Such-Template, das die Attributwerte festlegt, denen entsprochen werden muss.
- *DESTROYOBJECT, *GETOBJECTSIZE, *FINDOBJECTS, *FINDOBJECTSFINAL: TEMPLAT wird nicht benutzt.

COUNT — Anzahl der Attribute im Template
*DESTROYOBJECT, *GETOBJECTSIZE, *FINDOBJECTS, *FINDOBJECTSFINAL: COUNT wird nicht genutzt.

OBJLIST — zeigt auf den Speicherbereich, der die Liste (Array) von zusätzlichen Objekt-Handles empfängt;
wird nur von *FINDOBJECTS genutzt.

OBJSIZE — empfängt die Größe des Objekts in Bytes;
wird nur von *GETOBJECTSIZE genutzt.

MAXOBJ — maximale Anzahl von Objekt-Handles, die zurückgegeben werden;
wird nur von *FINDOBJECTS genutzt.

OBJCNT — empfängt die aktuelle Anzahl von Objekt-Handles, die zurückgegeben wird;
wird nur von *FINDOBJECTS genutzt.

BOID — Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD — Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG Makro-Aufruf ausgibt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL — Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CCRYINI – Kryptographische Operation initialisieren

Der Makro CCRYINI umfasst folgende Funktionen:

- eine Verschlüsselungsoperation initialisieren
- eine Entschlüsselungsoperation initialisieren
- eine message-digesting Operation initialisieren
- eine mehrteilige message-digesting Operation fortsetzen, indem der Wert des geheimen Schlüssels integriert wird
- eine Signatur-Operation initialisieren, bei der die Signatur ein Anhang der Daten ist
- eine Signatur-Operation initialisieren, bei der die Daten von der Signatur zurückgewonnen werden können
- eine Überprüfungsoperation initialisieren, wobei die Signatur ein Anhang der Daten ist
- eine Signatur-Überprüfungsoperation initialisieren, wobei die Daten von der Signatur zurückgewonnen werden

Alle Funktionen werden immer synchron ausgeführt.

Eine detaillierte Beschreibung zu den Funktionen des Makros CCRYINI finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard in den Abschnitten 11.8 bis 11.12 unter „C_EncryptInit“, „C_DecryptInit“, „C_DigestInit“, „C_DigestKey“, „C_SignInit“, „C_SignRecoverInit“, „C_VerifyInit“ und „C_VerifyRecoverInit“.

Makro	**Operanden**	
CCRYINI	MF=	C / D / L / M / E
	,ACTION=	*ENCRYPTINIT / *DECRYPTINIT / *DIGESTINIT / *DIGESTKEY / *SIGNINIT / *SIGNRECOVERINIT / *VERIFYINIT / *VERIFYRECOVERINIT / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4>/ <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,KEY=	<var: int:4> / <integer 0..2147483647> / 0

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*ENCRYPTINIT
entspricht der PKCS#11-Funktion *C_EncryptInit*;
initialisiert eine Verschlüsselungsoperation.

=*DECRYPTINIT
entspricht der PKCS#11-Funktion *C_DecryptInit*;
initialisiert eine Entschlüsselungsoperation.

=*DIGESTINIT
entspricht der PKCS#11-Funktion *C_DigestInit*;
initialisiert eine message-digesting Operation.

=*DIGESTKEY
entspricht der PKCS#11-Funktion *C_DigestKey*;
setzt eine mehrteilige message-digesting Operation fort, indem der Wert des geheimen Schlüssels in die bereits zusammengefassten Daten integriert wird.

=*SIGNINIT entspricht der PKCS#11-Funktion *C_SignInit*;
initialisiert eine Signatur-Operation, bei der die Signatur ein Anhang der Daten ist.

=*SIGNRECOVERINIT
entspricht der PKCS#11-Funktion *C_SignRecoverInit*;
initialisiert eine Signatur-Operation, bei der die Daten von der Signatur zurückgewonnen werden können.

i Diese Funktion wird nicht unterstützt.

=*VERIFYINIT
entspricht der PKCS#11-Funktion *C_VerifyInit*;
initialisiert eine Überprüfungs-Operation, wobei die Signatur ein Anhang der Daten ist.

=*VERIFYRECOVERINIT
entspricht der PKCS#11-Funktion *C_VerifyRecoverInit*;
initialisiert eine Signatur-Überprüfungs-Operation, bei der die Daten von der Signatur zurückgewonnen werden.

SESSION Identifier der Sitzung

MECHAN Mechanismus

KEY Handle des Schlüssels
*DIGESTINIT: KEY wird nicht genutzt.

CCRY – Kryptographische Operation ausführen

Der Makro CCRY umfasst folgende Funktionen:

- ein Datenpaket verschlüsseln
- eine mehrteilige Verschlüsselungsoperation fortsetzen
- verschlüsselte Daten in einem Teil entschlüsseln
- eine mehrteilige Entschlüsselungsoperation fortsetzen
- aus Daten einen Hash-Wert (Message-Digest) berechnen
- eine mehrteilige message-digesting Operation fortsetzen
- Daten in einem Teil signieren, wobei die Signatur ein Anhang der Daten ist
- eine mehrteilige Signatur-Operation fortsetzen, wobei die Signatur ein Anhang der Daten ist
- Daten in einer einzelnen Operation signieren, wobei die Daten von der Signatur zurückgewonnen werden können
- eine Signatur in einer einteiligen Operation überprüfen, wobei die Signatur ein Anhang der Daten ist
- eine mehrteilige Überprüfungsoperation fortsetzen, bei der die Signatur ein Anhang der Daten ist
- eine Signatur in einer einteiligen Operation überprüfen, wobei die Daten von der Signatur zurückgewonnen werden
- eine mehrteilige Hash-Wert-Berechnungs- und Verschlüsselungsoperation fortsetzen
- eine mehrteilige Entschlüsselungs- und Hash-Wert-Berechnungsoperation fortsetzen
- eine mehrteilige Signatur- und Verschlüsselungsoperation fortsetzen
- eine mehrteilige Entschlüsselungs- und Überprüfungsoperation fortsetzen

Alle Funktionen werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zu den Funktionen des Makros CCRY finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard in den Abschnitten 11.8 bis 11.13 unter „C_Encrypt“, „C_EncryptUpdate“, „C_Decrypt“, „C_DecryptUpdate“, „C_Digest“, „C_DigestUpdate“, „C_DigestKey“, „C_Sign“, „C_SignUpdate“, „C_SignRecover“, „C_Verify“, „C_VerifyUpdate“, „C_VerifyRecover“, „C_DigestEncryptUpdate“, „C_DecryptDigestUpdate“, „C_SignEncryptUpdate“ und „C_DecryptVerifyUpdate“.

Makro	**Operanden**	
CCRY	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*ENCRYPT / *ENCRYPTUPDATE / *DECRYPT / *DECRYPTUPDATE / *DIGEST / *DIGESTUPDATE / *SIGN / *SIGNUPDATE / *SIGNRECOVER / *VERIFY / *VERIFYUPDATE / *VERIFYRECOVER / *DIGESTENCRYPTUPDATE / *DECRYPTDIGESTUPDATE / *SIGNENCRYPTUPDATE / *DECRYPTVERIFYUPDATE <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,DATAIN=	<var: pointer> / NULL
	,INLEN=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,DATAOUT=	<var: pointer> / NULL
	,OUTLEN=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*ENCRYPT entspricht der PKCS#11-Funktion *C_Encrypt*;
verschlüsselt ein Datenpaket.

=*ENCRYPTUPDATE
entspricht der PKCS#11-Funktion *C_EncryptUpdate*;
setzt eine mehrteilige Verschlüsselungsoperation fort.

=*DECRYPT entspricht der PKCS#11-Funktion *C_Decrypt*;
entschlüsselt verschlüsselte Daten in einem Teil.

=*DECRYPTUPDATE
entspricht der PKCS#11-Funktion *C_DecryptUpdate*;
setzt eine mehrteilige Entschlüsselungsoperation fort.

=*DIGEST entspricht der PKCS#11-Funktion *C_Digest*;
fasst Daten zu einem Hash-Wert zusammen.

=*DIGESTUPDATE
entspricht der PKCS#11-Funktion *C_DigestUpdate*;
setzt eine mehrteilige message-digesting Operation fort.

=*SIGN entspricht der PKCS#11-Funktion *C_Sign*;
signiert Daten in einem Teil, in dem die Signatur ein Anhang der Daten ist.

=*SIGNUPDATE
entspricht der PKCS#11-Funktion *C_SignUpdate*;
setzt eine mehrteilige Signatur-Operation fort, bei der die Signatur ein Anhang der Daten ist.

=*SIGNRECOVER
entspricht der PKCS#11-Funktion *C_SignRecover*;
signiert Daten in einer einzelnen Operation, bei der die Daten von der Signatur zurückgewonnen werden können.

Diese Funktion wird nicht unterstützt.

=*VERIFY entspricht der PKCS#11-Funktion *C_Verify*;
überprüft die Signatur in einer einteiligen Operation, wo die Signatur ein Anhang der Daten ist.

=*VERIFYUPDATE
entspricht der PKCS#11-Funktion *C_VerifyUpdate*;
setzt eine mehrteilige Überprüfungsoperation fort, wo die Signatur ein Anhang der Daten ist.

=*VERIFYRECOVER
entspricht der PKCS#11-Funktion *C_VerifyRecover*;
überprüft eine Signatur in einer einteiligen Operation, bei der die Daten von der Signatur zurückgewonnen werden.

=*DIGESTENCRYPTUPDATE
entspricht der PKCS#11-Funktion *C_DigestEncryptUpdate*;
setzt eine mehrteilige Hash-Wert-Berechnungs- und Verschlüsselungsoperation fort.


Diese Funktion wird nicht unterstützt.

=*DECRYPTDIGESTUPDATE
entspricht der PKCS#11-Funktion *C_DecryptDigestUpdate*;
setzt eine mehrteilige Entschlüsselungs- und Hash-Wert-Berechnungsoperation fort.


Diese Funktion wird nicht unterstützt.

=*SIGNENCRYPTUPDATE
entspricht der PKCS#11-Funktion *C_SignEncryptUpdate*;
setzt eine mehrteilige Signatur- und Verschlüsselungsoperation fort.


Diese Funktion wird nicht unterstützt.

=*DECRYPTVERIFYUPDATE
entspricht der PKCS#11-Funktion *C_DecryptVerifyUpdate*;
setzt eine mehrteilige Entschlüsselungs- und Überprüfungsoperation fort.


Diese Funktion wird nicht unterstützt.

