

User Guide

General Cryptographic Algorithm Library User Guide

Introduction

The user guide mainly introduces the encryption module interface of N32A455 series MCU. The hardware encryption algorithm can be easily realized by using the encryption module of Nsing Technologies.

Terms and abbreviations

Abbreviations	Terms
AES	Advance Encryption Standard
DES	Data Encryption standard
TDES	Triple Data Encryption standard
RNG	Random Number Generator
SHA	Secure Hashing Algorithm are required for digital signature applications

Contents

Contents	4
1. Introduction	8
1.1. Supported Algorithms	8
1.2. Basic Data Type	8
2. DES/TDES Algorithm API Description	9
2.1. Algorithm Library Usage Method	9
2.2. Data Type Definition	9
2.3. API Description	10
2.3.1. DES/TDES Algorithm Initialization	10
2.3.2. DES/TDES Algorithm Encryption and Decryption	10
2.3.3. DES/TDES Close	11
2.3.4. Get DES/TDES Library Version Information	12
3. AES Algorithm API Description	13
3.1. Algorithm Library Usage Method	13
3.2. Data Type Definition	13
3.3. API Description	14
3.3.1. AES Algorithm Initialization	14
3.3.2. AES Algorithm Encryption and Decryption	14
3.3.3. AES Close	15
3.3.4. Get AES Library Version Information	15
4. HASH API Description	16
4.1. Algorithm Library Usage Method	16
4.2. Data Type Definition	16

4.3.	API Description	17
4.3.1.	HASH Algorithm Initialization	18
4.3.2.	HASH Start Operation	18
4.3.3.	HASH Distributed Processing Data	18
4.3.4.	Get the Result after HASH Operation	19
4.3.5.	HASH Operation Close	20
4.3.6.	Get HASH Library Version Information	20
5.	SM7 API Description	22
5.1.	Algorithm Library Usage Method	22
5.2	Data type definition	22
5.3	API Description	23
5.3.1	SM7 Algorithm Initialization	23
5.3.2	SM7 Algorithm Encryption and Decryption	23
5.3.3	SM7 Operation Close	24
5.3.4	Get SM7 Library Version Information	24
6	SM4 API Description	25
6.1	Algorithm Library Usage Method	25
6.2	Data Type Definition	25
6.3	API Description	25
6.3.1	SM4 Algorithm Initialization	26
6.3.2	SM4 Algorithm Encryption and Decryption	26
6.3.3	SM4 Operation Close	27
6.3.4	Get SM4 Library Version Information	27
7	RNG API Description	29
7.1	Algorithm Library Usage method	29
7.2	Data Type Definition	29

7.3	API Description	29
7.3.1	Pseudo-random Generating Function	30
7.3.2	True-random Generating Function	30
7.3.3	Get RNG Library Version Information	30
8	SM1 API Description	32
8.1	Algorithm Library Usage Method	32
8.2	Data Type Definition	32
8.3	API Description	33
8.3.1	SM1 Algorithm Initialization	33
8.3.2	SM1 Algorithm Encryption And Decryption	33
8.3.3	SM1 Operation Close	34
8.3.4	Get SM1 Library Version Information	35
9	SM2 API Description	36
9.1.	Algorithm Library Usage Method	36
9.2.	Data Type Definition	36
9.3.	API Description	37
9.3.1.	Judge whether the Point is on the Curve	38
9.3.2.	Key Pair Generation	38
9.3.3.	Private Key Generation Public Key	39
9.3.4.	Signature Generation	39
9.3.5.	Signature Verification	40
9.3.6.	Key Negotiation (Exchange)	40
9.3.7.	User ID Hash Calculation Function	41
9.3.8.	Message Hash Value	42
9.3.9.	Encryption	42
9.3.10.	Deciphering	43
9.3.11.	Get SM2 Library Version Information	44

<i>i.</i>	<i>Appendix I DES Algorithm Library Function Demo</i>	<u>45</u>
<i>ii.</i>	<i>Appendix II TDES Algorithm Library Function Demo</i>	<u>48</u>
<i>iii.</i>	<i>Appendix III AES Algorithm Library Function Demo</i>	<u>53</u>
<i>iv.</i>	<i>Appendix IV HASH Algorithm Library Function Demo</i>	<u>65</u>
<i>v.</i>	<i>Appendix V SM7 Algorithm Library Function Demo</i>	<u>69</u>
<i>vi.</i>	<i>Appendix VI SM4 Algorithm Library Function Demo</i>	<u>73</u>
<i>vii.</i>	<i>Appendix VII RNG Algorithm Library Function Demo</i>	<u>77</u>
<i>viii.</i>	<i>Appendix VIII SM1 Algorithm Library Function Demo</i>	<u>79</u>
<i>ix.</i>	<i>Appendix IX SM2 Algorithm Library Function Demo</i>	<u>83</u>
10	Version History	<u>88</u>
11	Disclaimer	<u>89</u>

1. Introduction

This document is applicable to N32A455 that have downloaded relevant algorithms. It mainly describes the algorithm interfaces and usage methods of these chips.

