

English



FUJITSU Software BS2000

CRYPT V2.0

Security with Cryptography

User Guide

Edition July 2017

Comments... Suggestions... Corrections...

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:

manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH

www.cognitas.de

Copyright and Trademarks

Copyright © 2017 Fujitsu Technology Solutions GmbH.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Contents

1	Introduction	7
1.1	Objectives and target groups of this manual	7
1.2	Security information	7
1.3	Summary of contents	8
1.4	Changes since the last edition of the manual	9
1.5	Notational conventions	10
2	Encryption in BS2000/OSD	11
2.1	PKCS#11 standard	11
3	PKCS#11 functions and the corresponding interfaces in BS2000	13
4	PKCS#11 mechanisms in CRYPT Services	17
5	PKCS#11 – implementation in BS2000	19
5.1	Mechanisms	20
5.2	General data types	23
5.3	Objects	23
5.4	Functions	24

6	Description of the Assembler macro calls	27
<hr/>		
6.1	Metasyntax for macros	28
6.2	Macro syntax for format operands	30
6.3	Asynchronous execution	32
6.4	Description of the macro calls	34
	CGENRAL – general functions	35
	CGTSTMI – display information	38
	CWTFSL – waiting for a slot event	42
	CINITTK – initialize token	43
	CPIN – initialize or modify PIN	44
	CSESSION – session management	45
	COPSTAT – display/set operation state	48
	CLOG – login/logout	50
	COBJMGT – object management	51
	CCRYINI – initialize cryptographic operation	56
	CCRY – execute cryptographic operation	58
	CCRYFIN – finalize cryptographic operation	63
	CGENKEY – generate secret key	66
	CGENKPR – generate key pair	68
	CWRPKEY – wrap key	70
	CUNWKEY – unwrap key	72
	CDRVKEY – derive key	74
	CRANDOM – generate random numbers	76
6.5	Sample programs	78
6.5.1	Synchronous execution – example	78
6.5.2	Asynchronous execution – example	83
7	Description of the functions in C	93
<hr/>		
7.1	Notes about the description in PKCS#11	93
7.2	Sample program	97

8	Creating diagnostic documents	103
9	Return codes	105
	CPKC11T – general data types	106
	CRYASC2 – sub return code 2	109
	Glossary	113
	Related publications	119
	Index	121

1 Introduction

CRYPT is a subsystem that provides cryptographic functions and interfaces in BS2000.

1.1 Objectives and target groups of this manual

This manual is designed for use by system administrators and users of the cryptographic interface in BS2000.

To ensure cryptographically secure application programming it is assumed that the programmer has knowledge of the cryptographic functions implemented.

1.2 Security information



DISCLAIMER!

If you implement a security architecture within the framework of a project, you will need to configure and use the selectable functions and mechanisms according to your actual requirements.

Fujitsu Technology Solutions takes no responsibility for any incidents or damage that may occur as a result of incorrectly used parameter values, or incorrectly specified mechanisms or functions of the CRYPT interfaces.

1.3 Summary of contents

This manual describes the architecture of CRYPT and the cryptographic functions and interfaces in BS2000.

It is based on the description of the “PKCS#11 V2.20: Cryptographic Token Interface Standard” from RSA Laboratories dated December 1999. This standard is freely available on the Internet under <http://germany.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm> and will be required in order to correctly implement the parameters. Additional references to chapters or sections of the PKCS#11 standard are made throughout the manual as and when this additional information may prove useful.

The [chapter “Description of the Assembler macro calls” on page 27](#) provides detailed information about the Assembler macro calls using both syntax diagrams and detailed operand descriptions. A programming example finishes off the chapter. The PKCS#11 standard provides additional detailed information.

The [chapter “Description of the Assembler macro calls” on page 27](#) [chapter “Description of the functions in C” on page 93](#) helps the user to find their way around and makes it easier to locate the detailed descriptions in the PKCS#11 standard. This chapter provides an overview of the functions of the standard that are implemented in C in the CRYPT interface and cross-references the appropriate chapters or sections in the PKCS#11 standard. Chapter 8 also contains a complete programming example.

The [chapter “Creating diagnostic documents” on page 103](#) explains how diagnostic documents are created.

The PKCS#11 standard is needed if you wish to use the C interface.

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at <http://manuals.ts.fujitsu.com>. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

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

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at <http://manuals.ts.fujitsu.com>.

1.4 Changes since the last edition of the manual

This manual describes the functionality of CRYPT V2.0A. Compared to the previous edition of this manual the following changes have been introduced:

No hardware is used any more to execute the CRYPT functions. The functions are executed in BS2000 without configuration of the subsystem.

1.5 Notational conventions

This manual uses the following notation conventions:



is used to identify general notes



CAUTION!

is used to identify security and warning notes

fixed pitch text

identifies programming text in examples

text in italics

identifies names of cryptographic functions and operations (in C), function parameters and files in descriptive text; syntax variables

The conventions used to describe the Assembler macro calls are described in the sections [“Metasyntax for macros” on page 28](#) and [“Macro syntax for format operands” on page 30](#).

References to places within this manual will specify the relevant page number and, if necessary, also the section or chapter. References to topics that are described in other manuals will give the short title of the appropriate manual. The complete title can be found in the list of references at the back of this manual.

2 Encryption in BS2000/OSD

2.1 PKCS#11 standard

RSA Laboratories, in conjunction with developers of security systems from industry, educational establishments and governmental sectors, have developed specifications, the aim of which is to speed the development of encryption and decryption technology using public keys.

These specifications are known as the Public-Key Cryptography Standards, or PKCS for short. Sections of this set of PKCS standards have since become components of many formal and de-facto standards, for example, ANSI X9 documents, PKIX, SET, S/MIME and SSL.

PKCS#11 is the Cryptographic Token Interface Standard. The CRYPT product is based on Version 2.10 of PKCS#11. CRYPT also comprises functions, such as the AES support, which are based on the newer version 2.11 of PKCS#11.

This de-facto standard specifies a program interface (API) to devices that save cryptographic information and carry out cryptographic functions. The short form for the Cryptographic Token Interface is Cryptoki.

The specification for the PKCS#11 standard can be found on the Internet under:

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

3 PKCS#11 functions and the corresponding interfaces in BS2000

The table below provides a summary of which BS2000 interface the various PKCS#11 functions are assigned to.

Key to the table

- The C_ prefix in the PKCS#11 functions indicates a function.
- “no” in the “Functionality available” column:
The interfaces are, for the sake of portability, implemented in BS2000, the macro calls have no function, however since, for this function, just an interface is provided without any associated functionality, a call is answered with MAINCODE = LINKAGE_ERROR and SUBCODE1 = FCT_NOT_AVAILABLE.

PKCS#11 function	BS2000 interface (SVC/ISL)	Functionality available
General-purpose functions		
C_Initialize	CGENRAL	yes
C_Finalize	CGENRAL	yes
C_GetInfo	CGENRAL	yes
C_GetFunctionList	CGENRAL	yes, via C interface
Slot and token management functions		
C_GetSlotList	CGTSTMI	yes
C_GetSlotInfo	CGTSTMI	yes
C_GetTokenInfo	CGTSTMI	yes
C_WaitForSlotEvent	CWTFSL	no
C_GetMechanismList	CGTSTMI	yes
C_GetMechanismInfo	CGTSTMI	yes
C_InitToken	CINITTK	no

PKCS#11 functions and the corresponding interfaces in BS2000

(Part 1 of 4)

PKCS#11 function	BS2000 interface (SVC/ISL)	Functionality available
C_InitPIN	CPIN	no
C_SetPIN	CPIN	no
Session management functions		
C_OpenSession	CSESSION	yes
C_CloseSession	CSESSION	yes
C_CloseAllSessions	CSESSION	yes
C_GetSessionInfo	CSESSION	no
C_GetOperationState	COPSTAT	no
C_SetOperationState	COPSTAT	no
C_Login	CLOG	yes
C_Logout	CLOG	yes
Object management functions		
C_CreateObject	COBJMGT	yes
C_CopyObject	COBJMGT	yes
C_DestroyObject	COBJMGT	yes
C_GetObjectSize	COBJMGT	no
C_GetAttributeValue	COBJMGT	yes
C_SetAttributeValue	COBJMGT	yes
C_FindObjectsInit	COBJMGT	yes
C_FindObjects	COBJMGT	yes
C_FindObjectsFinal	COBJMGT	yes
Encryption functions		
C_EncryptInit	CCRYINI	yes
C_Encrypt	CCRY	yes
C_EncryptUpdate	CCRY	yes
C_EncryptFinal	CCRYFIN	yes
Decryption functions		
C_DecryptInit	CCRYINI	yes
C_Decrypt	CCRY	yes

PKCS#11 functions and the corresponding interfaces in BS2000

(Part 2 of 4)

PKCS#11 function	BS2000 interface (SVC/ISL)	Functionality available
C_DecryptUpdate	CCRY	yes
C_DecryptFinal	CCRYFIN	yes
Message digesting functions		
C_DigestInit	CCRYINI	yes
C_Digest	CCRY	yes
C_DigestUpdate	CCRY	yes
C_DigestKey	CCRYINI	yes
C_DigestFinal	CCRYFIN	yes
Signing and MACing functions		
C_SignInit	CCRYINI	yes
C_Sign	CCRY	yes
C_SignUpdate	CCRY	yes
C_SignFinal	CCRYFIN	yes
C_SignRecoverInit	CCRYINI	no
C_SignRecover	CCRY	no
Functions for verifying signatures and MACs		
C_VerifyInit	CCRYINI	yes
C_Verify	CCRY	yes
C_VerifyUpdate	CCRY	yes
C_VerifyFinal	CCRYFIN	yes
C_VerifyRecoverInit	CCRYINI	yes
C_VerifyRecover	CCRY	yes
Dual-function cryptographic functions		
C_DigestEncryptUpdate	CCRY	no
C_DecryptDigestUpdate	CCRY	no
C_SignEncryptUpdate	CCRY	no
C_DecryptVerifyUpdate	CCRY	no

PKCS#11 functions and the corresponding interfaces in BS2000

(Part 3 of 4)

PKCS#11 function	BS2000 interface (SVC/ISL)	Functionality available
Key management functions		
C_GenerateKey	CGENKEY	yes
C_GenerateKeyPair	CGENKPR	yes
C_WrapKey	CWRPKEY	yes
C_UnwrapKey	CUNWKEY	yes
C_DeriveKey	CDRVKEY	yes
Random number generation functions		
C_SeedRandom	CRANDOM	yes
C_GenerateRandom	CRANDOM	yes
Parallel function management functions		
C_GetFunctionStatus	–	no
C_CancelFunction	–	no
Callback functions		
Surrender callbacks	–	no
Vendor-defined callbacks	–	no

PKCS#11 functions and the corresponding interfaces in BS2000

(Part 4 of 4)

4 PKCS#11 mechanisms in CRYPT Services

A mechanism is a process used to implement cryptographic operations.

The CRYPT subsystem covers the mechanisms of the PKCS#11 standard as listed below.

The explanations for the various mechanisms can be found in chapter [“Glossary” on page 113](#), and in chapter 12 “Mechanisms” of the PKCS#11 standard.

The prefix CKM_ in the standard stands for mechanism type.

Symmetric algorithms

- Block ciphers
 - DES
 - DES3
 - SD2 (corresponds to RC2)
 - AES
- Operating modes
 - ECB
 - CBC
 - OFB
- Stream ciphers
 - SD4 (corresponds to RC4)
- Generation of keys for the supported algorithms

Hash algorithms and integrity codes

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

Public key procedure

- Code/key exchange procedures
 - RSA (PKCS#1)
 - RSA (pure)
 - Diffie-Hellman
- Signature procedures
 - RSA (PKCS#1)
 - RSA (pure)
 - DSA
- Generation of keys for the supported algorithms

Random generators

Pseudo random generators DES OFB, DES3 OFB

5 PKCS#11 – implementation in BS2000

This chapter describes all the features of the CRYPT interface in BS2000 that do not correspond to the specifications of the PKCS#11 standard.