SESSION Identifier der Sitzung

DATAIN zeigt auf Eingabedaten

INLEN Länge der Eingabedaten in Bytes

DATAOUT zeigt auf Ausgabedaten
- *VERIFY: Zeiger auf die Signatur
- *DIGESTUPDATE, *SIGNUPDATE, *VERIFYUPDATE: wird nicht genutzt

OUTLEN
Länge der Ausgabedaten in Bytes
– *VERIFY: Länge der Signatur
– *DIGESTUPDATE, *SIGNUPDATE, *VERIFYUPDATE: wird nicht genutzt

BOID
Ereigniskennung
– bei synchroner Ausführung: BOID wird nicht genutzt.
– bei asynchroner Ausführung: Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD
Adresse des Postcodes
– bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
– bei asynchroner Ausführung: spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
Länge des Postcodes: 4 oder 8 bytes

RPOSTL
Länge des Postcodes
– bei synchroner Ausführung: RPOSTL wird nicht genutzt.
– bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CCRYFIN – Kryptographische Operation beenden

Der Makro CCRYFIN umfasst folgende Funktionen:

- eine mehrteilige Verschlüsselungsoperation beenden
- eine mehrteilige Entschlüsselungsoperation beenden
- eine mehrteilige message-digesting Operation beenden
- eine mehrteilige Signatur-Operation mit der Rückgabe der Signatur beenden
- eine mehrteilige Überprüfungsoperation mit der Überprüfung der Signatur beenden

Alle Funktionen werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zu den Funktionen des Makros CCRYFIN finden Sie in PK-CS#11 V2.20: Cryptographic Token Interface Standard in den Abschnitten 11.8 bis 11.12 unter „C_EncryptFinal“, „C_DecryptFinal“, „C_DigestFinal“, „C_SignFinal“ und „C_VerifyFinal“.

Makro	**Operanden**	
CCRYFIN	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*ENCRYPTFINAL / *DECRYPTFINAL / *DIGESTFINAL / *SIGNFINAL / *VERIFYFINAL / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,DATA=	<var: pointer> / NULL
	,LEN=	<var: int:4> / <integer 0 .. 2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das von CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*ENCRYPTFINAL
entspricht der PKCS#11-Funktion *C_EncryptFinal*;
beendet eine mehrteilige Verschlüsselungsoperation.

=*DECRYPTFINAL
entspricht der PKCS#11-Funktion *C_DecryptFinal*;
beendet eine mehrteilige Entschlüsselungsoperation.

=*DIGESTFINAL
entspricht der PKCS#11-Funktion *C_DigestFinal*;
beendet eine mehrteilige Operation, die einen Hash-Wert berechnet.

=*SIGNFINAL
entspricht der PKCS#11-Funktion *C_SignFinal*;
beendet eine mehrteilige Signatur-Operation mit der Rückgabe der Signatur.

=*VERIFYFINAL
entspricht der PKCS#11-Funktion *C_VerifyFinal*;
beendet eine mehrteilige Überprüfungsoperation mit der Überprüfung der Signatur.

SESSION Identifier der Sitzung

DATA zeigt auf Daten
- *ENCRYPTFINAL, *DECRYPTFINAL, *DIGESTFINAL, *SIGNFINAL: DATA zeigt auf Ausgabedaten.
- *VERIFYFINAL: DATA zeigt auf die Signatur.

LEN
Länge der Daten in Bytes
– *ENCRYPTFINAL, *DECRYPTFINAL, *DIGESTFINAL, *SIGNFINAL: Länge der Ausgabedaten in Bytes
– *VERIFYFINAL: Länge der Signatur in Bytes

BOID
Ereigniskennung
– bei synchroner Ausführung: BOID wird nicht genutzt.
– bei asynchroner Ausführung: Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD
Adresse des Postcodes
– bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
– bei asynchroner Ausführung: spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
Länge des Postcodes: 4 oder 8 bytes

RPOSTL
Länge des Postcodes
– bei synchroner Ausführung: RPOSTL wird nicht genutzt.
– bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CGENKEY – Geheimen Schlüssel generieren

Der Makro CGENKEY generiert einen geheimen Schlüssel, indem er ein neues Schlüsselobjekt erzeugt.

Die Funktion wird asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zur Funktion des Makros CGENKEY finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.14 „Key management functions“ unter „C_GenerateKey“.

Makro	**Operanden**	
CGENKEY	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,TEMPL=	<var: pointer> / NULL
	,COUNT=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION — gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 — Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 — Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

SESSION — Identifier der Sitzung

MECHAN — zeigt auf den Mechanismus für die Schlüsselgenerierung

TEMPL — zeigt auf das Template für den neuen Schlüssel

COUNT — Anzahl der Attribute im Template

BOID Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CGENKPR – Schlüsselpaar generieren

Der Makro CGENKPR generiert ein Schlüsselpaar aus einem öffentlichen und einem privaten Schlüssel, indem er neue Schlüsselobjekte erzeugt.

Die Funktion wird asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zur Funktion des Makros CGENKPR finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.14 „Key management functions“ unter „C_GenerateKeyPair“.

Makro	**Operanden**	
CGENKPR	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,PUBTEMP=	<var: pointer> / NULL
	,PUBACNT=	<var: int:4> / <integer 0..2147483647> / 0
	,PRVTEMP=	<var: pointer> / NULL
	,PRVACNT=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION — gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 — Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 — Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

SESSION — Identifier der Sitzung

MECHAN — zeigt auf den Mechanismus für die Schlüsselgenerierung

PUBTEMP — zeigt auf das Template für den öffentlichen Schlüssel

PUBACNT Anzahl der Attribute im Template für den öffentlichen Schlüssel

PRVTEMP zeigt auf das Template für den privaten Schlüssel

PRVACNT Anzahl der Attribute im Template für den privaten Schlüssel

BOID Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf ausgibt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CWRPKEY – Schlüssel einpacken

Der Makro CWRPKEY packt einen privaten oder einen geheimen Schlüssel ein.

Die Funktion wird asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zur Funktion des Makros CWRPKEY finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.14 „Key management functions“ unter „C_WrapKey“.

Makro	**Operanden**	
CWRPKEY	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,KEK=	<var: int:4> / <integer 0..2147483647> / 0
	,KEY=	<var: int:4> / <integer 0..2147483647> / 0
	,WRPDKEY=	<var: pointer> / NULL
	,WRPDLEN=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION — gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 — Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 — Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

SESSION — Identifier der Sitzung

MECHAN — zeigt auf den Schlüsseleinpack-Mechanismus

KEK — Handle des einpackenden Schlüssels

KEY — Handle des Schlüssels, der eingepackt wird

WRPDKEY — zeigt auf den Speicherbereich, der den eingepackten Schlüssel empfängt

WRPDLEN — Länge des eingepackten Schlüssels

BOID — Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD — Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL — Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CUNWKEY – Schlüssel auspacken

Der Makro CUNWKEY packt einen eingepackten Schlüssel aus, indem er ein neues Objekt für einen privaten oder einen geheimen Schlüssel erzeugt.

Die Funktion wird asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zur Funktion des Makros CUNWKEY finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.14 „Key management functions“ unter „C_UnwrapKey“.

Makro	**Operanden**	
CUNWKEY	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,KEK=	<var: int:4> / <integer 0..2147483647> / 0
	,WRPDKEY=	<var: pointer> / NULL
	,WRPDLEN=	<var: int:4> / <integer 0..2147483647> / 0
	,TEMPL=	<var: pointer> / NULL
	,COUNT=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION — gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 — Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 — Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

SESSION — Identifier der Sitzung

MECHAN — zeigt auf den Schlüsseleinpack-Mechanismus

KEK — Handle des auspackenden Schlüssels

WRPDKEY	zeigt auf den eingepackten Schlüssel
WRPDLEN	Länge des eingepackten Schlüssels
TEMPL	zeigt auf das Template für den neuen Schlüssel
COUNT	Anzahl der Attribute im Template
BOID	Ereigniskennung – bei synchroner Ausführung: BOID wird nicht genutzt. – bei asynchroner Ausführung: Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.
RPOSTAD	Adresse des Postcodes – bei synchroner Ausführung: RPOSTAD wird nicht genutzt. – bei asynchroner Ausführung: spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]). Länge des Postcodes: 4 oder 8 bytes
RPOSTL	Länge des Postcodes – bei synchroner Ausführung: RPOSTL wird nicht genutzt. – bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CDRVKEY – Schlüssel ableiten

Der Makro CDRVKEY leitet einen Schlüssel vom einem Basisschlüssel ab, indem er ein neues Schlüsselobjekt erzeugt.

Die Funktion wird asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zur Funktion des Makros CDRVKEY finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.14 „Key management functions“ unter „C_DeriveKey“.

Makro	**Operanden**	
CDRVKEY	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,MECHAN=	<var: pointer> / NULL
	,BASEKEY=	<var: int:4> / <integer 0..2147483647> / 0
	,TEMPL=	<var: pointer> / NULL
	,COUNT=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION — gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 — Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 — Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

SESSION — Identifier der Sitzung

MECHAN — zeigt auf den Mechanismus zum Ableiten des Schlüssels

BASEKEY — Handle des Basisschlüssels

TEMPL	zeigt auf das Template für den neuen Schlüssel
COUNT	Anzahl der Attribute im Template
BOID	Ereigniskennung – bei synchroner Ausführung: BOID wird nicht genutzt. – bei asynchroner Ausführung: Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.
RPOSTAD	Adresse des Postcodes – bei synchroner Ausführung: RPOSTAD wird nicht genutzt. – bei asynchroner Ausführung: spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]). Länge des Postcodes: 4 oder 8 bytes
RPOSTL	Länge des Postcodes – bei synchroner Ausführung: RPOSTL wird nicht genutzt. – bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

CRANDOM – Zufallszahlen generieren

Der Makro CRANDOM umfasst folgende Funktionen:

- zusätzliches Startparameter-Material in den Zufallszahlen-Generator für die Token mischen
- Zufallsdaten generieren

Alle Funktionen werden asynchron ausgeführt, falls für die Task mit C_Initialize asynchrone Funktionsausführung festgelegt wurde.

Eine detaillierte Beschreibung zu den Funktionen des Makros CRANDOM finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Abschnitt 11.15 „Random number generation functions“.

Makro	**Operanden**	
CRANDOM	MF=	C / D / L / M / E
	,VERSION=	001 / 002
	,ACTION=	*SEEDRANDOM / *GENERATERANDOM / <var: enum-of _action_set: 1> / default: _action_set.undefined
	,SESSION=	<var: int:4> / <integer 0..2147483647> / 0
	,DATA=	<var: pointer> / NULL
	,DATALEN=	<var: int:4> / <integer 0..2147483647> / 0
	,BOID=	<var: int:4> / 0
	,RPOSTAD=	<var: pointer> / NULL
	,RPOSTL=	<integer 1..2> / <var: int:4> / 0

VERSION gibt an, welche Version des Parameterbereichs generiert werden soll. Es empfiehlt sich, immer die neueste Version zu verwenden.

=001 Es wird das Format generiert, das von CRYPT V1.0 unterstützt wurde. Dieses Format unterstützt nur die bereits in CRYPT V1.0 bekannten Parameter.
VERSION=001 ist Voreinstellung.

=002 Es wird das Format generiert, das ab CRYPT V1.1 unterstützt wird.

ACTION Art der Aktion.
Die korrespondierende PKCS#11-Funktion ist bei jedem Aktionscode angegeben.

=*SEEDRANDOM
entspricht der PKCS#11-Funktion *C_SeedRandom*;
mischt zusätzliches Startparameter-Material in den Zufallszahlen-Generator für die Token.

=*GENERATERANDOM
entspricht der PKCS#11-Funktion *C_GenerateRandom*;
generiert Zufallsdaten.

SESSION Identifier der Sitzung

DATA zeigt auf folgende Daten:
- bei *SEEDRANDOM: DATA zeigt auf das Startparameter-Material.
- bei *GENERATERANDOM: DATA zeigt auf den Speicherbereich, der die Zufallsdaten empfängt.

