CASTLES TECHNOLOGY

EFT-POS Terminal

Key Management System II

User Manual

Confidential

Version 0.99

April 2014

Castles Technology Co., Ltd.

2F, No. 205, Sec. 3, Beixin Rd., Xindian District,

New Taipei City 23143, Taiwan R.O.C.

http://www.castech.com.tw

WARNING

Information in this document is subject to change without prior notice.

Revision History

Version	*Date*	*Descriptions*
0.1	June 12, 2013	Created
0.9	June 27, 2013	Draft
0.91	July 8, 2013	1. Change the control member of the structure CTOS_KMS2PINGET_PARA
0.92	July 22, 2013	1. Support CTOS_KMS2PINGET_PARA_VERSION_2. 2. Add certificate format definition tables in section 2.3
0.93	October 21, 2013	1. Section 2.3. Table Description Modified. 2. Add function “CTOS_KMS2IntermediateKeyGenerate”. “CTOS_KMS2IntermediateKeyWrite”, “CTOS_KMS2IntermediateKeyFlush”. “CTOS_KMS2IntermediateKeyErase”.
0.94	October 22, 2013	1. Add Section 1.2.4 Intermediate Key Register
0.95	November 21, 2013	1. Support CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA_VERSION_2 2. Support additional key attributes Freeze KeyWritebyCertificate function, Freeze RSAEncrypt function, Protected Mode.
0.96	November 29, 2013	1. Add (Single DES) DUKPT which is compliant with ANS X9.24 1998.
0.97	December 10, 2013	1. Upgrade 3DES-DUKPT from ANS x9.24-2004 to ANS x9.24-2009. 2. Add note in function CTOS_KMS2DataEncrypt and CTOS_KMS2MAC for 3DES-DUKPT.
0.98	January 16, 2014	1. Add key attribute KMS2_KEYATTRIBUTE_VALUE_UNIQUE for 3DES and AES key.
0.99	April 2, 2014	1. Add operation mode KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY for intermediate key function. 2. New function CTOS_KMS2KeySwap. 3. Add KMS2_KEYPROTECTIONMODE_KPK_CBC mode for key injection. 4. Add 3DUKPT_ECB_POUND_x and 3DUKPT_CBC_POUND_x methods for data encryption with 3DES-DUKPT. 5. Add d_KMS2_KEY_TYPE_NOT_MATCH and d_KMS2_DUKPT_KEY_EXPIRED into error code list.

Contents

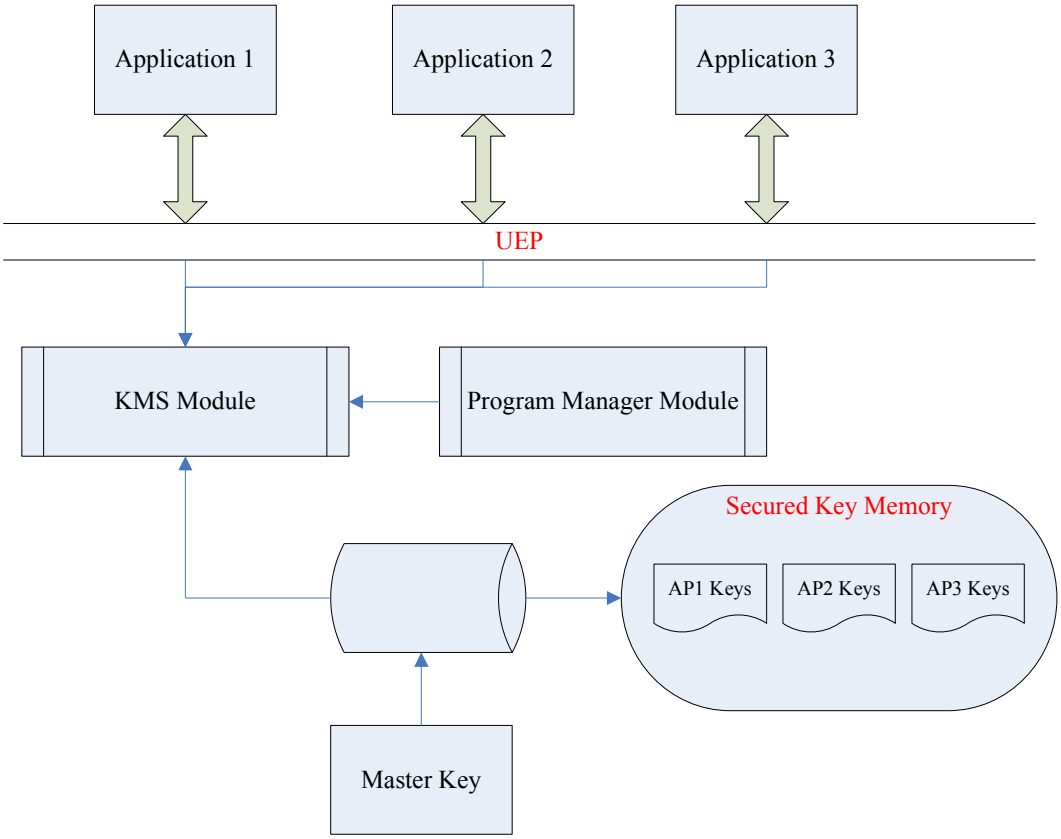
1. Introduction

This document describes all the necessary information for application developers to understand how to use the Key Management System II (KMS-II) to store their application keys and perform cryptographic operations with them.

The KMS is designed to securely store sensitive data such as keys and provide cryptographic operation, which includes:

- KEK (Highest Level Master Key) maintenance.
- Key integrity checking.
- Hardware tamper for key protection.
- User keys maintenance in key storage (Injection, deletion, and etc).
- Corresponding cryptographic operations with key for user usage (PIN Cipher, Data Encryption, etc).

1.1. KMS Framework



The key storage is provided for storing sensitive data. All data in key storage are encrypted by KEK. [KEK is automatically random generated during first booting.] Each application has its own keys. The key belonged to one application cannot be accessed by others. Accessing application keys is always allowed via “KMS Module”. When an application tries to use their keys by the APIs provided in KMS, the KMS Module will provide its cryptographic operations only if the application is the key owner. When an application is removed, the keys belonged to this application will be removed as well. The “Program Manager Module” is responsible for this key deletion.

1.2. Key Storage

1.2.1. Key Types

KMS-II supports the following key types

- 16-bytes / 24-byets 3DES Key (KMS2_KEYTYPE_3DES)
- 3DES DUKPT Key [ANS X9.24-2009)] (KMS2_KEYTYPE_3DES_DUKPT)
- AES 128 bit Key (KMS2_KEYTYPE_AES)
- RSA Key up to 2048 bits (KMS2_KEYTYPE_RSA), e.g. 1024/1536/2048,etc.
- DES DUKPT Key [ANS X9.24-1998)] (KMS2_KEYTYPE_DES_DUKPT)

1.2.2. Key Set & Key Index

A Working Key (WK) is identified by its Key Set and Key Index.

- Each key set is unique for all applications.
- Key Index is unique in one key set.
- Each key set has its own key type, that is, all keys stored in a key set must be with the same key type.
- Each key set has its owner (an application). Unless the key set is a common key set, only the owner is allowed to do cryptographic functions with any key of the key set.
- The key set values ranging from 0xC000 to 0xCFFF are as a common key set, which mean any application can use the keys of this kind of key set no matter it is the owner or not.
- The key set values ranging from 0xFF00 to 0xFFFF or equal to 0x0000 are reserved for system.

1.2.3. Key Attributes

Each working key has its own attributes. Corresponding attribute provides corresponding cryptographic function. The attributes are bit-mask and listed as below.

- PIN Encryption (01h)
  Allow to perform PIN encryption

- Data Encryption (02h)
  Allow to perform data encryption

- MAC (04h)
  Allow to perform MAC calculation

- Key Protection Key (08h)
  Allow to protect key during key injection

- Data Decryption (10h)
  Allow to perform data decryption
  Note: Data Decryption will be performed only if both of "Data Encryption" and "Data Decryption" of key attributes are set.

- SK Encryption (40h)
  Normally, Session Key (SK) used to perform cryptographic operations instead of WK will be retrieved by decrypting sSK with specified WK if the length of sSK is not zero. The WK with this attribute set will retrieve SK by "encrypting" sSK instead of "decrypting".

- Intermediate Key (80h)
  Allow to load the key into intermediate key register when using intermediate key functions.

- Freeze KeyWritebyCertificate function (100h)
  Disallow to use the key for CTOS_KMS2KeyWriteByCertificate function. This is only for RSA key type.

- Freeze RSAEncrypt function (200h)
  Disallow to use the key for CTOS_KMS2RSAEncrypt. This is only for RSA key type.
  Note: It is strongly recommended to set this attribute to the RSA key that is used for key injection via CTOS_KMS2KeyWriteByCertificate function.

- Key Value Unique (20000000h)
  This attribute ensure that all keys with this attribute are unique between each other. This is only for 3DES and AES key type.

- Protected Mode (40000000h)
  Only the key owner allowed to change the key. This is used to prevent other applications rather than the owner from changing the key.

1.2.4. Intermediate Key Register

KMS-II provides intermediate key function for key generation. Intermediate key registers are used to store the intermediate key which derive from a seed key with specified operation.

Note.
The intermediate key registers are volatility. All the keys stored here will be erased after power off or reboot!

Key set 0xFF00 is reserved for intermediate key register, and maximum of 16 key registers are provided for use. The key index of intermediate key register is form 0x00 to 0x0F. Each key stored in the intermediate key register has its own attributes. User not only use them for intermediate key operation but also use for cryptographic function directly with corresponding attributes.

Note.
For intermediate key operation except for GENERATE RANDOM KEY, at least one working key should be injected as seed key, and the key attribute "Intermediate Key (80h)" should be set.

KMS-II provides various methods for intermediate key operation as below.

- XOR
- DES
- TAKE
- COMBINE
- AES
- GENERATE RANDOM KEY

2. Key Injection

KMS-II provides the following methods for applications to inject their keys into secure key storage:

- By Plaintext Key
- By Key Protection Key (3DES or AES)
- By RSA Certificate

2.1. By Plaintext Key

To inject keys in plaintext, set the "***Mode***" of the "***Protection***" of *CTOS_KMS2KEYWRITE_PARA* to **KMS2_KEYPROTECTIONMODE_PLAINTEXT** within the *CTOS_KMS2KeyWrite* or *CTOS_KMS2KeyWriteEx* function.

