CASTLES TECHNOLOGY

EFT-POS Terminal

Key Management System II

User Manual

Confidential

Version 2.02

June 2015

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

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.
1.00	December 04, 2014	1. Wording amended. 2. Add 3DUKPT_CBC_POUND_30 & 3DUKPT_CBC_POUND_31

		methods for CTOS_KMS2_DataEncrypt 3. Add section 2.4 By Key Block Binding Method 4. Add KMS2_PINCIHERMETHOD_EMV_OFFLIEN_PIN cihper method 5. Add key attribute “Consider Invalid Bits as Valid Bits during Key Value Unique Checking”. 6. Add the new error code “d_KMS2_PURPOSE_NOT_UNIQUE” 7. Add functions “CTOS_KMS2KeyWriteByTR31” and “CTOS_KMS2KeyWriteByTR31Ex”
1.01	December 19, 2014	1. Correct the common user data area from 256K to 64K 2. Correct the private user data area from 64 K to 16K
2.00	March 30, 2015	Rearrangement for this document.
2.01	June 01, 2015	More description for key storage.
2.02	June 03, 2015	1. Correct some wording. 2. Add section Appendix A: KMS-II Example.

Contents

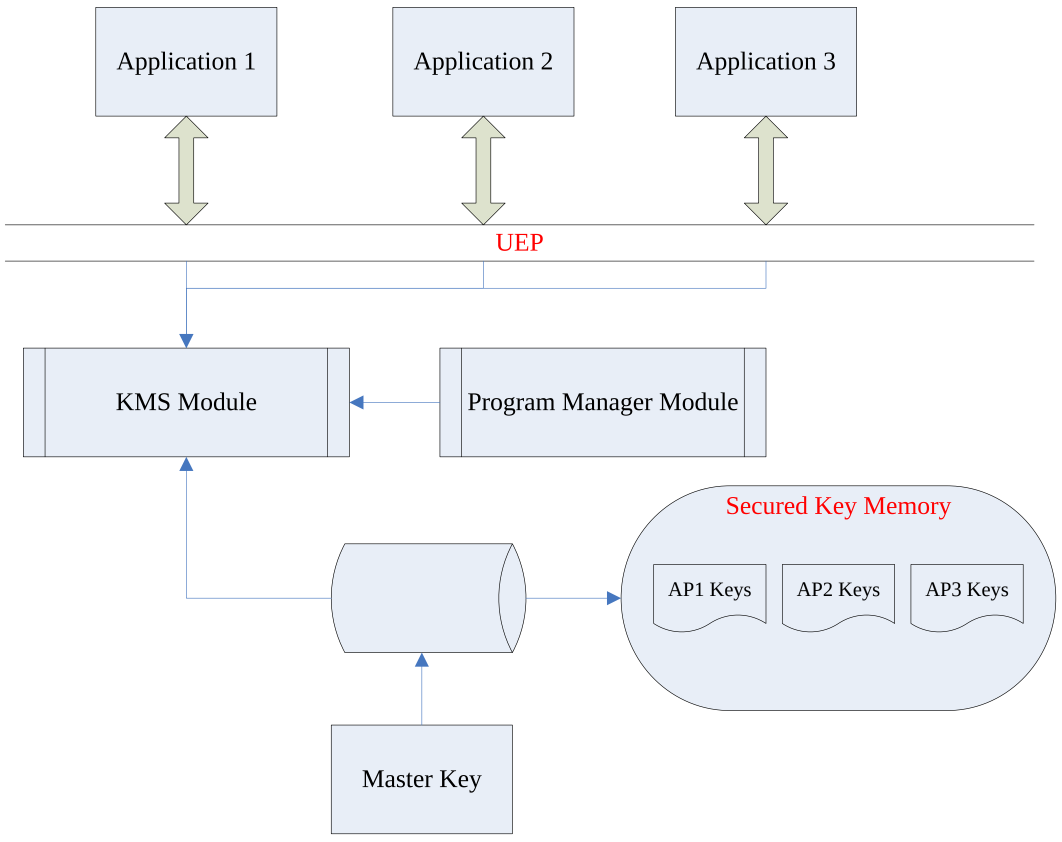
1. Introduction

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

Each key stored in the key storage has below main attributes:

- Key Set & Key Index
- Key Usage
- Key Algorithm
- Mode of Use

These attributes are helpful for KMS-II to identifiy each key and to decide the scope of each key clearly.

