

# Algorithm library User Guide

## *Terms and abbreviations*

Abbreviations	Whole put together
AES	Advance Encryption Standard
DES	Data Encryption standard
TDES	Triple Data Encryption standard
RNG	Random Number Generator
SHA	Secure Hashing Algorithm is required for digital signature applications

# CONTENTS

CONTENTS.....	- 3 -
<b>1. OVERVIEW.....</b>	<b>- 6 -</b>
1.1. SUPPORTED ALGORITHMS .....	- 6 -
1.2. BASIC DATA TYPES.....	- 6 -
<b>2. DES/TDES ALGORITHM API DESCRIPTION .....</b>	<b>- 7 -</b>
2.1. METHOD OF USING ALGORITHM LIBRARY .....	- 7 -
2.2. DATA TYPE DEFINITION.....	- 7 -
2.3. FUNCTION INTERFACE DESCRIPTION.....	- 8 -
2.3.1. <i>The DES/TDES algorithm init</i> .....	- 8 -
2.3.2. <i>DES/TDES algorithm encryption and decryption</i> .....	- 9 -
2.3.3. <i>DES/TDES close</i> .....	- 9 -
2.3.4. <i>Obtain DES/TDES library version information</i> .....	- 10 -
<b>3. AES ALGORITHM API DESCRIPTION .....</b>	<b>- 11 -</b>
3.1. METHOD OF USING ALGORITHM LIBRARY .....	- 11 -
3.2. DATA TYPE DEFINITION.....	- 11 -
3.3. FUNCTION INTERFACE DESCRIPTION.....	- 12 -
3.3.1. <i>The AES algorithm is initialized</i> .....	- 12 -
3.3.2. <i>AES algorithm for encryption and decryption</i> .....	- 13 -
3.3.3. <i>Close the AES</i> .....	- 13 -
3.3.4. <i>Obtain the AES library version information</i> .....	- 13 -
<b>4. HASH ALGORITHM API DESCRIPTION.....</b>	<b>- 15 -</b>
4.1. METHOD OF USING ALGORITHM LIBRARY .....	- 15 -
4.2. DATA TYPE DEFINITION.....	- 15 -
4.3. FUNCTION INTERFACE DESCRIPTION.....	- 17 -
4.3.1. <i>HASH initialization</i> .....	- 17 -
4.3.2. <i>HASH start operation</i> .....	- 17 -

4.3.3.	<i>HASH processes data step by step</i> .....	- 18 -
4.3.4.	<i>HASH completes and takes the result</i> .....	- 19 -
4.3.5.	<i>HASH operation close</i> .....	- 19 -
4.3.6.	<i>Obtain HASH library version information</i> .....	- 19 -
<b>5.</b>	<b>SM4 ALGORITHM API DESCRIPTION</b> .....	<b>- 21 -</b>
5.1.	METHOD OF USING ALGORITHM LIBRARY .....	- 21 -
5.2.	DATA TYPE DEFINITION.....	- 21 -
5.3.	FUNCTION INTERFACE DESCRIPTION.....	- 22 -
5.3.1.	<i>The SM4 algorithm is initialized</i> .....	- 22 -
5.3.2.	<i>SM4 algorithm for encryption and decryption</i> .....	- 23 -
5.3.3.	<i>Close the SM4</i> .....	- 23 -
5.3.5	<i>Obtain the SM4 library version information</i> .....	- 23 -
<b>6.</b>	<b>RNG ALGORITHM API DESCRIPTION</b> .....	<b>- 25 -</b>
6.1.	METHOD OF USING ALGORITHM LIBRARY .....	- 25 -
6.2.	DATA TYPE DEFINITION.....	- 25 -
6.3.	FUNCTION INTERFACE DESCRIPTION.....	- 25 -
6.3.1.	<i>Pseudo random generating function</i> .....	- 26 -
6.3.2.	<i>Random number generating function</i> .....	- 26 -
6.3.3.	<i>Obtain the RNG library version information</i> .....	- 26 -
<b>7.</b>	<b>VERSION HISTORY</b> .....	<b>- 28 -</b>
<b>8.</b>	<b>NOTICE</b> .....	<b>- 29 -</b>
<b>I.</b>	<b>APPENDIX I DES ALGORITHM LIBRARY FUNCTION CALL ROUTINE</b> .....	<b>- 30 -</b>
<b>II.</b>	<b>APPENDIX II TDES LIBRARY FUNCTION CALL ROUTINE</b> .....	<b>- 36 -</b>
<b>III.</b>	<b>APPENDIX III AES ALGORITHM LIBRARY FUNCTION CALL ROUTINE</b> .....	<b>- 46 -</b>
<b>IV.</b>	<b>APPENDIX IV HASH LIBRARY FUNCTION CALL ROUTINES</b> .....	<b>- 68 -</b>
<b>V.</b>	<b>APPENDIX V SM4 ALGORITHM LIBRARY FUNCTION CALL ROUTINE</b> .....	<b>- 74 -</b>

**VI. APPENDIX VI RNG LIBRARY CALL ROUTINE..... - 81 -**

# 1. Overview

This document is applicable to N3249X series chips, and mainly describes the algorithm interface and usage methods in these chips.

For U32 data type parameters, when using U8 casts to U32 format, it is necessary to ensure that the U8 address is word-aligned.

## 1.1. Supported algorithms

The algorithms provided are as follows:

- DES: encryption/decryption
- TDES: encryption/decryption
- AES: encryption/decryption (AES-128/192/256)
- SM4: encryption/decryption
- HASH: obtain the digest. Supported (SHA-1/SHA-224/SHA-256/MD5/SM3)
- RNG: generates random numbers

## 1.2. Basic data types

<i>typedef unsigned char</i>	<i>bool;</i>
<i>typedef unsigned char</i>	<i>u8;</i>
<i>typedef signed char</i>	<i>s8;</i>
<i>typedef unsigned short</i>	<i>u16;</i>
<i>typedef signed short</i>	<i>s16;</i>
<i>typedef unsigned int</i>	<i>u32;</i>
<i>typedef signed int</i>	<i>s32;</i>
<i>typedef unsigned long long</i>	<i>u64;</i>
<i>typedef signed long long</i>	<i>s64;</i>

## 2. DES/TDES algorithm API description

### 2.1. Method of using algorithm library

The method library is used as follows:

1. Place n32h49x\_des.h, Type.h, and n32h49x\_algo\_common.h in the folder. Add n32h49x\_algo\_common.lib and n32h49x\_des.lib in project.
2. Call the function according to the function description in Section 2.3. See the demo provided in Appendix I and Appendix II for the routine

### 2.2. Data type definition

```
#define DES_ECB (0x11111111)
#define DES_CBC (0x22222222)
#define DES_ENC (0x33333333)
#define DES_DEC (0x44444444)
#define DES_KEY (0x55555555)
#define TDES_2KEY (0x66666666)
#define TDES_3KEY (0x77777777)

enum DES
{
    DES_Crypto_OK = 0x0,    //DES/TDES operation success
    DES_Init_OK = 0x0,    //DES/TDES Init operation success
    DES_Crypto_ModeError = 0x5a5a5a5a,    //Working mode error(Neither ECB nor CBC)
    DES_Crypto_EnOrDeError,    //En&De error(Neither encryption nor decryption)
    DES_Crypto_ParaNull,    // the part of input(output/iv) Null
    DES_Crypto_LengthError,    //the length of input message must be 2 times and cannot be zero
    DES_Crypto_KeyError, //keyMode error(Neither DES_KEY nor TDES_2KEY nor TDES_3KEY)
    DES_Crypto_UnInitError, //DES/TDES uninitialized
};
```

*typedef struct*

```

{
    u32 *in; // the part of input to be encrypted or decrypted
    u32 *iv; // the part of initial vector
    u32 *out; // the part of out
    u32 *key; // the part of key
    u32 inWordLen; // the length(by word) of plaintext or cipher
    u32 En_De; // 0x33333333- encrypt, 0x44444444 - decrypt
    u32 Mode; // 0x11111111 - ECB, 0x22222222 - CBC
    u32 keyMode; //TDES key mode: 0x55555555-key,0x66666666-2key, 0x77777777-3key
}DES_PARM;
  
```

## 2.3. Function Interface Description

DES library contains the following list of functions:

**Table 2-1 DES/TDES algorithm library functions**

function	Description
u32 DES_Init(DES_PARM *parm);	DES/TDES initialization function
u32 DES_Crypto(DES_PARM *parm)	DES/TDES encryption
void DES_Close(void)	DES/TDES close
void DES_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	DES version get function

### 2.3.1. The DES/TDES algorithm init

#### **DES\_Init**

The DES/TDES algorithm init

#### **Function**

u32 DES\_Init(DES\_PARM \*parm)

#### **Parameter**

parm: input, a pointer to the DES\_PARM structure

#### **Return**

DES\_Init\_OK: initialization succeed. Other: initialization error

**Note**

1. In ECB mode, parameter iv can be directly replaced with NULL.

### 2.3.2. DES/TDES algorithm encryption and decryption

**DES\_Crypto**

DES/TDES algorithm initialization, encryption and decryption

**Function**

u32 DES\_Crypto(DES\_PARM \*parm)

**Parameter**

parm: input, a pointer to the DES\_PARM structure

**Return**

DES\_Crypto\_OK: the operation is correct. Other: the operation is incorrect

**Note**

Before calling this function, call DES\_Init if it has not been initialized or switched to another algorithm.

1. In ECB mode, iv1 can be replaced with NULL.
2. When a large amount of data is encrypted as a whole but divided into multiple CBC blocks, note the following: The initial vector IV (IV = IV1) used for block X data (X>1) must be updated to the last group (8 bytes) of the ciphertext obtained by calling this function for block X-1 data.
3. Call this function to decrypt block X (X>1). The initial vector IV (IV = iv1) used must be updated to the last group of block X-1 (8 bytes).
4. Refer to Appendix 1 and Appendix 2 for the invocation method.