For uint32_t data type parameter, if uint8_t is used to cast uint32_t, ensure that uint8_t addresses are word-aligned.

1.1. Supported Algorithms

The algorithm provided by N32A455 is as follows:

- DES: encryption/decryption
- TDES: encryption/decryption
- AES: encryption/decryption (AES-128/192/256)
- SM4: encryption/decryption
- SM1: encryption/decryption
- SM7: encryption/decryption
- HASH: get digest (SHA-1/SHA-224/SHA-256/MD5/SM3)
- RNG: generation of random number

1.2. Basic Data Type

<i>typedef unsigned char</i>	<i>bool;</i>
<i>typedef unsigned char</i>	<i>uint8_t;</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>uint32_t;</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. Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_des.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_des.lib in the folder to the project
2. Call the function according to the function description in Section 2.3. Refer to the demo provided in Appendix I and Appendix II for the demo

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

```
{
    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 inWordLen;    // the length(by word) of plaintext or cipher
    uint32_t En_De; // 0x33333333- encrypt, 0x44444444 - decrypt
    uint32_t Mode;    // 0x11111111 - ECB, 0x22222222 - CBC
};
```



```
uint32_t keyMode; //TDES key mode: 0x55555555-key,0x66666666-2key, 0x77777777-3key
}DES_PARM;
```

2.3. API Description

The DES algorithm library contains the following list of functions:

Table 2-1 DES/TDES Algorithm Library Functions

Functions	Descriptions
uint32_t DES_Init(DES_PARM *parm)	DES/TDES algorithm initialization function
uint32_t DES_Crypto(DES_PARM *parm)	DES/TDES algorithm encryption and decryption
void DES_Close(void)	DES/TDES algorithm close
void DES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get DES algorithm library version information

2.3.1. DES/TDES Algorithm Initialization

DES_Init	<u>DES/TDES algorithm initialization</u>
Function prototype	uint32_t DES_Init(DES_PARM *parm)
Parameter description	parm: input, point to DES_PARM structure
Return value	DES_Init_OK: initialization succeeded. Other: initialization error
Note	If it is in ECB mode, parameter iv can be directly replaced with NULL.

2.3.2. DES/TDES Algorithm Encryption and Decryption

DES_Crypto	<u>DES/TDES algorithm encryption and decryption</u>
-------------------	---

Function prototype	uint32_t DES_Crypto(DES_PARM *parm)
Parameter description	parm: input, point to DES_PARM structure
Return value	DES_Crypto_OK: correct operation. Other: operation error
Note	<p>Before calling this function, if it has not been initialized or switched to another algorithm, call DES_Init function first</p> <ol style="list-style-type: none"> 1. If it is in ECB mode, parameter iv1 can be directly replaced with NULL 2. When a large amount of data is encrypted as a whole but in 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 block (8 bytes) of the ciphertext obtained by calling this function to encrypt the X-1 block of data. 3. When a large amount of data is decrypted as a whole but in multiple CBC blocks, it should be noted that: The initial vector IV (IV=iv1) used for X data (X > 1) must be updated to the last block (8 bytes) of the X-1 block of data. 4. Please refer to Appendix I and Appendix II for the calling method.

2.3.3. DES/TDES Close

DES Close

Close DES/TDES algorithm clock and system clock

Function prototype void DES_Close(void)

Parameter description

Return value

2.3.4. Get DES/TDES Library Version Information

DES Version	Get DES/TDES library version information
Function prototype	<pre>void DES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)</pre>
Parameter description	<p>type: commercial or fast version</p> <p>customer: standard or customized version</p> <p>date: year, month, day</p> <p>version: version x.x</p>
Return value	
Note	<pre>*type = 0x05; // Business and Express *customer = 0x00; // Standard version date[0] = 18; //Year() date[1] = 12; //Month() date[2] = 28; //Day () *version = 0x10; // Version 1.0</pre>

3. AES Algorithm API Description

3.1. Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_des.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_aes.lib in the folder to the project
2. Call the function according to the function description in Section 3.3. Refer to the demo provided in Appendix III for the demo

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 cannot be zero;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. API Description

The AES algorithm library contains the following functions:

Table 3-1 AES Algorithm Library Functions

Functions	Descriptions
uint32_t AES_Init(AES_PARM *parm)	AES algorithm initialization function
uint32_t AES_Crypto(AES_PARM *parm)	AES algorithm encryption and decryption
void AES_Close(void)	AES algorithm close
void AES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get AES algorithm library version information

3.3.1. AES Algorithm Initialization

AES_Init	<u>AES algorithm initialization</u>
Function prototype	uint32_t AES_Init(AES_PARM *parm)
Parameter description	parm: input, point to AES_PARM structure
Return value	AES_Init_OK: correct operation. Other: arithmetic error
Note	1. Please refer to Appendix III for calling method.