1.2.1. Key Usage

KMS-II supports the following key usage.

Key Usage	Description
DUKPT Initial Key (IPEK) *	This key is used in the device as the initial key in a DUKPT key management scgeme.
PIN Encryption Key	This key is used to protect PIN blocks. It can be used with PIN block format and with any PIN block encryption method we supported. It cannot be used with any type of data other then a PIN or PIN block.
Data Encryption Key	This key can be used to encrypher or decripher data, but it cannot be used in any more specific cryptographic operations such as PIN block encryption or key encryption.
MAC Key	This key can be used to generate or verify a MAC. The MAC alrorithm is decided during the key injection.
TR-31 Key Block Protection Key	The derivation key from which the Key Block Encryption Key and the Key Block MAC Key are derived. This key cannot be used for other

		purpose.

** DUKPT implemented in KMS-II follows the specification ANS x9.24-2009.*

1.2.2. Algorithm

The algorithm is identified as below,

Value	Hex	Definition
'T'	0x54	Triple DES

1.2.3. Mode of Use

The Mode of Use defines the operation performed with the key. The mode is identified as below,

Value	Hex	Definition
'D'	0x44	Decrypt / Unwarp Only
'E'	0x45	Encrypt / Warp Only
'G'	0x46	Generate Only
'X'	0x58	Key used to derive other keys

1.2.4. Key Set & Key Index

Each key is identified by Key Set and Key Index.

- Each key set is unique for all applications, and the key index is unique in the key set.
- Each key set has its owner (an application), only the owner is allowed to do cryptographic functions with specified key in the key set.

1.2.5. limitations

The limitations of KMS-II are defined as below,

Key Value

Each key shall have the unique value for the application due to the security reason.

Key Set Range

The key set values 0x0000, and ranging from 0xC000 to 0xFFFF are reserved for system. The application shall not use the value in this range for the key set.

Key Storage

Each key set can only be used to store the key which has the same key type. The key type which KMS-II supports are listed as below:

- **KMS2_KEYTYPE_3DES** **0x01**
  - PIN Encryption Key
  - Data Encryption Key
  - MAC Key
  - TR-31 Key Block Protection Key
- **KMS2_KEYTYPE_3DES_DUKPT** **0x02**
  - DUKPT Initial Key (IPEK) [ANS X9.24-2009]

2. Key Injection

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

- By Key Block Binding Method

2.1. By Key Block Binding Method

This section introduces the key injection method using key block binding method which is following X9 TR31 2010. Currently KMS-II supports Key Derivation Binding Method only.

A key block in TR31 method contains several parts as below,

- The Key Header Block (KBH), which is not encrypted and contains attribute information about protected key.
    - The 1st section is 16bytes with a fixed format.
    - The 2nd section is optional
- The encrypted confidential data.
    - Two bytes indicating the key length
    - The key data which is being stored
    - Optional random padding
- A MAC Block, which is 16 bytes in Hex-ASCII for Key Derivation Binding Method.

Header	Header (Optional)	Encrypted Key Data (Key Length + Key + Padding)	MAC

2.1.1. Key Header Block

The 1st section of Key Header Block is a fixed format as below,

Byte #	Field Name	Description
0	Key Block Version	Identifies the version of the key block, this field should be: ◆ **'B' (0x42) – Key Block Protected using the Key Derivation Binding Method 2010**
1-4	Key Block Length	ASCII numeric digits providing total key block length. E.g., a 112 bytes key block would contain, '0' in byte #1, and '1' in byte #2, and '1' in byte #3, and '2' in byte #4.
5-6	Key Usage	Provides information about the intended function of the protected key. The available options as below, ◆ **'B1' (0x42, 0x31) – DUKPT Initial Key** ◆ **'D0' (0x44, 0x30) – Data Encryption Key** ◆ **'P0' (0x50, 0x30) – PIN Encryption Key** ◆ **'K1' (0x4B, 0x31) – TR-31 Key Block Protection Key** ◆ **'M0' (0x4D, 0x30) – MAC Key**
7	Algorithm	The algorithm for the protected key may be used. The available options as below, ◆ **'T' (0x54) – Triple DES**
8	Mode of Use	Defines the operation the protected key can perform. For **DUKPT Initial Key,** the available options as below, ◆ **'X' (0x58) – Key used to derive other keys**