### 2.3.3. DES/TDES close

**DES\_Close**

Disable the DES/TDES algorithm clock and system clock

**Function**

void DES\_Close(void)

**Parameter****Return**

## 2.3.4. Obtain DES/TDES library version information

**DES\_Version**

Obtain DES/TDES library version information

**Function**

```
void DES_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)
```

**Parameter**

type	Commercial or fast version
customer	Standard or customized version
Date	Year, month, day
version	version x. x

**Return****Note**

```
*type = 0x05;      // Commercial and fast version
*customer = 0x00;  // Standard version
date[0] = 18;     //Year()
date[1] = 12;     //Month()
date[2] = 28;     //Day ()
*version = 0x10;  // Indicates version 1. 0
```

## 3. AES algorithm API description

### 3.1. Method of using algorithm library

The method library is used as follows:

1. Place n32h49x\_aes.h, Type.h, and n32h49x\_algo\_common.h in the folder. Add n32h49x\_algo\_common.lib and n32h49x\_aes.lib in project.
2. Call the function as described in Section 3. 3. See the demo provided in Appendix III for the routines

### 3.2. Data type definition

```
#define AES_ECB (0x11111111)
#define AES_CBC (0x22222222)
#define AES_CTR (0x33333333)

#define AES_ENC (0x44444444)
#define AES_DEC (0x55555555)

enum
{
    AES_Crypto_OK = 0x0,    //AES operation success
    AES_Init_OK = 0x0,     //AES Init operation success
    AES_Crypto_ModeError = 0x5a5a5a5a,    //Working mode error(Neither ECB nor CBC nor CTR)
    AES_Crypto_EnOrDeError,    //En&De error(Neither encryption nor decryption)
    AES_Crypto_ParaNull,    // the part of input(output/iv) Null
    AES_Crypto_LengthError,    // if Working mode is ECB or CBC,the length of input message must be 4 times and
    //if Working mode is CTR,the length of input message cannot be zero; othets: return
    AES_Crypto_LengthError

    AES_Crypto_KeyLengthError, //the keyWordLen must be 4 or 6 or 8; othets:return AES_Crypto_KeyLengthError
```

```

    AES_Crypto_UnInitError, //AES uninitialized
};

typedef struct
{
    uint32_t *in;          // the part of input to be encrypted or decrypted
    uint32_t *iv;          // the part of initial vector
    uint32_t *out;         // the part of out
    uint32_t *key;         // the part of key
    uint32_t keyWordLen;   // the length(by word) of key
    uint32_t inWordLen;    // the length(by word) of plaintext or cipher
    uint32_t En_De;        // 0x44444444- encrypt, 0x55555555 - decrypt
    uint32_t Mode;         // 0x11111111 - ECB, 0x22222222 - CBC, 0x33333333 - CTR
}AES_PARM;

```

### 3.3. Function Interface Description

The AES algorithm library contains the following list of functions:

**Table 3-1 FUNCTIONS of the AES algorithm library**

function	description
u32 AES_Init(AES_PARM *parm)	AES initialization
u32 AES_Crypto(AES_PARM *parm)	AES encryption and decryption function
void AES_Close(void)	AES closing function
void AES_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	AES version get function

#### 3.3.1. The AES algorithm is initialized

##### **AES\_Init**

The AES algorithm is initialized

##### **Function**

u32 AES\_Init(AES\_PARM \*parm)

<b>Parameter</b>	parm:	input, a pointer to the AES_PARM structure
<b>Return</b>	AES_Init_OK:	the operation is correct. Other: the operation is incorrect
<b>Note</b>		1. Refer to Appendix 3 for the invocation method.

### 3.3.2. AES algorithm for encryption and decryption

<b>AES_Crypto</b>	<u>AES algorithm for encryption and decryption</u>	
<b>Function</b>	u32 AES_Crypto(AES_PARM *parm)	
<b>Parameter</b>	parm:	input, a pointer to the AES_PARM structure
<b>Return</b>	AES_Crypto_OK:	the operation is correct. Other: the operation is incorrect
<b>Note</b>	Calling this function, if the function is not initialized or switched to another algorithm, call the AES_Init function first. 1. Refer to Appendix 3 for the invocation method.	

### 3.3.3. Close the AES

<b>AES_Close</b>	<u>Disable the AES clock and system clock</u>	
<b>Function</b>	void AES_Close (void)	
<b>Parameter</b>		
<b>Return</b>		

### 3.3.4 Obtain the AES library version information

<b>AES_Version</b>	<u>Obtain the AES library version information</u>	
<b>Function</b>	void AES_Version (u8 *type, u8 *customer, u8 date[3], u8 *version)	
<b>Parameter</b>	type	Commercial or fast version
	customer	Standard or customized version

date                    Year, month, day  
version                version x. x

**Return****Note**

```
*type = 0x05; // Commercial and fast version  
*customer = 0x00; // Standard version  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; // Indicates version 1.0
```

## 4. HASH algorithm API description

Including SHA1 / SHA224 / SHA256 / MD5 / SM3 algorithms library.

### 4.1. Method of using algorithm library

Data input and output are in byte big-endian order. The method library is used as follows:

1. Place Type.h, n32h49x\_hash.h, and n32h49x\_algo\_common.h in the folder. Add n32h49x\_algo\_common.lib and n32h49x\_hash in project.
2. Call the function as described in Section 4. 3. See the demo provided in Appendix IV for the routine

### 4.2. Data type definition

*enum*

{

```
HASH_SEQUENCE_TRUE = 0x0105A5A5, //save IV
HASH_SEQUENCE_FALSE = 0x010A5A5A, //not save IV
HASH_Init_OK = 0, //hash init success
HASH_Start_OK = 0, //hash update success
HASH_Update_OK = 0, //hash update success
HASH_Complete_OK = 0, //hash complete success
HASH_Close_OK = 0, //hash close success
HASH_ByteLenPlus_OK = 0, //byte length plus success
HASH_PadMsg_OK = 0, //message padding success
HASH_ProcMsgBuf_OK = 0, //message processing success
SHA1_Hash_OK = 0, //sha1 operation success
SM3_Hash_OK = 0, //sm3 operation success
SHA224_Hash_OK = 0, //sha224 operation success
SHA256_Hash_OK = 0, //sha256 operation success
MD5_Hash_OK = 0, //MD5 operation success
```

```
HASH_Init_ERROR = 0x01044400, //hash init error
HASH_Start_ERROR, //hash start error
HASH_Update_ERROR, //hash update error
HASH_ByteLenPlus_ERROR, //hash byte plus error
};

typedef struct _HASH_CTX_HASH_CTX;

typedef struct
{
    const uint16_t HashAlgID; //choice hash algorithm
    const uint32_t * const K, KLen; //K and word length of K
    const uint32_t * const IV, IVLen; //IV and word length of IV
    const uint32_t HASH_SACCR, HASH_HASHCTRL; //relate registers
    const uint32_t BlockByteLen, BlockWordLen; //byte length of block, word length of block
    const uint32_t DigestByteLen, DigestWordLen; //byte length of digest, word length of digest
    const uint32_t Cycle; //iteration times
    uint32_t (* const ByteLenPlus)(uint32_t *, uint32_t); //function pointer
    uint32_t (* const PadMsg)(HASH_CTX *); //function pointer
}HASH_ALG;

typedef struct _HASH_CTX_
{
    const HASH_ALG *hashAlg; //pointer to HASH_ALG
    uint32_t sequence; // TRUE if the IV should be saved
    uint32_t IV[16];
    uint32_t msgByteLen[4];
    uint8_t msgBuf[128+4];
    uint32_t msgIdx;
}
```

}HASH\_CTX;

## 4.3. Function Interface Description

The HASH library contains the following list of functions:

**Table 4-1 List of HASH algorithm library functions**

Function	Description
u32 HASH_Init(HASH_CTX *ctx)	HASH initializes function
u32 HASH_Start(HASH_CTX *ctx)	HASH starts the operation
u32 HASH_Update(HASH_CTX *ctx, u8 *in, u32 byteLen)	HASH processing step by step
u32 HASH_Complete(HASH_CTX *ctx, u8 *out)	HASH completing
u32 HASH_Close(void)	Close HASH function
void HASH_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	Gets the HASH library version

### 4.3.1. HASH initialization

#### **HASH\_Init**

HASH initialization

#### **Function**

u32 HASH\_Init (HASH\_CTX \*ctx)

#### **Parameter**

ctx: input, a pointer to the HASH\_CTX structure

#### **Return**

HASH\_Init\_OK: correct operation. Other: incorrect operation

#### **Note**

1. CTX must point to the RAM area and what it points to cannot be changed (intermediate state for hash computation and temporary content storage), similarly below
2. When calculating the hash value of a message, you must call this function first

### 4.3.2. HASH start operation

#### **HASH\_Start**

HASH start operation

#### **Function**

u32 HASH\_Start (HASH\_CTX \*ctx)

<b>Parameter</b>	ctx:           input, a pointer to the HASH_CTX structure
<b>Return</b>	HASH_Start_OK: correct operation. Other: incorrect operation
<b>Note</b>	<p>1. If you want to support interruption during HASH operation, set CTX -&gt;sequence to HASH_SEQUENCE_TRUE. After the interruption, you need to call HASH_Init again and then call HASH_Update. Otherwise, set it to HASH_SEQUENCE_FALSE.</p> <p>2. Refer to Appendix 4 for the invocation method.</p>

### 4.3.3.HASH processes data step by step

#### **HASH\_Update**    HASH processes data step by step

<b>Function</b>	u32 HASH_Update (HASH_CTX *ctx, u8 *in, u32 byteLen)
<b>Parameter</b>	ctx:           input, a pointer to the HASH_CTX structure In:            input, meaning information to be hashed byteLen:       input, the length of hashed information in bytes
<b>Return</b>	HASH_Update_OK: correct operation. Other: incorrect operation
<b>Note</b>	<p>Before calling this function, call the HASH_Init and HASH_Start functions if you have not initialized or switched to another algorithm.</p> <ol style="list-style-type: none"> <li>1. The initialization functions HASH_Init and HASH_Start must be called before calling this function</li> <li>2. CTX must point to the RAM area and what it points to cannot be changed (intermediate state for hash computation and temporary content storage).</li> <li>3. The contents of 'in' can point to RAM or Flash area. 'in' can be NULL, and the calculated result is the digest value of NULL.</li> <li>4. byteLen can be 0 or NULL, and the result is the digest value of NULL.</li> <li>5. After initialization, the whole message can be arbitrarily divided into many small pieces. For each small piece of message, this function can be called in turn, and finally call the HASH_Complete function to obtain the hash result of the whole message.</li> <li>6. For cascading applications, you need to set ctx -&gt; sequence = HASH_SEQUENCE_TRUE; copy external IV to ctx -&gt;IV; Add len (length of updated data) to CTX -&gt;msgByteLen via</li> </ol>

ctx->hashAlg->ByteLenPlus(ctx->msgByteLen,len) , then call HASH\_Update to cascade successfully

7. Refer to Appendix 4 for the invocation method.

#### 4.3.4.HASH completes and takes the result

<b>HASH_Complete</b>	<u>HASH completes and takes the result</u>
<b>Function</b>	u32 HASH_Complete (HASH_CTX *ctx, u8 *out)
<b>Parameter</b>	ctx: input, a pointer to the HASH_CTX structure out: output, a pointer to the HASH result
<b>Return</b>	HASH_Complete_OK: Correct operation. Other values: error operation
<b>Note</b>	Before calling this function, call the HASH_Init and HASH_Start functions if you have not initialized or switched to another algorithm.  1. Call this function to obtain the final result after the message is input. 2. ctx must point to the RAM area and what it points to cannot be changed (intermediate state for hash computation and temporary content storage). 3. Refer to Appendix 4 for the invocation method.