A range of different prefixes are used in the PKCS#11 standard. The table shown below lists the prefixes used in this chapter and their meanings. You will also find a description of additional prefixes in chapter 5 “Symbols and Abbreviations” of the PKCS#11 standard.

Prefix	Meaning
C_	function
CKA_	attribute
CKM_	mechanism type
CKR_	return code

5.1 Mechanisms

A mechanism specifies how a particular cryptographic process is to be performed.

The table below shows which Cryptoki mechanisms are supported by which cryptographic operations. It replaces the table at the beginning of chapter 12 “Mechanisms” of the PKCS#11 V2.20 standard.

XX: The function(s) is/are supported.

X: Single-part operations are supported.

Wrap and unwrap are only possible on secret keys.

Mechanisms	Functions						
	Encrypt & Decrypt	Sign & Verify	Sign- / Verify- Recover	Digest	Generate 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					

Mechanisms and functions supporting these mechanisms

(Part 1 of 3)

Mechanisms	Functions						
	Encrypt & Decrypt	Sign & Verify	Sign- / Verify-Recover	Digest	Generate Key/ KeyPair	Wrap & Unwrap	Derive
CKM_RC4_KEY_GEN					xx		
CKM_RC4	xx						
CKM_RC5_KEY_GEN					xx		
CKM_RC5_ECB	xx					xx	
CKM_RC5_CBC	xx					xx	
CKM_RC5_CBC_PAD	xx					xx	
CKM_RC5_MAC_GENERAL		xx					
CKM_RC5_MAC		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_IDEA_KEY_GEN					xx		
CKM_IDEA_ECB	xx					xx	
CKM_IDEA_CBC	xx					xx	
CKM_IDEA_CBC_PAD	xx					xx	
CKM_IDEA_MAC_GENERAL		xx					
CKM_IDEA_MAC		xx					
CKM_MD2				xx			
CKM_MD2_HMAC_GENERAL		xx					
CKM_MD2_HMAC		xx					

Mechanisms and functions supporting these mechanisms

(Part 2 of 3)

Mechanisms	Functions						
	Encrypt & Decrypt	Sign & Verify	Sign- / Verify-Recover	Digest	Generate Key/KeyPair	Wrap & Unwrap	Derive
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_RIPEMD128				xx			
CKM_RIPEMD128_HMAC_GENERAL		xx					
CKM_RIPEMD128_HMAC		xx					
CKM_RIPEMD160				xx			
CKM_RIPEMD160_HMAC_GENERAL		xx					
CKM_RIPEMD160_HMAC		xx					
CKM_AES_KEY_GEN					xx		
CKM_AES_ECB	xx					xx	
CKM_AES_CBC	xx					xx	
CKM_AES_CBC_PAD	xx					xx	
CKM_AES_MAC_GENERAL		xx					
CKM_AES_MAC		xx					

Mechanisms and functions supporting these mechanisms

(Part 3 of 3)

5.2 General data types

slotId

In CRYPT V2.0 the SlotId is meaningless. The slotId needs not to be provided in the functions that require a slotId

5.3 Objects

Attributes when generating objects

The application must ensure that the transferred templates are both correct and complete. An incorrect or incomplete attribute template when generating an object is not always recognized and expanded (see also PKCS#11 V2.20 standard: chapter 10).

This affects the following functions:

- C_CreateObject
- C_CopyObject
- C_GenerateKey
- C_GenerateKeyPair
- C_UnwrapKey
- C_DeriveKey

The attributes CKA_SENSITIVE, CKA_EXTRACTABLE, CKA_LOCAL, CKA_TOKEN

The notes in this section refer to section 10.4 “Storage objects” in the PKCS#11 V2.20 standard.

The persistent saving of keys and other objects is not supported. The attribute CKA_TOKEN must, as a result, always be set to FALSE.



CAUTION!

No continuous protection of secret data of an object.

The attributes CKA_SENSITIVE and CKA_EXTRACTABLE are capable of preventing the secret data being read using *C_GetAttributeValue* or by exporting this data using *C_WrapKey*. However, the flags that are set by this process are ignored by other Cryptoki functions, thus allowing the protective function to be circumvented.

The attribute CKA_LOCAL is also not always set correctly.

5.4 Functions

An operation is a series of several functions.

The notes in this section refer to chapter 11 “Functions” of the PKCS#11 V2.20 standard.

- In BS2000 most functions from version 1.1 can be executed not just synchronously via the BS2000 specific assembler interfaces, but also asynchronously.

You can find more detailed information on this in [section “Asynchronous execution” on page 32](#).

- The following general functions are not required in BS2000.

C_InitToken

C_Login

C_Logout

- The maximum output data length for the functions, *encryptFinal*, *decryptFinal*, *digestFinal*, *signFinal*, *verifyFinal*, *wrapKey* and *generateRandom* is 2048 bytes.
- For certain functions you should **not** initially determine the size of the output area. This has an adverse effect on performance. See section 11.2 “Conventions for functions returning output in a variable-length buffer” of the PKCS#11 V2.20 standard.
- All operations initiated using *...Init* will **not** be terminated by follow-up calls that supply the return code CKR_SESSION_HANDLE_INVALID or CKR_ARGUMENTS_BAD. The return codes CKR_KEY_HANDLE_INVALID, CKR_MECHANISM_INVALID, CKR_ATTRIBUTE_VALUE_INVALID will **not** usually terminate the active operation either. See the section 11.4 “General-purpose functions” of the PKCS#11 V2.20 standard.
- The number of sessions that a user can open simultaneously is limited to 999. You can set this limit to a lower value with the CRYPTO-SESSION-LIMIT operand of the ADD-USER or MODIFY-USER-ATTRIBUTES command. Further information regarding this command you will find in the “BS2000 OSD/BC Commands” user manual [2]

If this limit of parallel session is exceeded the return code `session_count` provided.

C_Initialize, C_Finalize:

- **C_Initialize:**
In addition to the functionality described in the PKCS#11 standard, C_Initialize in BS2000 controls whether the program works synchronously or asynchronously with CRYPT:

If you select synchronous function execution, a C_Initialize in BS2000 is not required. *pInitArgs* must be a NULL_PTR.
- **C_Finalize:**
C_Finalize has no effect in BS2000.

Compare section 11.4 “General-purpose functions” in the PKCS#11 V2.20 standard.

C_GetMechanismInfo:

In the mechanism information data of the mechanisms CKM_RSA_PKCS and CKM_RSA_X_509, the flags for the operations *Sign* and *Verify* are not set. Despite this, the corresponding operations are still supported.
See the section 11.5 “Slot and token management functions” of the PKCS#11 V2.20 standard.

C_CopyObject:

The flags CKA_SENSITIVE and CKA_EXTRACTABLE which are used to protect the security-relevant data of a key from being read or from being extracted can be changed in both directions.
See section 11.7 “Object management functions” of the PKCS#11 V2.20 standard.

C_SetAttributeValue:**CAUTION!**

No complete check of transferred values is carried out.
It is possible that inconsistent states may occur since it is possible to set incorrect attribute values and to modify attributes which (according to the standard) may not be modified.
The calling application must ensure that these states do not occur.

C_GenerateKeyPair:

In order to generate an RSA key it is not necessary to specify the attribute CKA_PUBLIC_EXPONENT. See the section 11.14 “Key management functions” of the PKCS#11 V2.20 standard.

C_Encrypt, C_Decrypt, C_Digest, C_Sign, C_Verify

The cryptographic single-part operations (*C_Encrypt*, *C_Decrypt*, *C_Digest*, *C_Sign*, *C_Verify*) correspond to an update operation followed by a final operation.

As a result, you can terminate a sequence of *C_EncryptUpdate* calls using either *C_EncryptFinal* or *C_Encrypt*.

Single-part operation	Corresponding update and final operations
<i>C_Encrypt</i>	<i>C_EncryptUpdate</i> + <i>C_EncryptFinal</i>
<i>C_Decrypt</i>	<i>C_DecryptUpdate</i> + <i>C_DecryptFinal</i>
<i>C_Digest</i>	<i>C_DigestUpdate</i> + <i>C_DigestFinal</i>
<i>C_Sign</i>	<i>C_SignUpdate</i> + <i>C_SignFinal</i>
<i>C_Verify</i>	<i>C_VerifyUpdate</i> + <i>C_VerifyFinal</i>

The input data length of an update operation need not necessarily meet the criteria described in chapter 12ff of the PKCS#11 V2.20 standard. These criteria only refer to the overall length.

See sections 11.8 through 11.12 “Encryption / Decryption / Message digesting / Signing and MACing functions and functions for verifying signatures and MACs” of the PKCS#11 V2.20 standard.

6 Description of the Assembler macro calls

The introductory part of the chapter provides information about the following topics:

- Metasyntax for macros
- Macrosyntax for format operands



You will also find detailed information in the “Executive Macros” user guide [\[3\]](#) about the following:

- Use of registers
- Returned information and error messages (return codes) and their transfer in the standard header
- Standard header
- Eventing
- Contingency processes

After this introductory section, the individual macro calls are listed for the user along with syntax diagrams and operand descriptions. You will also find a reference to the relevant detailed descriptions in the PKCS#11 standard.

6.1 Metasyntax for macros

Kennzeichnung	Meaning	Example
UPPERCASE LETTERS	Uppercase letters denote keywords or constants which must be specified exactly as shown. Keywords begin with *.	ACTION=*INITIALIZE
lowercase letters	Lowercase letters denote the types of values or variables for which current values must be specified by the user, i.e. their value may be different from case to case.	AUTKEY=<var: int:4>
=	The equals sign links an operand to the associated operand value.	SLOTID =<var: int:4>
/	The slash separates simple alternative operand values.	ACTION= *OPENSESSION / *CLOSESESSION
< >	Angle brackets enclose the data type of the operand.	<var: int:4>
<u>underscored values</u>	The underscore indicates the default value, i.e. the value assumed by the system, if no value is specified by the user. Default values which contain the character “_” are indicated with “default:” at the beginning instead of the underscore.	TEMPL = <var: pointer> / <u>NULL</u> no specification implies TEMPL=NULL

Metasyntax for macros

Data types of the operand values

Data type	Character set	Special features
integer	0..9,+,-	“+” or “-” may be specified only as the first character. The suffix n..m specifies the permissible value range. Example: in syntax diagram: SESSION=<integer 0 .. 2147483647> actual input: SESSION=5
var:		Starts a variable specification. The colon is followed by the data type of the variable. Example: in syntax diagram: LEN=<var: int:4> actual input: LEN=100

Data types of variables and register contents

integer (n)	An integer which occupies n bytes, where $n \leq 4$. If the length specification is omitted, $n=1$ is assumed.
enum NAME(n)	A list which occupies n bytes, where $n \leq 4$. If the length specification is omitted, $n=1$ is assumed.
pointer	Pointer (the address is passed).

6.2 Macro syntax for format operands

The macro operands may be subdivided into two groups:

Control operands	Operands which determine the macro form and generation
Function operands	Interface-specific operands

Control operands

MF	controls code generation (“macro form”)
PREFIX	controls name generation (first character)
MACID	controls name generation (2nd to 4th characters)
PARAM	controls parameter area addressing
XPAND	controls data structure expansion

Macro forms

The MF operand determines the macro form and can have any of the following values:

MF = C | D | L | E | M

MF = C The layout of the data structure (usually the parameter area) is generated, with names being specified for each individual field and equate. The data structure becomes part of the current control section (CSECT/DSECT).

The function operands of the macro are not evaluated.