2.2. By Key Protection Key

To inject keys in symmatric encryption, set the "***Mode***" of the "***Protection***" of *CTOS_KMS2KEYWRITE_PARA* to **KMS2_KEYPROTECTIONMODE_KPK_ECB** or **KMS2_KEYPROTECTIONMODE_KPK_CBC** with indicating *CipherKeySet* and *CipherKeyIndex* in the *CTOS_KMS2KeyWrite* or *CTOS_KMS2KeyWriteEx* function. The field *CipherKeySet* and *CipherKeyIndex* is used to indicate the key that is used to do symmatric (**3DES** or **AES**) encryption. Note that this protection key shall have the key attribute **KMS2_KEYATTRIBUTE_KPK**.

2.3. By Ceritificate

Certificate for 3DES/AES key (Fomat 20h) –

Length	Description
1	Header, should be 0x6A
1	Format, should be 0x20
1	HashAlgorithm, 0x00 – SHA1 0x01 – SHA2
20	Key Owner
2	Key Set
2	Key Index
1	Key Type, should be 3DES(0x01) or AES(0x03)
1	Key Version

4	Key Attribute
2	Key Length
16/24	KeyData
1	CV(Check Value) Length
0 to 8	CV(Check Value Data, 0 to 8 bytes
Var.	Padding with 0xBB
20/32	Hash Data, calculated with the input data from “Format” to “Padding”
1	Tailer, should be 0xBC

Certificate for 3DES DUKPT key (Fomat 21h) –

Length	Description
1	Header, should be 0x6A
1	Format, should be 0x21
1	HashAlgorithm, 0x00 – SHA1 0x01 – SHA2
20	Key Owner
2	Key Set
2	Key Index
1	Key Type, should be 3DES_DUKPT(0x02)
1	Key Version
4	Key Attribute
2	Key Length
16	KeyData
10	Key Serial Number (KSN)
1	CV(Check Value) Length
0 to 8	CV(Check Value Data, 0 to 8 bytes
Var.	Padding with 0xBB
20/32	Hash Data, calculated with the input data from “Format” to “Padding”
1	Tailer, should be 0xBC

Certificates for RSA key (Fomat 22h) –

RSA Key Data := Modulus || Exponent Length (2 bytes) || Exponent

⇨ KeyData 1 of KeyCertPart1 [|| KeyData 2 of KeyCertPart2] [|| KeyData 3 of KeyCertPart3]

KeyCertificate = KeyCertPart1 [|| KeyCertPart2] [|| KeyCertPart3]

,where || indicates concatenation, [ ] indicates optional

For each KeyCertPart –

Length	Description
1	Header, should be 0x6A
1	Format, should be 0x22
1	HashAlgorithm,

	0x00 – SHA1 0x01 – SHA2
20	Key Owner
2	Key Set
2	Key Index
1	Key Type, should be RSA(0x04)
1	Key Version
4	Key Attribute
2	Key Length
1	Total Number of CertParts
1	CertPart No., 0-based
2	KeyData Length in this CertPart
Var.	KeyData
Var.	Padding with 0xBB
20/32	Hash Data, calculated with the input data from "Format" to "Padding"
1	Tailer, should be 0xBC

Certificate for DES DUKPT key (Fomat 23h) –

Length	Description
1	Header, should be 0x6A
1	Format, should be 0x23
1	HashAlgorithm, 0x00 – SHA1 0x01 – SHA2
20	Key Owner
2	Key Set
2	Key Index
1	Key Type, should be DES_DUKPT(0x05)
1	Key Version
4	Key Attribute
2	Key Length
8	KeyData
10	Key Serial Number (KSN)
1	CV(Check Value) Length
0 to 8	CV(Check Value Data, 0 to 8 bytes
Var.	Padding with 0xBB
20/32	Hash Data, calculated with the input data from "Format" to "Padding"
1	Tailer, should be 0xBC

3. KMS Application Programming Interfaces

Management

- void CTOS_KMS2Init(void);
- USHORT CTOS_KMS2KeyCheckAll(void);
- USHORT CTOS_KMS2KeyCheck(IN USHORT KeySet, IN USHORT KeyIndex);
- void CTOS_KMS2Erase(void);
- USHORT CTOS_KMS2KeyDeleteAll(void);
- USHORT CTOS_KMS2KeyDelete(IN USHORT KeySet, IN USHORT KeyIndex);
- USHORT CTOS_KMS2KeySwap(CTOS_KMS2KEYSWAP_PARA *para);

Key Injection

- USHORT CTOS_KMS2KeyWrite(CTOS_KMS2KEYWRITE_PARA* pKeyWritePara);
- USHORT CTOS_KMS2KeyWriteEx(CTOS_KMS2KEYWRITEEX_PARA* pKeyWriteExPara);
- USHORT CTOS_KMS2KeyWriteByCertificate(CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA* pKeyWriteByCertificate);

Key Crypto Functions

- USHORT CTOS_KMS2PINGet(CTOS_KMS2PINGET_PARA *pPinGetPara);
- USHORT CTOS_KMS2DataEncrypt(CTOS_KMS2DATAENCRYPT_PARA *pDataEncPara);
- USHORT CTOS_KMS2MAC(CTOS_KMS2MAC_PARA *pMacPara);
- USHORT CTOS_KMS2RSAEncrypt(CTOS_KMS2RSAENCRYPT_PARA *pRSAEncryptPara);
- USHORT CTOS_KMS2KeyGetInfo(IN CTOS_KMS2KEYGETINFO_PARA *pKeyGetInfoPara);
- USHORT CTOS_KMS2DUKPTGetKSN(IN USHORT KeySet, IN USHORT KeyIndex, OUT BYTE* pKSN, INOUT BYTE* KSNLen);

Additional Storage Functions

- USHORT CTOS_KMS2UserDataWrite(IN BOOL IsCommon, IN ULONG Offset, IN BYTE *pData, IN USHORT usLen);
- USHORT CTOS_KMS2UserDataRead(IN BOOL IsCommon, IN ULONG Offset, OUT BYTE *pData, IN USHORT usLen);

Intermediate Key Function

- USHORT CTOS_KMS2IntermediateKeyGenerate(CTOS_KMS2INTERMEDIATEKEYGENERATE_PARA* pIntermediateKeyGeneratePara);
- USHORT CTOS_KMS2IntermediateKeyWrite(CTOS_KMS2INTERMEDIATEKEYWRITE_PARA* pIntermediateKeyWritePara);

- USHORT CTOS_KMS2IntermediateKeyFlush(IN USHORT KeySet, IN USHORT KeyIndex);
- USHORT CTOS_KMS2IntermediateKeyErase(void);

KMS2 Error Codes

Constants	***Value***	***Description***
d_KMS2_INVALID_PARA	0x2901	The parameter is invalid
d_KMS2_FAILED	0x2902	General Failure
d_KMS2_SYSTEM_ERROR	0x2903	System Error
d_KMS2_NOT_OWNER	0x2904	The key does not belong to this application
d_KMS2_KEY_NOT_EXIST	0x2905	The key does not exist
d_KMS2_KEYTYPE_INCORRECT	0x2906	The key type is incorrect
d_KMS2_KEY_NOT_ALLOWED	0x2907	The key attribute is not allowed to use this operation,
d_KMS2_KEY_VERIFY_INCORRECT	0x2908	The verification code is incorrect.
d_KMS2_NOT_SUPPORTED	0x2909	The function (with the input argument) is not supported
d_KMS2_CERTIFICATE_INCORRECT	0x290A	The certificate format is incorrect.
d_KMS2_HASH_INCORRECT	0x290B	The hash is incorrect.
d_KMS2_CERTIFICATE_PARA_INCORRECT	0x290C	The parameter value(s) in the certificate is incorrect.
d_KMS2_INSUFFICIENT_BUFFER	0x290D	The buffer is insufficient
d_KMS2_DUKPT_KEY_NOT_GENERATED	0x290E	The dukpt key has not yet been generated.
d_KMS2_GET_PIN_ABORT	0x290F	User presses CANCEL key during getting PIN
d_KMS2_GET_PIN_TIMEOUT	0x2910	Timeout occurs during getting PIN
d_KMS2_GET_PIN_NULL_PIN	0x2911	User enters empty PIN.
d_KMS2_PKCS_FORMAT_ERROR	0x2912	PKCS#1.2 format error
d_KMS2_KEY_VALUE_NOT_UNIQUE	0x2913	Key value is not unique.
d_KMS2_KEY_TYPE_NOT_MATCH	0x2914	The key type between source and destination are different.
d_KMS2_DUKPT_KEY_EXPIRED	0x2915	The KSN reaches the maximum value.

CTOS_KMS2Init

```
void CTOS_KMS2Init(void);
```

Description	Initiate KMS-II Library. Please call this function in your main() function.
Parameters	None
Return Value	None

Example	`void main()`
	`{`
	`  CTOS_KMS2Init();`
	`}`

Note	This function does not erase the keys in key storage.

CTOS_KMS2KeyCheck

```
USHORT CTOS_KMS2KeyCheck(IN USHORT KeySet, IN USHORT KeyIndex);
```

Description Check if the specified key exists or not.

Parameters

[IN] *KeySet*

Used to indicate which key set it belong to.

[IN] *KeyIndex*

Specify its index in the key set.

Return Value

Constants	***Value***
d_OK (Key Exists)	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2KeyCheck(0x1000, 0x0001);
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "Key Check OK");
  else
      CTOS_LCDTPrintXY(1, 1, "Key Check Failed");
  while (1);
}
```

CTOS_KMS2KeyCheckAll

```
USHORT CTOS_KMS2KeyCheck(void);
```

Description Check all the keys belong to the (caller) application.

Parameters None

Return Value

Constants	***Value***
d_OK (Key Exists)	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2KeyCheckAll();
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "Keys Check OK");
  else
      CTOS_LCDTPrintXY(1, 1, "Keys Check Failed");
  while (1);
}
```

CTOS_KMS2KeyDelete

```
USHORT CTOS_KMS2KeyDelete(IN USHORT KeySet, IN USHORT KeyIndex);
```

Description This function is used to delete a key with specified key index.

Parameters

[IN] *KeySet*

Used to indicate which key set it belong to.