DATALEN Länge der Daten in Bytes

BOID Ereigniskennung
- bei synchroner Ausführung: BOID wird nicht genutzt.
- bei asynchroner Ausführung:
  Ereigniskennung, an die das Ende der Funktionsbearbeitung signalisiert wird.

RPOSTAD Adresse des Postcodes
- bei synchroner Ausführung: RPOSTAD wird nicht genutzt.
- bei asynchroner Ausführung:
  spezifiziert ein Feld, das Postcode-Information enthält, die zum korrespondierenden Programm übertragen werden soll, das den SOLSIG-Aufruf absetzt (siehe auch Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3]).
  Länge des Postcodes: 4 oder 8 bytes

RPOSTL Länge des Postcodes
- bei synchroner Ausführung: RPOSTL wird nicht genutzt.
- bei asynchroner Ausführung: gibt die Länge der Postcode-Information in Worten an (1 oder 2).

6.5 Beispielprogramme

In diesem Abschnitt finden Sie jeweils ein Programmbeispiel für synchronen und für asynchronen Ausführungsmodus.

6.5.1 Synchrone Ausführung – Beispiel

Im Beispielprogramm werden folgende Makros von CRYPT verwendet:

- Der Makro CPKC11T enthält Datenbeschreibungen und Equates, die von den folgenden Makros genutzt werden.
- Der Makro CSESION eröffnet mit der Aktion *OPENSESSION eine Sitzung zwischen einer Anwendung und einem Token in einem bestimmten Slot.
- Der Makro CGENKEY generiert einen geheimen Schlüssel.
- Anschließend initialisiert die Aktion *ENCRYPTINIT des Makros CCRYINI eine Verschlüsselungsoperation.
- Mit der Aktion *ENCRYPT des Makros CCRY wird die Verschlüsselungsoperation fortgesetzt und beendet.
- Die Aktion *DECRYPTINIT des Makros CCRYINI initialisiert eine Entschlüsselungsoperation.
- Anschließend setzt der Makro CCRY mit der Aktion *DECRYPT die Entschlüsselung fort und beendet sie.
- Mit der Aktion *CLOSESESSION des Makros CSESION wird die Sitzung geschlossen.

```
         TITLE 'CPKC11T LAYOUT'
         CPKC11T MF=D
         TITLE 'CSESION PARAM LIST'
         CSESION MF=D
         TITLE 'CCRYINI PARAM LIST'
         CCRYINI MF=D
         TITLE 'CCRY PARAM LIST'
         CCRY  MF=D
         TITLE 'CGENKEY PARAM LIST'
         CGENKEY MF=D
         TITLE 'CRY2EX - EXAMPLE'
CRY2EX   RMODE ANY
CRY2EX   AMODE ANY
         SPACE 3
```

```
*****
***** ENTRIES
*****
         SPACE
CRY2EX   @ENTR TYP=M,ENV=SPLSPEC,FUNCT='EXAMPLE OF CRYPT ASS PROGRAM', -
               LOCAL=ZEXALOC
         SPACE 4
* PRESET ILLEGAL SESSION HANDLE
         MVC   ZSESSION,=F'0'
* OPEN SESSION
         LA    R3,CSESIONC
         MVC   CSESIONC,CSESIONL
         @DATA BASE=R3,CLASS=B,DSECT=CRYO_MDL
         MVI   CRYOACTION,CRYOOPENSESSION
*        MVC   CRYOSLOTID,=F'0'
         CSESION MF=E,PARAM=(R3),CALLER=USER
         @IF   EQ
         CLC   CRYORET,=F'0'
         @THEN
         MVC   ZSESSION,CRYOSESSION

* GENERATE SECRET KEY
         LA    R3,CGENKEYC
         MVC   CGENKEYC,CGENKEYL
         @DATA BASE=R3,CLASS=B,DSECT=CRYD_MDL
         MVC   CRYDSESSION,ZSESSION
         MVC   CRYDMECHANISM,=A(MDESKGEN)
*        MVC   CRYDTEMPLAT,=F'0'
*        MVC   CRYDCOUNT,=F'0'
         CGENKEY MF=E,PARAM=(R3),CALLER=USER
         @IF   EQ
         CLC   CRYDRET,=F'0'
         @THEN
         MVC   ZDESKEY,CRYDKEY
         SPACE 4

* INITIALIZE ENCRYPTION OPERATION
         LA    R3,CCRYINIC
         MVC   CCRYINIC,CCRYINIL
         @DATA BASE=R3,CLASS=B,DSECT=CRYA_MDL
         MVI   CRYAACTION,CRYAENCRYPTINIT
         MVC   CRYASESSION,ZSESSION
         MVC   CRYAKEY,ZDESKEY
         MVC   CRYAMECHANISM,=A(MDESECB)
         CCRYINI MF=E,PARAM=(R3),CALLER=USER
         SPACE
         @IF   EQ
         CLC   CRYARET,=F'0'
         @THEN
```

```
* ENCRYPT OPERATION
         LA    R3,CCRYC
         MVC   CCRYC,CCRYL
         @DATA BASE=R3,CLASS=B,DSECT=CRYB_MDL
         MVI   CRYBACTION,CRYBENCRYPT
         MVC   CRYBSESSION,ZSESSION
         MVC   CRYBDATAIN,=A(ZINPUT)
         MVC   CRYBDATAINLEN,=A(L'ZINPUT)
         LA    R15,ZENCOUT
         ST    R15,CRYBDATAOUT
         MVC   CRYBDATAOUTLEN,=A(L'ZENCOUT)
         CCRY  MF=E,PARAM=(R3),CALLER=USER
         @IF   EQ
* CCRY SUCCESSFUL ?
         CLC   CRYBRET,=F'0'
         @THEN
* SAVE LENGTH OF ENCRYPTED STRING
         MVC   ZENCOUTL,CRYBDATAOUTLEN
* ENCRYPT OPERATION WAS TERMINATED BY SINGLE STEP ENCRYPTION.

* INITIALIZE DECRYPT OPERATION
         LA    R3,CCRYINIC
         MVC   CCRYINIC,CCRYINIL
         @DATA BASE=R3,CLASS=B,DSECT=CRYA_MDL
         MVI   CRYAACTION,CRYADECRYPTINIT
         MVC   CRYASESSION,ZSESSION
         MVC   CRYAKEY,ZDESKEY
         MVC   CRYAMECHANISM,=A(MDESECB)
         CCRYINI MF=E,PARAM=(R3),CALLER=USER
         SPACE
         @IF   EQ
         CLC   CRYARET,=F'0'
         @THEN

* DECRYPT OPERATION
         LA    R3,CCRYC
         MVC   CCRYC,CCRYL
         @DATA BASE=R3,CLASS=B,DSECT=CRYB_MDL
         MVI   CRYBACTION,CRYBDECRYPT
         MVC   CRYBSESSION,ZSESSION
         LA    R15,ZENCOUT
         ST    R15,CRYBDATAIN
         MVC   CRYBDATAINLEN,ZENCOUTL
         LA    R15,ZDECOUT
         ST    R15,CRYBDATAOUT
         MVC   CRYBDATAOUTLEN,=A(L'ZDECOUT)
         CCRY  MF=E,PARAM=(R3),CALLER=USER
         @IF   EQ

* CCRY SUCCESSFUL ?
         CLC   CRYBRET,=F'0'
         @THEN
* SAVE LENGTH OF DECRYPTED STRING
         MVC   ZDECOUTL,CRYBDATAOUTLEN
* DECRYPT OPERATION WAS TERMINATED BY SINGLE STEP DECRYPTION.

* NO ERROR FROM CRYPT CALLS
         LA    R3,0
* CHECK RESULT
         @IF   EQ
* LENGTH IDENTICAL
         CLC   ZDECOUTL,=A(L'ZINPUT)
         @AND  EQ
* DECRYPTED STRING IDENTICAL
         CLC   ZINPUT,ZDECOUT
         @THEN
```

```
* REPORT SUCCESS
         WROUT SUCCESS,SUCCESSE,PARMOD=31
SUCCESSE DS    OH
         @ELSE
* REPORT FAILURE
         WROUT FAILURE,FAILUREE,PARMOD=31
FAILUREE DS    OH
         @BEND
         @BEND
         @BEND
         @BEND
         @BEND
         @BEND
         @BEND
         @IF   NE
* SESSION WAS INITIALIZED ?
         CLC   ZSESSION,=F'0'
         @THEN

* CLOSE SESSION
         LA    R3,CSESIONC
         MVC   CSESIONC,CSESIONL
         @DATA BASE=R3,CLASS=B,DSECT=CRYO_MDL
         MVI   CRYOACTION,CRYOCLOSESESSION
         MVC   CRYOSESSION,ZSESSION
         CSESION MF=E,PARAM=(R3),CALLER=USER
         @BEND
         SPACE
         @EXIT

* DATA
CSESIONL CSESION MF=L
CGENKEYL CGENKEY MF=L
CCRYINIL CCRYINI MF=L
CCRYL    CCRY  MF=L
*
* MECHANISM DES_KEY_GEN (NO PARAMETER)
MDESKGEN DC    A(CRYOMDES_KEY_GEN),A(0),A(0)
* MECHANISM DES_ECB (NO PARAMETER)
MDESECB  DC    A(CRYOMDES_ECB),A(0),A(0)
*
* STRING TO BE ENCRYPTED (FOR DES-ECB, LENGTH MUST BE A MULTIPLE OF 8)
ZINPUT   DC    CL16'DAS IST GEHEIM !'
*
*

SUCCESS  DC    Y(SUCCESSL)
         DC    X'000001'
         DC    C'SUCCESSFUL ENCRYPTION AND DECRYPTION'
SUCCESSL EQU   *-SUCCESS
*
FAILURE  DC    Y(FAILUREL)
         DC    X'000001'
         DC    C'DECRYPTION OUTPUT DIFFERS FROM ENCRYPTION INPUT'
FAILUREL EQU   *-FAILURE
*
```

```
ZEXALOC  @PAR  D=YES
         DS    0F
CSESIONC DS    XL(CRYO#)
CGENKEYC DS    XL(CRYD#)
CCRYINIC DS    XL(CRYA#)
CCRYC    DS    XL(CRYB#)
* ENCRYPTED STRING AREA
ZENCOUT  DS    XL24
* ENCRYPTED STRING AREA
ZDECOUT  DS    XL24
* SESSION #
ZSESSION DS    F
* SECRET KEY HANDLE
ZDESKEY  DS    F
* LENGTH OF ENCRYPTED STRING
ZENCOUTL DS    F
* LENGTH OF DECRYPTED STRING
ZDECOUTL DS    F
         SPACE
ZEXALOC  @PAR  LEND=YES
         @END
         END

/START-ASSEMBH
//COMPILE SOURCE=...
//MACRO-LIBRARY=(.....)
//SOURCE-PROPERTIES=*PAR(LOW-CASE-CONVERSION=YES,...)
//....
//END

/START-BINDER ...
//START-LLM-CREATION INTERNAL-NAME=...
//INCLUDE-MODULES ELEMENT=CRY2EX,LIB=...
//INCLUDE-MODULES ELEMENT=ITSP1PMS,LIB=PM.MODULE
//RESOLVE-BY-AUTOLINK LIBRARY=PM.MODULE
//SAVE-LLM LIB=...
//END
```