PREFIX

The PREFIX operand is used for name generation. PREFIX is one, and only one, letter that is used as the first letter of all names. The identifier of the functional unit of which the macro is a part is used as the default PREFIX. In order to avoid the use of identical names, PREFIX should be specified explicitly if the same data structure is used more than once within a module.

MACID

The MACID operand is also used for name generation and defines the 2nd to 4th characters of a name. The default value is formed by two characters serving as the development group ID and one character serving as a macro-specific identifier. The default value guarantees that the same name will not occur twice within one component group.

- MF = D as with MF = C;
in addition, a DSECT statement is generated. The MACID operand is ignored, i.e. the default value is assumed.
- MF = L generates a parameter area entity by evaluating the function operands. This macro form generates no field names; the label specification is used for naming the generated constants.
- MF = E generates the instructions required for the function call. The function operands are ignored. The PARAM control operand must be used to ensure that the parameter area can be addressed correctly:

PARAM

PARAM=<address>

parameter area address, given as a name.

PARAM=(<reg>)

parameter area address, to be fetched from the register with the name <reg>.

- MF = M modifies a parameter area previously initialized by copying a MF=L form, evaluating the specified function operands in the process. Any operands that are not specified retain their original state.

The macro caller is responsible for ensuring the consistency of the parameter area.

MF=M cannot be used unless the MF=D form or MF=C form was invoked with the same values for the PREFIX and MACID operands and a USING statement was issued to address the DSECT (MF=D form).

6.3 Asynchronous execution

In BS2000 most functions from version 1.1 up can be performed both synchronously and asynchronously. This is thanks to the BS2000-specific interfaces.

Synchronous execution

In the case of synchronous execution, control is only restored to the program when the function has been executed. This may entail some waiting, for example, whilst keys are being generated. The advantage of synchronous execution is its simple operation.

Asynchronous execution

In the case of asynchronous execution, control is restored to the program before the function has been fully executed. The program can therefore make use of waiting time.

Proceed as follows for asynchronous execution:

1. Define an event code using the ENAEI call.
2. If an interrupt routine is to be started when an event occurs, define this routine with the ENACO call.
3. Using the call CGENRAL ACTION=*INITIALIZE, EXEC=*ASYNCHRON specify that the task works asynchronously with CRYPT.
4. Specify the event code for CRYPT macro calls with the operand BOID=....

In addition, specify a postcode area (RPOSTAD=, RPOSTL=).

CRYPT uses the postcode1 which comprises the following:

ETC	SC1	MC2	MC1
-----	-----	-----	-----

ETC Event Code Type
for CRYPT is '23'
The macro call CPKC11T contains a symbolic value for this.

SC1 Subcode1
corresponds to the value in the standard header

MC2 Maincode2
corresponds to the value in the standard header

MC1 Maincode1
 corresponds to the value in the standard header

The main code X'8000' – parameter area not accessible – can only appear in postcode1: CRYPT could not access the parameter area at the time the function was being performed.

Use postcode2, especially when using a contingency, for a relationship to the corresponding function call (see [section “Asynchronous execution – example” on page 83](#)).

Write access to the parameter area and any data areas specified there must be assured until the end of function processing has been signalled.

The session will remain locked for other function calls until the end of the function processing has been signalled.

5. CRYPT sends a signal for the event code as soon as execution is complete. The program can request the processing signal using the SOLSIG call.

You can find more details about eventing and the necessary macro calls in the “Executive Macros” user guide [3].

6.4 Description of the macro calls

The individual macro calls are explained using syntax diagrams and operand descriptions. For more information you will find a reference to the relevant detailed description in the PKCS#11 standard for each macro call.

The C_ prefix for the PKCS#11 functions stands for function.



The following five macro calls have no function. They are not supported in CRYPT. The interfaces are however implemented for reasons of portability within BS2000.

- CWTFSL
- CINITTK
- CPIN
- COPSTAT

CGENRAL – general functions


The CGENRAL macro covers all of the following general functions:

- initializing a Cryptoki library
- finalizing an application with the Cryptoki library
- outputting general information about Cryptoki
- outputting the function list of the Cryptoki library

All functions are always performed synchronously

A detailed description of the functions of the CGENRAL macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.4 “General-purpose functions”.

Macro	Operands
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= *SYNCHRONOUS / *ASYNCHRONOUS / <var: enum-of _exec_set:1> / default: _exec_set.synchronous ,INFO= <var: pointer> / <u>NULL</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u>

VERSION	specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
=001	This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
=002	This generates the format that is supported as of CRYPT V1.1.
ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*INITIALIZE	corresponds to the PKCS#11 function <i>C_Initialize</i> ; initializes the Cryptoki library.
=*FINALIZE	corresponds to the PKCS#11 function <i>C_Finalize</i> ; indicates that an application has been executed with the Cryptoki library.
=*GETINFO	corresponds to the PKCS#11 function <i>C_GetInfo</i> ; outputs general information about Cryptoki.
=*GETFUNCTIONLIST	corresponds to the PKCS#11 function <i>C_GetFunctionList</i> ; outputs the function list.
	This function is not supported.
EXEC	specifies the execution mode of the following functions
=*SYNCHRONOUS	only restores control to the caller once the function has been executed.
=*ASYNCHRONOUS	restores control to the caller once the function has been sent to the token.

INFO	Type of information output depending on the action: <ul style="list-style-type: none">– *INITIALIZE: INFO points to an <code>_INITIALIZE_ARGS</code> structure– *FINALIZE: INFO should be set to <code>NULL_PTR</code>.– *GETINFO: INFO points to the storage area which will receive the information.– *GETFUNCTIONLIST: INFO points to a value which receives a pointer to the <code>_FUNCTION_LIST</code> structure of the library.
BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used– in the case of asynchronous execution: event identification which informs the program about the scheduling of CRYPT.
RPOSTAD	Address of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing the postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CGTSTMI – display information

The CGTSTMI macros covers the following functions of slot and token management

- outputting a list of the slots in the system
- outputting information about a single slot in the system
- outputting information about a single token in the system
- outputting a list of types of mechanism that are supported by a token
- outputting information about a single mechanism that is supported by a token

The C_GetMechanismList and C_GetMechanismInfo functions are performed asynchronously if asynchronous function execution has been specified for the task with C_Initialize.

The functions C_GetSlotList, C_GetSlotInfo and C_GetTokenInfo are always executed synchronously.

A detailed description of the functions of the CGTSTMI macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.5 “Slot and token management functions” under “C_GetSlotList”, “C_GetSlotInfo”, “C_GetTokenInfo”, “C_GetMechanismList”, “C_GetMechanismInfo”.

Macro	Operands
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> / <u>NULL</u> ,COUNT= <var: int:4> / <integer 0 .. 2147483647> / 0 ,BOID= <var: int:4> / 0 ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / 0

VERSION	specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
=001	This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
=002	This generates the format that is supported as of CRYPT V1.1.
ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*GETSLOTLIST	corresponds to the PKCS#11 function <i>C_GetSlotList</i> ; outputs the list of slots in the system.
=*GETSLOTINFO	corresponds to the PKCS#11 function <i>C_GetSlotInfo</i> ; outputs information about a single slot in the system
=*GETTOKENINFO	corresponds to the PKCS#11 function <i>C_GetTokenInfo</i> ; outputs information about a single token in the system
=*GETMECHANISMLIST	corresponds to the PKCS#11 function <i>C_GetMechanismList</i> ; outputs the list of types of mechanism that are supported by a token
=*GETMECHANISMINFO	corresponds to the PKCS#11 function <i>C_GetMechanismInfo</i> ; outputs information about a single mechanism that is supported by a token.
TOKNPRS	Information about the current token; is only relevant for the action *GETSLOTLIST.
=*TRUE	A token must be available.
=*FALSE	It is not relevant whether a token is available or not.
SLOTID	ID of the slot
TYPE	Mechanism type: value from <code>_mechanism_set</code> (see also section "CPKC11T – general data types" on page 106); is only relevant for the action *GETMECHANISMINFO.

- INFO
- The type of information output depends on the action:
- *GETSLOTLIST: NULL_PTR or pointer to a memory location that receives the slot list.
 - *GETSLOTINFO: INFO points to the memory location that receives the slot information.
 - *GETTOKENINFO: INFO points to the memory location that receives the token information.
 - *GETMECHANISMLIST: NULL_PTR or pointer to a memory location that receives the mechanism list.
 - *GETMECHANISMINFO: INFO points to the memory location that receives the information about the mechanism.
- COUNT
- The memory size depends on the action:
- *GETSLOTLIST:
INFO = NULL_PTR: Returns the number of slots;
INFO <> NULL_PTR: Must contain the size of the memory that INFO is pointing to. The size of the memory corresponds to the number of SLOT_ID elements.
 - *GETSLOTINFO: COUNT is not evaluated.
 - *GETTOKENINFO: COUNT is not evaluated.
 - *GETMECHANISMLIST:
INFO = NULL_PTR: Returns the number of slots;
INFO <> NULL_PTR: Must contain the size of the memory that INFO is pointing to. The size of the memory corresponds to the number of MECHANISM_TYPE elements.
 - *GETMECHANISMINFO: COUNT is not evaluated.
- BOLD
- Event identification
- in the case of synchronous execution: BOLD is not used.
 - in the case of asynchronous execution:
event identification to which the end of function processing is signalled.

- RPOSTAD Postcode address
- in the case of synchronous execution: RPOSTAD is not used.
 - in the case of asynchronous execution:
specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]).
Length of postcode: 4 or 8 bytes
- RPOSTL Length of postcode
- in the case of synchronous execution: RPOSTL is not used.
 - in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CWTFSSLE – waiting for a slot event

The CWTFSSLE macro waits for a slot event.



This macro call has no function. The interface is implemented for reasons of portability within BS2000.

A detailed description of the function of the CWTFSSLE macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.5 “Slot and token management functions” under “C_WaitForSlotEvent”.

Macro	Operands
CWTFSSLE	MF= C / D / L / M / E ,DONTBLK= <var: bit:1> / *TRUE / *FALSE ,SLOTID= <var: int:4> / <integer 0..2147483647> / <u>0</u>

DONTBLK specifies whether the CWTFSSLE call blocks or not, if, for example, the macro is waiting for the arrival of a slot event.

SLOTID ID of the slot

CINITTK – initialize token

The CINITTK macro initializes a token.



This macro call has no function. The interface is implemented for reasons of portability within BS2000.

A detailed description of the function of the CINITTK macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.5 “Slot and token management functions” under “C_InitToken”.

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

SLOTID ID of the slot

PIN points to the first PIN of the security officer (SO)

PINLEN Length of the PIN in bytes

LABEL 32-byte label of the token.
 The label must be padded with spaces and may not end in a null.

CPIN – initialize or modify PIN

The CPIN macro initializes or modifies a PIN.



This macro call has no function. The interface is implemented for reasons of portability within BS2000.

A detailed description of the functions of the CPIN macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.5 “Slot and token management functions” under “C_InitPIN” and “C_SetPIN”.

Macro	Operands
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> / <u>NULL</u> ,OLDPINL= <var: int:4> / <integer 0..2147483647> / 0 ,NEWPIN= <var: pointer> / <u>NULL</u> ,NEWPINL= <var: int:4> / <integer 0..2147483647> / 0

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*INITPIN	corresponds to the PKCS#11 function <i>C_InitPIN</i> ; initializes a PIN.
=*SETPIN	corresponds to the PKCS#11 function <i>C_SetPIN</i> ; modifies a PIN.
SESSION	Session identifier
OLDPIN	points to the old PIN; is not used for the action *INITPIN.
OLDPINL	Length of the old PIN in bytes; is not used for the action *INITPIN.
NEWPIN	points to the new PIN.
NEWPINL	Length of the new PIN in bytes.