[IN] *KeyIndex*

Specify its index in the key set.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
      USHORT rtn;
      CTOS_KMS2Init();
      rtn = CTOS_KMS2KeyDelete(0x1000, 0x0001);
      if (rtn == d_OK)
            CTOS_LCDTPrintXY(1, 1, "Delete OK!");
      else
            CTOS_LCDTPrintXY(1, 1, "Delete Failed!");
      while (1);
}
```


CTOS_KMS2KeyDeleteAll

```
USHORT CTOS_KMS2KeyDeleteAll(void);
```

Description Delete all the keys belong to the (caller) application.

Parameters None

Return Value

Constants	***Value***
d_OK (Key Exists)	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2KeyDeleteAll();
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "All Keys Deleted");
  else
      CTOS_LCDTPrintXY(1, 1, "Key Delete Failed");
  while (1);
}
```


CTOS_KMS2Erase

```
USHORT CTOS_KMS2Erase(void);
```

Description Physically erase all the secure memory.

Parameters None

Return Value

Constants	***Value***
d_OK (Key Exists)	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2Erase();
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "Erase KMS OK");
  else
      CTOS_LCDTPrintXY(1, 1, "Erase KMS Failed");
  while (1);
}
```


CTOS_KMS2KeySwap

```
USHORT CTOS_KMS2KeySwap(CTOS_KMS2KEYSWAP_PARA *para);
```

Description Specify 2 keys and swap each other.

Note. Source1 & source2 should have the same key type, otherwise the error code d_KMS2_KEY_TYPE_NOT_MATCH will be thrown.

```
typedef struct
{
    // Should be 0x00 or 0x01
    IN BYTE Version;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;

    }Source1;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;

    }Source2;

}CTOS_KMS2KEYSWAP_PARA;
```

Parameters

[IN] *Version*
Structure Format Version. It shall be 0x00 or 0x01.

[IN] *Source1.KeySet*
Specify the 1st key set.

[IN] *Source1.KeyIndex*

Specify the 1st key Index.

[IN] *Source2.KeySet*

Specify the 2nd key set.

[IN] *Source2.KeyIndex*

Specify the 2nd key Index.

Return Value

Constants	***Value***
d_OK (Key Exists)	0000h
KMS2 error codes	29xxh

Example

```
USHORT ret;
BYTE str[17];
CTOS_KMS2KEYSWAP_PARA KMS2KeySwap;

KMS2KeySwap.Version = 0x01;
KMS2KeySwap.Source1.KeySet = 0x0100;
KMS2KeySwap.Source1.KeyIndex = 0x0001;
KMS2KeySwap.Source2.KeySet = 0x0102;
KMS2KeySwap.Source2.KeyIndex = 0x0004;
ret = CTOS_KMS2KeySwap(&KMS2KeySwap);
if(ret)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2KeyWrite

```
USHORT CTOS_KMS2KeyWrite(CTOS_KMS2KEYWRITE_PARA* pKeyWritePara);
```

Description This function is used to write or update a key into KMS. All the keys written by this function will take the caller application as their owner.

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            IN USHORT KeySet;
            IN USHORT KeyIndex;
            IN BYTE KeyType;
            IN BYTE KeyVersion;

            // For KeyType RSA, only accepts
            // KMS2_KEYATTRIBUTE_PROTECTED
            IN DWORD KeyAttribute;
      }Info;

      // This is not used for KeyType RSA
      struct
      {
            IN BYTE Mode;

            // This is used for KPK_ECB and TR31 Mode
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;

            struct
            {
            // This is used for TR31 Mode
            // This is used as ICV for
            // SELF_UPDATE_CBC mode
                  IN USHORT Length;
                  IN BYTE* pData;
            }AdditionalData;

      }Protection;

      struct
      {
```

```
        // For KeyType RSA, this is used for Modulus.
            IN USHORT KeyLength;
            IN BYTE* pKeyData;
        }Value;

        // Optional. This is not used if
        // Protection.Mode is
        // KMS2_KEYPROTECTIONMODE_PLAINTEXT
        // This is not used for KeyType RSA
        struct
        {
            IN BYTE Method;
            IN USHORT KeyCheckValueLength;
            IN BYTE* pKeyCheckValue;
        }Verification;

        // Optional. Used only for KeyType RSA
        struct
        {
            IN USHORT Length;
            IN BYTE* pValue;
        }Exponent;

        // Optional. Used only for KeyType
3DES_DUKPT/DES_DUKPT
        struct
        {
            IN BYTE KSNLength;
            IN BYTE *pKSN;
        }DUKPT_DATA;

}CTOS_KMS2KEYWRITE_PARA;
```

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00 or 0x01.

[IN] *Info.KeySet*

Specify the key set the new/update key belongs to.

[IN] *Info.KeyIndex*

Specify the key index the new/update key belongs to.

[IN] *Info.KeyType*

Specify the key type of the new/update key. **Note that if specified the key set does not exist, the key set will be created with this key type. If the specified key set exists already, the key type of the new/update key shall be identical with the key type of the key set.**

[IN] Info.KeyVersion

Specify the key version.

[IN] Info.KeyAttribute

Specify the key attribute.

[IN] *Protection.Mode*

Specify which mode is used to protect the new/update key.

- Plaintext mode

KMS2_KEYPROTECTIONMODE_PLAINTEXT (0x00)

- KPK_ECB mode

KMS2_KEYPROTECTIONMODE_KPK_ECB (0x01)

- KPK_CBC mode

KMS2_KEYPROTECTIONMODE_KPK_CBC (0x05)

[IN] *Protection.CipherKeySet*

This is used for KPK_ECB mode.

Specify the key set of the key used for encryption.

[IN] *Protection.CipherKeyIndex*

This is used for KPK_ECB mode.

Specify the key index of the key used for encryption.

[IN] *Protection.AdditionalData*

This is used to specify ICV when Protection,Mode is KPK_CBC mode.

[IN] *Value.KeyLength*

Specify the new/update key length.

For RSA key type, this field is put with RSA key length (in

byte).

[IN] *Value.pKeyData*

Point to a buffer containing the ciphered or plaintext key data.

For RSA key type, this field is put with its Modulus.

[IN] Verification.Method

This field is used only for the 3DES/3DES-DUKPT/AES/DES-DUKPT key types.

Specify the method for verifying if the key data after decryption is correct or not.

The default value 0x00 of verification method with 3DES key type, 3DES-DUKPT key type, AES key type or DES-DUKPT key type to calculate the key check values is as below.

- For 3DES, 3DES-DUKPT or DES-DUKPT key type,

key check values := take the result of **3DES(key, 8 bytes of zeros)** from left to right with the length specified by the parameter “KeyCheckValueLength”.

- For AES key type,

key check values := take the result of **AES(key, 16 bytes of zeros)** from left to right with the length specified by the parameter “KeyCheckValueLength”.

Note that the field is not used for the plaintext protection mode.

[IN] Verification.KeyCheckValueLength

Specify the length of key check values stored in Verification.pKeyCheckValue.

Note that the field is not used for the plaintext protection mode.

[IN] Verification.pKeyCheckValue

Point to a buffer containing the key check values.

Note that the field is not used for the plaintext protection mode.

[IN] Exponent.Length

This field is used only for the RSA key type.

Specify the length of exponent of a RSA key.

[IN] Exponent.pValue

This field is used only for the RSA key type.

Point to a buffer containing the exponent of a RSA key.

[IN] DUKPT_DATA.KSNLength

This field is used only for the 3DES-DUKPT or DES-DUKPT key type.

Specify the length of KSN (Key Serial Number). This value shall be 10.

[IN] DUKPT_DATA.pKSN

This field is used only for the 3DES-DUKPT or DES-DUKPT key type.

Point to a buffer containing KSN.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

//

```
BYTE const Key_3DES_1000_0001[] = "3DES_1000_0001_0";
USHORT KeySet;
USHORT KeyIndex;
CTOS_KMS2KEYWRITE_PARA para;
USHORT ret;
BYTE KeyData[16];
BYTE str[17];

// Write 3DES Key in plaintext
KeySet = 0x1000;
KeyIndex = 0x0001;
memcpy(KeyData, Key_3DES_1000_0001, 16);

memset(&para, 0x00, sizeof(CTOS_KMS2KEYWRITE_PARA));
para.Version = 0x01;
para.Info.KeySet = KeySet;
para.Info.KeyIndex = KeyIndex;
para.Info.KeyType = KMS2_KEYTYPE_3DES;
```

```
para.Info.KeyVersion = 0x01;
para.Info.KeyAttribute = KMS2_KEYATTRIBUTE_PIN |
KMS2_KEYATTRIBUTE_ENCRYPT | KMS2_KEYATTRIBUTE_MAC |
KMS2_KEYATTRIBUTE_KPK;
para.Protection.Mode = KMS2_KEYPROTECTIONMODE_PLAINTEXT;
para.Value.pKeyData = KeyData;
para.Value.KeyLength = 16;

ret = CTOS_KMS2KeyWrite(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2KeyWriteEx

```
USHORT CTOS_KMS2KeyWriteEx(CTOS_KMS2KEYWRITEEX_PARA*
pKeyWriteExPara);
```

Description This function is used to write or update a key into KMS with specifying the key owner.

```
typedef struct
{
      struct
      {
            IN BYTE Length;
            IN BYTE Name[20];
      }KeyOwner;

      CTOS_KMS2KEYWRITE_PARA KeyWritePara;

}CTOS_KMS2KEYWRITEEX_PARA;
```

Parameters

[IN] KeyOwner.Length

Specify the length of key owner name.

[IN] KeyOwner.Name

A buffer used to fill with the key owner name.

[IN] KeyWritePara

A structure of **CTOS_KMS2KEYWRITE_PARA**. For more detail, please refer to the **CTOS_KMS2KeyWrite** section.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example //

CTOS_KMS2KeyWriteByCertificate

```
USHORT
CTOS_KMS2KeyWriteByCertificate(CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA*
pKeyWriteByCertificate);
```

Description This function is used to write or update a key into KMS by RSA ceriticate.