#### 4.3.5.HASH operation close

<b>HASH_Close</b>	<u>HASH operation close</u>
<b>Function</b>	u32 HASH_Close(void)
<b>Parameter</b>	
<b>Return</b>	HASH_Close_OK: The operation is correct
<b>Note</b>	

#### 4.3.6.Obtain HASH library version information

<b>HASH_Version</b>	<u>Obtain HASH library version information</u>
---------------------	--

**Function** void HASH\_Version(u8 \*type, u8 \*customer, u8 date[3], u8 \*version)

**Parameter**

type	Commercial or fast version
customer	Standard or customized version
date	Year, month, day
version	version x. x

**Return**

**Note**

```
*type = 0x05; // Commercial and fast version
*customer = 0x00; // Standard version
date[0] = 18; //Year()
date[1] = 12; //Month()
date[2] = 28; //Day ()
*version = 0x10; // Indicates version 1. 0
```

## 5. SM4 algorithm API description

### 5.1. Method of using algorithm library

Data input and output are in byte big-endian order. The method library is used as follows:

1. Place Type.h, n32h49x\_sm4.h, and n32h49x\_algo\_common.h in the folder. Add n32h49x\_algo\_common.lib and n32h49x\_sm4.lib in project.
2. Call the function as described in Section 5. 3. See the demo provided in Appendix V for the routine.

### 5.2. Data type definition

```
#define SM4_ECB (0x11111111)
#define SM4_CBC (0x22222222)
#define SM4_ENC (0x33333333)
#define SM4_DEC (0x44444444)

enum{
    SM4_Crypto_OK=0, //SM4 operation success
    SM4_Init_OK=0, //SM4 Init operation success
    SM4_ADRNULL =0x27A90E35, //the address is NULL
    SM4_ModeErr, //working mode error(Neither ECB nor CBC)
    SM4_EnDeErr, // En&De error(Neither encryption nor decryption)
    SM4_LengthErr, //the word length of input error(the word length is 0 or is not as times as 4)
    SM4_UnInitError, //SM4 uninitialized
};

typedef struct{
    uint32_t *in; // the first part of input to be encrypted or decrypted
    uint32_t *iv; // the first part of initial vector
    uint32_t *out; // the first part of out
    uint32_t *key; // the first part of key
```

```

uint32_t inWordLen; //the word length of input or output

uint32_t EnDeMode; //encrypt/decrypt

uint32_t workingMode; // ECB/CBC

}SM4_PARM;
  
```

## 5.3. Function Interface Description

The SM4 algorithm library contains the following list of functions:

**Table 5-1 FUNCTIONS of the SM4 algorithm library**

Function	Description
u32 SM4_Init(SM4_PARM *parm)	SM4 initialization
u32 SM4_Crypto(SM4_PARM *parm)	SM4 encryption and decryption function
void SM4_Close(void)	SM4 closing function
void SM4_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	SM4 version get function

### 5.3.1. The SM4 algorithm is initialized

**SM4\_Init**                    The SM4 algorithm is initialized

**Function**                    u32 SM4\_Init(SM4\_PARM \*parm)

**Parameter**                    parm:                    input, a pointer to the SM4\_PARM structure

**Return**                        SM4\_Init\_OK:            the operation is correct. Other: the operation is incorrect

**Note**

**SM4\_Init**                    The SM4 algorithm is initialized

**Function**                    u32 SM4\_Init(SM4\_PARM \*parm)

**Parameter**                    parm                    input, a pointer to the SM4\_PARM structure

**Return**                        SM4\_Init\_OK:            the operation is correct. Other: the operation is incorrect

**Note**

### 5.3.2. SM4 algorithm for encryption and decryption

<b>SM4_Crypto</b>	<u>SM4 algorithm for encryption and decryption</u>
<b>Function</b>	u32 SM4_Crypto(SM4_PARM *parm)
<b>Parameter</b>	parm: input, a pointer to the SM4 _PARAM structure
<b>Return</b>	SM4_Crypto_OK: the operation is correct. Other: the operation is incorrect
<b>Note</b>	Calling this function, if the function is not initialized or switched to another algorithm, call the SM4_Init function first.

### 5.3.3. Close the SM4

<b>SM4_Close</b>	<u>Disable the SM4 clock and system clock</u>
<b>Function</b>	void SM4_Close (void)
<b>Parameter</b>	
<b>Return</b>	

### 3.3.5 Obtain the SM4 library version information

<b>SM4_Version</b>	<u>Obtain the SM4 library version information</u>
<b>Function</b>	void SM4_Version (u8 *type, u8 *customer, u8 date[3], u8 *version)
<b>Parameter</b>	type Commercial or fast version customer Standard or customized version date Year, month, day version version x. x
<b>Return</b>	
<b>Note</b>	*type = 0x05; // Commercial and fast version *customer = 0x00; // Standard version date[0] = 18; //Year() date[1] = 12; //Month()

```
date[2] = 28; //Day ()
```

```
*version = 0x10; // Indicates version 1.0
```

## 6. RNG algorithm API description

### 6.1. Method of using algorithm library

The method library is used as follows:

1. Place Type.h, n32h49x\_rng.h, and n32h49x\_algo\_common.h in the folder. Add n32h49x\_algo\_common.lib and n32h49x\_rng.lib in project.
2. Call the function as described in Section 6. 3. See the demo provided in Appendix VI for the routine.

### 6.2. Data type definition

```
enum{
    RNG_OK = 0x5a5a5a5a,
    LENError = 0x311ECF50, //RNG generation of key length error
    ADDRNULL = 0x7A9DB86C, //This address is empty
};
```

### 6.3. Function Interface Description

The RNG library contains the following list of functions:

**Table 7-1 Functions of RNG algorithm library**

Function	Describe
u32 GetPseudoRand_U32(u32 *rand, u32 wordLen, u32 seed[2])	Pseudo random number according to word generation function
u32 GetTrueRand_U32(u32 *rand, u32 wordLen)	True random number by word generation function
void RNG_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	Obtain the RNG library version information

### 6.3.1. Pseudo random generating function

**GetPseudoRand\_U32**pseudorandom number according to word generation function**Function**

u32 GetPseudoRand\_U32 (u32 \*rand, u32 wordLen, u32 seed[2])

**Parameter**

rand: pointer to a generated random number

wordlen: pseudo random number word length

seed [2]: input, pseudo-random array of seed variables

**Return**

RNG\_OK: succeeded. Other: pseudo-random number generation error

**Note**

Generate pseudorandom number by word

The user can input the seed array. If the user inputs the seed array as NULL, the internal seed will be generated automatically.

### 6.3.2. Random number generating function

**GetTrueRand\_U32**True random number generating function**Function**

u32 GetTrueRand\_U32 (u32 \*rand, u32 wordLen)

**Parameter**

rand: pointer to a random memory address

wordlen: the length of the word intended to get a truly random number

**Return**

RNG\_OK: success. Others: True random number generation error, see enumeration type value definition

**Note**

### 6.3.3. Obtain the RNG library version information

**RNG\_Version**Obtain the RNG library version information**Function**

void RNG\_Version(u8 \*type, u8 \*customer, u8 date[3], u8 \*version)

**Parameter**

type Commercial or fast version

customer Standard or customized version

date Year, month, day

version version x. x

**Return****Note**

```
*type = 0x05; // Commercial and fast version
```

```
*customer = 0x00; // Standard version
```

```
date[0] = 18; //Year()
```

```
date[1] = 12; //Month()
```

```
date[2] = 28; //Day ()
```

```
*version = 0x10; // Indicates version 1. 0
```

## 7. Version History

<b>Version</b>	<b>Date</b>	<b>Changes</b>
V1.0.0	2025.11.04	Initial release.

## 8. Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections, enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, 'Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.

## i. Appendix I DES algorithm library function call routine

```
u32 DES_test()
{
    u32 i,flag1,flag2,flag3,flag4;
    u32 ret;
    DES_PARM DES_Parm={0};
    /* If you need to modify the test instance, if the actual value of the parameter is 0x0102030405060708, enter
    0x04030201,0x08070605 during initialization because u32 data is stored in byte endian sequence. Unless otherwise
    specified, the routine parameters are set in this manner */
    u32 in1 [16]={
        0x5FE2D4C0,0xAEAE3F30,0x692930A8,0x1DA69A51,0xDD34B34B,0xAF8D237A,0x2114F489,
        0xE461FF17,0x47C795FD,0x8FF62B49,0x62E9BD63,0x1AF52817,0xECB9DFD4,0xE04421C9,
        0x87B4B22E,0x9FF98759
    };
    u32 key1 [2]={0x946AB06B,0x2276E632};
    u32 iv1 [2]={0x482A8C66,0xC324FC78};
    u32 out[16];
    U32 DES_ECB_EN [16] = {0x2FD8D31F,0xC3E2E705,0x4B6D1C4C,0x31EB4154,0xDA273EEC,
        0x8EED57DA,0x26FDE038,0x15B0D57D,0xBCE7464F,0x78D7997A,
        0x4F9917D7,0xAE9C1DA9,0x749FEAEE,0xDFE6A911,0x34D556D5,
        0xA32FA0A2};
    /*DES_ECB_EN=0x1FD3D82F05E7E2C34C1C6D4B5441EB31EC3E27DADA57ED8E38E0FD267DD5B0154F
    46E7BC7A99D778D717994FA91D9CAEEEEAA9F741 1A9E6DFD556D534A2A02FA3*/
```

```
U32 DES_ECB_DE [16] = {0xBD77D94A,0xCF5698BB,0xF113743F,0x0FCFC898,0x7DD21DA8,  
0x3908A674,0x65303E6C,0x56CB0E02,0xF0B14651,0x3BBB36AB,  
0x8C129CC3,0xC42D5DD0,0x74549F20,0x5A7E5029,0xE5334FE2,  
0xD5ED9CA8};
```

```
/*DES_ECB_DE=0x4AD977BDBB9856CF3F7413F198C8CF0FA81DD27D74A608396C3E3065020ECB565146  
B1F0AB36BB3BC39C128CD05D2DC4209F54742 9507E5AE24F33E5A89CEDD5*/
```

```
U32 DES_CBC_EN [16] = {0x236813B0,0x14D3A0CA,0xDB57CA2F,0x073FADB0,0x83577985,  
0x7DEBA1CB,0xD5410854,0x2C0E74D8,0x8B8019BB,0xBAB789EF,  
0xF93DEC2E,0xD1BFE8F4,0xE061C81D,0x2F620219,0x662759FF,  
0x77CABBF6};
```

```
/*DES_CBC_EN=0xB0136823CAA0D3142FCA57DBB0AD3F0785795783CBA1EB7D540841D5D8740E2CBB  
19808BEF89B7BA2EEC3DF9F4E8BFD11DC861E01 902622FFF592766F6BBCA77*/
```

```
U32 DES_CBC_DE [16] = {0xF55D552C,0x0C7264C3,0xAEF1A0FF,0xA161F7A8,0x14FB2D00,  
0x24AE3C25,0xB8048D27,0xF9462D78,0xD1A5B2D8,0xDFDAC9BC,  
0xCBD5093E,0x4BDB7699,0x16BD2243,0x408B783E,0x098A9036,  
0x35A9BD61};
```

```
/*DES_CBC_DE=0x2C555DF5C364720CFFA0F1AEA8F761A1002DFB14253CAE24278D04B8782D46F9D8B2  
A5D1BCC9DADF3E09D5CB9976DB4B4322BD163 E788B4036908A0961BDA935*/
```

```
Cpy_U32 (out, in1, 16);
```

```
DES_Parm. in = out;
```

```
DES_Parm. key = key1;
```

```
DES_Parm. out = out;
```

```
DES_Parm. inWordLen = 16;
```

```
DES_Parm. keyMode = DES_KEY;
```

```
DES_Parm. Mode = DES_ECB;
```

```
DES_Parm. En_De = DES_ENC;
```

```
ret = DES_Init(&DES_Parm);
ret = DES_Crypto(&DES_Parm);
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag1=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(DES_ECB_EN,16, out,16))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }
}
Cpy_U32 (out, in1, 16);
DES_Parm. En_De = DES_DEC;
ret = DES_Init(&DES_Parm);
ret=(DES_Crypto(&DES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{
```

```
if(Cmp_U32(DES_ECB_DE,16, out,16))
{
    flag2=0x5A5A5A5A;
}
else
{
    flag2=0;
}
}
Cpy_U32 (out, in1, 16);
DES_Parm. iv = iv1;
DES_Parm. Mode = DES_CBC;
DES_Parm. En_De = DES_ENC;
ret = DES_Init(&DES_Parm);
ret=(DES_Crypto(&DES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(DES_CBC_EN,16, out,16))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}
```

```
    }  
}  
Cpy_U32 (out, in1, 16);  
DES_Parm. iv = iv1;  
DES_Parm. En_De = DES_DEC;  
ret = DES_Init(&DES_Parm);  
ret=(DES_Crypto(&DES_Parm));  
DES_Close();  
if (ret!= DES_Crypto_OK)  
{  
    flag4=0x5A5A5A5A;  
}  
else  
{  
    if(Cmp_U32(DES_CBC_DE,16, out,16))  
    {  
        flag4=0x5A5A5A5A;  
    }  
    else  
    {  
        flag4=0;  
    }  
}  
  
if (flag1|flag2|flag3|flag4)  
{  
    return 0x5A5A5A5A;  
}  
else
```

```
{  
    return 0;  
}
```