CSESSION – session management


The CSESSION macro covers the following session management functions

- opening a session between an application and a token in a specified slot
- closing a session between an application and a token
- closing all sessions of an application with a token
- outputting information about a session

The functions C_OpenSession, C_CloseSession and C_CloseAllSessions are performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the functions of the CSESSION macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.6 “Session management functions” under “C_OpenSession”, “C_CloseSession”, “C_CloseAllSessions” and “C_GetSessionInfo”.

Macro	Operands
CSESSION	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> / <u>0</u> ,SLOTID= <var: int:4> / <u>0</u> ,RWSESS= <var: bit:1> / *NO / *YES ,SERIAL= <var: bit:1> / *NO / *YES ,INFO= <var: pointer> / <u>NULL</u> ,NOTIFY= <var: pointer> / <u>NULL</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u> ,GENKPR= <var: bit:1> / *ALLOWED / *NOTALLOWED

VERSION	specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
=001	This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
=002	This generates the format which is supported as of CRYPT V1.1.
ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*OPENSESSION	corresponds to the PKCS#11 function <i>C_OpenSession</i> ; opens a session between an application and a token in a specified slot
=*CLOSESESSION	corresponds to the PKCS#11 function <i>C_CloseSession</i> ; closes a session between an application and a token
=*CLOSEALLSESSIONS	corresponds to the PKCS#11 function <i>C_CloseAllSessions</i> ; closes all sessions between an application and a token
=*GETSESSIONINFO	corresponds to the PKCS#11 function <i>C_GetSessionInfo</i> ; outputs information about a session.
 This function is not supported.	
SESSION	Session identifier
SLOTID	ID of the slot
RWSESS	indicates Read/Write and Read-only sessions.
=*NO	Read-only session
=* <u>YES</u>	Read/Write session
SERIAL	Serial session <ul style="list-style-type: none"> – *OPENSESSION: SERIAL should always be set to *YES. – *GETSESSIONINFO / *CLOSESESSION / *CLOSEALLSESSIONS: SERIAL is not used.

INFO	<p>The type of information depends on the action:</p> <ul style="list-style-type: none">– *OPENSESSION: INFO is a pointer defined in the application which is passed to the notification callback function; is not supported.– *GETSESSIONINFO: INFO points to the memory location that receives the session information.– *CLOSESESSION / *CLOSEALLSESSIONS: INFO is not used.
NOTIFY	<p>Callback function</p> <ul style="list-style-type: none">– *OPENSESSION: Address of the notification callback function; is not supported– *CLOSESESSION / *CLOSEALLSESSIONS / *GETSESSIONINFO: The callback function is not used.
BOLD	<p>Event identification</p> <ul style="list-style-type: none">– in the case of synchronous execution: BOLD is not used.– in the case of asynchronous execution: event identification to which the end of the function processing is signalled.
RPOSTAD	<p>Postcode address</p> <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	<p>Length of postcode</p> <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).
GENKPR	<p>This operand has no function.</p>

COPSTAT – display/set operation state

The COPSTAT macro covers the following session management functions

- outputting a copy of the state of the cryptographic operations of a session
- restoring the state of the cryptographic operations of a session



This macro call has no function. The interface is implemented for reasons of portability within BS2000.

A detailed description of the functions of the COPSTAT macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.6 “Session management functions” under “C_GetOperationState” and “C_SetOperationState”.

Macro	Operands
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> / <u>0</u> ,STATE= <var: pointer> / <u>NULL</u> ,STATEL= <var: int:4> / <integer 0..2147483647> / <u>0</u> ,CRYKEY= <var: int:4> / <integer 0..2147483647> / <u>0</u> ,AUTHKEY= <var: int:4> / <integer 0..2147483647> / <u>0</u>

ACTION Type of action.
 The corresponding PKCS#11 function is specified for each action code.

=*GETOPERATIONSTATE
 corresponds to the PKCS#11 function *C_GetOperationState*;
 outputs a copy of the state of the cryptographic operations of a session

=*SETOPERATIONSTATE
 corresponds to the PKCS#11 function *C_SetOperationState*;
 restores the state of the cryptographic operations of a session

SESSION Session identifier

STATE	State of the cryptographic operation: <ul style="list-style-type: none">– *GETOPERATIONSTATE: Memory location that receives the state– *SETOPERATIONSTATE: Memory location in which the stored state is located
STATEL	Length of the *...OPERATIONSTATE memory location
CRYKEY	Encryption and decryption key, only in *SETOPERATIONSTATE; *GETOPERATIONSTATE: CRYKEY is not used.
AUTHKEY	signs/verifies the authentication key, only with *SETOPERATINSTATE; *GETOPERATIONSTATE: AUTHKEY is not used.

CLOG – login/logout

The CLOG macro covers the following session management functions

- logging a user into a token
- logging a user out of a token

All functions are always performed synchronously.

A detailed description of the functions of the CLOG macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.6 “Session management functions” under “C_Login” and “C_Logout”.

Macro	Operands
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> / <u>0</u> ,PIN= <var: pointer> / <u>NULL</u> ,PINL= <var: int:4> / <integer 0..2147483647> / <u>0</u>

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*LOGIN	corresponds to the PKCS#11 function <i>C_Login</i> ; logs a user into a token.
=*LOGOUT	corresponds to the PKCS#11 function <i>C_Logout</i> ; logs a user out of a token.
SESSION	Session identifier
PIN	points to the PIN; *LOGOUT: PIN is not used.
PINL	Length of the PIN in bytes; *LOGOUT: PINL is not used.

COBJMGT – object management

The COBJMGT macro covers the following object management functions

- generating a new object
- copying an object
- deleting an object
- outputting the size of an object in bytes
- outputting the value of one or more attributes of an object
- modifying the value of one or more attributes of an object
- initializing a search for token and session objects that correspond to a template
- continuing a search for token and session objects that correspond to a template, where additional object handles are output
- terminating a search for token and session objects


The functions `C_FindObjectsInit` and `C_FindObjectsFinal` are always performed synchronously.

All other functions are performed asynchronously if asynchronous function execution has been specified for the task with `C_Initialize`.

A detailed description of the functions of the COBJMGT macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.7 “Object management functions”.

Macro	Operands
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> / <u>0</u> ,OBJECT= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,TEMPLAT= <var: pointer> / <u>NULL</u> ,COUNT= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,OBJLIST= <var: pointer> / <u>NULL</u> ,OBJSIZE= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,MAXOBJ= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,OBJCNT= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u>

- VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
- =001 This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
- =002 This generates the format which is supported as of CRYPT V1.1.

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*CREATEOBJECT	corresponds to the PKCS#11 function <i>C_CreateObject</i> ; creates a new object.
=*COPYOBJECT	corresponds to the PKCS#11 function <i>C_CopyObject</i> ; copies an object.
=*DESTROYOBJECT	corresponds to the PKCS#11 function <i>C_DestroyObject</i> ; deletes an object.
=*GETOBJECTSIZE	corresponds to the PKCS#11 function <i>C_GetObjectSize</i> ; outputs the size of an object in bytes.
	This function is not supported.
=*GETATTRIBUTEVALUE	corresponds to the PKCS#11 function <i>C_GetAttributeValue</i> ; outputs the value of one or more attributes of an object.
=*SETATTRIBUTEVALUE	corresponds to the PKCS#11 function <i>C_SetAttributeValue</i> ; modifies the value of one or more attributes of an object.
=*FINDOBJECTSINIT	corresponds to the PKCS#11 function <i>C_FindObjectsInit</i> ; initializes a search for token and session objects that correspond to a template.
=*FINDOBJECTS	corresponds to the PKCS#11 function <i>C_FindObjects</i> ; continues a search for token and session objects that correspond to a template, where additional object handles are output.
=*FINDOBJECTSFINAL	corresponds to the PKCS#11 function <i>C_FindObjectsFinal</i> ; terminates a search for token and session objects.
SESSION	Session identifier

OBJECT	<p>Object handle</p> <ul style="list-style-type: none">– *CREATEOBJECT: OBJECT receives the new object handle– *COPYOBJECT, *DESTROYOBJECT, *GETOBJECTSIZE, *GETATTRIBUTEVALUE, *SETATTRIBUTEVALUE: The object handle– *FINDOBJECTSINIT, *FINDOBJECTS, *FINDOBJECTSFINAL: Object handle is not used
TEMPLAT	<p>Object template</p> <ul style="list-style-type: none">– *CREATEOBJECT, *COPYOBJECT: The template of the object– *GETATTRIBUTEVALUE: TEMPLAT points to a template which specifies which attribute values must be output and which receives attribute values.– *SETATTRIBUTEVALUE: TEMPLAT points to a template which specifies which attribute values must be modified and specifies the new values.– *FINDOBJECTSINIT: TEMPLAT points to a search template which specifies the attribute values that are to be matched.– *DESTROYOBJECT, *GETOBJECTSIZE, *FINDOBJECTS, *FINDOBJECTSFINAL: TEMPLAT is not used.
COUNT	<p>Number of attributes in the template</p> <p>*DESTROYOBJECT, *GETOBJECTSIZE, *FINDOBJECTS, *FINDOBJECTSFINAL: COUNT is not used.</p>
OBJLIST	<p>points to the memory location which receives the list (array) of additional object handles;</p> <p>is only used by *FINDOBJECTS.</p>
OBJSIZE	<p>outputs the size of an object in bytes;</p> <p>is only used by *GETOBJECTSIZE.</p>
MAXOBJ	<p>Maximum number of object handles that are returned.</p> <p>is only used by *FINDOBJECTS.</p>
OBJCNT	<p>receives the current number of object handles that are returned;</p> <p>is only used by *FINDOBJECTS.</p>

BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– in the case of asynchronous execution: event identification to which the end of the function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that outputs the SOLSIG macro call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CCRYINI – initialize cryptographic operation


The CCRYINI macro covers the following functions

- initializing an encryption operation
- initializing a decryption operation
- initializing a message-digesting operation
- continuing a multiple-part message-digesting operation by integrating the value of the secret key.
- initializing a signature operation where the signature is an appendix of the data
- initializing a signature operation where the data can be recovered from the signature
- initializing a verification operation where the signature is an appendix of the data
- initializing a signature verification operation where the data can be recovered from the signature

All functions are always performed synchronously.

A detailed description of the functions of the CCRYINI macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in the sections 11.8 through 11.12 under “C_EncryptInit”, “C_DecryptInit”, “C_DigestInit”, “C_DigestKey”, “C_SignInit”, “C_SignRecoverInit”, “C_VerifyInit” and “C_VerifyRecoverInit”.

Macro	Operands
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> / <u>0</u> ,MECHAN= <var: pointer> / <u>NULL</u> ,KEY= <var: int:4> / <integer 0..2147483647> / <u>0</u>

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*ENCRYPTINIT	corresponds to the PKCS#11 function <i>C_EncryptInit</i> ; initializes an encryption operation.
=*DECRYPTINIT	corresponds to the PKCS#11 function <i>C_DecryptInit</i> ; initializes a decryption operation.
=*DIGESTINIT	corresponds to the PKCS#11 function <i>C_DigestInit</i> ; initializes a message-digesting operation.
=*DIGESTKEY	corresponds to the PKCS#11 function <i>C_DigestKey</i> ; continues a multiple-part message-digesting operation by integrating the value of the secret key in data which has already been summarized.
=*SIGNINIT	corresponds to the PKCS#11 function <i>C_SignInit</i> ; initializes a signature operation where the signature is an appendix of the data.
=*SIGNRECOVERINIT	corresponds to the PKCS#11 function <i>C_SignRecoverInit</i> ; initializes a signature operation where the data can be recovered from the signature.
	This function is not supported.
=*VERIFYINIT	corresponds to the PKCS#11 function <i>C_VerifyInit</i> ; initializes a verification operation where the signature is an appendix of the data
=*VERIFYRECOVERINIT	corresponds to the PKCS#11 function <i>C_VerifyRecoverInit</i> ; initializes a signature verification operation where the data can be recovered from the signature.
SESSION	Session identifier
MECHAN	Mechanism
KEY	Key handle *DIGESTINIT: KEY is not used.