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            // Indicate the RSA key for decrypting
            // the certificate.
            // The default RSA private key is in
            // KMS2_DEFAULT_RSA_KEY_SET
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;
      }Protection;

      struct
      {
            IN USHORT Length;
            IN BYTE* pData;
      }Certificate;

}CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA;
```

```
typedef struct
{
      // Should be 0x02
      IN BYTE Version;

      struct
      {
        // Indicate the RSA key for decrypting
        // the certificate.
        // The default RSA private key is in
        // KMS2_DEFAULT_RSA_KEY_SET
        IN USHORT CipherKeySet;
        IN USHORT CipherKeyIndex;
```

```
}Protection;

struct
{
  //0x00 : PKCS#1.2
  IN BYTE Format;
  IN USHORT Length;
  IN BYTE* pData;
}Certificate;

struct
{
  struct
  {
      IN BYTE Length;
      IN BYTE Name[20];
  }KeyOwner;

  struct
  {
     // Note that 0x0000 and 0xFF00 to 0xFFFF
     //are reserved for system.
     IN USHORT KeySet;
     IN USHORT KeyIndex;
     //Not support KMS2_KEYTYPE_RSA when
     //format is 00h(PKCS#1.2)
     IN BYTE KeyType;
     IN BYTE KeyVersion;
     IN DWORD KeyAttribute;
   }KeyInfo;

     // This is not used for KeyType RSA
     struct
     {
          IN BYTE Method;
          IN USHORT KeyCheckValueLength;
          IN BYTE* pKeyCheckValue;
     }Verification;

     // Optional. Used only for KeyType
     // 3DES_DUKPT
     struct
     {
          IN BYTE KSNLength;
          IN BYTE *pKSN;
     }DUKPT_DATA;

}KMS2_Info;
```

}CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA_VERSION_2;

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00, 0x01 or 0x02.

[IN] Protection.CipherKeySet

Specify the key set of the key used for decryption.

[IN] Protection.CipherKeyIndex

Specify the key index of the key used for decryption.

[IN] Certificate. Format (for PARA_VERSION_2)

Specify the format of ceriticate data.

The 0x00 indicate PKCS#1.2 format.

[IN] Certificate. Length

Specify the length of ceriticate data.

[IN] Certificate. pData

Point to a buffer containing the certificate data.

[IN] KMS2_Info.KeyOwner.Length

Specify the length of key owner name.

[IN] KMS2_Info.KeyOwner.Name

A buffer used to fill with the key owner name.

[IN] KMS2_Info.*KeyInfo.KeySet*

Specify the key set the new/update key belongs to.

[IN] KMS2_Info.*KeyInfo.KeyIndex*

Specify the key index the new/update key belongs to.

[IN] KMS2_Info.*KeyInfo.KeyType*

Specify the key type of the new/update key. **Note that if specified the key set does not exist, the key set will be created with this key type. If the specified key set exists already, the key type of the new/update key shall be**

identical with the key type of the key set.

[IN] KMS2_Info.*Key*Info.KeyVersion
Specify the key version.

[IN] KMS2_Info.*Key*Info.KeyAttribute
Specify the key attribute.

[IN] KMS2_Info.Verification.Method
This field is used only for the 3DES/3DES-DUKPT/AES/DES_DUKPT key types.
Specify the method for verifying if the key data after decryption is correct or not.
The default value 0x00 of verification method with 3DES key type, 3DES-DUKPT key type, AES key type or DES-DUKPT key type to calculate the key check values is as below.

- For 3DES, 3DES-DUKPT or DES-DUKPT key type,

key check values := take the result of **3DES(key, 8 bytes of zeros)** from left to right with the length specified by the parameter "KeyCheckValueLength".

- For AES key type,

key check values := take the result of **AES(key, 16 bytes of zeros)** from left to right with the length specified by the parameter "KeyCheckValueLength".
Note that the field is not used for the plaintext protection mode.

[IN] KMS2_Info.Verification.KeyCheckValueLength
Specify the length of key check values stored in Verification.pKeyCheckValue.
Note that the field is not used for the plaintext protection mode.

[IN] KMS2_Info.Verification.pKeyCheckValue
Point to a buffer containing the key check values.
Note that the field is not used for the plaintext protection

mode.

[IN] KMS2_Info.DUKPT_DATA.KSNLength

This field is used only for the 3DES-DUKPT or DES-DUKPT key type.

Specify the length of KSN (Key Serial Number). This value shall be 10.

[IN] KMS2_Info.DUKPT_DATA.pKSN

This field is used only for the 3DES-DUKPT or DES-DUKPT key type.

Point to a buffer containing KSN.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
//
typedef struct
{
      USHORT M_Len;
      BYTE *pModulus;
      USHORT E_Len;
      BYTE *Exponent;
}RSA_KEY;
RSA_KEY Key_RSA_1030_0002_1024P =
{
128,
“A7A93F7BB662791E41F43BBE3FE39C53
4DC372834EA454691094ABF77F39F470
CF32B08BC18065E8DF1B6A094634AB40
F61A0F233F776D07EA4169C7964C955E
2DF39C1C2E28F856B589A302C631C58B
25413B1F5A5B9C69FD81BF0636568697
4E1133BF85ADF031BCEF5C4BAA8EF347
F77C7E28CEE7D1E43CD7DC370009814F",
3,
“010001”,
};

void GenerateCertificate(CTOS_KMS2KEYWRITE_PARA *pPara,
BOOL IsSHA2, BYTE* CipherData, USHORT* pCipherLength)
{
      RSA_KEY *pRSAkey;
```

```
USHORT index;
BYTE HashLength;
BYTE HashData[32];
BYTE PlainData[512];
SHA_CTX stInfo;
BYTE Modulus[256];
BYTE Exponent[256];

pRSAkey = &Key_RSA_1030_0002_1024P;

memset(PlainData, 0xBB, sizeof(PlainData));

// 0, Header
PlainData[0] = 0x6A;
// 1, Format
if(pPara->Info.KeyType == KMS2_KEYTYPE_3DES ||
pPara->Info.KeyType == KMS2_KEYTYPE_AES)
{
      PlainData[1] = 0x20;
}
else if(pPara->Info.KeyType ==
KMS2_KEYTYPE_3DES_DUKPT)
{
      PlainData[1] = 0x21;
}
else if(pPara->Info.KeyType == KMS2_KEYTYPE_RSA)
{
      PlainData[1] = 0x22;
}
else
{
      return;
}
// 2, Hash Algorithm
if(IsSHA2)
{
      PlainData[2] = 0x01; // SHA2
}
else
{
      PlainData[2] = 0x00; // SHA1
}
// 3 to 22, Owner
memset(&PlainData[3], 0x00, 20); // Current App as
the owner
// 23 to 24, KeySet
PlainData[23] = (BYTE)(pPara->Info.KeySet >> 8);
PlainData[24] = (BYTE)(pPara->Info.KeySet & 0xFF);
// 25 to 26, KeyIndex
PlainData[25] = (BYTE)(pPara->Info.KeyIndex >> 8);
PlainData[26] = (BYTE)(pPara->Info.KeyIndex &
0xFF);
// 27, KeyType
```

```
PlainData[27] = pPara->Info.KeyType;
// 28, KeyVersion
PlainData[28] = pPara->Info.KeyVersion;
// 29 to 32, KeyAttribute
PlainData[29] = (BYTE)(pPara->Info.KeyAttribute >>
24);
PlainData[30] = (BYTE)(pPara->Info.KeyAttribute >>
16);
PlainData[31] = (BYTE)(pPara->Info.KeyAttribute >>
8);
PlainData[32] = (BYTE)(pPara->Info.KeyAttribute &
0xFF);
// 33 to 34, KeyLength
PlainData[33] = (BYTE)(pPara->Value.KeyLength >>
8);
PlainData[34] = (BYTE)(pPara->Value.KeyLength &
0xFF);
// 35 to 50/58. KeyData
memcpy(&PlainData[35], pPara->Value.pKeyData,
pPara->Value.KeyLength);
index = 35 + pPara->Value.KeyLength;

if(pPara->Info.KeyType == KMS2_KEYTYPE_3DES_DUKPT)
{
      // KSN
      memcpy(&PlainData[index], pPara-
      >DUKPT_DATA.pKSN, pPara-
      >DUKPT_DATA.KSNLength);
      index += pPara->DUKPT_DATA.KSNLength;
}
// CheckValueLength
PlainData[index++] = pPara-
>Verification.KeyCheckValueLength;
// CheckValue
memcpy(&PlainData[index], pPara-
>Verification.pKeyCheckValue, pPara-
>Verification.KeyCheckValueLength);
index += pPara->Verification.KeyCheckValueLength;
// Padding with 0xBB
// => Done in the initialization of PlainData
// Hash Data
if(IsSHA2)
{
      sha256_ctx ctx;

      HashLength = 32;
      SHA256_Init(&ctx);
       SHA256_Update(&ctx, &PlainData[1], pRSAkey-
       >M_Len - 1 - 1 - HashLength);
      SHA256_Final(&ctx, HashData);
}
else
{
```

```
            HashLength = 20;
            CTOS_SHA1Init(&stInfo);
            CTOS_SHA1Update(&stInfo, &PlainData[1],
            pRSAkey->M_Len - 1 - 1 - HashLength);
            CTOS_SHA1Final(HashData, &stInfo);
      }
      memcpy(&PlainData[pRSAkey->M_Len - 1 - HashLength],
      HashData, HashLength);
      // Tail
      PlainData[pRSAkey->M_Len - 1] = 0xBC;

      Pack((BYTE*)pRSAkey->pModulus, pRSAkey->M_Len * 2,
      Modulus);
      Pack((BYTE*)pRSAkey->Exponent, pRSAkey->E_Len * 2,
      Exponent);

      CTOS_RSA(Modulus, pRSAkey->M_Len,
            Exponent, pRSAkey->E_Len, PlainData,
      CipherData);

      *pCipherLength = pRSAkey->M_Len;
}


BYTE const Key_3DES_1002_0001[] = "3DES_1002_0001_0";
CTOS_KMS2KEYWRITE_PARA para;
BYTE CCode[8];
BYTE Certifcate[256];
USHORT CertificateLength;
CTOS_KMS2KEYWRITEBYCERTIFICATE_PARA CertPara;
USHORT ret;
BYTE str[17];
USHORT KeySet;
USHORT KeyIndex;
BYTE Zero[8];


KeySet = 0x1002;
KeyIndex = 0x0001;

memset(&para, 0x00, sizeof(CTOS_KMS2KEYWRITE_PARA));
para.Version = 0x01;
para.Info.KeySet = KeySet;
para.Info.KeyIndex = KeyIndex;
para.Info.KeyType = KMS2_KEYTYPE_3DES;
para.Info.KeyVersion = 0x01;
para.Info.KeyAttribute = KMS2_KEYATTRIBUTE_PIN |
KMS2_KEYATTRIBUTE_ENCRYPT | KMS2_KEYATTRIBUTE_MAC;
para.Value.KeyLength = 16;
para.Value.pKeyData = (BYTE*)Key_3DES_1002_0001;
para.Verification.KeyCheckValueLength = 3;
para.Verification.pKeyCheckValue = CCode;
```

```
memset(Zero, 0x00, 8);
CTOS_DES(d_ENCRYPTION, (BYTE*)Key_3DES_1002_0001, 16,
Zero, 8, CCode);

GenerateCertificate(&para, FALSE, Certifcate,
&CertificateLength);

CertPara.Version = 0x01;
CertPara.Protection.CipherKeySet = 0x1030;
CertPara.Protection.CipherKeyIndex = 0x0001;
CertPara.Certificate.Length = CertificateLength;
CertPara.Certificate.pData = Certifcate;

ret = CTOS_KMS2KeyWriteByCertificate(&CertPara);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2PINGet