## ii. Appendix II TDES library function call routine

```
u32 TDES_2Key_test()
```

```
{
```

```
    u32 i,flag1,flag2,flag3,flag4;
```

```
    u32 ret;
```

```
    DES_PARM TDES_Parm={0};
```

```
    /* If you need to modify the test instance, if the actual value of the parameter is 0x0102030405060708, enter  
    0x04030201,0x08070605 during initialization because u32 data is stored in byte endian sequence. Unless otherwise  
    specified, the routine parameters are set in this manner */
```

```
    u32 in1[16]={
```

```
        0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,
```

```
        0xED8A71D7,0xA66B2FAA,0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,
```

```
        0x05CDB663,0xF7D812E4,0x86341DEB,0xBA52B237
```

```
    };
```

```
    u32 key1 [4] = {0x81F08C18,0x5C6BE38C,0x4D6A6563,0xFF220031};
```

```
    u32 iv1 [2] = {0xB5CC3A62,0xC96EF050};
```

```
    u32 out[16];
```

```
    u32 TDES_ECB_EN [16] = {0x42976179,0x3A15FDA5,0x278639E4,0x3F4D2DDD,0x987EAF74,
```

```
    0x17376CD5,0x9BE1CAB1,0x5501A0BA,0xD18D511B,0x11054F45,
```

```
    0x7EAC1828,0x375B9DAD,0x3823A312,0x8EE802FF,0xF2F00328,
```

```
    0x3F81CF19};
```

```
    /*TDES_ECB_EN=0x79619742A5FD153AE4398627DD2D4D3F74AF7E98D56C3717B1CAE19BBAA001551B  
518DD1454F05112818AC7EAD9D5B3712A32338 FF02E88E2803F0F219CF813F*/
```

```
u32 TDES_ECB_DE [16] = {0x58AD407C,0x76B43ED7,0x23B44DDA,0x22EC376C,0x50311263,  
0xECC57D42,0x2FA5ADAA,0xE7A099A0,0x287DBD9B,0x3951FD62,  
0x530A3728,0x9AFA2D3,0x0C41708F,0x5BFE1BCC,0x3B21EE97,  
0xE29E749A };
```

```
/*TDES_ECB_DE=0x7C40AD58D73EB476DA4DB4236C37EC2263123150427DC5ECAADA52FA099A0E79  
BBD7D2862FD513928370A53D3A2AF9A8F70410C CC1BFE5B97EE213B9A749EE2*/
```

```
u32 TDES_CBC_EN [16] = {0x3723A485,0x3E2EEB10,0x9E5434C4,0x2692C8FD,0x978D5743,  
0x10CBCFD7,0x873A396C,0xD9CF6AEB,0x5C8953FC,0xD62F3744,  
0xDE2D0B60,0x1DA22B35,0x00793D6F,0x543CD424,0x833BE660,  
0x05703F52};
```

```
/*TDES_CBC_EN=0x85A4233710EB2E3EC434549EFDC8922643578D97D7CFCB106C393A87EB6ACFD9FC  
53895C44372FD6600B2DDE352BA21D6F3D7900 24D43C5460E63B83523F7005*/
```

```
u32 TDES_CBC_DE[16]={0xED617A1E,0xBFDAACE87,0x1FCAFD57,0x8D3ECA85,0x72154F73,  
0xF84F987F,0xE8AABC7B,0xEFB3677F,0xC5F7CC4C,0x9F3AD2C8,  
0x40779B72,0x00D7F205,0xF1A8B424,0x9A389EA2,0x3EEC58F4,  
0x1546667E};
```

```
/*TDES_CBC_DE=0x1E7A61ED87CEDABF57FDCA1F85CA3E8D734F15727F984FF87BBCAAEE87F67B3EF4  
CCCF7C5C8D23A9F729B774005F2D70024B4A8F1 A29E389AF458EC3E7E664615*/
```

```
TDES_Parm. in = in1;  
TDES_Parm. key = key1;  
TDES_Parm. out = out;  
TDES_Parm. inWordLen = 16;  
TDES_Parm. keyMode = TDES_2KEY;  
TDES_Parm. Mode = DES_ECB;  
TDES_Parm. En_De = DES_ENC;  
ret = DES_Init(&TDES_Parm);
```

```
ret=(DES_Crypto(&TDES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag1=0x5A5A5A5A;
}
else
{

if(Cmp_U32(TDES_ECB_EN,16, out,16))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }
}

TDES_Parm. En_De = DES_DEC;
ret = DES_Init(&TDES_Parm);
ret=(DES_Crypto(&TDES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{
```

```
if(Cmp_U32(TDES_ECB_DE,16, out,16))
{
    flag2=0x5A5A5A5A;
}
else
{
    flag2=0;
}
}
```

```
TDES_Parm. iv = iv1;
TDES_Parm. Mode = DES_CBC;
TDES_Parm. En_De = DES_ENC;
ret = DES_Init(&TDES_Parm);
ret=(DES_Crypto(&TDES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
```

```
if(Cmp_U32(TDES_CBC_EN,16, out,16))
{
    flag3=0x5A5A5A5A;
}
else
```

```
    {  
        flag3=0;  
    }  
}
```

```
TDES_Parm. iv = iv1;
```

```
TDES_Parm. En_De = DES_DEC;
```

```
ret = DES_Init(&TDES_Parm);
```

```
ret=(DES_Crypto(&TDES_Parm));
```

```
DES_Close();
```

```
if (ret!= DES_Crypto_OK)
```

```
{  
    flag4=0x5A5A5A5A;
```

```
}
```

```
else
```

```
{
```

```
    if(Cmp_U32(TDES_CBC_DE,16, out,16))
```

```
    {
```

```
        flag4=0x5A5A5A5A;
```

```
    }
```

```
else
```

```
{
```

```
    flag4=0;
```

```
}
```

```
}
```

```
if (flag1|flag2|flag3|flag4)
```

```
{
```

```
        return 0x5A5A5A5A;
    }
else
{
    return 0;
}
}
```

```
u32 TDES_3Key_test()
```

```
{
    u32 i,flag1,flag2,flag3,flag4,ret=0;
    DES_PARM TDES_Parm={0};
    u32 in1[16]= {
        0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,0xED8A71D7,0xA66B2FA
A,
        0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,0x05CDB663,0xF7D812E4,0x86341DEB,0xBA52B237
    };
    u32 key1[6]={0x675BE5D2,0x1641A6AD,0x14531A6B,0xEBFA006E,0x90DFD0CD,0x2D029B93};

    u32 iv1[2]={0xB5CC3A62,0xC96EF050};

    u32 out[16];
```

u32

TDES\_ECB\_EN[16]={0x5D6C633C,0x8EDFC4C7,0x3D02A02C,0x97431789,0x83EF4C36,0xFF591C67,0xE869DB08,0xAB82D05B,

0x11771439,0xDC6F79BB,0x5B46D128,0xF52114F5,0x2C758CB4,0x1A4D1A6A,0x0DC3FBCA,0x82222BB2};

u32

TDES\_ECB\_DE[16]={0x6780A75A,0x62EC1AC8,0xD0341FF5,0x2260C44E,0xF2720589,0xB0EBBBE0,0xBFE0991D,0x1EA78C1C,

0xBAB53D00,0xE3FA25D6,0x9430DEF4,0xC465511C,0xEE9D2DFB,0x9796AADC,0x4FFFEF58,0x172D00A2};

u32

TDES\_CBC\_EN[16]={0x048BD8AD,0xF98F2C51,0x5F6FD563,0xA26A1038,0x8017FC81,0xBBD5AF4C,0x0A7AE EFF,0xB7D428A1,

0x316E31F7,0xD8F283E1,0xDDD4395F,0x8076C2D0,0x0434D1E9,0xD1A94D4D,0xFF3E3B5E,0x77C93116};

u32

TDES\_CBC\_DE[16]={0xD24C9D38,0xAB82EA98,0xEC4AAF78,0x8DB239A7,0xD0565899,0xA4615EDD,0x78EF88CC,0x16B472C3,

0x573F4CD7,0x45910A7C,0x874D72AE,0x5E1D01CA,0x1374E950,0x56502FB2,0x4A32593B,0xE0F51246};;

TDES\_Parm. in = in1;

TDES\_Parm. key = key1;

TDES\_Parm. out = out;

TDES\_Parm. inWordLen = 16;

TDES\_Parm. keyMode = TDES\_3KEY;

TDES\_Parm. Mode = DES\_ECB;