6.5.2 Asynchrone Ausführung – Beispiel

In dem folgenden Ausschnitt eines Beispielprogramms werden folgende Makros von CRYPT verwendet:

1. Der Makro CPKC11T enthält Datenbeschreibungen und Equates, die von den folgenden Makros genutzt werden.
2. Der Makro CGENRAL stellt für die Task asynchrone Verarbeitung mit CRYPT ein.
3. Die Ereigniskennung CRYPTTST wird definiert. Die Adresse der Kurzkennung ist OUTEIID.
4. Die Routine CRY2ABC wird als Contingency-Prozess definiert:
   CONTAAD gibt die Anfangsadresse an.
   OUTCOID gibt die Adresse der Kurzkennung an.
5. Das Programm fordert mit einem SOLSIG-Aufruf von der Ereigniskennung CRYPTTST ein Signal und gibt den Contingency-Prozess CRY2ABC an. Wenn nach einer Wartezeit von 600 Sekunden das Signal nicht eingetroffen ist, soll die Ereignissteuerung den Contingency-Prozess CRY2ABC starten. Das Programm läuft nach diesem SOLSIG-Aufruf weiter.
6. Der Makro CSESION eröffnet mit der Aktion *OPENSESSION eine Sitzung zwischen einer Anwendung und einem Token in einem bestimmten Slot.
7. Der Makro CGENKEY generiert einen geheimen DES-Schlüssel.
8. Anschließend initialisiert die Aktion *ENCRYPTINIT des Makros CCRYINI eine Verschlüsselungsoperation mit dem erzeugten DES-Schlüssel.
9. Mit der Aktion *ENCRYPT des Makros CCRY wird die Verschlüsselungsoperation durchgeführt.
10. Routine CRY2ABC, die als Contingency-Prozess fungiert.
11. Das Programm fordert mit einem SOLSIG-Aufruf erneut ein Signal an.
12. Es wird überprüft, ob ein Ereignis von CRYPT eingetreten ist.
13. Je nach eingetretenem Ereignis wird eine Folgeverarbeitung durchgeführt.

Informationen zu ereignisgesteuerter Verarbeitung und Contingency-Prozess finden Sie im Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3].

Makroaufrufe, die in dem vorliegenden Handbuch nicht beschrieben sind (z. B. SOLSIG), sind ebenfalls im Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [3] beschrieben.

```
FHDR  MF=D
TITLE 'CPKC11T layout'  ------------------------------------------(1.)
CPKC11T MF=D
TITLE 'CSESION'
CSESION MF=D,VERSION=002
TITLE 'CCRY'
CCRY  MF=D,VERSION=002
TITLE 'CGENKEY'
CGENKEY MF=D,VERSION=002
TITLE 'CRY2ABS - example'

*
AREA     DSECT
OUTEIID  DS    F
OUTCOID  DS    F
AREA#    EQU   *-AREA
*
CRY2AB   CSECT
CRY2AB   AMODE ANY
CRY2AB   RMODE ANY
*
         ENTRY CRY2ABV
CRY2ABV  DS    0D
         DS    XL(AREA#)

*
         SPACE
CRY2ABS  @ENTR TYP=M,ENV=SPLSPEC,FUNCT='example of crypt ass program', -
               LOCAL=ZEXALOC
CRY2ABS  AMODE ANY
CRY2ABS  RMODE ANY
*
         L     R9,=V(CRY2ABV)
         @DATA BASE=R9,CLASS=B,DSECT=AREA
*
         LA    R3,CGENRALC
         MVC   CGENRALC,CGENRALL
         @DATA BASE=R3,CLASS=B,DSECT=CRYJHEADER
         MVI   CRYJACTION,CRYJINITIALIZE
         MVI   CRYJEXEC,CRYJASYNCHRON

CGENRALE CGENRAL MF=E,PARAM=(R3),CALLER=USER  ----------------------------(2.)
*
         @IF   ZE
         CLC   CRYJRET,=F'0'
         @THEN , INIT ok
*
         @ELSE , error
* error handling
         B     EXIT
         @BEND , INIT ok/error
*
         LA    R2,OUTEIID
         ENAEI EINAME=CRYPTTST,EIIDRET=(R2),PARMOD=31  --------------------(3.)
         ST    R15,ENAEIRC
         @IF   EQ
* ok?
         CLI   ENAEIMC,X'00'
         @THEN , event item created
```

```
*
         @ELSE , event item not created
* error handling
         B     EXIT
         @BEND , event item (not) created
*
         LA    R2,OUTCOID
         ENACO CONAME=CRYPTST,COADAD=CONTAAD,COIDRET=(R2),PARMOD=31  ------(4.)
         ST    R15,ENACORC
         @IF   EQ
* ok?
         CLI   ENACOMC,X'00'
         @THEN , contingency created

*
         @ELSE , contingency not created
* error handling
         B     EXIT
         @BEND , contingency (not) created
*
         LA    R4,OUTCOID
         LA    R2,OUTEIID
         SOLSIG EIID=(R2),COID=(R4),LIFETIM=600,PARMOD=31  ----------------(5.)

*
         @IF   NZ
*   SOLSIG ok?
         LTR   R15,R15
         @THEN , error
* error handling
         B     EXIT
         @BEND , error

*
* REQM for openSession PA
         REQM  1,PARMOD=31
         @IF   NZ
         LTR   R15,R15
         @THEN , error
* error handling
         B     EXIT
         @BEND , error
*
         LR    R6,R1
         @DATA BASE=R6,CLASS=B,DSECT=CRYO_MDL
```

```
*
* set up CSESION call
         MVC   CRYOHEADER(CRYO#),CSESIONL
         MVI   CRYOACTION,CRYOOPENSESSION
         MVC   CRYOBOID,OUTEIID
         LA    R1,OPSTKEY1
         ST    R1,CRYORPOSTAD
         ST    R6,OPSTKEY2
         MVC   CRYORPOSTL,=F'2'
         CSESION MF=E,PARAM=(R6),CALLER=USER  ------------------------------(6.)
*
         @IF   EQ
         CLC   CRYORET,=F'0'
         @THEN , open session accepted
*
*  wait for the completion of openSession
*
         @ELSE , open session not accepted
* error handling
         B     EXIT
         @BEND , open session (not) accepted
*

*
* REQM for genKey PA
         REQM  1,PARMOD=31
         @IF   NZ
         LTR   R15,R15
         @THEN , error
* error handling
         B     EXIT
         @BEND , error
*
         LR    R7,R1
         @DATA BASE=R7,CLASS=B,DSECT=CRYDHEADER
*
         MVC   CRYDHEADER(CRYD#),CGENKEYL
         MVC   CRYDSESSION,CRYOSESSION
         MVC   CRYDMECHANISM,=A(MDESKGEN)
         MVC   CRYDBOID,OUTEIID
         LA    R1,DPSTKEY1
         ST    R1,CRYDRPOSTAD
         ST    R7,DPSTKEY2
         MVC   CRYDRPOSTL,=F'2'
         CGENKEY MF=E,PARAM=(R7),CALLER=USER  ------------------------------(7.)
         @IF   EQ
         CLC   CRYDRET,=F'0'
         @THEN , generate key accepted
*
*  wait for the completion of generate key
*
         @ELSE , generate key not accepted
* error handling
         B     EXIT
         @BEND , generate key (not) accepted
*
```

```
*
* set up encrypt CCRYINI call
         MVC   CCRYINIC,CCRYINIL
         LA    R8,CCRYINIC
         @DATA BASE=R8,CLASS=B,DSECT=CRYAHEADER
         MVI   CRYAACTION,CRYAENCRYPTINIT
         MVC   CRYASESSION,CRYOSESSION
         MVC   CRYAKEY,CRYDKEY
         MVC   CRYAMECHANISM,=A(MDESECB)
*
         CCRYINI MF=E,PARAM=(R8),CALLER=USER  ----------------------------(8.)
*
         @IF   EQ
         CLC   CRYARET,=F'0'
         @THEN , encrypt init ok
*
         @ELSE , encrypt init not ok
* error handling
         B     EXIT
         @BEND , encrypt init (not) ok

*
* REQM for encrypt PA
         REQM  1,PARMOD=31
         @IF   NZ
         LTR   R15,R15
         @THEN , error
* error handling
         B     EXIT
         @BEND , error
*
         LR    R8,R1
         @DATA BASE=R8,CLASS=B,DSECT=CRYB_MDL
*
         MVC   CRYBHEADER(CRYB#),CCRYL
         MVC   CRYBSESSION,CRYOSESSION
         MVI   CRYBACTION,CRYBENCRYPT
         MVC   CRYBDATAIN,=A(ZINPUT)
         MVC   CRYBDATAINLEN,=A(L'ZINPUT)
         LA    R15,ZENCOUT
         ST    R15,CRYBDATAOUT
         MVC   CRYBDATAOUTLEN,=A(L'ZENCOUT)
         MVC   CRYBBOID,OUTEIID
         LA    R1,BPSTKEY1
         ST    R1,CRYBRPOSTAD
         ST    R8,BPSTKEY2
         MVC   CRYBRPOSTL,=F'2'
         CCRY  MF=E,PARAM=(R8),CALLER=USER  ------------------------------(9.)
*
         @IF   EQ
         CLC   CRYBRET,=F'0'
         @THEN , encrypt accepted
```

```
*
*  wait for the completion of encrypt
*
         @ELSE , encrypt not accepted
* error handling
         B     EXIT
         @BEND , encrypt (not) accepted
*
EXIT     EQU   *
         @EXIT
* DATA
*
CONTAAD  DC    A(CRY2ABC)
*
CGENRALL CGENRAL MF=L,VERSION=002
CSESIONL CSESION MF=L,VERSION=002
CGENKEYL CGENKEY MF=L,VERSION=002
CCRYINIL CCRYINI MF=L
CCRYL    CCRY  MF=L,VERSION=002
*
* mechanism DES_KEY_GEN (no parameter)
MDESKGEN DC    A(CRY0MDES_KEY_GEN),A(0),A(0)
* mechanism DES_ECB (no parameter)
MDESECB  DC    A(CRY0MDES_ECB),A(0),A(0)
*
* string to be encrypted (for DES-ECB, length must be a multiple of 8)
ZINPUT   DC    CL16'das ist geheim!'
*

*
ZEXALOC  @PAR  D=YES
*
CGENRALC CGENRAL MF=C,VERSION=002
*
CCRYINIC CCRYINI MF=C
*
ENAEIRC  DS    0F
ENAEISC  DS    X
         DS    X
         DS    X
ENAEIMC  DS    X
*
ENACORC  DS    0F
ENACOSC  DS    X
         DS    X
         DS    X
ENACOMC  DS    X
*
OPOSTKEY DS    0D
OPSTKEY1 DS    F
OPSTKEY2 DS    F
*
DPOSTKEY DS    0D
DPSTKEY1 DS    F
DPSTKEY2 DS    F
*
BPOSTKEY DS    0D
BPSTKEY1 DS    F
BPSTKEY2 DS    F
*
* encrypted string area
ZENCOUT  DS    XL24
*
ZEXALOC  @PAR  LEND=YES
         @END
*
*********************************************************************
```

```
*
ENTRY CRY2ABC
CRY2ABC  @ENTR TYP=B,BASE=R10,FUNCT='Contingency '  ----------------------(10.)
CRY2ABC  AMODE ANY
CRY2ABC  RMODE ANY
*
*  register contents at start of contingency
*  R1: contingency message - not used
*  R2: event information code
*  R3: post code 1 (byte1: EC type of Crypt; rest: RC)
*  R4: post code 2 (A(PA))
*
         LR    R10,R15
*
         CONTXT STACKR=(R12,R13),OWNR=(R12,R13),FUNCT=READ,PROCESS=LAST
*
*
         L     R9,=V(CRY2ABV)
         @DATA BASE=R9,CLASS=B,DSECT=AREA

*
         LA    R14,OUTCOID
         LA    R15,OUTEIID
         SOLSIG EIID=(R15),COID=(R14),LIFETIM=600,PARMOD=31  --------------(11.)
*
         @IF   NZ
*  error at SOLSIG?
         LTR   R15,R15
         @THEN , Fehler
* error handling
         B     RETCO
         @BEND , Fehler
*
         ST    R2,CONTIRC
         @IF   EQ
* ok?
         CLI   CONTIMC,X'00'
         @THEN , cont correctly started
         @IF   EQ
         CLI   CONTISC,X'28'
         @OR   EQ
         CLI   CONTISC,X'2C'
         @THEN , ok
*
* nothing to do

*
         @ELSE , what's wrong
         @IF   EQ
* timeout?
         CLI   CONTIMC,X'04'
         @THEN , timeout or EI killed
* timeout handling
*
         B     RETCO
         @ELSE , something wrong

*
* error handling
*
         B     RETCO
         @BEND , something wrong/ timeout or EI killed
         @BEND , what's wrong
         @ELSE , something wrong
* error handling
         B     RETCO
         @BEND , something wrong
```

```
*
* cont correctly started
*
         @IF   NE
*  contains post code 1 the ETC of CRYPT?
         CLM   R3,B'1000',=AL1(CRYOEVENT)  ---------------------------------(12.)
         @THEN , not a CRYPT event

*
* error handling
*
         B     RETCO
         @BEND , not a CRYPT event
*
         @IF   EQ
*  PA not allocated?
         CLM   R3,B'0011',=AL2(CRYOPA_NOT_ALLOC)
         @THEN , PA not allocated
*
* error handling
         B     RETCO
         @BEND , PA not allocated
*

*
         LR    R3,R4
         @DATA BASE=R3,DSECT=ESMFHDR
*
         @CAS2 ESMFCT,COMP=CLI  --------------------------------------------(13.)
*
         @OF   ESESION
*
         @DATA BASE=R3,DSECT=CRYO_MDL
         @IF   EQ
         CLC   CRYORET,=F'0'
         @THEN , session function ok
*
         @IF   EQ

* action = opensession
         CLI   CRYOACTION,CRYOOPENSESSION
         @THEN , openSession
*
* handle open session
*
         @BEND , openSession
*
         @ELSE , session function not ok
* error handling
         B     RETCO
         @BEND , session function (not) ok

*
* end ESESION
*
         @DATA BASE=R3,DSECT=ESMFHDR
         @OF   EGENKEY
*
         @DATA BASE=R3,DSECT=CRYD_MDL
         @IF   EQ
         CLC   CRYDRET,=F'0'
         @THEN , generate key ok
```

```
*
* handle generate key
*
         @ELSE , generate key not ok
* error handling
         B     RETCO
         @BEND , generate key (not) ok

*
* end EGENKEY
*
         @DATA BASE=R3,DSECT=ESMFHDR
         @OF   ECRY
* CRY
         @DATA BASE=R3,DSECT=CRYB_MDL
*
         @IF   EQ
         CLC   CRYBRET,=F'0'
         @THEN , crypt function ok
*
         @IF   EQ
* action = encrypt?
         CLI   CRYBACTION,CRYBENCRYPT
         @THEN , encrypt

*
* handle encrypt
*
         @ELSE , <> encrypt
         @IF   EQ
* action = decrypt?
         CLI   CRYBACTION,CRYBDECRYPT
         @THEN , decrypt

*
* handle decrypt
*
         @ELSE , <> decrypt
*
         @BEND , decrypt ...
         @BEND , encrypt ...
         @ELSE , crypt function not ok
* error handling
         B     RETCO
         @BEND , crypt function (not) ok

*
* end ECRY
         @DATA BASE=R3,DSECT=ESMFHDR
         @OFRE
* error: unknown function
*
* error handling
*
         B     RETCO
         @BEND , CAS  ESMFCT,COMP=CLI
*
*
*
```

```
RETCO    EQU   *
         RETCO
         @EXIT
*********************************************************
* EQUates for the CRYPT functions
EGENRAL  EQU   1    GENeRAL-purpose functions
ESESION  EQU  20    SESsION management
EOBJMGT  EQU  30    OBJect ManaGemenT
ECRYINI  EQU  40    INIt a CRYptographic function
ECRY     EQU  41    CRYptographic function
ECRYFIN  EQU  42    FINalize a CRYptographic function
EGENKEY  EQU  80    GENerateKEY
EGENKPR  EQU  81    GENerateKeyPaiR
EWRPKEY  EQU  82    WRaPKEY
EUNWKEY  EQU  83    UNWrapKEY
EDRVKEY  EQU  84    DeRiVeKEY
ERANDOM  EQU  90    RANDOM number generation
*--------------------------------------------------------
*
*
CONTIRC  DS    OF
CONTISC  DS    X
         DS    X
         DS    X
CONTIMC  DS    X
*
```