```
USHORT CTOS_KMS2PINGet(CTOS_KMS2PINGET_PARA *pPinGetPara);
```

Description This function is used to get encryted PIN block. It will display a simple user interface to notify the user to enter the PIN. The plaintext input PIN block will be encrypted by the specified key. If the field *SK_Length* is 0x00, this function will directly encrypt the data with specified working key. If the field *SK_Length* is not 0x00, this function will encrypt the data with the session key (SK), which is retrieved by decrypting the field *pSK* with specified working key.

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            IN BYTE BlockType;
            IN BYTE PINDigitMaxLength;
            IN BYTE PINDigitMinLength;
      }PIN_Info;

      struct
      {
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;
            IN BYTE    CipherMethod;

            // This is used for KeyType
            // KMS2_KEYTYPE_3DES/KMS2_KEYTYPE_AES
            // If SK_Length is 0, SK will not be
            // calculated and used.
            IN BYTE    SK_Length;
            IN BYTE* pSK;
      }Protection;

      struct
      {
            // This is used for PAN if BlockType is  //
            KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.
            // Ths is used for terminal pseudo random
            // if BlockType  is
  // KMS2_PINBLOCKTYPE_ISBAN_REVERSIBLE_PIN_4B_MODE.
            // This is used for PK if BlockType   is
// KMS2_PINBLOCKTYPE_ISBAN_IRREVERSIBLE_PIN_4B_MODE.
            IN BYTE InLength;
            IN BYTE* pInData;
```

```
        }AdditionalData;

        // This is used for KeyType
        // KMS2_KEYTYPE_3DES_DUKPT/DES_DUKPT
        struct
        {
              IN BOOL IsUseCurrentKey;
        }DUKPT_PARA;

        struct
        {
              INOUT USHORT EncryptedBlockLength;
              OUT   BYTE* pEncryptedBlock;
              OUT BYTE PINDigitActualLength;
        }PINOutput;

        struct
        {
              IN DWORD Timeout;
              IN BYTE AsteriskPositionX;
              IN BYTE AsteriskPositionY;
              IN BYTE NULLPIN;
              IN int (*piTestCancel)(void);
        }Control;

}CTOS_KMS2PINGET_PARA;

------------------------------------------------------

typedef struct
{
        // Should be 0x02
        IN BYTE Version;

        struct
        {
              IN BYTE BlockType;
              IN BYTE PINDigitMaxLength;
              IN BYTE PINDigitMinLength;
        }PIN_Info;

        struct
        {
              IN USHORT CipherKeySet;
              IN USHORT CipherKeyIndex;
              IN BYTE    CipherMethod;

              // This is used for KeyType is
              // KMS2_KEYTYPE_3DES/KMS2_KEYTYPE_AES
              // If SK_Length is 0, SK will not be
              // calculated and used.
              IN BYTE    SK_Length;
              IN BYTE* pSK;
        }Protection;
```

```
        struct
        {
              // This is used for PAN if BlockType is
              // KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.
              // Ths is used for terminal pseudo random
              // if BlockType  is
KMS2_PINBLOCKTYPE_ISBAN_REVERSIBLE_PIN_4B_MODE.
              // This is used for PK if BlockType   is
KMS2_PINBLOCKTYPE_ISBAN_IRREVERSIBLE_PIN_4B_MODE.
              IN BYTE InLength;
              IN BYTE* pInData;

        }AdditionalData;

        // This is used for KeyType is
       // KMS2_KEYTYPE_3DES_DUKPT/DES_DUKPT
        struct
        {
              IN BOOL IsUseCurrentKey;
        }DUKPT_PARA;

        struct
        {
              INOUT USHORT EncryptedBlockLength;
              OUT   BYTE* pEncryptedBlock;
              OUT BYTE PINDigitActualLength;
        }PINOutput;

        struct
        {
              IN DWORD Timeout;
              BYTE NULLPIN;
        }Control;

        struct
        {
              void (*OnGetPINDigit)(BYTE NoDigits);
              void (*OnGetPINCancel)(void);
              void (*OnGetPINBackspace)(BYTE NoDigits);

        }EventFunction;

}CTOS_KMS2PINGET_PARA_VERSION_2;
```

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00 or 0x01.

[IN] *PIN_Info.BlockType*

Specify the type/format of PIN block.

- ANSI X9.8 ISO-0 format

  KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0 (0x00)

[IN] *PIN_Info.PINDigitMaxLength*

Specifiy the maximum number of PIN digits. This value ranges from 4 to 12.

[IN] *PIN_Info.PINDigitMinLength*

Specifiy the minimum number of PIN digits. This value ranges from 4 to 12.

[IN] *Protection.CipherKeySet*

Specify the key set of the working key used for PIN block encryption.

[IN] *Protection.CipherKeyIndex*

Specify the key index of the working key used for PIN block encryption.

[IN] *Protection.CipherMethod*

Specify which method is used for PIN block encryption.

- ECB mode

  KMS2_PINCIHERMETHOD_ECB (0x00)

[IN] *Protection.SK_Length*

This field is used only for the 3DES or AES key type.

Specify the length of session key.

For 3DES key type, this value shall be 0, 16, or 24.

For AES key type, this value shall be 0 or 16.

[IN] *Protection.pSK*

This field is used only for the 3DES or AES key type.

Point to a buffer containing the ciphered session key. The ciphered session key is encrypted by the working key with **3DES** or **AES** cipher operation in ECB mode. If the attribute of the working key is set with KMS2_KEYATTRIBUTE_SK_ENCRYPT, the ciphered session key is encrypted by the working key with **3DES**$^{-1}$ or **AES**$^{-1}$ cipher operation in ECB mode.

[IN] *AdditionalData.InLength*

This field is used as the length of PAN if BlockType is KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.

Specify the length of additional input data pointed by pInData.

[IN] *AdditionalData.pInData*

This field is used as the PAN data if BlockType is KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.

Point to a buffer containing the additional input data.

Note that for PAN data, it shall also contain the last check digit.

[IN] *DUKPT_PARA. IsUseCurrentKey*

This field is used only for the key type KMS2_KEYTYPE_3DES_DUKPT or KMS2_KEYTYPE_DES_DUKPT.

Indicate whether to increase the KSN and generate the session key or not. If this field is TRUE, the KSN won't be increased and session key won't be re-generated, but it is required the session key already generated before.

[INOUT] *PINOutput.EncryptedBlockLength*

Specify the size of the buffer pointed by *pEncryptedBlock* and return the actual length of encrypted PIN block.

[OUT] *PINOutput.pEncryptedBlock*

Point to a buffer used to retrieve the encrypted PIN block.

[OUT] *PINOutput.PINDigitActualLength*

Return the actual number of PIN digits.

[IN] *Control.TimeOut*

Waiting Time in seconds for PIN entry (0 means infinite)

[IN] *Control.AsteriskPositionX [Version 1]*

***Starting* column number of Asterisk.**

[IN] *Control.AsteriskPositionY [Version 1]*

***Starting* row number of Asterisk.**

[IN] *Control.NULLPIN*

Indicate if the function will accept ENTER and return if no PIN is typed.

[IN] *Control.piTestCancel [Version 1]*

Point to a callback function which is called during PIN entering. If the returning value of this function is non-zero, the getting PIN action will be aborted.

[IN] *EventFunction.OnGetPINDigit [Version 2]*

Point to a callback function which is called during PIN entering. The input value "NoDigits" indicates how many PIN digits the user has already entered.

[IN] *EventFunction.OnGetPINCancel [Version 2]*

Point to a callback function which is called when the user presses the CANCEL button during PIN entering.

[IN] *EventFunction.OnGetPINBackSpace [Version 2]*

Point to a callback function which is called when the user presses the BACKSPACE button during PIN entering. The number of PIN digits that the user entered will be decrease with 1. The input value "NoDigits" indicates how many PIN digits remains.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
BYTE const Key_3DES_1000_0001[] = "3DES_1000_0001_0";
BYTE const TestPAN[] = "4067111122223333";
BYTE const TestLine1Msg[] = "Require 123456";
BYTE const TestLine2Msg[] = "Enter PIN :";
BYTE const TestProcMsg[] = "Get PIN OK";
CTOS_KMS2PINGET_PARA para;
```

```
USHORT ret;
BYTE PINBlock[16];
BYTE *pCipherKey;
BYTE CipherKeyLength;
BYTE str[17];

pCipherKey = (BYTE*)Key_3DES_1000_0001;
CipherKeyLength = 16;

memset(&para, 0x00, sizeof(CTOS_KMS2PINGET_PARA));
para.Version = 0x01;
para.PIN_Info.BlockType =
KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0;
para.PIN_Info.PINDigitMinLength = 4;
para.PIN_Info.PINDigitMaxLength = 12;
para.Protection.CipherKeySet = 0x1000;
para.Protection.CipherKeyIndex = 0x0001;
para.Protection.CipherMethod = KMS2_PINCIHERMETHOD_ECB;
para.Protection.SK_Length = 0;
para.AdditionalData.InLength = 16;
para.AdditionalData.pInData = (BYTE*)TestPAN;
para.PINOutput.EncryptedBlockLength = 8;
para.PINOutput.pEncryptedBlock = PINBlock;
para.Control.Timeout = 10;
para.Control.pLine1Msg = (BYTE*)TestLine1Msg;
para.Control.pLine2Msg = (BYTE*)TestLine2Msg;
para.Control.pProcessingMsg = (BYTE*)TestProcMsg;
para.Control.NULLPIN = FALSE;
para.Control.piTestCancel = NULL;

ret = CTOS_KMS2PINGet(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2DataEncrypt

```
USHORT CTOS_KMS2DataEncrypt(CTOS_KMS2DATAENCRYPT_PARA *pDataEncPara);
```

Description

This function is used to perform data encryption. If the field *SK_Length* is 0x00, this function will directly encrypt the data with specified working key. If the field *SK_Length* is not 0x00, this function will encrypt the data with the session key (SK), which is retrieved by decrypting the field *pSK* with specified working key.