TDES\_Parm. En\_De = DES\_ENC;

ret = DES\_Init(&TDES\_Parm);

```
DES_Crypto(&TDES_Parm);
DES_Close();

if(Cmp_U32(TDES_ECB_EN,16, out,16))
{
    flag1=0x5A5A5A5A;
}
else
{
    flag1=0;
}

TDES_Parm. En_De = DES_DEC;
ret = DES_Init(&TDES_Parm);
DES_Crypto(&TDES_Parm);
DES_Close();

if(Cmp_U32(TDES_ECB_DE,16, out,16))
{
    flag2=0x5A5A5A5A;
}
else
{
    flag2=0;
}

TDES_Parm. iv = iv1;

TDES_Parm. Mode = DES_CBC;
```

```
TDES_Parm. En_De = DES_ENC;
ret = DES_Init(&TDES_Parm);
DES_Crypto(&TDES_Parm);
DES_Close();

if(Cmp_U32(TDES_CBC_EN,16, out,16))
{
    flag3=0x5A5A5A5A;
}
else
{
    flag3=0;
}

TDES_Parm. iv = iv1;
TDES_Parm. En_De = DES_DEC;
ret = DES_Init(&TDES_Parm);
DES_Crypto(&TDES_Parm);
DES_Close();

if(Cmp_U32(TDES_CBC_DE,16, out,16))
{
    flag4=0x5A5A5A5A;
}
else
{
    flag4=0;
}
```

```
if (flag1|flag2|flag3|flag4)
{
    return 0x5A5A5A5A;
}

else
{
    return 0;
}
}
```

### iii. Appendix III AES algorithm library function call routine

```
u32 AES_128_test()
{
    u32 flag1,flag2,flag3,flag4,flag5,flag6;
    u32 ret;
    AES_PARM AES_Parm={0};
    /* If you need to modify the test instance, if the actual value of the parameter is 0x0102030405060708, enter
    0x04030201,0x08070605 during initialization because u32 data is stored in byte endian sequence. Unless otherwise
    specified, the routine parameters are set in this manner */
    u32 in [32] = {0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207D,
    0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,
    0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,
    0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,
    0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,
    0x85D4EFC8,0x9BAAD700};
    /*in=0xA570874A98DAC273D1522DF5464A885FD572CF8D7D200F2ACEF579749EBEB53FFE98793D6D585
    97C4B29E1309077E1B3BD0C08CA1379B42A0
    3B8093BE70F411BEF3742E65EBE7635F6AAD7E4B05F4919B1C32D817A9F33DD5F496FBE3A80B08C60970DC
    DD4D6C0A4BB1D389810CA699799DD673B1 A5A334CC8EFD48500D7AA9B*/
    u32 key [4] = {0x7FDDA35D,0x7D5C725B,0x1960F327,0x4FD9DDA2};
    /*key=0x5DA3DD7F5B725C7D27F36019A2DDD94F*/
    u32 iv [4] = {0x7B00FE39,0xD3E06638,0xD52BC983,0x38E98017};
    /*iv=0x39FE007B3866E0D383C92BD51780E938*/
    u32 out[32];
```

```
u32 AES_ECB_EN[32]={0xB24E5438,0x0145A303,0xC450A27F,0x2ADEEE70,0x906F314E,  
0xB24229AD,0x1312360E,0x949C8B22,0xE2C1BC02,0x1960239E,  
0xCAD2D5E5,0x8DC57DE2,0x13429CE1,0xE8FC0876,0xCA4581DB,  
0x08019050,0x4B2942F8,0xD6073C62,0x113FB648,0x1967CC27,  
0x250B9989,0x861180E0,0x1A450E0C,0x81D727AF,0xB679608E,  
0x53D31669,0x1D071E99,0x42CEB6DB,0x44094205,0xD0331668,  
0x2704B798,0x6E347E9C};
```

```
/*AES_ECB_EN=0x38544EB203A345017FA250C470EEDE2A4E316F90AD2942B20E361213228B9C9402BCC  
1E29E236019E5D5D2CAE27DC58DE19C42137  
608FCE8DB8145CA50900108F842294B623C07D648B63F1127CC671989990B25E08011860C0E451AAF27D7818E  
6079B66916D353991E071DDBB6CE420 5420944681633D098B704279C7E346E*/
```

```
u32 AES_ECB_DE[32]={0x818D1AFD,0xEC4B4F8E,0x69D9F9FF,0x5567B549,0x42DD5C4B,  
0x3BCA1DD3,0xF318E616,0x89297FEC,0x2A3E0A06,0xFDA90D61,  
0x93DCAE5D,0xCF1AFEAE,0x3CF5A889,0x4CFFEF3,0xB2C42607,  
0x37D43F8A,0x9C1CD1D8,0x2FE878E8,0x22D941C3,0x239B9D2D,  
0xD9FEB719,0xA4F9E01C,0xC9C39FE8,0x336B01FA,0xFD12E415,  
0x2B6A0006,0x4A35AFBC,0xA7942FAB,0x09DF0A3A,0x9545521B,  
0x7E009336,0x030A5DA5};
```

```
/*AES_ECB_DE=0xFD1A8D818E4F4BECFF9D96949B567554B5CDD42D31DCA3B16E618F3EC7F2989060  
A3E2A610DA9FD5DAEDC93AEFE1ACF89A8F53CE  
3EFFF4C0726C4B28A3FD437D8D11C9CE878E82FC341D9222D9D9B2319B7FED91CE0F9A4E89FC3C9FA016B3  
315E412FD06006A2BBCAF354AAB2F94A73 A0ADF091B5245953693007EA55D0A03*/
```

```
u32 AES_CBC_EN [32] = {0x8A83E006,0xAC3AB610,0x0CD2C4CB,0x21F22AA9,0x61963E3C,  
0x992FDE54,0x7E408523,0x749261FF,0xE159802D,0xBC807E3C,  
0x1C16AF67,0xE7574629,0x73573225,0xEE88600D,0x324FE0BB,  
0x7426A48C,0x8EA9E470,0x4DB1BE0F,0x9DC49C2E,0xAD41A05B,  
0x9E7C9143,0x15F55BF2,0xF4E7195D,0x2D9E1E46,0xB78E9809,
```

0xF8F831D0,0x12F1890A,0x0CABFF9C,0x49E6FCE6,0x6156CDA5,  
0xFFE38EF7,0x4962AF1D };

/\*AES\_CBC\_EN=0x06E0838A10B63AACCCBC4D20CA92AF2213C3E966154DE2F992385407EFF6192742D80  
59E13C7E80BC67AF161C294657E7253257730  
D6088EEBBE04F328CA4267470E4A98E0FBEB14D2E9CC49D5BA041AD43917C9EF25BF5155D19E7F4461E9E2  
D09988EB7D031F8F80A89F1129CFFAB0CE 6FCE649A5CD5661F78EE3FF1DAF6249\*/

u32 AES\_CBC\_DE[32]={0xFA8DE4C4,0x3FAB29B6,0xBCF2307C,0x6D8E355E,0x085A2CEE,  
0x4808C74B,0x0635B4C7,0xD6A135AA,0xA7F178D3,0xD7A62D1C,  
0xE7A55B93,0xF0AF4030,0x018C3077,0x30A6B78E,0x82250F4C,  
0x8435481A,0x5614DD65,0x055C01FB,0x19D0F9C0,0x38DA92CA,  
0x3FBC80F6,0x918F5E42,0x2D14351E,0x2A225E4A,0x7C3F27A4,  
0xF6599F7C,0xF45AE6E3,0x2B24AF91,0xC4D29D5A,0x318584CF,  
0xE6388E8D,0x946397B5};

u32 AES\_CTR\_EN[32]={0xF14C3DA0,0xA74E1089,0x81480939,0x5C8D4E8D,0x655E20AB,  
0x6D797028,0x1E355F48,0x58184929,0x52B1495A,0xC15EB91D,0xFBD499AB,  
0xF59B39FE,0x96DAE1C3,0x6ECC9CDA,0xDA1FB535,0xAA1C74B2,0xA3F19C5E,  
0x9944E1A6,0xDAA05E9A,0xB96278E3,0x1E4915FC,0xB77FBB2,0x92BA80B9,  
0xCA97857E,0x509D0365,0x78A6FD99,0xB56F5B3C,0xFB5FF5B2,0xF9E928C6,  
0xBC28AE3A,0xD8B82D7A,0xA99BF98D};

u32 AES\_CTR\_DE[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207D,  
0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,  
0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,  
0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,  
0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,  
0x85D4EFC8,0x9BAAD700};

```
/*AES_CBC_DE=0xC4E48DFAB629AB3F7C30F2BC5E358E6DEE2C5A084BC70848C7B43506AA35A1D6D3
78F1A71C2DA6D7935BA5E73040AFF077308C018
EB7A6304C0F25821A48358465DD1456FB015C05C0F9D019CA92DA38F680BC3F425E8F911E35142D4A5E222A
A4273F7C7C9F59F6E3E65AF491AF242B5 A9DD2C4CF8485318D8E38E6B5976394*/
```

```
Cpy_U32(out, in,32);
```

```
AES_Parm. in = out;
```

```
AES_Parm. key = key;
```

```
AES_Parm. iv = iv;
```

```
AES_Parm. out = out;
```

```
AES_Parm. keyWordLen = 4;
```

```
AES_Parm. inWordLen = 32;
```

```
AES_Parm. Mode = AES_ECB;
```

```
AES_Parm. En_De = AES_ENC;
```

```
ret =AES_Init(&AES_Parm);
```

```
ret = AES_Crypto(&AES_Parm);
```

```
AES_Close();
```

```
if(ret!= AES_Crypto_OK)
```

```
{
```

```
    flag1=0x5A5A5A5A;
```

```
}
```

```
else
```

```
{
```

```
    if(Cmp_U32(AES_ECB_EN, 32, out, 32))
```

```
    {
```

```
        flag1=0x5A5A5A5A;
```

```
    }
```

```
    else
    {
        flag1=0;
    }
}
Cpy_U32(out, in,32);
AES_Parm. En_De = AES_DEC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{

    if(Cmp_U32(AES_ECB_DE, 32, out, 32))
    {
        flag2=0x5A5A5A5A;
    }
    else
    {
        flag2=0;
    }
}
//CBC
Cpy_U32(out, in,32);
AES_Parm. Mode = AES_CBC;
```

```
AES_Parm. En_De = AES_ENC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(AES_CBC_EN, 32, out, 32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}
Cpy_U32(out, in,32);
AES_Parm. En_De = AES_DEC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
```

```
{
    if(Cmp_U32(AES_CBC_DE, 32, out, 32))
    {
        flag4=0x5A5A5A5A;
    }
    else
    {
        flag4=0;
    }
}

//CTR
Cpy_U32(out, in,32);
AES_Parm. Mode = AES_CTR;
AES_Parm. En_De = AES_ENC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag5=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(AES_CTR_EN, 32, out, 32))
    {
        flag5=0x5A5A5A5A;
    }
    else
    {
```

```
        flag5=0;
    }
}
Cpy_U32(out, AES_CTR_EN,32);
AES_Parm. En_De = AES_DEC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag6=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(AES_CTR_DE, 32, out, 32))
    {
        flag6=0x5A5A5A5A;
    }
    else
    {
        flag6=0;
    }
}

if (flag1|flag2|flag3|flag4|flag5|flag6)
{
    return 0x5A5A5A5A;
}
else
```

```
{
    return 0;
}

}

u32 AES_192_test()
{
    u32 flag1,flag2,flag3,flag4,flag5,flag6,ret=0;
    AES_PARM AES_Parm={0};

    u32
in[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9E1,0xE1A33077,0xE3AEA2
15,
0x2670CBEB,0x160CA5C2,0x86808BEA,0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,0xC4BD0D05,
0xAB21B81D,0x7477E171,0xDE6FFE4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,0x9C3A69BA,
0x6F7CD7A9,0x2BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,0x03D4B04F,0xE0EAAC56,0x73CC7E9C};
    u32 key[6]={0xA1148977,0xCFA42A1F,0x9D983F36,0x521C1313,0xDAD2CB6F,0xC6254819};

    u32 iv[4]={0xFCAA7077,0x44DB6BB5,0xDC74178D,0xA91A44D6};
    u32 out[32];

    u32
AES_ECB_EN[32]={0x9FCB396D,0xF9A6B55C,0x4CCE7669,0x917CAF2F,0x71F8907D,0xC6893936,0x5ABA1DF
B,0xA933FF81,
```