7 Beschreibung der Funktionen in C

Zur Entwicklung von portablen Anwendungen, bzw. zur einfachen Portierung von Anwendungen, die eine PKCS#11-Schnittstelle nutzen, wird im Rahmen von CRYPT auch die in PKCS#11 beschriebene ANSI C-Schnittstelle mit den Include-Dateien PKCS11.H, PKCS11T.H und PKCS11F.H zur Verfügung gestellt.

Voraussetzung

Wenn Sie diese C-Schnittstelle nutzen möchten, binden Sie das Adaptermodul CRYADAP aus der Bibliothek SYSLIB.CRYPT.nnn zum Programm hinzu.

7.1 Hinweise zur Beschreibung in PKCS#11

Die PKCS#11-Spezifikation V2.20 ist die Basis für Ihre Arbeit mit der C-Schnittstelle von CRYPT.

Diese Spezifikation finden Sie im Internet unter:

http://germany.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

In der nachfolgenden Tabelle sind die Funktionen aufgelistet und die Stelle, an der sie im PKSC#11 V2.20 beschrieben sind. Das Präfix C_ bei den PKCS#11-Funktionen steht für Funktion.

Funktion	**Beschreibung in PKCS#11 V2.20**
Allgemeine Funktionen – C_Initialize – C_Finalize – C_GetInfo – C_GetFunctionList	Abschnitt 11.4
Funktionen für die Slot- und Token-Verwaltung – C_GetSlotList – C_GetSlotInfo – C_GetTokenInfo – C_WaitForSlotEvent – C_GetMechanismList – C_GetMechanismInfo – C_InitToken – C_InitPIN – C_SetPIN	Abschnitt 11.5
Funktionen für die Sitzungsverwaltung – C_OpenSession – C_CloseSession – C_CloseAllSessions – C_GetSessionInfo – C_GetOperationState – C_SetOperationState – C_Login – C_Logout	Abschnitt 11.6
Funktionen für die Objektverwaltung – C_CreateObject – C_CopyObject – C_DestroyObject – C_GetObjectSize – C_GetAttributeValue – C_SetAttributeValue – C_FindObjectsInit – C_FindObjects – C_FindObjectsFinal	Abschnitt 11.7

Funktionen und zugehörige Beschreibung in PKCS#11 (Teil 1 von 3)

Funktion	Beschreibung in PKCS#11 V2.20
Verschlüsselungsfunktionen – C_EncryptInit – C_Encrypt – C_EncryptUpdate – C_EncryptFinal	Abschnitt 11.8
Entschlüsselungsfunktionen – C_DecryptInit – C_Decrypt – C_DecryptUpdate – C_DecryptFinal	Abschnitt 11.9
Funktionen für den Message-Digest – C_DigestInit – C_Digest – C_DigestUpdate – C_DigestKey – C_DigestFinal	Abschnitt 11.10
Funktionen zum Signieren und MACing – C_SignInit – C_Sign – C_SignUpdate – C_SignFinal – C_SignRecoverInit – C_SignRecover	Abschnitt 11.11
Funktionen zum Überprüfen von Signaturen und MACs – C_VerifyInit – C_Verify – C_VerifyUpdate – C_VerifyFinal – C_VerifyRecoverInit – C_VerifyRecover	Abschnitt 11.12
Kryptographische Funktionen mit Doppelfunktion – C_DigestEncryptUpdate – C_DecryptDigestUpdate – C_SignEncryptUpdate – C_DecryptVerifyUpdate	Abschnitt 11.13

Funktionen und zugehörige Beschreibung in PKCS#11 (Teil 2 von 3)

Funktion	Beschreibung in PKCS#11 V2.20
Funktionen für die Schlüsselverwaltung – C_GenerateKey – C_GenerateKeyPair – C_WrapKey – C_UnwrapKey – C_DeriveKey	Abschnitt 11.14
Funktionen zur Generierung von Zufallszahlen – C_SeedRandom – C_GenerateRandom	Abschnitt 11.15
Funktionen zur parallelen Funktionsverwaltung – C_GetFunctionStatus – C_CancelFunction	Abschnitt 11.16
Rückruf-Funktionen – Surrender callbacks – Vendor-defined callbacks	Abschnitt 11.17

Funktionen und zugehörige Beschreibung in PKCS#11 (Teil 3 von 3)

7.2 Beispielprogramm

Im Beispielprogramm werden die folgenden Funktionen von CRYPT verwendet:

- Mit *C_OpenSession* wird eine Sitzung zwischen einer Anwendung und einem Token in einem bestimmten Slot eröffnet.
- *C_GenerateKey* generiert einen geheimen Schlüssel.
- Anschließend wird mit *C_EncryptInit* eine Verschlüsselungsoperation initialisiert und mit *C_EncryptUpdate* fortgesetzt.
- *C_EncryptFinal* beendet die mehrteilige Verschlüsselungsoperation.
- Mit *C_DecryptInit* wird eine Entschlüsselungsoperation initialisiert und mit *C_DecryptUpdate* fortgesetzt.
- *C_DecryptFinal* beendet die mehrteilige Entschlüsselungsoperation.
- Mit *C_CloseSession* wird die Sitzung geschlossen.