		For **Data Encryption Key**, the available options as below, ◆ **'D' (0x44) – Decrypt Only** ◆ **'E' (0x45) – Encrypt Only** For **PIN Encryption Key,** the available options as below, ◆ **'E' (0x45) – Encrypt Only** For **TR-31 Key Block Protection Key,** the available options as below, ◆ **'D' (0x44) – Decrypt Only** For **MAC Key,** the available options as below, ◆ **'G' (0x47) – Generate Only**
9-10	Key Version Number	Two-digit HEX ASCII ('0' - '9', 'A' - 'F') character version number, used to prevent re-injection of the old keys. E.g., version 160 (0xA0) would contain 'A' in byte #9, and '0' in byte #10
11	Exportability	This field should be filled with below value, ◆ **'N' (0x4E) – Non-exportable**
12-13	NoOB	This field should be filled by '0' in byte #12, and '0' in byte #13 to indicates no any optional header blocks are used here.
14-15	RFU	This field is reserved for future used and is filled with ASCII zero (0x30) character.

The 2nd section of Key Header Block is formatted as below,

Byte #	Field Name	Description
16-17	First Optional Block ID	KSN for IPEK, fixed to ‘KS’ (0x4B53)
18-19	Optional Block 1 Length	Fixed to ‘18’ (0x3138)
20-39	Optional Block 1	KSN for IPEK in HEX ASCII format.

	Data	

2.1.2. Encrypted Key Data Block

The encrypted key data block is formatted as below. For the calculation method, please refer to specification X9 TR-31 2010.

Byte #	Field Name	Description
VAR.	Encrypted Key Data Block	Encrypted key data in Hex-ASCII.

2.1.3. MAC block

The MAC block is 16 bytes message authentication code in Hex-ASCII. For the calculation method, please refer to specification X9 TR-31 2010.

Byte #	Field Name	Description
VAR.	MAC	16 bytes message authentication code in Hex-ASCII.

3. KMS-II Application Programming Interfaces

Management

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

Key Injection

- USHORT CTOS_KMS2KeyWriteByTR31(CTOS_KMS2KEYWRITEBYTR31_PARA* pKeyWriteByTR31Para);

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_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);

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.
d_KMS2_PURPOSE_NOT_UNIQUE	0x2916	The purpose of key attribute is not unique.

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. ***Notice!*** ***This function shall be called in the application before using any other KMS-II functions.***

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 Please refer to **KMS2 Error Codes** for more details.

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	Please refer to **KMS2 Error Codes** for more details.

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

Please refer to **KMS2 Error Codes** for more details.

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	Please refer to **KMS2 Error Codes** for more details.

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_KMS2KeySwap

USHORT CTOS_KMS2KeySwap(CTOS_KMS2KEYSWAP_PARA *para);

Description

Specify 2 keys which are belonged to the same owner (application), and exchange the key slot directly.

Structure Version 01

```
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

Structure Version 01

[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 Please refer to **KMS2 Error Codes** for more details.

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;
}
```

Note This function is only used to exchange the key slots with specified 2 keys. All the attributes of each key includes key type, key usage, and key algorithm don't be affected.

The key type of source1 & source2 should be the same key type, otherwise the error code d_KMS2_KEY_TYPE_NOT_MATCH will be thrown.

CTOS_KMS2KeyWriteByTR31

USHORT CTOS_KMS2KeyWriteByTR31(CTOS_KMS2KEYWRITEBYTR31_PARA*

pKeyWriteByTR31Para);

Description

This function is used to write or update a key into KMS by TR31. All the keys written by this function will take the caller application as their owner.