0xBD33847F,0x0F1B2F6C,0x1B4AACA7,0xE555E2EE,0x0CBD4683,0x76ECD138,0x7BFE81E8,0xE05FE788,

0xAF688124,0xED29ACF2,0xCE424458,0x8E304A1C,0xE5A21E6C,0x3C7D433A,0x32DC028D,0x697F9624,

0xB451070E,0xF82A4488,0x33D99F4C,0x7FBBCC3E,0x8BB01E57,0x0C1EE01B,0x6D96FF7F,0xDEC84BD8}

;

u32

AES\_ECB\_DE[32]={0x41F29D18,0x13C52105,0xB24DBDDD,0x46B6BAB9,0x95F63F1A,0x28B24F73,0xAA77429  
3,0xA086E548,

0xD446667D,0xF8D67CCE,0x7AC5BD02,0xE43EE791,0x25B857B4,0x30A3D7FB,0x8DB4C416,0xAE6B0B0C

,

0x0F7E89E1,0xBA900B96,0x516EC69B,0xBED1D082,0x3590FD32,0x878C5EE5,0x91B71430,0x6A005A7F,

0x0627EF04,0x28D96A77,0xF8DCDCFC,0x790D0304,0x02149E37,0xDC8E518D,0x80D75D77,0x80670408};

u32

AES\_CBC\_EN[32]={0xE5682F2E,0x07A087E9,0x37D60ED6,0x41262C81,0xD69A23B5,0x1800A3FD,0xAC50301D  
,0xB12F3C5E,

0x568A1F62,0xC1057524,0x7E7D09BC,0x26F42541,0x5C2FB09B,0x12C68EFC,0xE03B2AF8,0x6E2C9934,

0xD805445F,0x3876A6E4,0xCA85688F,0xD1116501,0x2DE18902,0xCBFD9B2,0x57911796,0x0719A673,

0x3915B680,0x3B760C23,0x23F715DE,0x6D3425B9,0x9C339EF5,0x6C91D7B0,0x050E91DA,0x286AB477};

u32

AES\_CBC\_DE[32]={0xBD58ED6F,0x571E4AB0,0x6E39AA50,0xEFACFE6F,0xCFB4F836,0x21432C5A,0x43CA36  
B8,0x148505B7,

0x6E05BE79,0x26A1C52F,0x9B668D75,0x07904584,0x03C89C5F,0x26AF7239,0x0B344FFC,0x9311957F,

0xBE10E141,0xA875B40E,0xDB762AC4,0x7A6CDD87,0x9EB1452F,0xF3FBBF94,0x4FD8EAC4,0xD20B3287,

0xA288EAA5,0x34AE4EED,0x4A1074FA,0xE5376ABE,0x6D68499E,0xF757B012,0xF8634844,0xAF390CFF};

u32

AES\_CTR\_EN[32]={0xF4EB3E15,0xCEC90E4B,0x1708E770,0x6A1297BB,0x045A69FD,0x7FC870A7,0x56BE6A2  
2,0x5A912CEA,

0xC22E6811,0x37177967,0x68D08A6A,0xCECA04AE,0x30EA7217,0x16992F79,0xF0DD4DAD,0x4710126B,0xCC0  
6BD7F,

0x03093EE5,0x596D2B9B,0xD9844F7C,0x130D4E24,0xD6C87ABF,0xE1745614,0xEF260225,0x0F90C354,0x7557E  
159,

0x4CBC3789,0xDB0552F8,0x28F27315,0x046363A6,0xAF1F0089,0x29AC2CC1};

u32

AES\_CTR\_DE[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9E1,0xE1A3307  
7,0xE3AEA215,

0x2670CBEB,0x160CA5C2,0x86808BEA,0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,0xC4BD0D05,

0xAB21B81D,0x7477E171,0xDE6FFEF4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,0x9C3A69BA,

```
0x6F7CD7A9,0x2BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,0x03D4B04F,0xE0EAAC56,0x73CC7E9C};
```

```
AES_Parm. in = in;
```

```
AES_Parm. key = key;
```

```
AES_Parm. iv = iv;
```

```
AES_Parm. out = out;
```

```
AES_Parm. keyWordLen = 6;
```

```
AES_Parm. inWordLen = 32;
```

```
AES_Parm. Mode = AES_ECB;
```

```
AES_Parm. En_De = AES_ENC;
```

```
ret =AES_Init(&AES_Parm);
```

```
ret =AES_Crypto(&AES_Parm);
```

```
AES_Close();
```

```
if(Cmp_U32(AES_ECB_EN, 32, out, 32))
```

```
{
```

```
    flag1=0x5A5A5A5A;
```

```
}
```

```
else
```

```
{
```

```
    flag1=0;
```

```
}
```

```
AES_Parm. En_De = AES_DEC;
```

```
ret =AES_Init(&AES_Parm);
```

```
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_ECB_DE, 32, out, 32))
```

```
{  
    flag2=0x5A5A5A5A;
```

```
}
```

```
else
```

```
{  
    flag2=0;
```

```
}
```

```
//cbc
```

```
AES_Parm. Mode = AES_CBC;
```

```
AES_Parm. En_De = AES_ENC;
```

```
ret =AES_Init(&AES_Parm);
```

```
ret =AES_Crypto(&AES_Parm);
```

```
AES_Close();
```

```
if(Cmp_U32(AES_CBC_EN, 32, out, 32))
```

```
{
```

```
    flag3=0x5A5A5A5A;
```

```
}
```

```
else
```

```
{  
    flag3=0;
```

```
}
```

```
AES_Parm. En_De = AES_DEC;
```

```
ret =AES_Init(&AES_Parm);
```

```
ret =AES_Crypto(&AES_Parm);  
AES_Close();  
  
if(Cmp_U32(AES_CBC_DE, 32, out, 32))  
{  
    flag4=0x5A5A5A5A;  
}  
else  
{  
    flag4=0;  
}
```

```
//ctr
```

```
AES_Parm. Mode = AES_CTR;  
AES_Parm. En_De = AES_ENC;  
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();  
  
if(Cmp_U32(AES_CTR_EN, 32, out, 32))  
{  
    flag5=0x5A5A5A5A;  
}  
else  
{  
    flag5=0;  
}  
  
AES_Parm. in = AES_CTR_EN;  
AES_Parm. En_De = AES_DEC;
```

```
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_CTR_DE, 32, out, 32))  
{  
    flag6=0x5A5A5A5A;  
}  
else  
{  
    flag6=0;  
}
```

```
if (flag1|flag2|flag3|flag4|flag5|flag6)  
{  
    return 0x5A5A5A5A;  
}  
else  
{  
    return 0;  
}  
}
```

```
u32 AES_256_test()  
{  
    u32 flag1,flag2,flag3,flag4,flag5,flag6,ret=0;  
    AES_PARM AES_Parm={0};
```

u32

in[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B3C0,0x8BB97CB6,0x99EA0BE6,

0x8B338E1D,0xFE104A1C,0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBDC3A3,0xA7C27969,

0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x438105F6,

0x986F81C9,0x639F51B2,0xA3169082,0x6CD5570C,0x39B678E4,0x84986F66,0x94BB95FA,0x976D9797};

u32

key[8]={0xB2591B82,0xD25676DB,0x2546F076,0xC8D01753,0xB4A620E7,0x4AADD91D,0x2E5EDF9B,0x596C1146};

u32 iv[4]={0xF0E72786,0xD272F169,0x0ECED17B,0x29D34319};

u32 out[32];

u32

AES\_ECB\_EN[32]={0x5766DACC,0x50DBB1F9,0x58720E73,0x2182AA3E,0x7D5A6D4D,0xA07EF43D,0x5A533E1E,0x34816CF3,

0xBA23F9CD,0x99A7BD14,0x6789D933,0xD14B2F0D,0xAF53E19E,0xB88DA31F,0xEFBE0472,0x03F077B1,

0x4489E477,0x97161707,0x6C24CB62,0x0FF361DC,0x60BBD2CF,0xEB7AB0C1,0xFA3421E5,0x2F5DB80E,

0x2D61A7CD,0x22988E98,0x51B195AF,0x22C8A4C0,0x7F8E90C3,0x6690789A,0x48AF0FAF,0xAC16F7A6};

u32

AES\_ECB\_DE[32]={0x0ADBDA93,0x93C512ED,0x6A99A60B,0x0A1841B5,0x135E685D,0xB9ADC987,0x6262573F,0x9090A7D3,

0x2B7DDAA3,0x7370FB9D,0xE7E739C6,0xCA013CA6,0x3509E08F,0x74A21641,0x3D2C9527,0xF8DF90F0,  
0xED8209E9,0x9DD57975,0x0A506603,0x7C2EFD3B,0x0937237E,0x2828BAAF,0x245E9D40,0xF3BB882A,  
0x66E82B24,0xF3E778E7,0x386802D1,0xD74C7057,0xEF8525C8,0x1EB7AA48,0x362EACDD,0x8AA0F286};

u32

AES\_CBC\_EN[32]={0x39AD6F3A,0xF8E3E1DD,0x2209A14B,0x241642CC,0x83FA4820,0xD82816B3,0xEF66B17  
A,0xB5B49FCC,

0xA7540FD7,0xCC11801C,0xC6126D93,0x8E6C259A,0x626135EB,0x3FEA411B,0x45FF91A3,0x1B91B51A,  
0x9169DD4C,0x2F42A1E6,0x4299E687,0xEB9FBAA4,0x3B667902,0xDCB4117A,0x45B78A05,0x5FECBFA7,  
0x54C54A81,0xBDF538B1,0xF2D5804D,0x568910A8,0x41655B32,0xD47D533B,0x5A82D212,0x63C07B46};

u32

AES\_CBC\_DE[32]={0xFA3CFD15,0x41B7E384,0x64577770,0x23CB02AC,0x95811940,0x0069DBAA,0x7154DC12  
,0xC335689C,

0x9682708F,0xC7A4485D,0x6C5E4570,0x53EB3740,0xBE3A6E92,0x8AB25C5D,0x733F40C4,0x505915DF,  
0x8AD021A8,0x00CA8C94,0xE5EDA5A0,0xDBEC8452,0x0D42E557,0xFCC3A85F,0x612E2967,0x0A92ED3C