Beispiel

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "pkcs11.h"

static CK_BBOOL aTrue = TRUE;
static CK_BBOOL aFalse = FALSE;

void main()

{

    CK_MECHANISM MGDES1 = {CKM_DES_KEY_GEN, 0, 0};
    CK_MECHANISM MCDES1E = {CKM_DES_ECB, 0, 0};
    CK_ATTRIBUTE AGDES[] =

    {

        {CKA_EXTRACTABLE, &aTrue, sizeof(aTrue)}
        ,{CKA_SENSITIVE, &aFalse, sizeof(aFalse)}
        ,{CKA_ENCRYPT, &aTrue, sizeof(aTrue)}
        ,{CKA_DECRYPT, &aTrue, sizeof(aTrue)}

    };

    CK_ULONG NGDES = sizeof(AGDES)/12;
    CK_MECHANISM_PTR mgdes1 = &MGDES1;
    void *encin = 0;
    void *encout = 0;
    void *decout = 0;
```

```
unsigned int encinlen = 32*1024;
unsigned int encoutlen = 34*1024;
unsigned int decoutlen = 34*1024;
CK_BYTE_PTR encAktIn;
CK_BYTE_PTR encAktOut;
CK_BYTE_PTR decAktOut;
CK_ULONG encAcrylOutLen = 0;
CK_ULONG decAcrylOutLen = 0;
unsigned int i;
CK_RV rc;
CK_SESSION_HANDLE session;
CK_OBJECT_HANDLE key;
CK_ULONG inLen;
CK_ULONG outLen;
char *nextChar;

encin = calloc(encinlen, 1);
if (!encin)
{
    printf("----no more memory\n");
    return;
}

nextChar = (char*) encin;
for (i = 0; i < encinlen; i++)
       *nextChar++ = i % 256;

encout = malloc(encoutlen);
if (!encout)
{
    printf("----no more memory\n");
    return;
}

decout = malloc(decoutlen);
if (!decout)
{
    printf ("----no more memory\n");
    return;
}

/* Sitzung eröffnen */
rc = C_OpenSession(0, CKF_SERIAL_SESSION | CKF_RW_SESSION,
       NULL_PTR, NULL_PTR, &session);

if (rc != CKR_OK)
{
    printf("---- open session rc: %08x\n", rc);
    return;
}
```

```
printf("++++ open session: ok; session: %08X\n", session);

/* geheimen Schlüssel generieren */
rc = C_GenerateKey(session, mgdes1, AGDES, NGDES, &key);
if (rc != CKR_OK)
{
   printf("---- genkey rc: %08x\n", rc);
   return;
}
printf("++++ genkey: ok; key: %08X\n", key);

/* Verschlüsselungsoperation initialisieren */
rc = C_EncryptInit(session, &MCDES1E, key);
if (rc != CKR_OK)
{
   printf("---- cryini rc: %08x\n", rc);
   return;
}
printf("++++ cryini: encryptinit DES_ECB ok\n");

encAktIn = (CK_BYTE_PTR) encin;
encAktOut = (CK_BYTE_PTR) encout;
for (i = 0; i < 32; i++)
{
   /* outLen = 1024; */
   outLen = encoutlen - encAcrylOutLen;

   /* mehrteilige Verschlüsselungsoperation fortsetzen */
   rc = C_EncryptUpdate(session, encAktIn, 1024, encAktOut, &outLen);
   if (rc != CKR_OK)
   {
      printf("---- cry rc: %08x\n", rc);
      return;
   }
   encAcrylOutLen += outLen;
   encAktIn += 1024;           /* next portion */
   encAktOut += outLen;
}                               /* for (i = 0; i < 32; i++) */

outLen = encoutlen - encAcrylOutLen;

/* Verschlüsselungsoperation beenden */
rc = C_EncryptFinal(session, encAktOut, &outLen);
if (rc != CKR_OK)
{
   printf("---- cryfin rc: %08x\n", rc);
   return;
}
```

```
encAcrylOutLen += outLen;
printf("++++ cry: encrypt DES_ECB ok\n");


/* Entschlüsselungsoperation initialisieren */
rc = C_DecryptInit(session, &MCDES1E, key);
if (rc != CKR_OK)
{
    printf("---- cryini rc: %08x\n", rc);
    return;
}
printf("++++ cryini: decryptinit DES_ECB ok\n");

encAktOut = (CK_BYTE_PTR) encout;
decAktOut = (CK_BYTE_PTR) decout;
inLen = encAcrylOutLen >= 1024 ? 1024 : encAcrylOutLen;
while (inLen > 0)
{
    /* outLen = 1024; */
    outLen = decoutlen - decAcrylOutLen;

    /* mehrteilige Entschlüsselungsoperation fortsetzen */
    rc = C_DecryptUpdate(session, encAktOut, inLen,
       decAktOut, &outLen);
    if (rc != CKR_OK)
    {
       printf("---- cry rc: %08x\n", rc);
       return;
    }
    encAcrylOutLen -= inLen;
    if (encAcrylOutLen < 1024)
       inLen = encAcrylOutLen;
    decAcrylOutLen += outLen;
    encAktOut += 1024;          /* next portion */
    decAktOut += outLen;
}                                /* while (encAcrylOutLen > 0) */
outLen = decoutlen - decAcrylOutLen;


/* Entschlüsselungsoperation beenden */
rc = C_DecryptFinal(session, decAktOut, &outLen);
decAcrylOutLen += outLen;
printf("++++ cry: decrypt DES_ECB ok\n");
if (decAcrylOutLen == encinlen)
{
    printf("++++ length ok \n");
}
```

```
    else
    {
        printf("---- enc/dec: length diff %d %d\n", encinlen,
            decAcrylOutLen);
        return;
    }

    if (memcmp(encin, decout, decAcrylOutLen) == 0)
    {
        printf("++++ output ok \n");
    }

    else
    {
        printf("---- enc/dec: diff \n");
        return;
    }


    /* Sitzung schließen */
    rc = C_CloseSession(session);
    if (rc != CKR_OK)
    {
        printf("---- close session rc: %08x\n", rc);
        return;
    }
    printf ("++++ close session: ok\n");
}

/START-CPLUS-COMPILER
//MODIFY-SOURCE-PROPERTIES                      -
// LANG=*C(MODE=*ANSI)
//MODIFY-INCLUDE-LIBRARIES                      -
// STD-INCLUDE-LIBRARY=*USER-INCLUDE-LIBRARY,   -
// USER-INCLUDE-LIBRARY=(                       -
//    $.SYSLIB.CRYPT.nnn ....
//...