Structure Version 01

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

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

        struct
        {
                IN USHORT CipherKeySet;
                IN USHORT CipherKeyIndex;
        }Protection;

        struct
        {
                IN USHORT KeyLength;
                IN BYTE* pKeyData;
        }Value;

}CTOS_KMS2KEYWRITEBYTR31_PARA;
```

Parameters

Structure Version 01

[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] *Protection.CipherKeySet*
Specify the key set of the key used for encryption.

[IN] *Protection.CipherKeyIndex*
Specify the key index of the key used for encryption.

[IN] *Value.KeyLength*
Specify the new/update key length.

[IN] *Value.pKeyData*
Point to a buffer containing the ciphered or plaintext key data.

Return Value Please refer to **KMS2 Error Codes** for more details.

Example

```
// 0081, 0001, 00810001000000000011223344556677
BYTE const KMS_KEY_PIN[] = {
   0x42, 0x30, 0x31, 0x32, 0x30, 0x50, 0x30, 0x54,
   0x45, 0x31, 0x32, 0x4E, 0x30, 0x31, 0x30, 0x30,
   0x4B, 0x53, 0x31, 0x38, 0x46, 0x46, 0x46, 0x46,
   0x39, 0x38, 0x37, 0x36, 0x35, 0x34, 0x33, 0x32,
   0x31, 0x30, 0x45, 0x30, 0x30, 0x30, 0x30, 0x30,
   0x33, 0x37, 0x45, 0x38, 0x44, 0x31, 0x34, 0x44,
   0x42, 0x42, 0x37, 0x38, 0x46, 0x38, 0x35, 0x34,
   0x42, 0x42, 0x42, 0x41, 0x45, 0x42, 0x31, 0x44,
   0x45, 0x31, 0x45, 0x34, 0x41, 0x44, 0x31, 0x44,
   0x45, 0x38, 0x42, 0x45, 0x44, 0x41, 0x35, 0x32,
    0x46, 0x31, 0x37, 0x33, 0x30, 0x35, 0x33, 0x45,
    0x30, 0x39, 0x39, 0x43, 0x34, 0x38, 0x37, 0x44,
    0x42, 0x32, 0x38, 0x43, 0x46, 0x31, 0x39, 0x37,
    0x37, 0x43, 0x39, 0x45, 0x31, 0x46, 0x44, 0x43,
    0x35, 0x38, 0x41, 0x42, 0x39, 0x35, 0x41, 0x32
};

int main(int argc,char *argv[])
{
        BYTE bKey;
        CTOS_KMS2KEYWRITEBYTR31_PARA stWriteKeyTR31;
        USHORT usRet;
        BYTE Message[24];

        CTOS_LCDTClearDisplay();

        stWriteKeyTR31.Version = 0;
        stWriteKeyTR31.Info.KeySet = 0x0081;
        stWriteKeyTR31.Info.KeyIndex = 0x0001;
        stWriteKeyTR31.Protection.CipherKeySet = 0xFF80;
        stWriteKeyTR31.Protection.CipherKeyIndex = 0x0001;
        stWriteKeyTR31.Value.KeyLength = sizeof(KMS_KEY_PIN);
        stWriteKeyTR31.Value.pKeyData = KMS_KEY_PIN;
        usRet = CTOS_KMS2KeyWriteByTR31(&stWriteKeyTR31);
        if(usRet != d_OK)
        {
            CTOS_LCDTPrintXY(1, 7, "Write PIN KEY ERR");
```

```
                sprintf(Message, "Ret: %04X", usRet);
                CTOS_LCDTPrintXY(1, 8, Message);
                CTOS_KBDGet(&bKey);
                exit(0);
            }
            CTOS_LCDTPrintXY(1, 7, "Write PIN KEY OK");
            sprintf(Message, "Ret: %04X", usRet);
            CTOS_LCDTPrintXY(1, 8, Message);
            CTOS_KBDGet(&bKey);
        }
```

CTOS_KMS2PINGet

USHORT CTOS_KMS2PINGet(CTOS_KMS2PINGET_PARA *pPinGetPara);

Description

This function is provided for application to get the ciphered PIN block with specified PIN key.

The keypad is controlled by KMS-II, but the prompts are application's responsibility to show the suitable message for the card holder.

Structure Version 01