CCRY – execute cryptographic operation

The CCRY macro covers the following functions

- encrypting a data package
- continuing a multiple-part encryption operation
- decrypting encrypted data in a single part
- continuing a multiple-part decryption operation
- digesting data in a single part
- continuing a multiple-part message-digesting operation
- signing data in a single part where the signature is an appendix of the data
- continuing a multiple-part signature operation where the signature is an appendix of the data
- signing data in a single operation where the data can be recovered from the signature
- verifying a signature in a single-part operation where the signature is an appendix of the data
- continuing a verification operation where the signature is an appendix of the data
- verifying a signature in a single-part operation where the data can be recovered from the signature
- continuing a multiple-part digesting and encryption operation
- continuing a multiple-part decryption and digesting operation
- continuing a multiple-part signature and encryption operation
- continuing a multiple-part decryption and verification operation

All functions are always performed asynchronously if asynchronous function execution was specified for the task with `C_Initialize`.


A detailed description of the functions of the CCRY macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in the sections 11.8 through 11.13 under “`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`” and “`C_DecryptVerifyUpdate`”.

Macro	Operands
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> / <u>0</u> ,DATAIN= <var: pointer> / <u>NULL</u> ,INLEN= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,DATAOUT= <var: pointer> / <u>NULL</u> ,OUTLEN= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u>

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format that is supported as of CRYPT V1.1.

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*ENCRYPT	corresponds to the PKCS#11 function <i>C_Encrypt</i> ; encrypts a data package.
=*ENCRYPTUPDATE	corresponds to the PKCS#11 function <i>C_EncryptUpdate</i> ; continues a multiple-part encryption operation.
=*DECRYPT	corresponds to the PKCS#11 function <i>C_Decrypt</i> ; decrypts encrypted data in a single part.
=*DECRYPTUPDATE	corresponds to the PKCS#11 function <i>C_DecryptUpdate</i> ; continues a multiple-part decryption operation.
=*DIGEST	corresponds to the PKCS#11 function <i>C_Digest</i> ; digests data in a single part.
=*DIGESTUPDATE	corresponds to the PKCS#11 function <i>C_DigestUpdate</i> ; continues a multiple-part message-digesting operation.
=*SIGN	corresponds to the PKCS#11 function <i>C_Sign</i> ; signs data in a single part where the signature is an appendix of the data.
=*SIGNUPDATE	corresponds to the PKCS#11 function <i>C_SignUpdate</i> ; continues a multiple-part signature operation where the signature is an appendix of the data.
=*SIGNRECOVER	corresponds to the PKCS#11 function <i>C_SignRecover</i> ; signs data in a single operation where the data can be recovered from the signature
	This function is not supported.
=*VERIFY	corresponds to the PKCS#11 function <i>C_Verify</i> ; checks a signature in a single-part operation where the signature is an appendix of the data.
=*VERIFYUPDATE	corresponds to the PKCS#11 function <i>C_VerifyUpdate</i> ; continues a multiple-part verification operation where the signature is an appendix of the data.

=*VERIFYRECOVER

corresponds to the PKCS#11 function *C_VerifyRecover*;
checks a signature verification operation where the data can be
recovered from the signature.

=*DIGESTENCRYPTUPDATE

corresponds to the PKCS#11 function *C_DigestEncryptUpdate*;
continues a multiple-part digesting and encryption operation.



This function is not supported.

=*DECRYPTDIGESTUPDATE

corresponds to the PKCS#11 function *C_DecryptDigestUpdate*;
continues a multiple-part decryption and digesting operation.



This function is not supported.

=*SIGNENCRYPTUPDATE

corresponds to the PKCS#11 function *C_SignEncryptUpdate*;
continues a multiple-part signature and encryption operation.



This function is not supported.

=*DECRYPTVERIFYUPDATE

corresponds to the PKCS#11 function *C_DecryptVerifyUpdate*;
continues a multiple-part decryption and verification operation.



This function is not supported.

SESSION

Session identifier

DATAIN

points to the input data

INLEN

Length of the input data in bytes

DATAOUT

points to the output data

- *VERIFY: Pointer to signature
- *DIGESTUPDATE, *SIGNUPDATE, *VERIFYUPDATE:
are not used.

OUTLEN	Length of the output data in bytes <ul style="list-style-type: none">– *VERIFY: Length of the signature– *DIGESTUPDATE, *SIGNUPDATE, *VERIFYUPDATE: are not used.
BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– in the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CCRYFIN – finalize cryptographic operation

The CCRYFIN macro covers the following functions

- terminating a multiple-part encryption operation
- terminating a multiple-part decryption operation
- terminating a multiple-part message-digesting operation
- terminating a multiple-part signature operation with return of signature
- terminating a multiple-part verification operation with the verification of the signature

All functions are performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the functions of the CCRYFIN macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in the sections 11.8 through 11.12 under “C_EncryptFinal”, “C_DecryptFinal”, “C_DigestFinal”, “C_SignFinal” and “C_VerifyFinal”.

Macro	Operands
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> / <u>0</u> ,DATA= <var: pointer> / <u>NULL</u> ,LEN= <var: int:4> / <integer 0 .. 2147483647> / <u>0</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u>

VERSION	specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
=001	This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
=002	This generates the format that is supported as of CRYPT V1.1.
ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*ENCRYPTFINAL	corresponds to the PKCS#11 function <i>C_EncryptFinal</i> ; terminates a multiple-part encryption operation.
=*DECRYPTFINAL	corresponds to the PKCS#11 function <i>C_DecryptFinal</i> ; terminates a multiple-part decryption operation.
=*DIGESTFINAL	corresponds to the PKCS#11 function <i>C_DigestFinal</i> ; terminates a multiple-part message-digesting operation.
=*SIGNFINAL	corresponds to the PKCS#11 function <i>C_SignFinal</i> ; terminates a multiple-part signature operation with return of signature
=*VERIFYFINAL	corresponds to the PKCS#11 function <i>C_VerifyFinal</i> ; terminates a multiple-part verification operation with the verification of the signature.
SESSION	Session identifier
DATA	points to data – *ENCRYPTFINAL, *DECRYPTFINAL, *DIGESTFINAL, *SIGNFINAL: DATA points to the output data. – *VERIFYFINAL: DATA points to the signature.
LEN	Length of the data in bytes – *ENCRYPTFINAL, *DECRYPTFINAL, *DIGESTFINAL, *SIGNFINAL: Length of the output data in bytes – *VERIFYFINAL: Length of the signature in bytes

BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– In the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Postcode length <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CGENKEY – generate secret key

The CGENKEY macro generates a secret key by creating a new key object.

The function is performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the function of the CGENKEY macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.14 “Key management functions” under “C_GenerateKey”.

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

VERSION	specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.
=001	This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.
=002	This generates the format supported as of CRYPT V1.1.
SESSION	Session identifier
MECHAN	points to the mechanism for key generation
TEMPL	points to the template for the new key
COUNT	Number of attributes in the template

BOLD	<p>Event identification</p> <ul style="list-style-type: none">– in the case of synchronous execution: BOLD is not used.– in the case of asynchronous execution: event identification to which the end of function processing is signalled.
RPOSTAD	<p>Postcode address</p> <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). <p>Length of postcode: 4 or 8 bytes</p>
RPOSTL	<p>Length of postcode</p> <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CGENKPR – generate key pair

The CGENKPR macro generates a key pair from a public and a private key by creating new key objects.

The function is performed asynchronously if asynchronous function execution was specified for the task with `C_Initialize`.

A detailed description of the function of the CGENKPR macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.14 “Key management functions” under “`C_GenerateKeyPair`”.

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

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format only supports the parameters known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format supported as of CRYPT V1.1.

SESSION Session identifier

MECHAN points to the mechanism for key generation

PUBTEMP points to the template for the public key

PUBACNT	Number of attributes in the template for the public key
PRVTEMP	points to the template for the private key
PRVACNT	Number of attributes in the template for the private key
BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– in the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of the postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CWRPKEY – wrap key

The CWRPKEY macro wraps a private or secret key.

The function is performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the function of the CWRPKEY macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.14 “Key management functions” under “C_WrapKey”.

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

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format supported as of CRYPT V1.1.

SESSION Session identifier

MECHAN points to the key wrap mechanism

KEK Handle of the wrapping key

KEY Handle of the key that is to be wrapped

WRPDKEY	points to the memory location that receives the wrapped key
WRPDLEN	Length of the wrapped key
BOLD	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOLD is not used.– in the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of the postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CUNWKEY – unwrap key

The CUNWKEY macro unwraps a wrapped key by creating a new object for a private or secret key.

The function is performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the function of the CUNWKEY macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.14 “Key management functions” under “C_UnwrapKey”.

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

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format only supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format that is supported as of CRYPT V1.1.

SESSION Session identifier

MECHAN points to the key wrap mechanism

KEK Handle of the unwrapping key

WRPDKEY	points to the wrapped key
WRPDLEN	Length of the wrapped key
TEMPL	points to the template for the new key
COUNT	Number of attributes in the template
BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– in the case of asynchronous execution: Event identification to which the end of signal processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CDRVKEY – derive key

The CDRVKEY macro derives a key from a basic key by generating a new key object.

The function is performed asynchronously if asynchronous function execution was specified for the task with C_Initialize.

A detailed description of the function of the CDRVKEY macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.14 “Key management functions” under “C_DeriveKey”.

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

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format supported as of CRYPT V1.1.

SESSION Session identifier

MECHAN points to the mechanism used to derive the key

BASEKEY Handle of the basic key

TEMPL points to the template for the new key

COUNT	Number of attributes in the template
BOID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOID is not used.– in the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

CRANDOM – generate random numbers

The CRANDOM macro covers the following functions

- mixing additional seed material into the token's random number generator
- generating random data

All functions are performed asynchronously if asynchronous function execution was specified for the task with `C_Initialize`.

A detailed description of the functions of the CRANDOM macro can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in section 11.15 "Random number generation functions".

Macro	Operands
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> / <u>0</u> ,DATA= <var: pointer> / <u>NULL</u> ,DATALEN= <var: int:4> / <integer 0..2147483647> / <u>0</u> ,BOID= <var: int:4> / <u>0</u> ,RPOSTAD= <var: pointer> / <u>NULL</u> ,RPOSTL= <integer 1..2> / <var: int:4> / <u>0</u>

VERSION specifies which version of the parameter area is to be generated. It is always advisable to use the latest version.

=001 This generates the format that was supported by CRYPT V1.0. This format supports the parameters already known in CRYPT V1.0. VERSION=001 is the default.

=002 This generates the format supported as of CRYPT V1.1.

ACTION	Type of action. The corresponding PKCS#11 function is specified for each action code.
=*SEEDRANDOM	corresponds to the PKCS#11 function <i>C_SeedRandom</i> ; mixes additional seed material in the token's random number generator
=*GENERATERANDOM	corresponds to the PKCS#11 function <i>C_GenerateRandom</i> ; generates random data.
SESSION	Session identifier
DATA	points to the following data: <ul style="list-style-type: none">– with *SEEDRANDOM: DATA points to the start parameter material.– with *GENERATERANDOM: DATA points to the memory location that receives the random data.
DATALEN	Length of the data in bytes
BOLID	Event identification <ul style="list-style-type: none">– in the case of synchronous execution: BOLID is not used.– in the case of asynchronous execution: Event identification to which the end of function processing is signalled.
RPOSTAD	Postcode address <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTAD is not used.– in the case of asynchronous execution: specifies a field containing postcode information which is to be transferred to the corresponding program that issues the SOLSIG call (see also “Executive Macros” user guide [3]). Length of postcode: 4 or 8 bytes
RPOSTL	Length of postcode <ul style="list-style-type: none">– in the case of synchronous execution: RPOSTL is not used.– in the case of asynchronous execution: specifies the length of the postcode information in words (1 or 2).