/START-BINDER ...
//INCLUDE-MODULES ELEMENT=
//INCLUDE-MODULES ELEMENT=CRYADAP,LIB=$.SYSLIB.CRYPT.nnn
//RESOLVE-BY-AUTOLINK LIBRARY=...
//...
//SAVE-LLM LIB=...
//END
```

8 Diagnose-Unterlagen erstellen

Sollte während des CRYPT-Betriebs eine Fehlersituation entstehen, die Sie selbst nicht beheben können, dann wenden Sie sich bitte an Ihren Ansprechpartner bei Fujitsu.

Für eine effiziente Fehlersuche benötigt Ihr Fujitsu-Ansprechpartner folgende Informationen:

- Genaue Beschreibung der Fehlersituation sowie Angaben, ob der Fehler reproduzierbar ist.
- Informationen zur Produkt- und Betriebsystemversion, sowie zum HSI (/390 bzw. x86).
- Die $SYSAUDIT.SYS.CONSLOG-Datei der BS2000-Session
- Dumps
- $.SYS.SERSLOG-Datei
- Ggf. CRYPDIAG-Unterlagen

  Die CRYPDIAG-Unterlagen erhalten Sie wie folgt:

  1. SYSLST auf eine Datei zuweisen.
  2. Programm CRYPDIAG starten:

     ► `CRYPDIAG`
  3. Anweisung TOTAL eingeben:

     ► `TOTAL`
  4. Anweisung END eingeben:

     ► `END`
  5. Umlenkung der SYSLST-Ausgabe zurücknehmen.
- Ggf. Trace-Dateien verwenden

  Den CRYPT-Benutzerschnittstellen-Trace CRYPT.COM schalten Sie mit dem Kommando /DCDIAG ein.

Weitere Einzelheiten zum Kommando /DCDIAG entnehmen Sie bitte dem Handbuch „openNet Server“ [1].

9 Returncodes

In diesem Kapitel finden Sie Returncodes zu CRYPT.

Die vom PKCS#11-Standard festgelegten Funktions-Returncodes werden im Standardheader im Main-Returncode (MRC) zurückgeliefert.

Die generische Komponente CPKC11T enthält die Definitionen für die MRCs. Darüber hinaus werden die Typ- und Strukturdefinitionen aus der Include-Datei PKCS11T.H in CPKC11T angeboten.

Die generische Komponente CRYASC2 enthält die Definitionen für den Sub-Returncode2 (SRC2), falls die MRC-Definition in der CPKC11T-Komponente „arguments_bad“ lautet.

Dieser Abschnitt listet jeweils alle möglichen MRCs und SRC2 mit ihrer Bedeutung auf.

CPKC11T – Allgemeine Datentypen

Die generische Komponente CPKC11T enthält Definitionen aus der Include-Datei PKCS11T.H.

Eine detaillierte Beschreibung zu CPKC11T finden Sie in PKCS#11 V2.20: Cryptographic Token Interface Standard im Kapitel 9 „General data types“.

Makro	Operanden
CPKC11T	MF= D / C

Main-Returncodes (MRC)

Eine detaillierte Beschreibung der Bedeutung der einzelnen MRCs finden Sie in der Spezifikation PKCS#11 V2.20 im Abschnitt 11.1 „Function return values“ ab Seite 112. Das Präfix CKR_ steht für Returncode. Die folgende Tabelle gibt einen Überblick über diese MRCs.

Wenn Sie die asynchrone Schnittstelle nutzen, können darüberhinaus zusätzliche Returncodes auftreten, die nicht in der PKCS#11-Spezifikation beschrieben sind. Diese sind in der Tabelle auf Seite 109 aufgelistet.

SRC2	SRC1	MRC	MRC-Identifier	Hinweis
00	00	0000	ok	
00	40	0001	cancel	
00	20	0002	host_memory	
00	80	0002	host_memory	
00	01	0003	slot_id_invalid	
xx	20	0005	general_error	
00	40	0006	function_failed	
aa	01	0007	arguments_bad	siehe CRYASC2
00	40	0008	no_event	
00	40	0009	need_to_create_threads	
00	40	000a	cant_lock	
00	40	0010	attribute_read_only	
00	40	0011	attribute_sensitive	

Main-Returncodes (MRC) (Teil 1 von 4)

SRC2	SRC1	MRC	MRC-Identifier	Hinweis
00	40	0012	attribute_type_invalid	
ii	01	0013	attribute_value_invalid	Der i. Wert ist ungültig.
00	40	0020	data_invalid	
00	40	0021	data_len_range	
00	20	0030	device_error	
00	20	0031	device_memory	
00	80	0032	device_removed	
00	40	0040	encrypted_data_invalid	
00	40	0041	encrypted_data_len_range	
00	80	0050	function canceled	
00	40	0051	function_not_parallel	
00	01	0054	function_not_supported	
aa	01	0060	key_handle_invalid	siehe CRYASC2
00	40	0062	key_size_range	
00	40	0063	key_type_inconsistent	
00	40	0064	key_not_needed	
00	40	0065	key_changed	
00	40	0066	key_needed	
00	40	0067	key_indigestible	
00	40	0068	key_function_not_permitted	
00	40	0069	key_not_wrappable	
00	40	006a	key_unextractable	
08	01	0070	mechanism_invalid	
08	01	0071	mechanism_param_invalid	
00	01	0082	object_handle_invalid	
00	40	0090	operation_active	
00	40	0091	operation_not_initialized	
00	40	00a0	pin_incorrect	
00	40	00a1	pin_invalid	
00	40	00a2	pin_len_range	

Main-Returncodes (MRC) (Teil 2 von 4)

SRC2	SRC1	MRC	MRC-Identifier	Hinweis
00	40	00a3	pin_expired	
00	40	00a4	pin_locked	
00	80	00b0	session_closed	
00	80	00b1	session_count	
00	40	00b3	session_handle_invalid	
00	01	00b4	session_parallel_not_supported	
00	40	00b5	session_read_only	
00	40	00b6	session_exists	
00	40	00b7	session_read_only_exists	
00	40	00b8	session_read_write_so_exists	
00	40	00c0	signature_invalid	
00	40	00c1	signature_len_range	
00	40	00d0	template_incomplete	
00	40	00d1	template_inconsistent	
00	80	00e0	token_not_present	
00	80	00e1	token_not_recognized	
00	80	00e2	token_write_protected	
00	40	00f0	unwrapping_key_handle_invalid	
00	40	00f1	unwrapping_key_size_range	
00	40	00f2	unwrapping_key_type_inconsist	
00	40	0100	user_already_logged_in	
00	40	0101	user_not_logged_in	
00	40	0102	user_pin_not_initialized	
00	40	0103	user_type_invalid	
00	40	0104	user_another_already_logged_in	
00	40	0105	user_too_many_types	
00	40	0110	wrapped_key_invalid	
00	40	0112	wrapped_key_len_range	
00	40	0113	wrapping_key_handle_invalid	
00	40	0114	wrapping_key_size_range	

Main-Returncodes (MRC) (Teil 3 von 4)

SRC2	SRC1	MRC	MRC-Identifier	Hinweis
00	40	0115	wrapping_key_type_inconsistent	
00	40	0120	random_seed_not_supported	
00	40	0121	random_no_rng	
00	40	0150	buffer_too_small	
00	40	0160	saved_state_invalid	
00	40	0170	information_sensitive	
00	40	0180	state_unsaveable	
00	80	0190	cryptoki_not_initialized	
00	40	0191	cryptoki_already_initialized	
00	40	01a0	mutex_bad	
00	40	01a1	mutex_not_locked	

Main-Returncodes (MRC) (Teil 4 von 4)

SRC2	SRC1	MRC	MRC-Identifier	Bedeutung
00	40	8000	PA_not_alloc	Parameterbereich nicht zugewiesen
00	40	8001	asynch_call_active	asynchroner Aufruf ist aktiv
02	00	8002	no_open_session	keine Session zu schließen
00	80	8003	CRYPT_down	Subsystem CRYPT wurde zwischen zwei Funktionsaufrufen beendet

Mögliche Main-Returncodes (MRC), die nicht in PKCS#11 V2.20 beschrieben sind

CRYASC2 – Sub-Returncodes 2

Die generische Komponente CRYASC2 enthält die Sub-Returncodes 2 für den Main-Returncode CPKC11T „arguments_bad“.

Makro	Operanden
CRYASC2	MF= D / C

Sub-Returncodes 2

SRC2	Kennung für Main-Returncode (Assembler)	Bedeutung
01	CRY2wrAction	ungültige Aktion
02	CRY2wrSession	ungültige Sitzung
03	CRY2wrInfo	ungültige Information
04	CRY2wrNotify	ungültige Notiz
05	CRY2wrSlotId	ungültige Slot-ID
06	CRY2wrDatIn	ungültige Dateneingabe
07	CRY2wrDatOut	ungültige Datenausgabe
08	CRY2wrMechanism	ungültiger Mechanismus
09	CRY2wrKey	ungültiger Schlüssel
0a	CRY2wrBKey	ungültiger Basisschlüssel
0b	CRY2wrTmplt	ungültiges Template
0c	CRY2wrAttr	ungültiges Attribut
0d	CRY2wrCnt	ungültige Anzahl
0e	CRY2wrPubKey	ungültiger öffentlicher Schlüssel
0f	CRY2wrPrvKey	ungültiger privater Schlüssel
10	CRY2wrPubTmplt	ungültiges Template für den öffentlichen Schlüssel
11	CRY2wrPrvTmplt	ungültiges Template für den privaten Schlüssel
14	CRY2wrTpkPr	ungültiges aktuelles Token
15	CRY2wrType	ungültiger Typ
16	CRY2wrLabel	ungültiger Label

Sub-Returncodes 2 (Teil 1 von 2)

SRC2	Kennung für Main-Returncode (Assembler)	Bedeutung
17	CRY2wrPin	ungültiger Pin
18	CRY2wrPinL	ungültige Pinlänge
19	CRY2wrUTyp	ungültiger Benutzertyp
1a	CRY2wrObj	ungültiges Objekt
1b	CRY2wrObjLst	ungültige Objektliste
1c	CRY2wrObjSz	ungültige Objektgröße
1d	CRY2wrObjCnt	ungültige Objektanzahl
1e	CRY2wrmaxObjCnt	ungültige Maximalanzahl von Objekten
1f	CRY2wrOpState	ungültiger Operationszustand
20	CRY2wrOpStateLen	ungültige Länge des Operationszustands
21	CRY2wrEncKey	ungültiger Verschlüsselungsschlüssel
22	CRY2wrAutKey	ungültiger Authentifikationsschlüssel
23	CRY2wrData	ungültige Daten
24	CRY2wrDataLen	ungültige Datenlänge
25	CRY2wrUnwrKey	ungültiger auspackender Schlüssel
26	CRY2wrWrKey	ungültiger eingepackter Schlüssel
27	CRY2wrWrKeyLen	ungültige Länge des eingepackten Schlüssels
28	CRY2wrWringKey	ungültiger einpackender Schlüssel
29	CRY2wrExec	ungültiger Ausführungsmodus
2a	CRY2wrRPostAd	ungültige Adresse des Postcodes
2b	CRY2wrRPostL	ungültige Länge des Postcodes
2c	CRY2wrBoid	ungültige Börsenkennung
2d	CRY2wrSigLen	ungültige Signaturlänge
2e	CRY2wrVers1	asynchrone Ausführung mit Version 1.0 nicht zugelassen
2f	CRY2wrAlloc	nicht allokierter Speicherbereich festgestellt
40	reserved	reservierter Parameterbereich ist nicht 0
F0	preceding	für alle Main-Returncodes: Der Fehler ist in der vorhergehenden Funktion aufgetreten (siehe auch nachfolgenden Abschnitt).

Sub-Returncodes 2 (Teil 2 von 2)

SRC2 „preceding“

Wenn SRC2 den Wert „preceding“ (X’F0’) liefert, bezieht sich der im MRC gelieferte Fehler nicht auf die Funktion, zu der er gemeldet wird, sondern auf eine vorausgegangene Funktion. Dies kann dann auftreten, wenn fehlerhafte Parameter nach Abschluss der vorausgegangenen Funktion erkannt werden.

Beispiel

Der Makroaufruf CCRYINI SES=1,ACT=*ENCRYPTINIT,MECH=mDes_key_gen,KEY=7 wird zunächst mit OK quittiert.
Der Folgeaufruf CCRY ACT=*ENCRYPT,DATAIN=...,INLEN=...,DATAOUT=..., OUTLEN=... wird mit X’F0400070’ beantwortet.
Der MRC „mechanism_invalid“ bezieht sich in diesem Fall nicht auf den Aufruf CCRY, sondern, wie Sie dem SRC2 entnehmen, auf den vorausgegangenen Aufruf CCRYINI.

Dieser Mechanismus kann für die Aktion *ENCRYPT nicht verwendet werden.

Fachwörter

Das vorliegende Fachwortverzeichnis ergänzt das Kapitel 4 „Definitions“ in der PKCS#11-Spezifikation.

AES (Advanced Encryption Standard)
Der AES ist eine FIPS-Veröffentlichung, die den Behörden der USA einen kryptographischen Algorithmus vorgibt. AES ist mittlerweile der Standard-Blockchiffrierer. AES wude von den belgischen Kryptologen Dr. Joan Daemen und Dr. Vincent Rijmen entwickelt.

ANSI (American National Standards Institute)
Entwickelt Standards über verschiedene akkreditierte Normen-Gremien (Accredited Standards Committee; ASC). Das X9-Komitee beschäftigt sich vorwiegend mit Sicherheitsstandards für Finanzdienstleistungen.

Asymmetrische Schlüssel (Asymmetric keys)
Ein zwar separates, jedoch integriertes Benutzerschlüsselpaar, bestehend aus einem öffentlichen und einem privaten Schlüssel. Es handelt sich dabei um Einwegschlüssel, d.h. ein Schlüssel, der zur Verschlüsselung bestimmter Daten verwendet wurde, kann nicht zur Entschlüsselung derselben Daten benutzt werden.

Blockchiffrierer (Block cipher)
Ein symmetrischer Chiffriercode, der auf der Basis von Blöcken (in der Regel 128-bit-Blöcke) von Klartext und verschlüsseltem Text funktioniert.

CBC (Cipher Block Chaining)
Der Prozess der Anwendung des XOR-Operators, um Klartext mit dem vorherigen chiffrierten Textblock in eine Entweder-Oder-Beziehung zu bringen, bevor dieser verschlüsselt wird. So wird einem Blockchiffrierer ein Rückkopplungsmechanismus hinzugefügt.

CFB oder CFM (Cipher Feedback Mode)
Ein Blockchiffrierer, der als selbstsynchronisierender Stromchiffriercode implementiert ist. Dabei wird eine vorgegebene Anzahl von Bits des Chiffrats als Eingabedaten für die Blockchiffre zurückgekoppelt und mit einem festen Schlüssel verschlüsselt.

Chiffrierter Text oder Chiffrat (Cipher text)