3.3.2. AES Algorithm Encryption and Decryption

AES_Crypto	<u>AES algorithm encryption and decryption</u>
-------------------	--

Function prototype	uint32_t AES_Crypto(AES_PARM *parm)
Parameter description	parm: input, point to AES_PARM structure
Return value	AES_Crypto_OK: correct operation. Other: arithmetic error
Note	1. Before calling this function, if it has not been initialized or switched to other algorithms, call AES_Init function first

3.3.3. AES Close

AES Close	Turn off AES algorithm clock and system clock
Function prototype	void AES_Close(void)
Parameter description	
Return value	

3.3.4 Get AES Library Version Information

AES Version	Get AES library version information
Function prototype	void AES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)
Parameter description	type: commercial or fast version customer: standard or customized version date: year, month, day version: version x.x
Return value	
Note	*type = 0x05; // Business and Express *customer = 0x00; // Standard Version date[0] = 18; //Year() date[1] = 12; //Month() date[2] = 28; //Day () *version = 0x10; // Version V1.0

4. HASH API Description

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

4.1. Algorithm Library Usage Method

Data input and output are in byte big-end order .Algorithm library usage method as follows:

1. Copy the n32a455_hash.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_hash.lib in the folder to the project.
2. Call the function according to the function description in Section 4.3. Refer to the demo provided in Appendix IV.

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. API Description

The HASH algorithm library contains the following functions:

Table 4-1 HASH Algorithm Library Functions

Functions	Descriptions
uint32_t HASH_Init(HASH_CTX *ctx)	HASH algorithm initialization function
uint32_t HASH_Start(HASH_CTX *ctx)	HASH step hash start function
uint32_t HASH_Update(HASH_CTX *ctx, uint8_t *in, uint32_t byteLen)	HASH step hash processing function
uint32_t HASH_Complete(HASH_CTX *ctx, uint8_t *out)	HASH step hash completion function
uint32_t HASH_Close(void)	HASH algorithm close
void HASH_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t	Get HASH algorithm

*version)	library version information
-----------	-----------------------------

4.3.1. HASH Algorithm Initialization

HASH_Init	<u>HASH algorithm initialization</u>
Function prototype	uint32_t HASH_Init(HASH_CTX *ctx)
Parameter description	ctx: input, Point to HASH_CTX structure
Return value	HASH_Init_OK: correct operation. Other: arithmetic error
Note	<ol style="list-style-type: none"> 1. Ctx must point to the RAM area, and the content pointed to cannot be changed (intermediate state and temporary content storage of hash calculation), the same below 2. When calculating the hash value of a message step by step, this function must be called first

4.3.2. HASH Start Operation

HASH_Start	<u>HASH start operation</u>
Function prototype	uint32_t HASH_Start(HASH_CTX *ctx)
Parameter description	ctx: input, Point to HASH_CTX structure
Return value	HASH_Start_OK: correct operation. Other: arithmetic error
Note	<ol style="list-style-type: none"> 1. If interrupt is required during HASH operation, set ctx->sequence to HASH_SEQUENCE_TRUE, HASH needs to be called again after the interruption_Init function, Then call HASH_Update function; otherwise, set ctx->sequence to false. 2. Please refer to Appendix IV for calling method.

4.3.3. HASH Distributed Processing Data

HASH_Update	<u>HASH distributed processing data</u>
Function prototype	uint32_t HASH_Update(HASH_CTX *ctx, uint8_t *in, uint32_t byteLen)

Parameter description	<p>ctx: input, point to HASH_CTX structure</p> <p>in: input, refers to the information to be pieced together</p> <p>byteLen: input, refers to the byte length of hash information</p>
Return value	HASH_Update_OK: correct operation. Other values: operation error
Note	<p>Before calling this function, if it has not been initialized or has switched to other algorithms, call HASH_Init and HASH_Start function</p> <ol style="list-style-type: none"> 1. The initialization function HASH_Init and HASH_Start must be called before calling this function. 2. Ctx must point to the RAM area, and the content pointed to cannot be changed (intermediate state and temporary content storage for hash calculation). 3. The in content can point to the RAM or Flash area. In can be NULL, and the calculation result is a summary value of NULL. 4. ByteLen can be 0 or NULL, and the calculation result is a summary value of NULL. 5. After initialization, a whole message can be divided into multiple pieces at will. For each piece of message, this function can be called in turn, and finally HASH_Complete function can be called to get the hash result of the whole message 6. If cascading application is required, set ctx->sequence as HASH_SEQUENCE_TRUE, copy external IV to ctx->IV, and add the updated data length len to ctx->msgByteLen using ctx->hashAlg->ByteLenPlus(ctx->msgByteLen, len), then call HASH_Update function to cascade successfully 7. Please refer to Appendix IV for calling method.