6.5 Sample programs

In this section you will find a sample program for both synchronous and asynchronous execution modes.

6.5.1 Synchronous execution – example

The following CRYPT macros are used in the sample program:

- The CPKC11T macro contains data descriptions and equates that are used by the subsequent macros.
- The CSESSION macro uses the *OPENSESSION action to open a session between an application and a token in a specified slot.
- The CGENKEY macro generates a secret key.
- Then the action *ENCRYPTINIT of the CCRYINI macro initiates an encryption operation.
- The *ENCRYPT action of the CCRY macro is used to continue and terminate the encryption operation.
- The action *DECRYPTINIT of the CCRYINI macro initiates a decryption operation.
- Then the CCRY macro continues and terminates the decryption using the *DECRYPT action.
- The session is terminated using the *CLOSESESSION action of the CSESSION macro.

```

                TITLE 'CPKC11T LAYOUT'
                CPKC11T MF=D
                TITLE 'CSESSION PARAM LIST'
                CSESSION 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,CSESSIONC
      MVC   CSESSIONC,CSESSIONL
      @DATA BASE=R3,CLASS=B,DSECT=CRYO_MDL
      MVI   CRYOACTION,CRYOOPENSESSION
*      MVC   CRYOSLOTID,=F'0'
      CSESSION 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,SUCSESSE,PARMOD=31
SUCSESSE 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,CSESSIONC
MVC CSESSIONC,CSESSIONL
@DATA BASE=R3,CLASS=B,DSECT=CRYO_MDL
MVI CRYOACTION,CRYOCLOSESESSION
MVC CRYOSESSION,ZSESSION
CSESSION MF=E,PARAM=(R3),CALLER=USER
@BEND
SPACE
@EXIT

* DATA
CSESSIONL CSESSION 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 OF
CSESSIONC 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 Asynchronous execution – example

CRYPT uses the following macros in the sample program section below:

1. The CPKC11T macro contains data descriptions and equates that are used by the following macros.
2. The CGENRAL macro implements asynchronous processing for the task with CRYPT.
3. The event identification CRYPTTST is defined. The address of the short ID is OUTEIID.
4. The CRY2ABC routine is defined as a contingency process:
CONTAAD specifies the start address.
OUTCOID specifies the address of the short ID.
5. The program requests a signal from the event identification CRYPTTST using a SOLSIG call and specifies the contingency process CRY2ABC. If the signal has not yet arrived after 600 seconds the event control should start the contingency process CRY2ABC. The program continues to run after this SOLSIG call.
6. The CSESSION macro opens a session between an application and a token in a certain slot using the *OPENSESSION action.
7. The CGENKEY macro generates a secret DES key.
8. The *ENCRYPTINIT action of the CCRYINI macro then initiates an encryption operation with the DES key created.
9. The *ENCRYPT action of the CCRY macro performs the encryption operation
10. Routine CRY2ABC which acts as a contingency process.
11. The program once again requests a signal with a SOLSIG call.
12. A check is carried out to determine whether an CRYPT event has occurred.
13. Follow-up processing takes place depending on the event that has occurred.



You can find information about eventing and the contingency process in the “Executive Macros” user guide [3].

Macro calls that are not described in this manual (e.g. SOLSIG) are also described in the “Executive Macros” user guide [3].

```

FHDR MF=D
TITLE 'CPKC11T layout' -----(1.)
CPKC11T MF=D
TITLE 'CSESSION'
CSESSION 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
*
CRY2ABV ENTRY CRY2ABV
DS OD
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 E1NAME=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 CSESSION call
MVC  CRYOHEADER(CRYO#),CSESSIONL
MVI  CRYOACTION,CRYOOPENSESSION
MVC  CRYOBOID,OUTEIID
LA   R1,OPSTKEY1
ST   R1,CRYORPOSTAD
ST   R6,OPSTKEY2
MVC  CRYORPOSTL,=F'2'
CSESSION 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,PARAMOD=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
CSESSION CSESSION 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(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'that is secret!'
*
*
ZEXALOC  @PAR   D=YES
*
CGENRALC CGENRAL MF=C,VERSION=002
*
CCRYINIC CCRYINI MF=C
*
ENAEIRC  DS      0F
ENAEISC  DS      X
          DS      X
          DS      X
          DS      X
ENAEIMC  DS      X
*
ENACORC  DS      0F
ENACOSC  DS      X
          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 , error
* error handling
B RETCO
@BEND , error
*
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 ESESSION

*
    @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 ESESSION
*
    @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
ESESSION 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     0F
CONTISC  DS     X
          DS     X
          DS     X
CONTIMC  DS     X
*

```

7 Description of the functions in C

In order to develop portable applications, or to ensure that applications that use a PKCS#11 interface can be easily ported CRYPT also provides the ANSI C interface described in PKCS#11 and the include files PKCS11.H, PKCS11T.H and PKCS11F.H.

Prerequisites

If you wish to use this C interface, you will need to link the CRYADAP adapter module from the SYSLIB.CRYPT.nnn library in to the program.

7.1 Notes about the description in PKCS#11

Version 2.10 of the PKCS#11 standard is used as the basis for your work with the C interface of CRYPT.

This standard can be found on the Internet under

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

The following table lists the functions and where you will find the description in the PKCS#11 V2.20 standard. The C_ prefix in the PKCS#11 functions indicates a function.

Function	Description in PKCS#11 V2.20
General-purpose functions <ul style="list-style-type: none"> – C_Initialize – C_Finalize – C_GetInfo – C_GetFunctionList 	section 11.4
Slot and token management functions <ul style="list-style-type: none"> – C_GetSlotList – C_GetSlotInfo – C_GetTokenInfo – C_WaitForSlotEvent – C_GetMechanismList – C_GetMechanismInfo – C_InitToken – C_InitPIN – C_SetPIN 	section 11.5
Session management functions <ul style="list-style-type: none"> – C_OpenSession – C_CloseSession – C_CloseAllSessions – C_GetSessionInfo – C_GetOperationState – C_SetOperationState – C_Login – C_Logout 	section 11.6
Object management functions <ul style="list-style-type: none"> – C_CreateObject – C_CopyObject – C_DestroyObject – C_GetObjectSize – C_GetAttributeValue – C_SetAttributeValue – C_FindObjectsInit – C_FindObjects – C_FindObjectsFinal 	section 11.7

Functions and corresponding description in PKCS#11

(Part 1 of 3)

Function	Description in PKCS#11 V2.20
Encryption functions <ul style="list-style-type: none"> – C_EncryptInit – C_Encrypt – C_EncryptUpdate – C_EncryptFinal 	section 11.8
Decryption functions <ul style="list-style-type: none"> – C_DecryptInit – C_Decrypt – C_DecryptUpdate – C_DecryptFinal 	section 11.9
Message digesting functions <ul style="list-style-type: none"> – C_DigestInit – C_Digest – C_DigestUpdate – C_DigestKey – C_DigestFinal 	section 11.10
Signing and MACing functions <ul style="list-style-type: none"> – C_SignInit – C_Sign – C_SignUpdate – C_SignFinal – C_SignRecoverInit – C_SignRecover 	section 11.11
Functions for verifying signatures and MACs <ul style="list-style-type: none"> – C_VerifyInit – C_Verify – C_VerifyUpdate – C_VerifyFinal – C_VerifyRecoverInit – C_VerifyRecover 	section 11.12
Dual-function cryptographic functions <ul style="list-style-type: none"> – C_DigestEncryptUpdate – C_DecryptDigestUpdate – C_SignEncryptUpdate – C_DecryptVerifyUpdate 	section 11.13

Functions and corresponding description in PKCS#11

(Part 2 of 3)

Function	Description in PKCS#11 V2.20
Key management functions <ul style="list-style-type: none">– C_GenerateKey– C_GenerateKeyPair– C_WrapKey– C_UnwrapKey– C_DeriveKey	section 11.14
Random number generation functions <ul style="list-style-type: none">– C_SeedRandom– C_GenerateRandom	section 11.15
Parallel function management functions <ul style="list-style-type: none">– C_GetFunctionStatus– C_CancelFunction	section 11.16
Callback functions <ul style="list-style-type: none">– Surrender callbacks– Vendor-defined callbacks	section 11.17

Functions and corresponding description in PKCS#11

(Part 3 of 3)

7.2 Sample program

The following CRYPT functions are used in the sample program:

- *C_OpenSession* is used to open a session between an application and a token in a specified slot.
- *C_GenerateKey* generates a secret key.
- Then *C_EncryptInit* is used to initialize an encryption operation and *C_EncryptUpdate* used to continue it.
- *C_EncryptFinal* terminates a multiple-part encryption operation.
- Then *C_DecryptInit* is used to initialize a decryption operation and *C_DecryptUpdate* used to continue it.
- *C_DecryptFinal* terminates a multiple-part decryption operation.
- The session is then closed using *C_CloseSession*.

Example

```
#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;
}

/* Opening the session */
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);

/* Generating a secret key */
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);

/* Initializing an encryption operation */
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;

    /* Continuing a multiple-part encryption operation */
    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;

/* Terminating an encryption operation */
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");

/* Initializing a decryption operation */
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;

    /* Continuing a multiple-part decryption operation */
    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;

/* Terminating a decryption operation */
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 Creating diagnostic documents

If an error occurs whilst CRYPT is running which you cannot repair yourself, please contact your partner at Fujitsu.

In order to be able to troubleshoot the problem efficiently, your Fujitsu contact person will require the following information:

- precise description of the fault, as well as information as to whether the fault is reproducible
- information about the version of the product and the operating system as well as about the HSI (/390 or x86).
- the \$SYSAUDIT.SYS.CONSOLE file of the BS2000 session
- dumps
- \$.SYS.SERSLOG file
- CRYPTDIAG documents if applicable

You will receive the CRYPTDIAG documents as follows:

1. Assign SYSLST to a file.
 2. Start CRYPTDIAG program:
 - ▶ CRYPTDIAG
 3. Enter TOTAL statement:
 - ▶ TOTAL
 4. Enter statement END:
 - ▶ END
 5. Undo the SYSLST assignment.
- Use trace files if necessary
 - Activate the CRYPT user interfaces trace CRYPT.COM using the /DCDIAG command.

Please see the “openNet Server” manual [1] for more details about the /DCDIAG command.

9 Return codes

The function return codes set out by the PKCS#11 standard are returned in the standard header in the main return code (MRC).

The generic component CPKC11T contains the definitions for the MRCs. In addition to this, the type and structure definitions from the PKCS11T.H include file are also available in CPKC11T.

The generic component CRYASC2 contains the definitions for the sub-return code 2 (SRC2) if the MRC definition in the CPKC11T components is “arguments_bad”.

This section lists all possible MRCs and SRCs and their meanings.

CPKC11T – general data types

The generic component CPKC11T contains the definitions from the PKCS11T.H include file.

A detailed description of CPKC11T can be found in PKCS#11 V2.20: Cryptographic Token Interface Standard in chapter 9 “General data types”.

Macro	Operands
CPKC11T	MF= D / C

Main return codes (MRC)

A detailed description of the meanings of the various MRCs can be found in section 11.1 “Function return values” as of page 112 of the PKCS#11 V2.20 standard. The CKR_ prefix indicates a return code. The following table provides an overview of these MRCs.

If you use the asynchronous interface, other return codes, which are not described in the PKCS#11 standard, may occur. These are listed in the [table on page 109](#).

SRC2	SRC1	MRC	MRC identifier	Note
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	see also 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	
00	40	0012	attribute_type_invalid	
ii	01	0013	attribute_value_invalid	The i th value is invalid.