For 3DES-DUKPT key type, the value of CipherMethod can be CBC mode, CBC_POUND_x or ECB_POUND_x. The variable x is used to indicate which key will be used:

1: PIN Encryption

2: Message Authentication, request or both ways

3: Data Encryption, request or both ways

4: Message Authentication, response

5: Data Encryption, response

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;
            IN BYTE    CipherMethod;

            // This is used for KeyType
            // KMS2_KEYTYPE_3DES or KMS2_KEYTYPE_AES
            // If SK_Length is 0, SK will not be
            // calculated and used.
            IN BYTE    SK_Length;
            IN BYTE* pSK;

      }Protection;

      // This is used only for KeyType
      // KMS2_KEYTYPE_3DES_DUKPT/DES_DUKPT
      struct
      {
            IN BOOL IsUseCurrentKey;
```

```
        }DUKPT_PARA;

        struct
        {
              IN USHORT Length;
              IN BYTE* pData;

              IN USHORT ICVLength;
              IN BYTE* pICV;
        }Input;

        struct
        {
              OUT USHORT Length;
              OUT BYTE* pData;
        }Output;

}CTOS_KMS2DATAENCRYPT_PARA;
```

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00 or 0x01.

[IN] *Protection.CipherKeySet*

Specify the key set of the working key used for encryption.

[IN] *Protection.CipherKeyIndex*

Specify the key index of the working key used for encryption.

[IN] *Protection.CipherMethod*

Specify which method is used for data encryption.

- ECB mode

KMS2_DATAENCRYPTCIHERMETHOD_ECB (0x00)

- CBC mode

KMS2_DATAENCRYPTCIHERMETHOD_CBC (0x01)

- CBC_POUND_1

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_CBC_POUND_1 (0x11)

- CBC_POUND_2

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_CBC_POUND_2 (0x12)

- CBC_POUND_3

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_CBC_POUND_3 (0x13)

- CBC_POUND_4

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_CBC_POUND_4 (0x14)

- CBC_POUND_5

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_CBC_POUND_5 (0x15)

- ECB_POUND_1

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_ECB_POUND_1 (0x16)

- ECB_POUND_2

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_ECB_POUND_2 (0x17)

- ECB_POUND_3

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_ECB_POUND_3 (0x18)

- ECB_POUND_4

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_ECB_POUND_4 (0x19)

- ECB_POUND_5

KMS2_DATAENCRYPTCIPHERMETHOD_3DUKPT_ECB_POUND_5 (0x1A)

[IN] *Protection.SK_Length*

This field is used only for the 3DES or AES key type.

Specify the length of session key.

For 3DES key type, this value shall be 0, 16, or 24.

For AES key type, this value shall be 0 or 16.

[IN] *Protection.pSK*

This field is used only for the 3DES or AES key type.

Point to a buffer containing the ciphered session key. The ciphered session key is encrypted by the working key with **3DES** or **AES** cipher operation in ECB mode. If the attribute of the working key is set with KMS2_KEYATTRIBUTE_SK_ENCRYPT, the ciphered session key is encrypted by the working key with **3DES**$^{-1}$ or **AES**$^{-1}$ cipher operation in ECB mode.

[IN] *DUKPT_PARA. IsUseCurrentKey*

This field is used only for the key type KMS2_KEYTYPE_3DES_DUKPT or KMS2_KEYTYPE_DES_DUKPT.

Indicate whether to increase the KSN and generate the session key or not. If this field is TRUE, the KSN won't be increased and session key won't be re-generated, but it is required the session key already generated before.

[IN] *Input.Length*

Specify the length of input data.

If the *CipherMethod* is ECB mode, this value shall be multiple of 8 if the key type is 3DES, 3DES-DUKPT or DES-DUKPT or shall be multiple of 16 if the key type is AES.
If the *CipherMethod* is CBC mode, the padding will pad 0x00 to the tail of input data to be multiple of 8 or multiple of 16 depending on the key type.

[IN] *Input.pData*
Point to a buffer containing the input data.

[IN] *Input.ICVLength*
This field is used only for the *CipherMethod* being CBC mode.
Specify the length of Initial Chaining Vector (ICV).

[IN] *Input.pICV*
This field is used only for the *CipherMethod* being CBC mode.
Point to a buffer containing the data of Initial Chaining Vector.

[OUT] *Output.Length*
Indicate the length of the output data.

[OUT] *Output.pData*
Point to a buffer used to retrieve the output data.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
//
BYTE const Key_3DES_1000_0001[] = "3DES_1000_0001_0";
USHORT ret;
CTOS_KMS2DATAENCRYPT_PARA para;
BYTE *pCihperKey;
BYTE CihperKeyLength;
BYTE plaindata[256];
BYTE cipherdata[256];
BYTE str[17];

pCihperKey = (BYTE*)Key_3DES_1000_0001;
CihperKeyLength = 16;

memset(&para, 0x00, sizeof(CTOS_KMS2DATAENCRYPT_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x1000;
para.Protection.CipherKeyIndex = 0x0001;
```

```
para.Protection.CipherMethod =
KMS2_DATAENCRYPTCIHERMETHOD_ECB;
para.Protection.SK_Length = 0;

memset(plaindata, 0x00, sizeof(plaindata));
para.Input.Length = sizeof(plaindata);
para.Input.pData = plaindata;
para.Output.pData = cipherdata;

ret = CTOS_KMS2DataEncrypt(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2MAC

```
USHORT CTOS_KMS2MAC(CTOS_KMS2MAC_PARA *pMacPara);
```

Description

This function is used to calculate MAC of the input data.. If the field *SK_Length* is 0x00, it will perform MAC calculation directly with the specified working key. If the field *SK_Length* is not 0x00, the MAC of input data will be calculated with the session key (SK), which is retrieved by decrypting the field *pSK* with the specified working key.
For 3DES-DUKPT key type, the value of CipherMethoud is required to be KMS2_MACMETHOD_X9_19.

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;
            IN BYTE    CipherMethod;

            // This is used for KeyType
            // KMS2_KEYTYPE_3DES/KMS2_KEYTYPE_AES
            // If SK_Length is 0, SK will not be
            // calculated and used.
            IN BYTE    SK_Length;
            IN BYTE* pSK;
      }Protection;

      struct
      {
            BYTE Length;
            BYTE* pData;
      }ICV;

      // This is used for KeyType
      // KMS2_KEYTYPE_3DES_DUKPT/DES_DUKPT
      struct
      {
            IN BOOL IsUseCurrentKey;
      }DUKPT_PARA;

      struct
```

```
        {
              IN USHORT Length;
              IN BYTE* pData;
        }Input;

        struct
        {
              OUT USHORT Length;
              OUT BYTE* pData;
        }Output;

}CTOS_KMS2MAC_PARA;
```

Parameters

[IN] Version

Structure Format Version. It shall be 0x00 or 0x01.

[IN] Protection.CipherKeySet

Specify the key set of the working key used for MAC calculation.

[IN] Protection.CipherKeyIndex

Specify the key index of the working key used for MAC calculation.

[IN] Protection.CipherMethod

Specify which method is used for MAC calculation.

- CBC mode

KMS2_MACMETHOD_CBC (0x00)

- ANSI X9.19 Retail MAC

KMS2_MACMETHOD_X9_19 (0x01)

[IN] Protection.SK_Length

This field is used only for the 3DES or AES key type.

Specify the length of session key.

For 3DES key type, this value shall be 0, 16, or 24.

For AES key type, this value shall be 0 or 16.

[IN] Protection.pSK

This field is used only for the 3DES or AES key type.

Point to a buffer containing the ciphered session key. The

ciphered session key is encrypted by the working key with 3DES or AES cipher operation in ECB mode. If the attribute of the working key is set with KMS2_KEYATTRIBUTE_SK_ENCRYPT, the ciphered session key is encrypted by the working key with 3DES-1 or AES-1 cipher operation in ECB mode.

[IN] ICV.Length
Specify the length if Initial Chaining Vector (ICV).

[IN] ICV.pData
Pointer to a buffer containing the Initial Chaining Vector.

[IN] DUKPT_PARA. IsUseCurrentKey
This field is used only for the key type KMS2_KEYTYPE_3DES_DUKPT.
Indicate whether to increase the KSN and generate the session key or not. If this field is TRUE, the KSN won't be increased and session key won't be re-generated, but it is required the session key already generated before.

[IN] Input.Length
Specify the length of input data.
The padding will pad 0x00 to the tail of input data to be multiple of 8 or multiple of 16 depending on the key type.

[IN] Input.pData
Point to a buffer containing the input data.

[OUT] Output.Length
Indicate the length of the output MAC data.

[OUT] Output.pData
Point to a buffer used to retrieve the output MAC data.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
CTOS_KMS2MAC_PARA para;
USHORT ret;
```

```
BYTE *pCipherKey;
BYTE CipherKeyLength;
BYTE Zero[8];
BYTE plaindata[255];
BYTE macdata[8];
BYTE str[17];

memset(Zero, 0x00, sizeof(Zero));
memset(plaindata, 0x55, sizeof(plaindata));

pCipherKey = (BYTE*)Key_3DES_1001_0001;
CipherKeyLength = 16;

memset(&para, 0x00, sizeof(CTOS_KMS2MAC_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x1001;
para.Protection.CipherKeyIndex = 0x0001;
para.Protection.CipherMethod = KMS2_MACMETHOD_CBC;
para.ICV.Length = 8;
para.ICV.pData = Zero;
para.Input.Length = sizeof(plaindata);
para.Input.pData = plaindata;
para.Output.pData = macdata;

ret = CTOS_KMS2MAC(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2RSAEncrypt

```
USHORT CTOS_KMS2RSAEncrypt(CTOS_KMS2RSAENCRYPT_PARA
*pRSAEncryptPara);
```

Description This function is used to perform RSA encryption.

```
typedef struct
{
      // Should be 0x00 or 0x01
      IN BYTE Version;

      struct
      {
            IN USHORT CipherKeySet;
            IN USHORT CipherKeyIndex;
            IN BYTE    CipherMethod;

      }Protection;

      struct
      {
            IN USHORT Length;
            IN BYTE* pData;
      }Input;

      struct
      {
            OUT USHORT Length;
            OUT BYTE* pData;
      }Output;

}CTOS_KMS2RSAENCRYPT_PARA;
```

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00 or 0x01.