4.3.4. Get the Result after HASH Operation

HASH Complete

Get the result after HASH operation

Function prototype

uint32_t HASH_Complete(HASH_CTX *ctx, uint8_t *out)

Parameter description	ctx: input, point to HASH_CTX structure Out: output, point to HASH result
Return value	HASH_Complete_OK: correct operation. Other values: operation error
Note	<ol style="list-style-type: none"> 1. Before calling this function, if it has not been initialized or has switched to other algorithms, call HASH_Init and HASH_Start function first. 2. Call this function to get the final result after entering the message. 3. ctx must point to the RAM area, and the content pointed to cannot be changed (intermediate state and temporary content storage for hash calculation). 4. Please refer to Appendix IV for calling method

4.3.5. HASH Operation Close

HASH_Close	HASH operation close
Function prototype	uint32_t HASH_Close(void)
Parameter description	
Return value	HASH_Close_OK: correct operation. Other values: operation error
Note	

4.3.6. Get HASH Library Version Information

HASH_Version	Get HASH library version information
Function prototype	void HASH_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)
Parameter description	type: commercial or fast version Customer: standard or customized version date: year, month, day version: version x.x
Return value	
Note	*type = 0x05; // Business and Express

```
*customer = 0x00; // Standard version  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; // Version V1.0
```

5. SM7 API Description

5.1. Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_sm7.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_sm7.lib in the folder to the project;
2. Call the function according to the function description in Section 5.3. Refer to the demo provided in Appendix V for the demo.

5.2 Data type definition

```
#define SM7_ECB (0x11111111)
#define SM7_CBC (0x22222222)
#define SM7_ENC (0x44444444)
#define SM7_DEC (0x55555555)
enum
{
    SM7_Crypto_OK = 0x0, //SM7 operation success
    SM7_Init_OK = 0x0, //SM7 operation success
    SM7_Crypto_ModeError = 0x5a5a5a5a, //Working mode error(Neither ECB nor CBC)
    SM7_Crypto_EnOrDeError, //En&De error(Neither encryption nor decryption)
    SM7_Crypto_ParaNull, // the part of input(output/iv) Null
    SM7_Crypto_LengthError, //the length of input message must be 2 times and cannot be zero
    SM7_UnInitError, //SM7 uninitialized
};

typedef struct
{
    uint32_t *in; // the part of input
    uint32_t *iv; // the part of initial vector
    uint32_t *out; // the part of output
    uint32_t *key; // the part 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
}SM7_PARM;
```

5.3 API Description

The SM7 algorithm library contains the following functions:

Table 5-1 SM7 Algorithm Library Functions

Functions	Descriptions
uint32_t SM7_Init(SM7_PARM *parm)	SM7 algorithm initialization function
uint32_t SM7_Crypto(SM7_PARM *parm);	SM7 algorithm encryption and decryption
uint32_t SM7_Close (void)	SM7 algorithm close
void SM7_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get SM7 algorithm library version information

5.3.1 SM7 Algorithm Initialization

SM7_Init	<u>SM7 algorithm initialization</u>
Function prototype	uint32_t SM7_Init(SM7_PARM *parm)
Parameter description	parm: input, point to SM7_PARM structure
Return value	SM7_Init_OK: Correct operation Other: Initialization failure. Refer to enumeration type definition for details
Note	1. SM7 initialization function, which must be executed before SM7 operation

5.3.2 SM7 Algorithm Encryption and Decryption

SM7_Crypto	<u>SM7 algorithm encryption and decryption</u>
Function prototype	uint32_t SM7_Crypto(SM7_PARM *parm)
Parameter description	parm: input, point to SM7_PARM structure
Return value	SM7_Crypto_OK: correct operation. Other: calculation error, see enumeration type definition for details

Note	1. Before calling this function, if it has not been initialized or switched to another algorithm, call SM7_Init function first
Demo	See Appendix V SM7 algorithm library function for the routine

5.3.3 SM7 Operation Close

SM7_Close	<u>Turn off the SM7 algorithm system clock and algorithm clock</u>
Function prototype	uint32_t SM7_Close (void)
Parameter description	
Return value	
Note	

5.3.4 Get SM7 Library Version Information

SM7_Close	<u>Get SM7 library version information</u>
Function prototype	void SM7_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)
Parameter description	type: commercial or fast version customer: standard or customized version date: year, month, day version: version x.x
Return value	
Note	*type = 0x05; // Business and Express *customer = 0x00; // Standard version date[0] = 18; //Year date[1] = 12; //Month date[2] = 28; //Day *version = 0x10; // version 1.0

6 SM4 API Description

6.1 Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_sm4.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_sm4.lib in the folder to the project;
2. Call the function according to the function description in Section 6.3. Refer to the demo provided in Appendix VI for the routine