,  
0x3E1FDF82,0xD97A448C,0x5D4E5630,0x94CD75A1,0x77EAA401,0x7D28FBFA,0x95383C5F,0xE675A58A};

u32

AES\_CTR\_EN[32]={0x85F1DD33,0xAE808F2F,0x26A40960,0xB2020DF8,0xB6C2006E,0xA22A35F6,0x33BB584A  
,0xBFEA7F68,

0x73E54E78,0xF3EB0368,0x80816676,0x6109DE39,0xE0001920,0x8D2B18B8,0x0E46A012,0xE43F1DD1,0x3  
CA4BC36,

0xD5101452,0x83020170,0x4B752F62,0x3D27A004,0x3C18B5DB,0x99DA9032,0xEA59B340,0x79BBD087,0x  
2EF8CB3D,

0xDC32D3CA,0x30F577EA,0x56774C66,0xC33DA1F8,0x0288B1D6,0x091C9666};

u32

AES\_CTR\_DE[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B3C0,0x8BB97  
CB6,0x99EA0BE6,

0x8B338E1D,0xFE104A1C,0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBDC3A3,0xA7C27969,

0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x438105F6,

0x986F81C9,0x639F51B2,0xA3169082,0x6CD5570C,0x39B678E4,0x84986F66,0x94BB95FA,0x976D9797};

AES\_Parm. in = in;

AES\_Parm. key = key;

AES\_Parm. iv = iv;

AES\_Parm. out = out;

AES\_Parm. keyWordLen = 8;

AES\_Parm. inWordLen = 32;

AES\_Parm. Mode = AES\_ECB;

```
AES_Parm. En_De = AES_ENC;  
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_ECB_EN, 32, out, 32))
```

```
{  
    flag1=0x5A5A5A5A;  
}
```

```
else
```

```
{  
    flag1=0;  
}
```

```
AES_Parm. En_De = AES_DEC;
```

```
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_ECB_DE, 32, out, 32))
```

```
{  
    flag2=0x5A5A5A5A;  
}
```

```
else
```

```
{  
    flag2=0;  
}
```

```
//CBC
```

```
AES_Parm. Mode = AES_CBC;
AES_Parm. En_De = AES_ENC;
ret =AES_Init(&AES_Parm);
ret =AES_Crypto(&AES_Parm);
AES_Close();

if(Cmp_U32(AES_CBC_EN, 32, out, 32))
{
    flag3=0x5A5A5A5A;
}
else
{
    flag3=0;
}

AES_Parm. En_De = AES_DEC;
ret =AES_Init(&AES_Parm);
ret =AES_Crypto(&AES_Parm);
AES_Close();

if(Cmp_U32(AES_CBC_DE, 32, out, 32))
{
    flag4=0x5A5A5A5A;
}
else
{
    flag4=0;
}

//CTR
```

```
AES_Parm. Mode = AES_CTR;  
AES_Parm. En_De = AES_ENC;  
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_CTR_EN, 32, out, 32))
```

```
{  
    flag5=0x5A5A5A5A;
```

```
}  
else
```

```
{  
    flag5=0;  
}
```

```
AES_Parm. in = AES_CTR_EN;
```

```
AES_Parm. En_De = AES_DEC;  
ret =AES_Init(&AES_Parm);  
ret =AES_Crypto(&AES_Parm);  
AES_Close();
```

```
if(Cmp_U32(AES_CTR_DE, 32, out, 32))
```

```
{  
    flag6=0x5A5A5A5A;
```

```
}  
else
```

```
{  
    flag6=0;  
}
```

```
if (flag1|flag2|flag3|flag4|flag5|flag6)
{
    return 0x5A5A5A5A;
}
else
{
    return 0;
}
}
```

## iv. Appendix IV HASH Library function Call routines

```
u32 MD5_fixed_steps_test(void)
{
    u8 out[16];
    char in[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    u8 MD5_fixout[16]=
    {
        0xd1,0x74,0xab,0x98,0xd2,0x77,0xd9,0xf5,0xa5,0x61,0x1c,0x2c,0x9f,0x41,0x9d,0x9f
    };
    HASH_CTX ctx[1];
    ctx->hashAlg = HASH_ALG_MD5;
    ctx->sequence = HASH_SEQUENCE_TRUE;

    HASH_Init(ctx);
    HASH_Start(ctx);
    HASH_Update(ctx, (u8*)in, 28);
    HASH_Update(ctx, ((u8*)in)+ 28, 28);
    HASH_Update(ctx, ((u8*)in)+ 56, 6);
    HASH_Complete(ctx, out);
    HASH_Close();
    if(memcmp(out,MD5_fixout,16))
    {
        //printf("MD5-FIX-Test fail\r\n");
        return 0x5a5a5a5a;
    }
    else
```

```
{
    //printf("MD5-FIX-Test success\r\n");
    return 0;
}
//return 0;
}
// SM3 fixed step test cases
u32 SM3_test(void)
{
    u8 out[32];
    //SM3 fixed step hash
    // Step message
    u8 SM3_fixin[48*3]=
    {
        0x02,0x89,0x00,0xD4,0x66,0x14,0xF9,0xA2,0x9E,0xC9,
        0xBC,0x05,0x5B,0xBE,0x10,0x33,0x0F,0x41,0x1B,0xDF,
        0x9A,0x20,0x44,0x2C,0xB1,0x51,0xBD,0xCA,0x8D,0xDB,
        0xAD,0x86,0x46,0x48,0xA3,0xC6,0x34,0x27,0xEB,0x8B,
        0x05,0x57,0x40,0x90,0x52,0xE9,0x92,0xA3,0x79,0xBB,
        0x2D,0x3D,0x48,0xEC,0xC2,0x9A,0x91,0xBE,0x47,0xD0,
        0x7C,0x6E,0x6B,0x4E,0xEF,0x68,0x46,0x03,0x72,0x44,
        0xD5,0xCA,0x96,0x17,0xE3,0xFB,0x92,0x3E,0x41,0x27,
        0x55,0x16,0x77,0x9F,0x93,0x1A,0x60,0x78,0x83,0x13,
        0xDF,0x76,0x09,0xC0,0xC1,0xBF,0x6F,0x0F,0xEB,0x11,
        0x6D,0x6A,0x0B,0x8C,0x0A,0x43,0x38,0xE6,0x05,0x8E,
        0xCD,0x84,0xE7,0xA3,0x9B,0x9D,0x6B,0x75,0x91,0xEB,
        0xA5,0x28,0xCF,0xEF,0x4F,0xED,0x61,0x35,0x43,0x2D,
        0x33,0xE2,0x25,0x99,0x14,0xB1,0x05,0xA8,0xFF,0x04,
        0x9C,0xC2,0x29,0x05
    }
```

```
};  
  
// Correct message digest  
u8 SM3_fixout[32]=  
{  
    0xC7,0x8B,0xF5,0x97,0x52,0xCD,0xFE,0x9F,0x70,0x21,  
    0x4F,0x5D,0x88,0x92,0x2E,0x60,0x35,0x22,0x3B,0x66,  
    0x94,0xFD,0x08,0x96,0x5E,0x26,0x44,0xF9,0x72,0xFE,  
    0xE2,0xB2  
};  
  
u8 i,byteLen=48;  
HASH_CTX ctx[1];  
  
// Set the operation to SM3  
// CTX ->hashAlg  
// such as HASH_ALG_SHA1,  
    // HASH_ALG_SHA224,  
    // HASH_ALG_SHA256,  
    //HASH_ALG_SM3  
ctx->hashAlg = HASH_ALG_SM3;  
  
ctx->sequence = HASH_SEQUENCE_TRUE;  
  
HASH_Init(ctx);  
HASH_Start(ctx);  
for(i=0;i<3;i++)  
{  
    HASH_Update(ctx,SM3_fixin+i*byteLen,byteLen);  
}  
  
HASH_Complete(ctx, out);  
HASH_Close();  
if (memcmp(out,SM3_fixout,32))  
{
```

```
// Step SM3 test fails
printf("SM3-FIX-Test fail\r\n");
return HASH_ATTACK;
}
else
{
// Step SM3 test successful
printf("SM3-FIX-Test success\r\n");
}
return SM3_Hash_OK;
}
// This routine performs a single step hash on the hash sha1/224/256
u32 HASH_test(void)
{
u32 TEST_BUF[200];
u8 in[48]=
{
0x1C,0xBB,0x9F,0x4A,0x43,0x6A,0xAD,0x81,0xFE,0x4F,0x52,0x4A,0x0A,0x76,0x22,0xC8,0x4F,0x90,0x18,
0x30,0xA4,0xD2,0x8C,0x6A,0xC3,0x40,0xA0,0xBD,0x0A,0x6A,0x37,0x18,0x8D,0x19,0x9D,0xE5,0xCB,0x8
4,0xA3,0xFC,0x39,0xDE,0x8C,0xD6,0xFC,0x2F,0xC8,0x88
};
u8 in2[10] = {0x1C,0x61,0xAD,0x6C,0x05,0xF3,0x98,0xA4,0x4C,0xFD};
u8 out[64];
u8 sha1_out[20]=
{
0x0E,0xEC,0x49,0xC5,0x36,0xBB,0xD7,0x87,0xD2,0xE2,0x0C,0x97,0xC4,0xF8,0x65,0x7C,0xCC,0x74,0x8D
,0x1E
};
u8 sha224_out[28]=
```

```
{
    0xC1,0x44,0x4F,0xD0,0xB8,0xA9,0xA3,0xD9,0xE8,0x04,0xA0,0xD1,0x9E,0x38,0xF3,0x5E,0x85,0xB4,0x0
    F,0x10,0x5A,0x1C,0x48,0xC4,0xF2,0x40,0x10,0x48
};
u8 sha256_out[32]=
{
    0xE2,0xE4,0x2C,0x8A,0x01,0x1A,0xE7,0x98,0x67,0x74,0x93,0xAF,0x9D,0x65,0x99,0xB3,0xA1,0x68,0x8B
    ,0x5A,0xF1,0x32,0x3D,0x5B,0xFF,0xFB,0x12,0x30,0x94,0xE4,0x81,0xDD
};

u8 SM3_out[32]=
{
    0xBD,0x77,0x63,0x33,0x0A,0x71,0x19,0x5C,0x5D,0x26,0xE7,0x99,0x7B,0x41,0x22,0xB0,0xBC,0xB0,0x
    BE,0x52,0x3E,0xDA,0x0F,0xBE,0xE6,0xA4,0x33,0x96,0xB8,0x83,0x76,0xD4
};

u32 ret=0x5123;

#if 1
    HASH_CTX *ctx;
    ctx = (HASH_CTX*)(TEST_BUF);
    ctx->hashAlg = HASH_ALG_SHA1;
    ctx->sequence = HASH_SEQUENCE_FALSE;
    HASH_Init(ctx);
    HASH_Start(ctx);
    HASH_Update(ctx, in, 48);
    ret=HASH_Complete(ctx, out);
    HASH_Close();
    if (memcmp(out,sha1_out,20))
```

```
{
    return 0x5a5a5a5a;
}
else
{
    printf("SHA1-Test success\r\n");
}
ctx->hashAlg = HASH_ALG_SHA224;
ctx->sequence = HASH_SEQUENCE_FALSE;
HASH_Init(ctx);
HASH_Start(ctx);
HASH_Update(ctx, in, 48);
//HASH_Update(ctx, in2, 10);
ret=HASH_Complete(ctx, out);
HASH_Close();
if (memcmp(out,sha224_out,28))
{
    return 0x5a5a5a5a;
}
else
{
    printf("SHA224-Test success\r\n");
}

ctx->hashAlg = HASH_ALG_SHA256;
ctx->sequence = HASH_SEQUENCE_FALSE;
HASH_Init(ctx);
HASH_Start(ctx);
HASH_Update(ctx, in, 48);
```

```
    ret=HASH_Complete(ctx, out);
    HASH_Close();
    if(memcmp(out,sha256_out,32))
    {
        return 0x5a5a5a5a;
    }
    else
    {
        printf("SHA256-Test success\r\n");
    }
#endif
    return 0;
}
```