Das Ergebnis der Veränderung von Buchstaben oder Bits durch Ersetzung, Vertauschung oder beides.

Cryptoki
Programm-Schnittstelle zu Geräten, die kryptographische Informationen speichern und kryptographische Funktionen ausführen; spezifiziert vom PKSC#11-Standard.

DES (Data Encryption Standard; Datenverschlüsselungsstandard)
Ein 64-bit-Blockchiffrierer oder symmetrischer Algorithmus, der auch als Data Encryption Algorithm (DEA) (von ANSI) bzw. DEA-1 (von ISO) bezeichnet wird. Seit über 20 Jahren weit verbreitet, 1976 übernommen als „FIPS 46“.

Diffie-Hellman
Der erste Verschlüsselungsalgorithmus für öffentliche Schlüssel, der diskrete Logarithmen in einem endlichen Feld verwendete. Er wurde 1976 erfunden.

Distinguished Encoding Rules (DER)
Eine Untermenge von BER.

DSA (Digital Signature Algorithm)
Ein vom NIST entworfener digitaler Unterschriftenalgorithmus für öffentliche Schlüssel zur Verwendung in DSS.

DSS (Digital Signature Standard)
Ein vom NIST vorgeschlagener Standard (FIPS) für digitale Unterschriften unter Verwendung des DSA.

ECB (electronic codebook)
Ein Blockchiffrierer, bei dem der Klartextblock direkt als Eingabe für den Verschlüsselungs-Algorithmus verwendet wird. Der aus der Verschlüsselung resultierende Ausgabeblock wird unmittelbar als Chiffrat verwendet.

Entschlüsselung (Decryption)
Der Prozess des Rück-Umwandelns von chiffriertem (verschlüsseltem) Text in Klartext.

FIPS (Federal Information Processing Standard)
Eine vom NIST veröffentlichte Norm der Regierung der USA.

Geheimer Schlüssel (Secret key)
Der „Sitzungsschlüssel“ in symmetrischen Algorithmen.

Handle
Ein Wert zur Identifizierung einer Sitzung oder eines Objektes, den Cryptoki zuordnet (siehe auch PKCS#11-Spezifikation, Abschnitt 6.6.5 „Session handles and object handles“).

Hash-Funktion (Hash function)
Eine Einweg-Hash-Funktion ist eine Funktion, die einen Nachrichtenkern erzeugt, der zur Erzeugung des Originals nicht umgekehrt werden kann.

HMAC
Eine schlüsselabhängige Einweg-Hash-Funktion, die speziell für die Verwendung mit MAC (Message Authentication Code) gedacht ist und auf IETF RFC 2104 basiert.

IETF (Internet Engineering Task Force)
Eine umfangreiche offene internationale Gemeinschaft, bestehend aus Netzwerk-Entwicklern, Betreibern, Händlern und Forschungsspezialisten, deren Aufgabe die Entwicklung der Internetarchitektur und des reibungslosen Betriebs des Internets ist. Eine Mitarbeit steht jedem Interessenten offen.

Initialisierungsvektor (Initialization vector, IV)
Ein Block aus willkürlichen Daten, der unter Verwendung eines „Chaining Feedback Mode“ (siehe „Cipher Block Chaining“) als Ausgangspunkt für einen Blockchiffrierer dient.

Integrität (Integrity)
Ein Beleg dafür, dass Daten bei der Speicherung oder Übertragung (durch unbefugte Personen) nicht verändert werden.

ISO (International Organization for Standardization)
Diese Organisation ist für eine Vielzahl von Normen verantwortlich, wie das OSI-Modell sowie internationale Beziehungen mit dem ANSI bezüglich X. 509.

Klartext (Plain text oder clear text)
Daten oder Nachrichten in einer für den Menschen lesbaren Form vor dem Verschlüsseln – auch unverschlüsselter Text genannt.

MAC (Message Authentication Code)
Eine schlüsselabhängige Einweg-Hash-Funktion, bei der zur Verifizierung des Hash der identische Schlüssel benötigt wird.

MD2 (Message Digest 2)
Eine von einer Zufallspermutation von Bytes abhängige Einweg-Hash-Funktion mit 128 bit. Sie wurde von Ron Rivest entwickelt.

MD4 (Message Digest 4)
Eine Einweg-Hash-Funktion mit 128 bit, in der ein einfacher Satz von Bit-Manipulationen mit 32-bit-Operanden verwendet wird. Sie wurde von Ron Rivest entwickelt.

MD5 (Message Digest 5)
Eine verbesserte, komplexere Version von MD4. Es handelt sich jedoch immer noch um eine Einweg-Hash-Funktion mit 128 bit.

Mechanismus
Prozess zur Implementierung von kryptographischen Operationen.

Message Digest
Eine Prüfsumme, die aus einer Nachricht berechnet wird. Wenn Sie ein einziges Zeichen in der Nachricht verändern, hat die Nachricht einen anderen Message Digest.

Mutex-Objekte
Mutex abgleitet von Mutual Exclusion;
einfache Objekte, die sich zu jeder Zeit nur in einem von zwei Zuständen befinden können: ge- oder entsperrt (siehe auch PKCS#11-Spezifikation, Abschnitt 6.5.2 „Applications and threads“).

NIST (National Institute for Standards and Technology)
Eine Abteilung des „U.S. Department of Commerce“ (Wirtschaftsministerium der USA). Veröffentlicht Normen bezüglich der Kompatibilität (FIPS).

NSA (National Security Agency)
Eine Abteilung des „U.S. Department of Defense“.

Öffentlicher Schlüssel (Public key)
Die öffentlich verfügbare Komponente eines integrierten asymmetrischen Schlüsselpaares, die oft als Verschlüsselungsschlüssel bezeichnet wird.

OFB (Output Feedback Mode)
Wie beim CFB wird eine Blockchiffre als Stromchiffre eingesetzt. Anders als beim CFB werden die Bits der Ausgabe direkt zurückgekoppelt. Außerdem ist OFB nicht selbstsynchronisierend.

Operation
Abfolge mehrerer kryptographischer Funktionen.

PKCS (Public Key Crypto Standards)
Eine Reihe von De-facto-Normen zur Verschlüsselung mit öffentlichen Schlüsseln, die in Zusammenarbeit mit einem informellen Konsortium (Apple, DEC, Lotus, Microsoft, MIT, RSA und Sun) entwickelt wurden. Dazu gehören Algorithmen-spezifische und von Algorithmen unabhängige Implementierungsnormen. Spezifikationen zur Definition von Nachrichtensyntax und anderen Protokollen, die von RSA Data Security, Inc., gesteuert werden.

Privater Schlüssel (Private key)
Die im privaten Besitz befindliche „geheime“ Komponente eines integrierten asymmetrischen Schlüsselpaares, die oft als Entschlüsselungsschlüssel bezeichnet wird.

Pseudo-Zufallszahl (Pseudo-random number)
Eine Zahl, die aus der Anwendung von Algorithmen errechnet wird, die Zufallswerte auf Eingabewerte erzeugen, die aus der Computerumgebung abgeleitet sind (z.B. Mauskoordinaten). Siehe Zufallszahl.

RC2 (Rivest Cipher 2)
Symmetrischer 64-bit-Blockchiffrierer mit variabler Schlüsselgröße, ein brancheninterner Schlüssel von RSA, SDI.

RC4 (Rivest Cipher 4)
Stromchiffriercode mit variabler Schlüsselgröße, war früher Eigentum von RSA Data Security, Inc. Da RC4 mittlerweile zu viele Schwächen zeigt, wird von seiner Verwendung dringend abgeraten. Siehe hierzu auch RFC 7465 "Prohibiting RC4 Cipher Suites".

RFC (Request for Comment)
Ein IETF-Dokument aus der Untergruppe FYI RFC, die Überblicke und Einführungen gibt, oder aus der Untergruppe STD RFC, die Internet-Normen angibt. Die Abkürzung FYI steht für „For Your Information“ – zu Ihrer Information. Jeder RFC hat zur Indizierung eine RFC-Nummer, anhand derer er abgerufen werden kann (www.ietf.org).

RSA
Abkürzung von RSA Data Security, Inc. Steht auch für die Firmenchefs Ron Rivest, Adi Shamir und Len Adleman oder bezieht sich auf den von ihnen erfundenen Algorithmus. Der RSA-Algorithmus wird in der Kryptographie mit öffentlichen Schlüsseln verwendet. Seine Funktionsweise beruht auf der Tatsache, dass zwei große Primzahlen zwar leicht miteinander zu multiplizieren sind, aber das Produkt nur schwer wieder in sie zu zerlegen ist.

Salt
Eine zufällige Zeichenkette, die mit Passwörtern (oder Zufallszahlen) verknüpft wird, bevor an ihnen mit einer Einweg-Funktion Operationen durchgeführt werden. Durch diese Verkettung wird das Passwort effektiv verlängert und verfremdet. Damit ist der chiffrierte Text besser gegen Wörterbuchangriffe geschützt.

Schlüssel (Key)
Ein Mittel zur Gewährung bzw. Verweigerung von Zugriff, Eigentumsrechten oder Steuerungsbefugnissen. Wird durch eine beliebige, große Anzahl von Werten dargestellt.

Schlüsselaustausch (Key exchange)
Ein Schema mit zwei oder mehreren Knoten zum Übertragen eines geheimen Sitzungsschlüssels über einen nicht gesicherten Kanal.

Schlüssellänge (Key length)
Die Anzahl der Bits zur Darstellung der Schlüsselgröße. Je länger der Schlüssel, desto stärker ist er.

Schlüsselverwaltung (Key management)
Das Verfahren zum sicheren Speichern und Verteilen akkurater kryptographischer Schlüssel. Der Gesamtprozess des sicheren Erstellens und Verteilens von kryptographischen Schlüsseln an befugte Empfänger.

SET (Secure Electronic Transaction)
Dient der sicheren Übertragung von Kreditkartennummern über das Internet.

SHA-1 (Secure Hash Algorithm)
Die 1994 vorgenommene Überarbeitung des vom NIST entwickelten SHA (FIPS 180-1). SHA-1 wird zusammen mit DSS zur Erzeugung eines 160-bit-Hash verwendet. Es ähnelt dem sehr beliebten und weitverbreiteten MD4.

Sitzungsschlüssel (Session key)
Der geheime (symmetrische) Schlüssel zum Verschlüsseln aller Datensätze auf Transaktionsbasis. Für jede Kommunikationssitzung wird ein anderer Sitzungsschlüssel verwendet.

Slot
Über die PKCS#11 können gemäß Definition logische kryptographische Funktionseinheiten (Slots) simultan zu einander genutzt werden. In CRYPT wird derzeit nur ein Slot unterstützt.

SSL (Secure Socket Layer)
Wurde von Netscape zur Gewährleistung von Sicherheit und zur Geheimhaltung im Internet entwickelt. Unterstützt die Server- und Client-Authentisierung und gewährleistet die Sicherheit und Integrität des Übertragungskanals. Wirkt auf der Übertragungsebene und dient als „Socket-Bibliothek“, wodurch eine anwendungsunabhängige Wirkungsweise ermöglicht wird. Verschlüsselt den gesamten Kommunikationskanal.

Stromchiffriercode (Stream cipher)
Eine Klasse der symmetrischen Schlüsselverschlüsselung, bei der die Umwandlung für jedes zu verschlüsselnde Symbol des Klartextes geändert werden kann; wird für Umgebungen mit geringer Speicherkapazität zum Puffern von Daten empfohlen.

Symmetrischer Algorithmus (Symmetric algorithm)
Wird auch als konventioneller, geheimer Schlüssel- oder Einzelschlüsselalgorithmus bezeichnet. Der Verschlüsselungsschlüssel ist entweder mit dem Entschlüsselungsschlüssel identisch, oder ein Schlüssel kann aus dem anderen abgeleitet werden. Es gibt zwei Unterkategorien – Block und Strom.

TLS (Transport Layer Security)
Ein IETF-Entwurf. Die Version 1 basiert auf der Version 3.0 des SSL-Protokolls und dient zur Wahrung der Privatsphäre bei der Kommunikation über das Internet.

Triple-DES
Eine Verschlüsselungskonfiguration, in der der DES-Algorithmus dreimal mit drei unterschiedlichen Schlüsseln verwendet wird.

Unverschlüsselter Text
siehe Klartext.

Zufallszahl (Random number)
Ein wichtiger Aspekt für viele Verschlüsselungssysteme sowie ein notwendiges Element beim Erzeugen von eindeutigen Schlüsseln, die für Gegner nicht berechenbar sind. Echte Zufallszahlen werden normalerweise aus analogen Quellen abgeleitet und erfordern in der Regel den Einsatz von besonderer Hardware.

Literatur

Die Handbücher finden Sie im Internet unter *http://manuals.ts.fujitsu.com*. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

[1] **openNet Server** (BS2000)
BCAM
Benutzerhandbuch

[2] **BS2000 OSD/BC**
Kommandos
Benutzerhandbuch

[3] **BS2000 OSD/BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zusätzliche Literatur

PKCS#11 V2.30: Cryptographic Token Interface Standard
RSA Laboratories, April 2009:
http://germany.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

Stichwörter