6.2 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;
```

6.3 API Description

The SM4 algorithm library contains the following functions:

Table 6-1 SM4 Algorithm Library Functions

Functions	Descriptions
uint32_t SM4_Init(SM4_PARM *parm)	SM4 algorithm initialization function
uint32_t SM4_Crypto(SM4_PARM *parm)	SM4 algorithm encryption and decryption
void SM4_Close(void)	SM4 algorithm close
void SM4_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get SM4 algorithm library version information

6.3.1 SM4 Algorithm Initialization

SM4_Init

SM4 algorithm initialization

Function prototype

uint32_t SM4_Init(SM4_PARM *parm)

Parameter description

parm: input, point to SM4_PARM structure

Return value

SM4_Init_OK: Correct operation. Other values: calculation error, refer to enumeration type value definition for details

Note

6.3.2 SM4 Algorithm Encryption and Decryption

SM4_Crypto

SM4 algorithm encryption and decryption

Function prototype

uint32_t SM4_Crypto(SM4_PARM *parm)

Parameter description

parm: input, point to SM4_PARM structure

Return value

SM4_Crypto_OK: Correct operation Other values: Calculation error, see enumeration type value definition for details

Note

1. Before calling this function, if it has not been initialized or switched to another algorithm, call SM4_Init function first
2. Structure SM4_PARM refers to the definition of SM4_PARM in Section 6.2.
3. If it is in ECB mode, the parameter iv1 can be directly replaced with NULL.
4. When a large amount of data is encrypted as a whole but in multiple CBC blocks, note the following:

The initial vector IV (IV=iv1) used for block X data (X>1) must be updated the last group of ciphertext obtained by calling this function to encrypt the X-1 block of data (16 bytes).

5. When a large amount of data is decrypted as a whole but in multiple CBC blocks, it should be noted that:

The initial vector IV (IV=iv1) used for block X data (X>1) must be updated to the last group (16 bytes) of the X-1 block of data.

6.3.3 SM4 Operation Close**SM4_Close**

Turn off the SM4 algorithm clock and system clock

Function prototype

void SM4_Close(void)

Parameter description**Return value****Note****6.3.4 Get SM4 Library Version Information****SM4_Version**

Get SM4 library version information

Function prototype

uint8_t *version)

void SM4_Version(uint8_t *type, uint8_t *customer, uint8_t date[3],

Parameter description

type: commercial or fast version

customer: standard or customized version

date: year, month, day

version: version x.x

Return value

Note

```
*type = 0x05; // Business and Express
```

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

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

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

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

```
*version = 0x10; // Version V1.0
```

7 RNG API Description

7.1 Algorithm Library Usage method

Algorithm library usage method as follows:

1. Copy the n32a455_rng.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_rng.lib in the folder to the project.
2. Call the function according to the function description in Section 7.3.

7.2 Data Type Definition

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

7.3 API Description

The RNG algorithm library contains the following functions:

Table 7-1 RNG Algorithm Library Functions

Functions	Descriptions
uint32_t GetPseudoRand_U32(uint32_t *rand, uint32_t wordLen, uint32_t seed[2])	Pseudo-random numbers generate functions by word
uint32_t GetTrueRand_U32(uint32_t *rand, uint32_t wordLen)	Word generating function of true random number
void RNG_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get RNG algorithm library version information

7.3.1 Pseudo-random Generating Function

GetPseudoRand_U32

Pseudo-random numbers generate functions by word

Function prototype

```
uint32_t GetPseudoRand_U32(uint32_t *rand, uint32_t
wordLen, uint32_t seed[2])
```

Parameter description

rand: pointer, point to the generated random number
wordlen: pseudo-random number word length to be obtained
seed[2]: input, pseudo-random seed variable array

Return value

RNG_OK: correct operation. Other values: error generating pseudo-random number

Description

Generate pseudo-random numbers by word

Note

The user can input the seed array. If the user input seed is NULL, the seed will be automatically generated internally.

7.3.2 True-random Generating Function

GetTrueRand_U32

Ture-random numbers generate functions by word

Function prototype

```
uint32_t GetTrueRand_U32(uint32_t *rand, uint32_t wordLen)
```

Parameter description

rand: pointer, point to a memory address of the generated random number
wordLen: word length to obtain true random number

Return value

RNG_OK: correct operation. Other values: error generating true random number. Refer to enumeration type value definition for details

Note

7.3.3 Get RNG Library Version Information

RNG_Version

Get RNG library version information

Function prototype

```
void RNG_Version(uint8_t *type, uint8_t *customer, uint8_t date[3],
uint8_t *version)
```

Parameter description

type: commercial or fast version
customer: standard or customized version
date: year, month, day
version: version x.x

Return value

Note

```
*type = 0x05; // Business and Express  
*customer = 0x00; // Standard version  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; // Version V1.0
```

8 SM1 API Description

8.1 Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_sm1.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_sm1.lib in the folder to the project;
2. Call the function according to the function description in Section 8.3. Refer to the demo provided in Appendix VIII for the routine

8.2 Data Type Definition

```
#define SM1_ECB (0x11111111)
#define SM1_CBC (0x22222222)
#define SM1_ENC (0x44444444)
#define SM1_DEC (0x55555555)
enum{
    SM1_Crypto_OK = 0x0,           //SM1 operation success
    SM1_Init_OK = 0x0,           //SM1 operation success
    SM1_Crypto_ModeError = 0x5a5a5a5a, //Working mode error(Neither ECB nor CBC)
    SM1_Crypto_EnOrDeError, //En&De error(Neither encryption nor decryption)
    SM1_Crypto_ParaNull, // the part of input(output/iv) Null
    SM1_Crypto_LengthError, //the length of input message must be 4 times and cannot be zero
    SM1_UnInitError, //SM1 uninitialized
};