## v. Appendix V SM4 algorithm library function call routine

```
u32 SM4_test(void)
```

```
{
```

```
    u32 flag1,flag2,flag3,flag4;
```

```
    u32 ret;
```

```
    SM4_PARM SM4_Parm={0};
```

*/\* If you need to modify the test instance, if the actual value of the parameter is 0x0102030405060708, enter 0x04030201,0x08070605 during initialization because u32 data is stored in byte endian sequence. Unless otherwise specified, the routine parameters are set in this manner \*/*

```
    u32 in1[32]={
```

```
        0x4B551C70,0xD54DA600,0xBAA2CA7F,0x0ABA6CD8,0x97BC9D7D,0xAD650748,
```

```
        0x0590F143,0x7288FD0F,0x9EDF1005,0xB7D4A607,0x8ED480C9,0x34FD4C59,
```

```
0x97C9286E,0xD0A23857,0x1ABE2026,0x6163578A,0xF5FB4FB4,0x72DB71B7,  
0x21217431,0xF8BE4ECA,0xB73D1018,0xACD37812,0x3FF19EE7,0x4C9575BE,  
0xF1FB289E,0x33694113,0x8EC5BB10,0x3B1DFF5F,0xA9D6A5A5,0xB98D90C8,  
0x91AB4E89,0x804343FD
```

```
};
```

```
u32 key1[4]={0x84853E30,0xB3D3154D,0x9A887F49,0xDC65910A};
```

```
u32 iv1[4]={0x2FA6B65A,0x1D0EC205,0xB90B8620,0x42E74F58};
```

```
u32 out[32];
```

```
u32 SM4_ECB_EN[32]={0xD61A389C,0xE136A0AD,0xBD626B7E,0x4277F173,0xAF3E5E82,  
0x876D84DF,0x7A065B7B,0x1CBBFFA8,0xC57C31DC,0x5BD86AFC,  
0x0825EAEF,0x600162A4,0x3E4787AC,0x58B32579,0x3A9135BF,  
0xB806A17C,0x9854F4C4,0x065CD28F,0x68FDF21F,0x9CA62C4C,  
0x5B2FA76E,0xEC693A2B,0xF028ADF6,0xFAA2ED18,0x6395B4B1,  
0x7A9B0069,0x9D55E04C,0xA5CDC23F,0x7FC56C92,0x89F199A1,  
0xF228D9E1,0xD705050A};
```

```
/*SM4_ECB_EN=0x9C381AD6ADA036E17E6B62BD73F17742825E3EAFDF846D877B5B067AA8FFBB1CDC  
317CC5FC6AD85BEFEA2508A4620160AC87473E7925B358BF35913A7CA106B8C4F454988FD25C061FF2FD684  
C2CA69C6EA72F5B2B3A69ECF6AD28F018EDA2FAB1B4956369009B7A4CE0559D3FC2CDA5926CC57FA199F1  
89E1D928F20A0505D7*/
```

```
u32 SM4_ECB_DE[32]={0x3107DFA0,0xC1EE3D0A,0x9025F9D5,0x90ACC081,0x7A72F90A,  
0x6481F1CE,0x76DF5450,0xCD262ACF,0xCE8E3C3B,0x208B7390,  
0xC9F8F526,0x1A73FFCC,0x0AB6E26F,0xA02B544A,0x760CD602,  
0x6D250CA4,0x2477FF67,0x44CBC39E,0x84ECF5CC,0x7DF30644,  
0x8746D41C,0xCB42B9EC,0xE975598C,0x28756C41,0x64C3C870,  
0x9EA8CBB3,0xBA2FA98E,0x1B10BA7B,0x1C50E8A0,0x1EE697FD,
```

0xA4E2DDD5,0xBB29D912};

```
/*SM4_ECB_DE=0xA0DF07310A3DEEC1D5F9259081C0AC900AF9727ACEF181645054DF76CF2A26CD3B3  
C8ECE90738B2026F5F8C9CCFF731A6FE2B60A4A542BA002D60C76A40C256D67FF77249EC3CB44CCF5EC844  
406F37D1CD44687ECB942CB8C5975E9416C752870C8C364B3CBA89E8EA92FBA7BBA101BA0E8501CFD97E61  
ED5DDE2A412D929BB*/
```

u32 SM4\_CBC\_EN[32]={0x304E1C3C,0x10DA649D,0x5EBCB5BE,0x2964AD84,0x18599756,  
0x2106AAD2,0x84364B24,0x57A9E62D,0xD160B03B,0x58293A74,  
0xEE57389F,0x398E69C2,0x63FD0959,0x5B4584FD,0x4DA6E8BE,  
0x578E4501,0x74B0159B,0x570E8604,0x38E2DB49,0xE028387E,  
0xCDDE4984,0x6B717E9F,0xE516D698,0x6520025E,0xC8D187A7,  
0x6E08373F,0xC3472666,0x654A0D41,0x7F363B95,0xAD8EB5D2,  
0x01F0F12A,0x8169D65A};

```
/*SM4_CBC_EN=0x3C1C4E309D64DA10BEB5BC5E84AD642956975918D2AA0621244B36842DE6A9573BB  
060D1743A29589F3857EEC2698E395909FD63FD84455BBEE8A64D01458E579B15B07404860E5749DBE2387E38  
28E08449DECD9F7E716B98D616E55E022065A787D1C83F37086E662647C3410D4A65953B367FD2B58EAD2AF1  
F0015AD66981*/
```

u32 SM4\_CBC\_DE[32]={0x1EA169FA,0xDCE0FF0F,0x292E7FF5,0xD24B8FD9,0x3127E57A,  
0xB1CC57CE,0xCC7D9E2F,0xC79C4617,0x5932A146,0x8DEE74D8,  
0xCC680465,0x68FB02C3,0x9469F26A,0x17FFF24D,0xF8D856CB,  
0x59D840FD,0xB3BED709,0x9469FBC9,0x9E52D5EA,0x1C9051CE,  
0x72BD7BA8,0xB999C85B,0xC8542DBD,0xD0CB228B,0xD3FED868,  
0x327BB3A1,0x85DE3769,0x5785CFC5,0xEDABC03E,0x2D8FD6EE,  
0x2A2766C5,0x8034264D};

```
/*SM4_CBC_DE=0xFA69A11E0FFFE0DCF57F2E29D98F4BD27AE52731CE57CCB12F9E7DCC17469CC746  
A13259D874EE8D650468CCC302FB686AF269944DF2FF17CB56D8F8FD40D85909D7BEB3C9FB6994EAD5529E  
CE51901CA87BBD725BC899B9BD2D54C88B22CBD068D8FED3A1B37B326937DE85C5CF85573EC0ABEDED6  
8F2DC566272A4D263480*/
```

```
Cpy_U32(out, in1,32);
SM4_Parm.in = out;
SM4_Parm.key = key1;
SM4_Parm.out = out;
SM4_Parm.inWordLen = 32;
SM4_Parm.workingMode = SM4_ECB;
SM4_Parm.EnDeMode = SM4_ENC;
ret=SM4_Init(&SM4_Parm);
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag1=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM4_ECB_EN,32, out,32))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }
}
Cpy_U32(out, in1,32);
SM4_Parm.EnDeMode = SM4_DEC;
ret=SM4_Init(&SM4_Parm);
```

```
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{

    if(Cmp_U32(SM4_ECB_DE,32, out,32))
    {
        flag2=0x5A5A5A5A;
    }
    else
    {
        flag2=0;
    }
}
Cpy_U32(out, in1,32);
SM4_Parm.iv = iv1;
SM4_Parm.workingMode = SM4_CBC;
SM4_Parm.EnDeMode = SM4_ENC;
ret=SM4_Init(&SM4_Parm);
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
```

```
else
{

    if(Cmp_U32(SM4_CBC_EN,32, out,32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}

Cpy_U32(out, in1,32);
SM4_Parm.iv= iv1;
SM4_Parm.EnDeMode = SM4_DEC;
ret=SM4_Init(&SM4_Parm);
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
{

    if(Cmp_U32(SM4_CBC_DE,32, out,32))
    {
        flag4=0x5A5A5A5A;
    }
}
```

```
    else
    {
        flag4=0;
    }
}

if (flag1|flag2|flag3|flag4)
{
    return 0x5A5A5A5A;
}
else
{
    return 0;
}
}
```

## vi. Appendix VI RNG library call routine

```
#define POKER RAND_BYTE 40 //320bit

u32 TrueRand_Poker_Test(void)
{
    u16 count[16] = {0};
    u32 sum = 0;
    u8 rand[POKER RAND_BYTE];
    u8 i, j, k, tmp;

    GetTrueRand_U32((u32*)rand, POKER RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER RAND_BYTE);
    //GetPseudoRand_U32((u32*)rand,POKER RAND_BYTE>>2);
    for(j = 0; j < POKER RAND_BYTE; j++)
    {
        for(k = 0; k < 2; k++)
        {
            (k == 1) ? tmp = (rand[j] >> 4) : (tmp = (rand[j] & 0x0F));
            for(i = 0; i < 16; i++)
            {
                if(tmp==i) count[i]++;
            }
        }
    }
    for(i = 0; i < 16; i++)
    {
        sum += ((u32)count[i]) * count[i];
    }
}
```

```
if(405 < sum && sum < 687)
    return 0;
else
    return 1;
}
u32 PseudoRand_Poker_Test(void)
{
    u16 count[16] = {0};
    u32 sum = 0;
    u8  rand[POKER_RAND_BYTE];
    u8 i, j, k, tmp;

    //GetTrueRand_U32((u32*)rand, POKER_RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER_RAND_BYTE);
    GetPseudoRand_U32((u32*)rand,POKER_RAND_BYTE>>2,NULL);
    for(j = 0; j < POKER_RAND_BYTE; j++)
    {
        for(k = 0; k < 2; k++)
        {
            (k == 1) ? tmp = (rand[j] >> 4) : (tmp = (rand[j] & 0x0F));
            for(i = 0; i < 16; i++)
            {
                if(tmp==i) count[i]++;
            }
        }
    }
    for(i = 0; i < 16; i++)
    {
        sum += ((u32)count[i]) * count[i];
    }
}
```

```
}  
  
if(405 < sum && sum < 687)  
    return 0;  
else  
    return 1;  
}
```