SRC2	SRC1	MRC	MRC identifier	Note
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	see also 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	
00	40	00a3	pin_expired	
00	40	00a4	pin_locked	

Main return codes (MRC)

(Part 2 of 4)

SRC2	SRC1	MRC	MRC identifier	Note
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	
00	40	0115	wrapping_key_type_inconsistent	
00	40	0120	random_seed_not_supported	

Main return codes (MRC)

(Part 3 of 4)

SRC2	SRC1	MRC	MRC identifier	Note
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 return codes (MRC)

(Part 4 of 4)

SRC2	SRC1	MRC	MRC-Identifier	Meaning
00	40	8000	PA_not_alloc	parameter area not allocated
00	40	8001	asynch_call_active	asynchronous call is active
02	00	8002	no_open_session	no session to close
00	80	8003	CRYPT_down	Subsystem CRYPT has been terminated between two function calls

Possible main return codes (MRC), that are not described in PKCS#11 V2.20

CRYASC2 – sub return code 2

The generic component CRYASC2 contains the sub return codes 2 for the main return code CPKC11T “arguments_bad”.

Macro	Operands
CRYASC2	MF= D / C

Sub return codes 2

SRC2	Identifier for main return code (Assembler)	Meaning
01	CRY2wrAction	invalid action
02	CRY2wrSession	invalid session
03	CRY2wrInfo	invalid information
04	CRY2wrNotify	invalid notify
05	CRY2wrSlotId	invalid slot ID
06	CRY2wrDatIn	invalid data input
07	CRY2wrDatOut	invalid data output
08	CRY2wrMechanism	invalid mechanism
09	CRY2wrKey	invalid key
0a	CRY2wrBKey	invalid base key
0b	CRY2wrTmplt	invalid template
0c	CRY2wrAttr	invalid attribute
0d	CRY2wrCnt	invalid count
0e	CRY2wrPubKey	invalid public key
0f	CRY2wrPrvKey	invalid private key
10	CRY2wrPubTmplt	invalid public key template
11	CRY2wrPrvTmplt	invalid private key template
14	CRY2wrTpkPr	invalid present token
15	CRY2wrType	invalid type
16	CRY2wrLabel	invalid label
17	CRY2wrPin	invalid pin
18	CRY2wrPinL	invalid pin length
19	CRY2wrUTyp	invalid user type
1a	CRY2wrObj	invalid object
1b	CRY2wrObjLst	invalid object list
1c	CRY2wrObjSz	invalid object size
1d	CRY2wrObjCnt	invalid object count

Sub return codes 2

(Part 1 of 2)

SRC2	Identifier for main return code (Assembler)	Meaning
1e	CRY2wrmaxObjCnt	invalid maximum object count
1f	CRY2wrOpState	invalid operation state
20	CRY2wrOpStateLen	invalid length of the operation state
21	CRY2wrEncKey	invalid encryption key
22	CRY2wrAutKey	invalid authentication key
23	CRY2wrData	invalid data
24	CRY2wrDataLen	invalid length of data
25	CRY2wrUnwrKey	invalid unwrapping key
26	CRY2wrWrKey	invalid wrapped key
27	CRY2wrWrKeyLen	invalid length of wrapped key
28	CRY2wrWringKey	invalid wrapping key
29	CRY2wrExec	invalid execution mode
2a	CRY2wrRPostAd	invalid postcode address
2b	CRY2wrRPostL	invalid postcode length
2c	CRY2wrBoid	invalid stock exchange identification
2d	CRY2wrSigLen	invalid signature length
2e	CRY2wrVers1	asynchronous execution with version 1.0 not permitted
2f	CRY2wrAlloc	non-allocated storage area detected
40	reserved	reserved parameter area is not 0
F0	preceding	for all main return codes: The error occurred in a preceding function (see also the following section).

Sub return codes 2

(Part 2 of 2)

SRC2 “preceding”

If SRC2 returns the value “preceding” (X'F0'), the error returned in the MRC does not refer to the function for which the value is returned, but instead to a preceding function. This may occur if an errored parameter is recognized after the preceding function has been completed.

Example

The macro call CCRYINI SES=1,ACT=*ENCRYPTINIT,MECH=mDes_key_gen,KEY=7 is first confirmed with OK.

The follow-up call CCRY ACT=*ENCRYPT,DATAIN=...,INLEN=...,DATAOUT=...,OUTLEN=... is answered with X'F0400070'.

The "mechanism_invalid" MRC refers, in this case, not to the CCRY call, but instead - as can be seen from the SRC2 - to the preceding CCRYINI call.

This mechanism cannot be used for the *ENCRYPT action.

Glossary

This glossary is designed to supplement chapter 4 “Definitions” of the PKCS#11 standard.

AES (Advanced Encryption Standard)

The AES is a FIPS publication that specifies a cryptographic algorithm for the USA authorities. AES is now the default block cipher. AES was developed by the Belgian cryptologists Dr. Joan Daemen and Dr. Vincent Rijmen.

ANSI (American National Standards Institute)

This institute develops standards for various accredited standard committees (ASC). The X9 committee is mainly concerned with security standards for financial services.

Asymmetric keys

A separate, yet integrated, user key pair consisting of a public key and a private key. This is a “one-way” key, or in other words, a key that is used to encrypt certain data, but which cannot be used to decrypt this data.

Block cipher

A symmetrical cipher code that is based on blocks (usually 128-bit blocks) of plain text and encrypted text.

CBC (Cipher Block Chaining)

The process of the application of the XOR operator which brings plaintext into an either-or relationship with the preceding text block before it is encrypted. This adds a “feedback” mechanism to a block cipher.

CFB or CFM (Cipher Feedback Mode)

A block cipher that is implemented as a self-synchronizing stream cipher. This feeds back a specified number of bits of the ciphertext as the input data for the block cipher and encrypts it using a fixed key.

Ciphertext

This is the result of a change made to letters or bits. This change is made by replacing, exchanging or replacing and exchanging information.

Cleartext

See plaintext

Cryptoki

A program interface to devices that save cryptographic information and carry out cryptographic functions; as specified by the PKCS#11 standard.

Decryption

The process of converting ciphered (or encrypted) text back to clear text.

DES (Data Encryption Standard)

A 64-bit block cipher or symmetric algorithm that is also referred to as the Data Encryption Algorithm (DEA) (ANSI) or DEA-1 (ISO). This has been used widely for the last 20 years, adopted in 1976 as “FIPS 46”.

Diffie-Hellman

The first encryption algorithm for public keys that used discrete logarithms in a finite field. It was created in 1976.

Distinguished Encoding Rules (DER)

A subset of BER.

DSA (Digital Signature Algorithm)

A digital signature algorithm created by NIST for public keys for use in DSS.

DSS (Digital Signature Standard)

A standard (FIPS) suggested by NIST for digital signatures using DSA.

ECB (electronic codebook)

A block cipher which uses the plain text block as direct input for the encryption algorithm. The output block resulting from the encryption process is then used directly as the ciphertext.

FIPS (Federal Information Processing Standard)

A standard of the USA government as published by NIST.

Handle

A value used to identify a session or an object assigned by Cryptoki (see also section 6.6.5 “Session handles and object handles” of the PKCS#11 standard).

Hash function

A single-direction hash function is a function that generates a message core, that cannot be reversed in order to obtain the original information.

HMAC

A key-dependent single-direction hash function specially designed for use with MAC (Message Authentication Code) and based on IETF RFC 2104.

IETF (Internet Engineering Task Force)

A comprehensive open international community made up of network developers, operators, dealers and research specialists whose job is the development of the Internet architecture and the smooth running of the Internet. This organization is open to all those who are interested.

Initialization vector or IV

A block of random data that uses “Chaining Feedback Mode” (see “Cipher Block Chaining (CBC)”) and serves as the starting point for a block cipher.

Integrity

Proof that data has not been changed (by unauthorized persons) during saving or transfer.

ISO (International Organization for Standardization)

This organization is responsible for a wide range of standards, for example, the OSI model, and also for international relations with ANSI for X. 509.

Key

This is a method of granting and rejecting the following access, ownership rights or control rights. This is represented by any number of values.

Key exchange

A procedure that uses two or more nodes to transfer a secret session key via an insecure channel.

Key length

The number of bits used to represent the key size. The longer the key - the more secure it is.

Key management

A procedure used to save and distribute accurate cryptographic keys. The entire process of secure creation and distribution of cryptographic keys to authorized recipients.

MAC (Message Authentication Code)

A key-dependant single-direction hash function which requires an identical key to verify the hash.

MD2 (Message Digest 2)

A single-direction hash function with 128-bit hash result, uses a random permutation of bytes.

Designed by Ron Rivest.

MD4 (Message Digest 4)

A single-direction hash function with 128-bit hash result, uses a simple set of bit manipulations with 32-bit operands. Designed by Ron Rivest.

MD5 (Message Digest 5)

An improved and more complex version of MD4. But still a single-direction hash function with 128-bit hash result.

Mechanism

Process used to implement cryptographic operations

Message Digest

A checksum that is calculated from a message. If you change just a single character in a message then the message will have a different Message Digest.

Mutex objects

Mutex is a short form of Mutual Exclusion; simple objects that can be in only one of two states at any time: locked or unlocked (see also PKCS#11 specification, section 6.5.2 “Applications and threads”).

NIST (National Institute for Standards and Technology)

A department of the “U.S. Department of Commerce”. Publishes standards on compatibility (FIPS).

NSA (National Security Agency)

A department of the “U.S. Department of Defense”.

OFB (Output Feedback Mode)

As in CFB a block cipher is used as a stream cipher. Unlike CFB, the bits of the output are directly fed back. OFB is not self-synchronizing.

Operation

A sequence of several cryptographic functions

PKCS (Public Key Crypto Standards)

A range of de-facto standards for encryption using public keys, developed by an informal consortium (Apple, DEC, Lotus, Microsoft, MIT, RSA and Sun). This also includes algorithm-specific and algorithm-independent implementation standards. Specifications for the definition of message syntax and other protocols that are controlled by the RSA Data Security, Inc.

Plaintext or cleartext

Data or messages before encryption, in a format that can be easily read by humans - also known as uncoded text.

Private key

The “secret” component of an integrated asymmetrical key pair, the component that is said to be in private possession, and is also often known as the decryption key.

Pseudo-random number

A number that is calculated by applying algorithms, this creates random values that are derived from the computer environment (e.g. mouse coordinates). See random number.

Public key

The publicly available component of an integrated asymmetrical key pair, often referred to as the encryption key.

Random number

An important aspect for many encryption systems and a necessary element when creating unique keys that a potential hacker is unable to calculate. Real random numbers are usually derived from analog sources and generally require the use of special hardware.

RC2 (Rivest Cipher 2)

Symmetrical 64-bit block cipher with variable key size, a branch-internal key from RSA, SDI.

RC4 (Rivest Cipher 4)

Stream cipher with variable key size, was originally the property of RSA Data Security, Inc. It is strongly recommended not to use RC4 as it meanwhile shows to many weaknesses. *has Da RC4 mittlerweile zu viele Schwächen zeigt, wird von seiner Verwendung dringend abgeraten.* See also RFC 7465 "Prohibiting RC4 Cipher Suites".

RFC (Request for Comment)

An IETF document from the subgroup FYI RFC giving an overview and introduction or from the subgroup STD RFC which gives Internet standard. The abbreviation FYI stands for "For Your Information". Each RFC has an RFC number that is used to identify it and call it up (www.ietf.org).

RSA

Short for RSA Data Security, Inc. Also refers to the names of the company founders Ron Rivest, Adi Shamir and Len Adleman, or to the algorithm developed by them. The RSA algorithm is used in cryptography with public keys. Its functionality is based on the fact that two large prime numbers may be easy to multiply together, but that it is very difficult to reduce the product back to the original two numbers.