typedef struct{
    uint32_t *in; // the part of input
    uint32_t *iv; // the part of initial vector
    uint32_t *out; // the part of output
    uint32_t *key; // the part 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
}SM1_PARM;
```

8.3 API Description

The SM1 algorithm library contains the following functions:

Table 8-1 SM1 Algorithm Library Functions

Functions	Descriptions
uint32_t SM1_Init(SM1_PARM *parm)	SM1 algorithm initialization function
uint32_t SM1_Crypto(SM1_PARM *parm)	SM1 algorithm encryption and decryption
void SM1_Close(void)	SM1 algorithm close
void SM1_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get SM1 algorithm library version information

8.3.1 SM1 Algorithm Initialization

SM1_Init	<u>SM1 algorithm initialization</u>
Function prototype	uint32_t SM1_Init(SM1_PARM *parm)
Parameter description	parm: input, point to SM1_PARM structure
Return value	SM1_Init_OK: correct operation. Other values: calculation error, refer to enumeration type value definition for details
Note	

8.3.2 SM1 Algorithm Encryption And Decryption

SM1_Crypto	<u>SM1 algorithm algorithm encryption and decryption</u>
-------------------	--

Function prototype	uint32_t SM1_Crypto(SM1_PARM *parm)
Parameter description	parm: Input, Pointer to SM1_PARM structure
Return value	SM1_Crypto_OK: Correct operation Other values: Calculation error, see enumeration type value definition for details
Note	<ol style="list-style-type: none"> 1. Before calling this function, if it has not been initialized or switched to another algorithm, call SM1_Init function first; 2. Structure SM1_PARM refers to the definition of SM1_PARM in Section 8.2. 3. If it is in ECB mode, the parameter iv1 can be directly replaced with NULL. 4. When a large amount of data is encrypted as a whole but in multiple CBC blocks, attention should be paid to: The initial vector IV (IV=iv1) used for block X data (X>1) must be updated the last group of ciphertext obtained by calling this function to encrypt the X-1 block of data (16 bytes). 5. When a large amount of data is decrypted as a whole but in multiple CBC blocks, it should be noted that: The initial vector IV (IV=iv1) used for block X data (X>1) must be updated to the last group (16 bytes) of the X-1 block of data.

8.3.3 SM1 Operation Close

SM1_Close	<u>Turn off the SM1 algorithm clock and system clock</u>
Function prototype	void SM1_Close(void)
Parameter description	
Return value	
Note	

8.3.4 Get SM1 Library Version Information

SM1 Version

Get SM1 library version information

Function prototype

```
void SM1_Version(uint8_t *type, uint8_t *customer, uint8_t  
date[3], uint8_t *version)
```

Parameter description

```
type:      commercial or fast version  
customer:  standard or customized version  
date:      year, month, day  
version:   // version x.x
```

Return value

Note

```
*type = 0x05; // Business and Express  
*customer = 0x00; // Standard version  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; // Version V1.0
```

9 SM2 API Description

9.1. Algorithm Library Usage Method

Algorithm library usage method as follows:

1. Copy the n32a455_sm2.h, Type.h, n32a455_algo_common.h, n32a455_algo_common.lib, n32a455_sm2.lib, n32a455_rng.lib in the folder to the project;
2. Call the function according to the function description in Section 9.3. Refer to the demo provided in Appendix 9 for the demo.

9.2. Data Type Definition

enum

```
{
    SM2_SUCCESS           = 0,
    SM2_DIV_OK           = 0,
    SM2_HCAL_OK          = 0,
    SM2_INIT_OK          = 0,
    SM2_MUL_OK           = 0,
    SM2_MODMUL_OK        = 0,
    SM2_Reverse_OK       = 0,
    SM2_SUB_OK           = 0,
    SM2_ADD_OK           = 0,
    SM2_Cpy_OK=0,         //copy success
    SM2_XOR_OK=0,         //XOR success
    SM2_SetZero_OK = 0,
    SM2_isCurve_Ok = 0,
    SM2_isCurve_Not = 0x05,
    SM2_PointAdd_Ok      = 0,
    SM2_PointDouble_Ok  = 0,
    SM2_PointMul_Ok     = 0,
    SM2_Sign_Ok = 0,
    SM2_Verif_Ok = 0,
    SM2_En_Ok = 0,
    SM2_De_Ok = 0,
    SM2_Exchange_Ok     = 0,
    SM2_FAIL             = 0x01,
    SM2_YES               = 0x02,
    SM2_NOT               = 0x03,
```