```
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;
                IN BYTE         SK_Length;
                IN BYTE* pSK;
        }Protection;

        struct
        {
                // This is used for PAN if BlockType is
                // KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.
                IN BYTE InLength;
                IN BYTE* pInData;
        }AdditionalData;

        // This field is used for DUKPT only
        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;
```

Structure Version 02

```
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;
        IN BYTE         SK_Length;
        IN BYTE* pSK;
}Protection;

struct
{
        // This is used for PAN if BlockType is
        // KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.
        IN BYTE InLength;
        IN BYTE* pInData;
}AdditionalData;

// This field is used for DUKPT only
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

Structure Version 01

[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*

Specify the length of session key.

[IN] *Protection.pSK*

Point to a buffer containing the ciphered session key. The session key is ciphered by the **specified key** with expected cipher operation in ECB mode.

[IN] *AdditionalData.InLength*

Specify the length of additional input data pointed by pInData.

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

[IN] *AdditionalData.pInData*

Point to a buffer containing the additional input data.

Note.
This field is used as the PAN data if BlockType is KMS2_PINBLOCKTYPE_ANSI_X9_8_ISO_0.
Note that for PAN data, it shall also contain the last check digit.

[IN] *DUKPT_PARA. IsUseCurrentKey*

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.

Note.
This field is used only for the DUKPT.

[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. The maximum is 15 minutes. Specified 0 for this field means the maximum value.

[IN] *Control.AsteriskPositionX **

Starting column number of Asterisk.

[IN] *Control.AsteriskPositionY **

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 **

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.

** The field doesn't exist in all the structure version.*

Structure Version 02

[IN] *EventFunction.OnGetPINDigit **
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 **
Point to a callback function which is called when the user presses the CANCEL button during PIN entering.

[IN] *EventFunction.OnGetPINBackSpace **
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.

** The field doesn't exist in all the structure version.*

Return Value Please refer to **KMS2 Error Codes** for more details.

Example

```
BYTE const TestPAN[] = "4067111122223333";
BYTE const TestLine1Msg[] = "Require 123456";
BYTE const TestLine2Msg[] = "Enter PIN :";
CTOS_KMS2PINGET_PARA para;
USHORT ret;
BYTE PINBlock[16];
BYTE str[17];

CTOS_LCDTPrintXY(1, 1, TestLine1Msg);
CTOS_LCDTPrintXY(1, 2, TestLine2Msg);

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.NULLPIN = FALSE;
para.Control.piTestCancel = NULL;
para.Control.AsteriskPositionX = 1;
para.Control.AsteriskPositionY = 4;

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.

Structure Version 01

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

    struct
    {
        IN USHORT CipherKeySet;
        IN USHORT CipherKeyIndex;
        IN BYTE     CipherMethod;
        IN BYTE     SK_Length;
        IN BYTE* pSK;
    }Protection;

    // This field is used for DUKPT only
    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 **Structure Version 01**

[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)

[IN] *Protection.SK_Length*

Specify the length of session key.

[IN] *Protection.pSK*

Point to a buffer containing the ciphered session key. The session key is ciphered by the **specified key** with expected cipher operation in ECB mode.

[IN] *DUKPT_PARA. IsUseCurrentKey*

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.

Note.
This field is used only for the DUKPT.

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

Note.
If the *CipherMethod* is CBC mode, the padding will pad 0x00 to the tail of input data to be multiple of crypto length.

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

[IN] *Input.ICVLength*
Specify the length of Initial Chaining Vector (ICV).

Note.
This field is used only for the CipherMethod being CBC mode.

[IN] *Input.pICV*
Point to a buffer containing the data of Initial Chaining Vector.

Note.
This field is used only for the CipherMethod being CBC mode.

[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 Please refer to **KMS2 Error Codes** for more details.

Example

```
USHORT ret;
CTOS_KMS2DATAENCRYPT_PARA para;
BYTE plaindata[256];
BYTE cipherdata[256];
BYTE str[17];

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.