Salt

A random string of characters that is linked using passwords (or random numbers) before operations are carried out using single-direction functions. This linking effectively extends the length of the password and makes it more obscure. Thus your cipher text is better protected against dictionary attack.

Secret key

The "session key" in symmetrical algorithms

Session key

The secret (symmetric) key used to encrypt all data records on a transaction basis. For each communication session a new session key is used.

SET (Secure Electronic Transaction)

Is used to transfer credit card details securely across the Internet.

SHA-1 (Secure Hash Algorithm)

In 1994 the SHA (FIPS 180-1) developed by NIST was revised and SHA-1 was the result. SHA-1 is used in conjunction with DSS to create a 160-bit hash result. It is similar to the popular and wide spread MD4.

Slot

According to the definition, PKCS#11 can simultaneously use logical cryptographic function units (slots). In CRYPT only one slot is supported at this time.

SSL (Secure Socket Layer)

This was developed by Netscape to ensure the security and non-disclosure of sensitive information on the Internet. Supports the server/client authentication and ensures the security and integrity of the transfer channel. This works on the transfer level and serves as the "socket library" and makes possible application-independent results. Encrypts the entire communication channel.

Stream cipher

A class of symmetric key encryption during which the conversion can be changed for each character of plaintext that is to be encrypted. This is recommended for environments with limited memory capacity available to buffer data.

Symmetric algorithm

Also called a conventional, secret key or single-key algorithm. The encryption key is either identical to the decryption key, or it is possible to derive one key from the other. There are two sub-categories - block and stream.

TLS (Transport Layer Security)

A draft from IETF. Version 1 is based on Version 3.0 of the SSL protocol and serves to maintain privacy when communicating across the Internet.

Triple-DES

An encryption configuration in which the DES algorithm is used three times with three different keys.

Related publications

You will find the manuals on the internet at <http://manuals.ts.fujitsu.com>. You can order printed copies of those manuals which are displayed with an order number.

- [1] **openNet Server** (BS2000/OSD)
BCAM
User Guide
- [2] **BS2000 OSD/BC**
Commands
User Guide
- [3] **BS2000 OSD/BC**
Executive Macros
User Guide

Additional references

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>

Index

A

- architecture, openCRYPT 11
- Assembler, sample program 78, 83
- asynchronous execution 32
- asynchronous execution (example) 83
- attribute
 - CKA_EXTRACTABLE 23
 - CKA_LOCAL 23
 - CKA_PUBLIC_EXPONENT 25
 - CKA_SENSITIVE 23

B

- BS2000
 - differences compared to PKCS#11
 - standard 19
 - functions not required 24
- BS2000 interfaces
 - associated PKCS#11 functions 13

C

- C function descriptions
 - reference to 94
- C sample program 97
- C_CopyObject 25
- C_Decrypt 26
- C_DecryptFinal 26
- C_DecryptUpdate 26
- C_Digest 26
- C_DigestFinal 26
- C_DigestUpdate 26
- C_Encrypt 26
- C_EncryptFinal 26
- C_EncryptUpdate 26
- C_Finalize 24, 25
- C_GenerateKeyPair 25

- C_GetMechanismInfo 25
- C_Initialize 24, 25
- C_InitToken 24
- C_Login 24
- C_Logout 24
- C_SetAttributeValue 25
- C_Sign 26
- C_SignFinal 26
- C_SignUpdate 26
- C_Verify 26
- C_VerifyFinal 26
- C_VerifyUpdate 26
- CCRY 58
- CCRYFIN 63
- CCRYINI 56
- CDRVKEY 74
- CGENKEY 66
- CGENKPR 68
- CGENRAL 35
- CGTSTMI 38
- CINITTK 43
- CKA_EXTRACTABLE 23, 25
- CKA_LOCAL 23
- CKA_PUBLIC_EXPONENT 25
- CKA_SENSITIVE 23, 25
- CKM_RSA_PKCS 25, 26
- CKM_RSA_X_509 25, 26
- CKR_ATTRIBUTE_VALUE_INVALID 24
- CKR_KEY_HANDLE_INVALID 24
- CKR_MECHANISM_INVALID 24
- CLOG 50
- close
 - all sessions 45
 - session 45
- COBJMGT 51

continue

- decryption and digesting operation [58](#)
- decryption and digesting operation (multiple-part) [58](#)
- decryption and verification operation (multiple-part) [58](#)
- decryption operation (multiple-part) [58](#)
- digesting and encryption operation [58](#)
- encryption operation (multiple-part) [58](#)
- message-digesting operation (multiple-part) [56, 58](#)
- search (according to template) [51](#)
- signature and encryption operation [58](#)
- signature operation [58](#)
- verification operation (multiple-part) [58](#)

continuing

- decryption and verification operation [58](#)
- signature and encryption operation [58](#)

COPSTAT [48](#)

copy

- object [51](#)

CPIN [44](#)

CPKC11T [105](#)

- main return code [105](#)

CRANDOM [76](#)

creating diagnostic documents [103](#)

CRYASC2

- sub return code 2 [105](#)

CRYPT

- differences compared to PKCS#11 standard [19](#)

Cryptoki library, initialize [35](#)

CSESSION [45](#)

CUNWKEY [72](#)

CWRPKEY [70](#)

CWTFSL [42](#)

D

data

- digest (single-part) [58](#)
- sign [58](#)

decrypt data (single-part) [58](#)

decryption operation

- continue (multiple-part) [58](#)
- initialize [56](#)
- terminate [63](#)

definition

- mechanism [17](#)
- operation [24](#)

delete

- object [51](#)

derive

- key [74](#)

description

- macro calls [34](#)

determine size of output area [24](#)

diagnostic documents, create [103](#)

differences compared to version 1.0 [9](#)

digest data [58](#)

disclaimer [7, 25](#)

display information

- CGTSTMI [38](#)

display operation state [48](#)

E

encrypt data (single-part) [58](#)

encryption operation

- continue (multiple-part) [58](#)
- initialize [56](#)
- terminate [63](#)

error codes [105](#)

execute operation [58](#)

execution

- asynchronous [32](#)
- asynchronous (example) [83](#)
- synchronous [32](#)
- synchronous (example) [78](#)

F

finalize application 35
flag
 CKA_EXTRACTABLE 25
 CKA_SENSITIVE 25
functions
 C_CopyObject 25
 C_Finalize 25
 C_GenerateKeyPair 25
 C_GetMechanismInfo 25
 C_Initialize 25
 C_SetAttributeValue 25
 maximum output data length 24
 special features 24

G

general data types
 special features compared to PKCS#11 23
general functions
 CGENERAL 35
 not required in BS2000 24
general-purpose functions
 not required in BS2000 24
generate
 key pair 68
 object 51
 random data 76
 random numbers 76
 secret key 66

H

hash algorithms 17

I

important note
 security 7, 25
include file
 PKCS11.H 93
 PKCS11F.H 93
 PKCS11T.H 93
initialize
 Cryptoki library 35
 decryption operation 56
 encryption operation 56
 message-digesting operation 56
 operation 56
 PIN 44
 search (according to template) 51
 signature operation 56
 token 43
 verification operation 56
input data length
 update operation 26
integrity codes 17

K

key
 derive 74
 generate (secret) 66
 unwrap 72
 wrap 70
key pair, generate 68

L

login 50
 user to token 50
logout 50
 user out of token 50

- M**
- macro
 - metasyntax 28
- macro call
 - CCRY 58
 - CCRYFIN 63
 - CCRYINI 56
 - CDRVKEY 74
 - CGENKEY 66
 - CGENKPR 68
 - CGENRAL 35
 - CGTSTMI 38
 - CINITTK 43
 - CLOG 50
 - COBJMGT 51
 - COPSTAT 48
 - CPIN 44
 - CRANDOM 76
 - CSESSION 45
 - CUNWKEY 72
 - CWRPKEY 70
 - CWTFSL 42
- macro calls
 - description 34
- macro syntax 30
 - control operands 30
 - macro forms 30
- main return code 105
 - list 106
- management
 - object 51
 - session 45
- mechanism
 - CKM_RSA_PKCS 25, 26
 - CKM_RSA_X_509 25, 26
 - definition 17
 - outputting information about 38
- mechanism types
 - supported by a token 38
- mechanisms
 - supporting cryptographic operations 20
- message-digesting operation
 - continue 58
 - continue (multiple-part) 56
 - initialize 56
 - terminate 63
- metasyntax
 - macro 28
- modify
 - object attributes 51
 - PIN 44
- MRC 105
- N**
- notational conventions 10
- note
 - security 7, 25
- O**
- object
 - copy 51
 - delete 51
 - generate 51
 - modifying attributes 51
 - output attributes 51
 - output size 51
 - special features 23
- object generation
 - attribute 23
- object management 51
- open
 - session 45
- openCRYPT
 - architecture 11
- openCRYPT-SERV
 - mechanisms 17
- operation 24
 - execute 58
 - initialize 56
 - sign 25, 26
 - single-part 26
 - terminate 63
 - verify 25, 26

- operation state
 - display 48
 - set 48
- output
 - data length, maximum 24
 - function list of Cryptoki library 35
 - general information about Cryptoki 35
 - object attributes 51
 - object size 51
 - slot list 38
 - state of the cryptographic operations 48
 - supported mechanism types 38
- output area
 - determine size 24
- outputting information
 - mechanism 38
 - session 45
 - slot 38
 - token 38
- P**
- PIN
 - initialize 44
 - modify 44
- PKCS#11
 - prefixes 19
- PKCS#11 functions
 - associated BS2000 interfaces 13
- PKCS#11 mechanisms 17
- PKCS#11 standard 11
- PKCS11.H 93
- PKCS11T.H 93, 105
- PKCSF.H 93
- prefixes 19
- prerequisites
 - use the C interface 93
- public key procedure 18
- R**
- random data, generate 76
- random generators 18
- random number generator
 - mix seed material into 76
- random numbers, generate 76
- Readme file 9
- reference
 - to C function description 94
- return code
 - CKR_ATTRIBUTE_VALUE_INVALID 24
 - CKR_KEY_HANDLE_INVALID 24
 - CKR_MECHANISM_INVALID 24
- S**
- sample program
 - Assembler 78, 83
 - C 97
- search
 - continue according to template 51
 - initiate according to template 51
 - terminate 51
- security information 7, 25
- seed material
 - mix into random number generator 76
- session
 - close 45
 - close (all) 45
 - open 45
 - outputting information 45
- session management 45
- set operation state 48
- sign 25, 26
 - data 58
- signature
 - verify 58
- signature operation
 - continue 58
 - initialize 56
 - terminate 63
- single-part operation
 - corresponding update and final operations 26
- slot
 - outputting information about 38
- slot event
 - waiting for 42
- slotId 23

special features

- functions [24](#)
- general data types [23](#)
- mechanisms [20](#)
- objects [23](#)

SRC2 [105](#)

standard, PKCS#11 [11](#)

state of the cryptographic operations

- output [48](#)
- restore [48](#)

sub return code 2 [105](#), [109](#)

- list [110](#)

supported mechanisms [20](#)

symmetric algorithms [17](#)

synchronous execution [32](#)

synchronous execution (example) [78](#)

T

target groups [7](#)

terminate

- decryption operation [63](#)
- encryption operation [63](#)
- message-digesting operation [63](#)
- operation [63](#)
- search [51](#)
- signature operation [63](#)
- verification operation [63](#)

token

- initialize [43](#)
- outputting information about [38](#)
- outputting supported mechanism types [38](#)

U

unwrap

- key [72](#)

update and final operations

- single-part operation [26](#)

update operation

- input data length [26](#)

use of registers [27](#)

user

- login to token [50](#)
- logout of token [50](#)

using the C interface

- prerequisites [93](#)

V

verification operation

- continue [58](#)
- initialize [56](#)
- terminate [63](#)

verify [25](#), [26](#)

- signature [58](#)

W

waiting for a slot event [42](#)