```

ZERO_VALUE_ERROR = 0x04
};

enum{
    SM2_IsZero_NOT = 0,    //Big number is not zero
    SM2_Cmp_EQUAL = 0,    //Two big number are equal
    SM2_IsOne_NOT = 0,    //big number is one
    SM2_IsOne_YES = 1,    //big number is not one
    SM2_IsZero_YES = 1,   //Big number is zero
    SM2_Cmp_LESS = -1,    //The former big number is less than the latter
    SM2_Cmp_GREATER = 1, //The former big number is greater than the latter
    SM2_Reverse_ERROR = 0x7A9E0863, //reverse fail due to src and dst are same
    SM2_ERROR = 3,
    POINT_MUL_ERROR = 4,
    PRIKEY_ERROR = 5,
    LENGTH_TOO_LONG = 6,
    PUBKEY_ERROR = 7,
    FAIL = 8,
    SM2_AddrErr,
    SM2_LengthErr,
    SM2_ROLE_ERR
};

```

9.3. API Description

The SM2 algorithm library contains the following functions:

Table 9-1 SM2 Algorithm Library Functions

Functions	Descriptions
uint32_t SM2_PointIsOnCrv(uint8_t pubKey[65])	Judge whether the point is on the curve
uint32_t SM2_GetKey(uint8_t priKey[32], uint8_t pubKey[65])	Generate SM2 key pair function
uint32_t SM2_GetPubKey(uint8_t priKey[32], uint8_t pubKey[65])	Get public key from private key
uint32_t SM2_Sign(uint8_t E[32], uint8_t priKey[32], uint8_t r[32], uint8_t s[32])	Signature function
uint32_t SM2_Verify(uint8_t E[32], uint8_t pubKey[65], uint8_t r[32], uint8_t s[32])	Signature Validation function
uint32_t SM2_ExchangeKey (uint8_t role, uint8_t *IDA, uint32_t	Key exchange

IDABYTELEN, uint8_t *IDB, uint32_t IDBByteLen, uint8_t dA[32], uint8_t PA[65], uint8_t PB[65], uint8_t rA[32], uint8_t RA[65], uint8_t RB[65], uint32_t kByteLen, uint8_t *KA, uint8_t S1[32], uint8_t SA[32])	
uint32_t SM2_getZ(uint8_t *ID, uint32_t IDByteLen, uint8_t pubKey[65], uint8_t Z[32])	Identity hash value calculation function
uint32_t SM2_GetE(uint8_t *M, uint32_t MByteLen, uint8_t Z[32], uint8_t E[32])	HUSH value of the message
uint32_t SM2_Encrypt(uint8_t *M, uint32_t MByteLen, uint8_t pubKey[65], uint8_t *C, uint32_t *CByteLen)	SM2 algorithm encryption
uint32_t SM2_Decrypt(uint8_t *C, uint32_t CByteLen, uint8_t priKey[32], uint8_t *M, uint32_t *MByteLen)	SM2 algorithm decryption
void SM2_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	Get SM2 algorithm library version information

9.3.1. Judge whether the Point is on the Curve

SM2 PointIsOnCrv

Judge whether the point is on the curve

Function prototype

uint32_t SM2_PointIsOnCrv(uint8_t pubKey[65])

Parameter description

Return value

SM2_isCurve_Ok: the point is on the curve. Other values: the point is not on the curve. Refer to enumeration type definition for details

Note

9.3.2. Key Pair Generation

SM2 GetKey

Generate SM2 key pair

Function prototype

uint32_t SM2_GetKey(uint8_t priKey[32], uint8_t pubKey[65])

Parameter description	<p>priKey: output, the private key is at the big endian, that is, the high bit is in the front and the low bit is in the back</p> <p>pubKey: output, the public key is at the big endian, and its first byte is 0x04, and then horizontal and vertical coordinates</p>
Return value	SM2_SUCCESS: correct calculation. Other values: calculation error, refer to enumeration type definition for details
Note	The private key is a large number. It must be in $[1, n-2]$. n is the SM2 curve parameter.

9.3.3. Private Key Generation Public Key

SM2_GetPubKey	<u>Get public key from private key</u>
Function prototype	uint32_t SM2_GetPubKey(uint8_t priKey[32], uint8_t pubKey[65])
Parameter description	<p>priKey: input, the private key is at the big endian, that is, the high key is in the front and the low key is in the back</p> <p>pubKey: output, public key is at the big endian, and its first byte is 0x04, and then horizontal and vertical coordinates</p>
Return value	SM2_SUCCESS: correct calculation. Other values: calculation error, refer to enumeration type definition for details
Note	The private key is a large number. It must be in $[1, n-2]$. n is the SM2 curve parameter.

9.3.4. Signature Generation

SM2_Sign	<u>Signature function</u>
Function prototype	uint32_t SM2_Sign(uint8_t E[32], uint8_t priKey[32], uint8_t r[32], uint8_t s[32])
Parameter description	<p>E: input, HASH value of the message to be signed</p> <p>priKey: input, Private key of signer</p> <p>r: output, signature results</p>

s: output, signature results

Return value

SM2_Sign_Ok: correct calculation. Other values: calculation error, refer to enumeration type definition for details

Note

E or prikey storage media can be RAM or FLASH.

1. If E or prikey is stored in RAM, the signature results r and s can reuse this space, but the space must be sufficient.
2. The prikey must be in [1, n-1].

9.3.5. Signature Verification

SM2_Verify

Signature verification

Function prototype

uint32_t SM2_Verify(uint8_t E[32], uint8_t pubKey[65], uint8_t r[32],uint8_t s[32])

Parameter description

E: input, HASH value of the message to be signed

pubKey:input, public key of signer

r: input, signature results

s: input, signature results

Return value

SM2_Verif_Ok: correct operation. Other values: calculation error, refer to enumeration type definition for details.

Note

E or pubKey storage media can be RAM or FLASH

9.3.6. Key Negotiation (Exchange)

SM2_ExchangeKey

Key negotiation (exchange)

Function prototype

uint32_t SM2_ExchangeKey (uint8_t role, uint8_t *IDA, uint32_t IDAByteLen, uint8_t *IDB, uint32_t IDBByteLen, uint8_t dA[32], uint8_t PA[65], uint8_t PB[65], uint8_t rA[32], uint8_t RA[65], uint8_t RB[65], uint32_t kByteLen, uint8_t *KA, uint8_t S1[32], uint8_t SA[32])

Parameter description

role: input, 1: initiator, 0: receiver

IDA: input, initiator ID
 IDAByteLen: input, number of bytes of IDB
 IDB: input, recipient ID
 IDBByteLen: input, number of bytes of IDB
 dA[32]: input, own private key
 PA[65]: input, own public key
 PB[65]: input, counterparty public key
 rA[32]: input, own temporary private key
 RA[65]: input, own temporary public key
 RB[65]: input, temporary public key of the other party
 kByteLen: input, negotiate key byte length
 KA [kByteLen]: output, negotiate key
 S1 [32]: output, own S1 value
 SA [32]: output, own SA value

Return value

SM2_SUCCESS: correct operation; Other values: calculation error, refer to enumeration type definition for details.

Note

1. If the calculated S1=SB, S2=SA, the key exchange is successful, and the negotiated key is KA
2. dA, PB, rA, RA, RB, ZA, ZB storage media can be RAM or FLASH
3. The output parameters KA, S1 and SA cannot use the same buffer as the input parameters dA, PB, rA, RA, RB, ZA and ZB.

9.3.7. User ID Hash Calculation Function

SM2_getZ

User ID hash calculation function

Function prototype

uint32_t SM2_getZ(uint8_t *ID, uint32_t IDByteLen, uint8_t pubKey[65], uint8_t Z[32])

Parameter description

ID: input, ID of user A
 IDByteLen: input, Number of bytes of user A's ID

pubKey: input, User A's public key, 65 bytes

Z[32]: output, The hash value of user A, 32 bytes

Return value

SM2_SUCCESS: calculation succeeded. Other values: calculation error, refer to enumeration type definition for details.

Note

9.3.8. Message Hash Value

SM2_GetE

Hash value calculation function of message

Function prototype