Structure Version 01

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

        struct
        {
                IN USHORT CipherKeySet;
                IN USHORT CipherKeyIndex;
                IN BYTE         CipherMethod;
                IN BYTE         SK_Length;
                IN BYTE* pSK;
        }Protection;

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

        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 **Structure Version 01**

[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.

[IN] Protection.SK_Length
Specify the length of session key.

[IN] Protection.pSK
Point to a buffer containing the ciphered session key. The session key is ciphered by the **specified key** with expected 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

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.

Note.
This field is used only for the DUKPT.

[IN] Input.Length

Specify the length of input data.

Note.
This field shall be multiple of the crypto length.

[IN] Input.pData

Point to a buffer containing the input data.

Note.
It's application's responsibility to pad the suitable value in the tail for this field. This field shall be multiple of the crypto length.

[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 Please refer to KMS2 Error Codes for more details.

Example

```
CTOS_KMS2MAC_PARA para;
USHORT ret;
BYTE Zero[8];
BYTE plaindata[256];
BYTE macdata[8];
BYTE str[17];

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

memset(&para, 0x00, sizeof(CTOS_KMS2MAC_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x1001;
para.Protection.CipherKeyIndex = 0x0001;
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_KMS2KeyGetInfo

USHORT CTOS_KMS2KeyGetInfo(IN CTOS_KMS2KEYGETINFO_PARA *pKeyGetInfoPara);

Description Get information of the specified working key.

Structure Version 01

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

        struct
        {
                IN USHORT KeySet;
                IN USHORT KeyIndex;
                IN BYTE CVLen;
                IN BYTE HashAlgorithm;
        }Input;

        struct
        {
                OUT BYTE KeyType;
                OUT BYTE KeyVersion;
                OUT DWORD KeyAttribute;
                OUT USHORT KeyLength;
                OUT BYTE* pCV;
                OUT USHORT KeyExponentLength;
                OUT BYTE* pHash;
        }Output;

}CTOS_KMS2KEYGETINFO_PARA;
```

Parameters **Structure Version 01**

[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 below key type:

- *DUKPT Initial Key (IPEK)*
- *PIN Encryption Key*
- *Data Encryption Key*
- *MAC Key*
- *TR-31 Key Block Protection Key*

[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 below key type:

- *DUKPT Initial Key (IPEK)*
- *PIN Encryption Key*
- *Data Encryption Key*
- *MAC Key*
- *TR-31 Key Block Protection Key*

[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 Please refer to **KMS2 Error Codes** for more details.

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 Please refer to **KMS2 Error Codes** for more details.

Example

Note.
At least one IPEK for DUKPT shall be injected first before execute 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.

[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 Please refer to **KMS2 Error Codes** for more details.

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;
}
```

Note The space of common user data is 64K, while the space of each application's user data (private) is 16K.

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. [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	Please refer to **KMS2 Error Codes** for more details.

Example

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

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

Note The space of common user data is 64K, while the space of each application's user data (private) is 16K.

4. Appendix A : KMS-II Example

4.1. CTOS_KMS2DataEncrypt

4.1.1. Cipher Method: CBC Decryption

This method is used for data decryption, and it can only be used when the specified key type is KMS2_KEYTYPE_3DES_DUKPT.

- KMS2_DATAENCRYPTCIPHERMETHOD_CBC_DECRYPTION 0x03
  Use the session key which is derivied from the specified 3DES_DUKPT key for data decryption. The session used here is defined in specification ANSI X9.24 as "Data encryption, request or both ways".

Example

```
USHORT ret;
CTOS_KMS2DATAENCRYPT_PARA para;
BYTE str[17];
BYTE plaindata[256];
BYTE cipherdata[256];

CTOS_LCDTClearDisplay();
CTOS_LCDTPrintXY(1, 1, "EncryptData");
CTOS_LCDTPrintXY(1, 2, "    with 3DES");