[IN] *Protection.CipherKeySet*

Specify the key set of the working key used for encryption.

[IN] *Protection.CipherKeyIndex*

Specify the key index of the working key used for encryption.

[IN] *Protection.CipherMethod*

Specify which method is used for data encryption.

- Default

KMS2_RSAENCRYPTCIHERMETHOD_DEFAULT (0x00)

[IN] *Input.Length*
Specify the length of input data.

[IN] *Input.pData*
Point to a buffer containing the input data.

[OUT] *Output.Length*
Indicate the length of the output data.

[OUT] *Output.pData*
Point to a buffer used to retrieve the output data.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
//
typedef struct
{
      USHORT M_Len;
      BYTE *pModulus;
      USHORT E_Len;
      BYTE *Exponent;
}RSA_KEY;
RSA_KEY Key_RSA_1030_0001_1024R =
{
128,
“A7A93F7BB662791E41F43BBE3FE39C53
4DC372834EA454691094ABF77F39F470
CF32B08BC18065E8DF1B6A094634AB40
F61A0F233F776D07EA4169C7964C955E
2DF39C1C2E28F856B589A302C631C58B
25413B1F5A5B9C69FD81BF0636568697
4E1133BF85ADF031BCEF5C4BAA8EF347
F77C7E28CEE7D1E43CD7DC370009814F",
128,
“03BECC243F56C3CDE13B4B7A5C830122
FB41BA752474974E2188B0AEBEB0D4BD
1063D97DC5BCD089FB31E9947B7501BE
59C10B45864D6CAA18998D7B5FE8260E
```

```
0347320750836C2932967C1E8F9E34A4
1D3FFF56C86C27EF9783C9C3C488C9E4
620441C327DC76632C94A9A0EB3E3704
B045965F2D961AF80C3EDD9B04732B81”,
};
CTOS_KMS2RSAENCRYPT_PARA para;
USHORT ret;
BYTE str[17];
BYTE Modulus[256];
BYTE Exponent[256];
RSA_KEY *pRSAkey;
USHORT CipherKeyLength;
BYTE plaindata[256];
BYTE cipherdata[256];

memset(plaindata, 0xBB, sizeof(plaindata));
plaindata[0] = 0x6A;
memset(cipherdata, 0x00, sizeof(cipherdata));
memset(matchdata, 0x00, sizeof(matchdata));

pRSAkey = &Key_RSA_1030_0001_1024R;
CipherKeyLength = pRSAkey->M_Len;
Pack((BYTE*)pRSAkey->pModulus, pRSAkey->M_Len * 2,
Modulus);
Pack((BYTE*)pRSAkey->Exponent, pRSAkey->E_Len * 2,
Exponent);

memset(&para, 0x00, sizeof(CTOS_KMS2RSAENCRYPT_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x1030;
para.Protection.CipherKeyIndex = 0x0001;
para.Protection.CipherMethod =
KMS2_RSAENCRYPTCIHERMETHOD_DEFAULT;
para.Input.Length = CipherKeyLength;
para.Input.pData = plaindata;
para.Output.pData = cipherdata;

ret = CTOS_KMS2RSAEncrypt(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2KeyGetInfo

```
USHORT CTOS_KMS2KeyGetInfo(IN CTOS_KMS2KEYGETINFO_PARA
*pKeyGetInfoPara);
```

Description

Get information of the specified working key.

```
typedef struct
{
    // Should be 0x00 or 0x01
    IN BYTE Version;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
        // Only used for KeyType 3DES/3DES-DUKPT/AES/DES-
DUKPT
        IN BYTE CVLen;

        // Only used for KeyType RSA
        IN BYTE HashAlgorithm;
    }Input;

    struct
    {
        OUT BYTE KeyType;
        OUT BYTE KeyVersion;
        OUT DWORD KeyAttribute;
        OUT USHORT KeyLength;

        // Only used for KeyType 3DES/3DES-DUKPT/AES
        OUT BYTE* pCV;

        // Only used for KeyType RSA
        OUT USHORT KeyExponentLength;
```

```
                // Only used for KeyType RSA
                // Calculated with the following input data in order:
                //          Modulus Length - 2 bytes, MSB to LSB
                //          Modulus
                //          Exponent Length - 2 bytes, MSB to LSB
                //          Exponent
                OUT BYTE* pHash;
            }Output;

        }CTOS_KMS2KEYGETINFO_PARA;
```

Parameters

[IN] *Version*

Structure Format Version. It shall be 0x00 or 0x01.

[IN] *Input.KeySet*

Used to indicate which key set it belong to.

[IN] *Input.KeyIndex*

Specify its index in the key set.

[IN] *Input.CVLen*

Specify the length of key check value to be returned.

Note that this field is used only for the key type 3DES, 3DES-DUKP, AES or DES-DUKPT.

[IN] *Input.HashAlgorithm*

Specify the hash algorithm.

- SHA1

  KMS2_KEYCERTIFICATEGENERATECIHERMETHOD_DEFAULT_WITH_SHA1 (0x00)

- SHA256

  KMS2_KEYCERTIFICATEGENERATECIHERMETHOD_DEFAULT_WITH_SHA2 (0x01)

Note that this field is used only for the key type RSA.

[OUT] *Output.KeyType*

Return the key type of the specified key.

[OUT] *Output*.*KeyVersion*

Return the key version of the specified key.

[OUT] *Output*.*KeyAttribute*

Return the key attribute of the specified key.

[OUT] *Output*.*KeyLength*

Return the key length of the specified key.

[OUT] *Output*.*pCV*

Point to a buffer used to retrieve the key check value.

Note that this field is used only for the key type 3DES, 3DES-DUKP, AES or DES-DUKPT.

[OUT] *Output*.*KeyExponentLength*

Return the exponent length of the specified RSA key.

Note that this field is used only for the key type RSA.

[OUT] *Output*. *pHash*

Point to a buffer used to retrieve the hash data generated for the specified RSA key.

The data used to calculate the hash is as below in order:

- ***Modulus Length - 2 bytes, MSB to LSB***
- ***Modulus***
- ***Exponent Length - 2 bytes, MSB to LSB***
- ***Exponent***

Note that this field is used only for the key type RSA.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
//
BYTE const Key_3DES_1000_0001[] = "3DES_1000_0001_0";
CTOS_KMS2KEYGETINFO_PARA para;
```

```
USHORT ret;
BYTE *pCipherKey;
BYTE CipherKeyLength;
BYTE CCode[8];
BYTE Hash[32];
BYTE str[17];

pCipherKey = (BYTE*)Key_3DES_1000_0001;
CipherKeyLength = 16;

memset(&para, 0x00, sizeof(CTOS_KMS2KEYGETINFO_PARA));
para.Version = 0x01;
para.Input.KeySet = 0x1000;
para.Input.KeyIndex = 0x0001;
para.Input.CVLen = 3;
para.Input.HashAlgorithm = 0x00;
para.Output.pCV = CCode;
para.Output.pHash = Hash;

ret = CTOS_KMS2KeyGetInfo(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      return;
}
```

CTOS_KMS2DUKPTGetKSN

```
USHORT CTOS_KMS2DUKPTGetKSN(IN USHORT KeySet, IN USHORT KeyIndex,
OUT BYTE* pKSN, INOUT BYTE* KSNLen);
```

Description Get current KSN of the specified DUKPT key.

Parameters

[IN] *KeySet*

Used to indicate which key set it belong to.

[IN] *KeyIndex*

Specify its index in the key set.

[OUT] *pKSN*

Pointer to a buffer used to retrieve KSN.

[INOUT] *KSNLen*

Specify the size of the buffer pointed by *pKSN* and return the actual length of KSN.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
//You need to download a DUKPT key first before running
this sample code.

void main()
{
      USHORT rtn
      BYTE length;
      BYTE ksn[10];

      CTOS_KMS2Init();
      length = 10;
      rtn = CTOS_KMS2DUKPTGetKSN(0x1010, 0x0001, ksn,
      &length);
      if (rtn == d_OK)
            CTOS_LCDTPrintXY(1, 1, "Get KSN ok!");
      else
            CTOS_LCDTPrintXY(1, 1, "Get KSN failed!");
```

```
        while (1);
}
```

CTOS_KMS2UserDataWrite

```
USHORT CTOS_KMS2UserDataWrite(IN BOOL IsCommon, IN ULONG Offset, IN
BYTE *pData, IN USHORT usLen);
```

Description A secure free-usage memory space provided for user applications to store their sensitive data by themselves.

Parameters

[IN] *IsCommon*
Indicate to access common user data area or private user data area.
The space of common user data is 256K, while the space of each application's user data (private) is 64K.

[IN] *Offset*
Specify the offset of the storage to write the data. If the ulOffset is 0, it means the start of the storage.

[IN] *pData*
The buffer to write.

[IN] *usLen*
The data length in the baBuf.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
USHORT ret;
BYTE WriteData[1024];
BYTE str[17];

// Private
ret = CTOS_KMS2UserDataWrite(FALSE, 0, WriteData, 1024);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
```

```
        return;
    }
```

CTOS_KMS2UserDataRead

```
USHORT CTOS_KMS2UserDataRead(IN BOOL IsCommon, IN ULONG Offset, OUT
BYTE *pData, IN USHORT usLen);
```

Description Get the data from the secure free-usage memory space.

Parameters

[IN] *IsCommon*
Indicate to access common user data area or private user data area.
The space of common user data is 256K, while the space of each application's user data (private) is 64K.

[IN] *Offset*
Specify the offset of the storage to read the data. If the ulOffset is 0, it means the start of the storage.

[OUT] *pData*
The buffer to read.

[IN] *usLen*
The data length to read from the storage.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
USHORT ret;
BYTE ReadData[1024];
BYTE str[17];

// private
ret = CTOS_KMS2UserDataRead(FALSE, 0, ReadData, 1024);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
```

```
        return;
    }
```

CTOS_KMS2IntermediateKeyGenerate

```
USHORT
CTOS_KMS2IntermediateKeyGenerate(CTOS_KMS2INTERMEDIATEKEYGENERATE_PA
RA* pIntermediateKeyGeneratePara);
```

Description Generate intermediate key with specified operation.

```
typedef struct
{
    IN BYTE Version;
    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
    }Src1KeyID;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
    }Src2KeyID;

    struct
    {
        IN USHORT ID;
        IN USHORT Length;
        IN BYTE* pData;
    }Operation;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
        IN BYTE KeyType;
        IN DWORD KeyAttribute;
    }DestIntermediateKeyInfo;
```

}CTOS_KMS2INTERMEDIATEKEYGENERATE_PARA;

Parameters

[IN] *Version*

Should be 0x00 or 0x01

[IN] *Src1KeyID.KeySet*

Specify the key set of the working key used for this function.