//--------------------------------------------------------
CTOS_LCDTPrintXY(1, 3, "CBC");

memset(&para, 0x00, sizeof(CTOS_KMS2DATAENCRYPT_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x1000;
para.Protection.CipherKeyIndex = 0x0001;
para.Protection.CipherMethod = KMS2_DATAENCRYPTCIPHERMETHOD_CBC_DECRYPTION;
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);
		goto __Leave;
	}

	CTOS_LCDTPrintXY(1, 3, "CBC OK");
```

4.2. CTOS_KMS2MAC

4.2.1. MAC Method: X9.19 Chaining Mode

KMS-II Extensions supports X9_19 chaning mode for 3DES key and 3DES_DUKPT key.

◆ KMS2_MACMETHOD_X9_19_START	0x02
◆ KMS2_MACMETHOD_X9_19_UPDATE	0x03
◆ KMS2_MACMETHOD_X9_19_FINAL	0x04

Operation Step

1. Using KMS2_MACMETHOD_X9_19_START to start the calculation.
2. Except the last data block, put all the data into MAC calculation by using KMS2_MACMETHOD_X9_19_UPDATE to update the result.
3. Using KMS2_MACMETHOD_X9_19_FINAL and put the last data block into MAC calculation for the MAC.

Example

```
void MAC(void)
{
      CTOS_KMS2MAC_PARA para;
      USHORT ret;
      BYTE str[17];
      BYTE key;
      BYTE *pCipherKey;
      BYTE CipherKeyLength;
      BYTE Zero[8];
      BYTE plaindata[32];
      BYTE macdata[8];
      BYTE strbuf[8];
      int i,j;
      BYTE buff_1[8],buff_2[8];

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

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

      //-----------------------------------------------------
      pCipherKey = (BYTE*)Key_3DES_1000_0001;
```

```
CipherKeyLength = 16;

memset(&para, 0x00, sizeof(CTOS_KMS2MAC_PARA));
para.Version = 0x01;
para.Protection.CipherKeySet = 0x9000;
para.Protection.CipherKeyIndex = 0x000F;
para.Protection.CipherMethod = MAC_MODE_X9_19_START;
para.ICV.Length = 8;
para.ICV.pData = Zero;
//para.DUKPT_PARA.IsUseCurrentKey = KMS2_KEYTYPE_3DES_DUKPT;
para.Input.Length = 8;
para.Input.pData = plaindata;
para.Output.Length = 8;
para.Output.pData = macdata;
ret = CTOS_KMS2MAC(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      goto __Leave;
}
memset(str,0x00,sizeof(str));

for(j=0;j<8;j++)
{
      sprintf(str+j*2, "%02x ", macdata[j]);
}
CTOS_LCDTPrintXY(1, 2, str);

para.Protection.CipherMethod = MAC_MODE_X9_19_UPDATE;
para.Input.Length = 24;
para.Input.pData = plaindata+8;
ret = CTOS_KMS2MAC(&para);
if(ret != d_OK)
{
      sprintf(str, "ret = 0x%04X", ret);
      CTOS_LCDTPrintXY(1, 8, str);
      goto __Leave;
}
memset(str,0x00,sizeof(str));

for(j=0;j<8;j++)
{
      sprintf(str+j*2, "%02x ", macdata[j]);
}
CTOS_LCDTPrintXY(1, 3, str);

para.Protection.CipherMethod = MAC_MODE_X9_19_FINAL;
para.Input.Length = 0;
para.Input.pData = plaindata;
ret = CTOS_KMS2MAC(&para);
if(ret != d_OK)
{
```

```
            sprintf(str, "ret = 0x%04X", ret);
            CTOS_LCDTPrintXY(1, 8, str);
            goto __Leave;
      }
      memset(str,0x00,sizeof(str));

      for(j=0;j<8;j++)
      {
            sprintf(str+j*2, "%02x ", macdata[j]);
      }
      CTOS_LCDTPrintXY(1, 4, str);

      CTOS_LCDTPrintXY(1, 7, "MAC Done");

__Leave:
      CTOS_KBDGet(&key);
}
```