Note.

Both Src1KeyID and Src2KeyID will be ignored when Operation.ID is KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *Src1KeyID.KeyIndex*

Specify the key index of the working key used for this function.

Note.

Both Src1KeyID and Src2KeyID will be ignored when Operation.ID is KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *Src2KeyID.KeySet*

Specify the key set of the working key used for this function.

Note.

If Src2KeyID.KeySet and Src2KeyID.KeyIndex are set to KMS2_NONE, Operation will be used to substitute Src2Key.

Note.

Both Src1KeyID and Src2KeyID will be ignored when Operation.ID is KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *Src2KeyID.KeyIndex*

Specify the key index of the working key used for this function.

Note.

If Src2KeyID.KeySet and Src2KeyID.KeyIndex are set to

KMS2_NONE, Operation will be used to substitute Src2Key.

Note.
Both Src1KeyID and Src2KeyID will be ignored when Operation.ID is KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *Operation.ID*

Specify the operation ID used for this function.

The operation ID are list as below:

KMS2_INTERMEDIATE_KEYGEN_OP_XOR
KMS2_INTERMEDIATE_KEYGEN_OP_DES_ENCRYPT
KMS2_INTERMEDIATE_KEYGEN_OP_DES_DECRYPT
KMS2_INTERMEDIATE_KEYGEN_OP_TAKE_LEFT
KMS2_INTERMEDIATE_KEYGEN_OP_TAKE_RIGHT
KMS2_INTERMEDIATE_KEYGEN_OP_COMBINE
KMS2_INTERMEDIATE_KEYGEN_OP_AES_ENCRYPT
KMS2_INTERMEDIATE_KEYGEN_OP_AES_DECRYPT
KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *Operation.Length*

Specify the length of operation data used for this function.

1. For XOR operation, it indicates how many bytes in Src1Key and Src2Key (Operation Data) will be calculated in this opersation.
2. For DES_ENCRYPT/DES_DECRYPT, it indicates how many bytes in Src1Key and Src2Key (Operation Data) will be calculated in this opersation.
3. For TAKE_LEFT/TAKE_RIGHT operation, it indicates how many bytes of Src1Key will be taked.
4. For GEN_RANDOM_KEYoperation, it indicates key length of requirement.

Note.
Operation length must not exceed a maximum of 32 bytes.

[IN] *Operation.pData*

Specify the operation data used for this function.

Note.
Operation.pData will be ignored when Operation.ID is KMS2_INTERMEDIATE_KEYGEN_OP_GEN_RANDOM_KEY

[IN] *DestIntermediateKeyInfo.KeySet*
Specify the key set of the destination intermediate key.

Note.
The key set must be KMS2_INTERMEDIATE_KEY_SET.

[IN] *DestIntermediateKeyInfo.KeyIndex*
Specify the key index of the destination intermediate key.

Note.
The range of key index must not exceed from 0x00 to 0x0F.

[IN] *DestIntermediateKeyInfo.KeyType*
Specify the key type of the destination intermediate key.

[IN] *DestIntermediateKeyInfo.KeyAttribute*
Specify the KeyAttribute to the destination intermediate key. The KeyAttribute is bit mask as use in normal KMS key. Besides, the destination intermediate key will be set with KMS2_KEYATTRIBUTE_INTERMEDIATE automatically.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
void KMS2_IntermediateKeyGenerate(void)
{
   CTOS_KMS2INTERMEDIATEKEYGENERATE_PARA para;
   BYTE Data[16];
   USHORT usRet;
   BYTE str[50];
   BYTE key;
   BYTE KeySet;
```

```
    BYTE KeyIndex;

    CTOS_LCDTClearDisplay();
    CTOS_LCDTPrintXY(1, 1, "Generate Key");

    // Intermediate Key Generate
    memset(&para, 0x00,
sizeof(CTOS_KMS2INTERMEDIATEKEYGENERATE_PARA));
    para.Version = 0x00;
    // Source1 Key Set
    para.Src1KeyID.KeySet = 0x1000;
    // Source1 Key Index
    para.Src1KeyID.KeyIndex = 0x0001;
    // Src2KeyID key set and key index is 0, Src2KeyID this field is not use
    para.Src2KeyID.KeySet = 0x0000;
    para.Src2KeyID.KeyIndex = 0x0000;

    // Operation is DES encrypt
    para.Operation.ID =
KMS2_INTERMEDIATE_KEYGEN_OP_DES_ENCRYPT;
    // DES encrypt length
    para.Operation.Length = 16;
    // Because Src2KeyID is not use, so must input data
    para.Operation.pData = Data;
    memcpy(Data,
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F\x10", 16);

    para.DestIntermediateKeyInfo.KeySet =
KMS2_INTERMEDIATE_KEY_SET;
    para.DestIntermediateKeyInfo.KeyIndex =
KMS2_INTERMEDIATE_KEY_4_INDEX;
    para.DestIntermediateKeyInfo.KeyType = KMS2_KEYTYPE_3DES;
    para.DestIntermediateKeyInfo.KeyAttribute = KMS2_KEYATTRIBUTE_PIN
| KMS2_KEYATTRIBUTE_ENCRYPT | KMS2_KEYATTRIBUTE_MAC;
    usRet = CTOS_KMS2IntermediateKeyGenerate(&para);
    if(usRet != d_OK)
    {
        memset(str, 0x00, sizeof(str));
        sprintf(str, "Fail:ret=%04X", usRet);
                    CTOS_LCDTPrintXY(1, 8, str);
```

```
        CTOS_KBDGet(&key);
        return;
    }

    CTOS_LCDTPrintXY(1, 2, "Generate Key OK");
    CTOS_KBDGet(&key);
    return;
}
```

CTOS_KMS2IntermediateKeyWrite

```
USHORT
CTOS_KMS2IntermediateKeyWrite(CTOS_KMS2INTERMEDIATEKEYWRITE_PARA*
pIntermediateKeyWritePara);
```

Description Write intermediate key into key storage with specified key set and key index.

```
typedef struct
{
    IN BYTE Version;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
    }IntermediateKeyID;

    struct
    {
        IN USHORT KeySet;
        IN USHORT KeyIndex;
        IN BYTE KeyType;
        IN BYTE KeyVersion;
        IN DWORD KeyAttribute;
    }DestKeyInfo;

}CTOS_KMS2INTERMEDIATEKEYWRITE_PARA;
```

Parameters

[IN] Version
Should be 0x00 or 0x01

[IN] IntermediateKeyID.KeySet
Specify the key set of intermediate key used for this function.

[IN] IntermediateKeyID.KeyIndex

Specify the key index of intermediate key used for this function.

[IN] DestKeyInfo.KeySet
Specify the key set of destination key.

[IN] DestKeyInfo.KeyIndex
Specify the key index of destination key.

[IN] DestKeyInfo.KeyType
Specify the key type of destination key.

[IN] DestKeyInfo.KeyVersion
Specify the key version of destination key.

[IN] DestKeyInfo.KeyAttribute
Specify the key attributeof destination key.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
void KMS2_WriteIntermediateKey(void)
{
   USHORT usRet;
   CTOS_KMS2INTERMEDIATEKEYWRITE_PARA para1;
   BYTE str[50];
   BYTE key;

   CTOS_LCDTClearDisplay();
   CTOS_LCDTPrintXY(1, 1, "WriteIntermediateKey");

   // Write Intermediate Key
   memset(&para1, 0x00,
sizeof(CTOS_KMS2INTERMEDIATEKEYWRITE_PARA));
   para1.Version = 0x00;
   para1.IntermediateKeyID.KeySet = KMS2_INTERMEDIATE_KEY_SET;
   para1.IntermediateKeyID.KeyIndex =
```

```
KMS2_INTERMEDIATE_KEY_0_INDEX;

    para1.DestKeyInfo.KeySet = 0x1000;
    para1.DestKeyInfo.KeyIndex = 0x0003;
    para1.DestKeyInfo.KeyType = KMS2_KEYTYPE_3DES;
    para1.DestKeyInfo.KeyAttribute = KMS2_KEYATTRIBUTE_ENCRYPT;
    para1.DestKeyInfo.KeyVersion = 0x01;

    usRet = CTOS_KMS2IntermediateKeyWrite(&para1);
    if(usRet != d_OK)
    {
        memset(str, 0x00, sizeof(str));
        sprintf(str, "Fail:ret=%04X", usRet);
                    CTOS_LCDTPrintXY(1, 8, str);
         CTOS_KBDGet(&key);
        return;
    }
    CTOS_LCDTPrintXY(1, 3, "Write Key OK");
    CTOS_KBDGet(&key);
    return;
}
```

CTOS_KMS2IntermediateKeyFlush

```
USHORT CTOS_KMS2IntermediateKeyFlush(IN USHORT KeySet, IN USHORT KeyIndex);
```

Description Flush the specified intermediate key.

Parameters

[IN] *KeySet*

Specify the key set of the intermediate key.

Note.

The key set must be KMS2_INTERMEDIATE_KEY_SET.

[IN] *KeyIndex*

Specify the key index of the intermediate key.

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2IntermediateKeyFlush(
      KMS2_INTERMEDIATE_KEY_SET,
      KMS2_INTERMEDIATE_KEY_0_INDEX);
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "Flush OK");
  else
      CTOS_LCDTPrintXY(1, 1, " Flush Failed");
  while (1);
}
```


CTOS_KMS2IntermediateKeyErase

```
USHORT CTOS_KMS2IntermediateKeyErase(void);
```

Description Erase all the intermediate keys.

Parameters None

Return Value

Constants	***Value***
d_OK	0000h
KMS2 error codes	29xxh

Example

```
void main()
{
  USHORT rtn;

  CTOS_KMS2Init();
  rtn = CTOS_KMS2IntermediateKeyErase();
  if (rtn == d_OK)
      CTOS_LCDTPrintXY(1, 1, "Erase OK");
  else
      CTOS_LCDTPrintXY(1, 1, "Erase Failed");
  while (1);
}
```

~ End ~