```
uint32_t SM2_GetE(uint8_t *M, uint32_t MByteLen, uint8_t Z[32],
uint8_t E[32])
```

Parameter description

M: input, message to be signed for verification

MByteLen: input, the number of bytes of the message to be signed for verification

Z: input, ID hash value of signer

E: output, input the hash value of M and Z, 32 bytes

Return value

SM2_SUCCESS: calculation succeeded. Other values: calculation error, refer to enumeration type definition for details.

Note

9.3.9. Encryption

SM2_Encrypt

SM2 encryption function

Function prototype

```
uint32_t SM2_Encrypt(uint8_t *M, uint32_t MByteLen, uint8_t
pubKey[65], uint8_t *C, uint32_t *CByteLen)
```

Parameter description

M: input, plaintext

MByteLen: input, byte length of plaintext, $0 < \text{MByteLen} < (2^{32}-97)$

pubKey: input, public key

C: output, ciphertext

CByteLen: output, byte length of ciphertext, should be (MByteLen +97)

Return value

SM2_En_Ok: calculation succeeded. Other values: calculation error, refer to enumeration type definition for details.

Note

1. M or pubKey storage media can be RAM or FLASH.
2. M and C cannot be the same buffer; pubKey and C cannot be the same buffer.
3. According to the new Nsing security specification, the ciphertext order is revised from C1 || C2 || C3 to C1 || C3 || C2.

9.3.10. Deciphering**SM2 Decrypt**SM2 deciphering function**Function prototype**

```
uint32_t SM2_Decrypt(uint8_t *C, uint32_t CByteLen, uint8_t
priKey[32], uint8_t *M, uint32_t *MByteLen)
```

Parameter description

C: input, plaintext

CByteLen: input, byte length of plaintext, $97 < \text{CByteLen} < 2^{32}$

priKey: input, private key

M: output, ciphertext

MByteLen: output, byte length of ciphertext, should be (CByteLen -97)

Return value

SM2_De_Ok: calculation succeeded. Other values: calculation error, refer to enumeration type definition for details.

Note

1. C or priKey storage media can be RAM or FLASH.
2. M and C cannot be the same buffer.
3. If the priKey is stored in RAM, M can reuse this space, but it needs to ensure enough space.
4. According to the new Nsing security specification, the ciphertext order is revised from C1 || C2 || C3 to C1 || C3 || C2.

9.3.11. Get SM2 Library Version Information

SM2 Version	Get SM2 library version information
Function prototype	<code>void SM2_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)</code>
Parameter description	<code>type:</code> commercial or fast version <code>customer:</code> standard or customized version <code>date:</code> year, month, day <code>version:</code> version x.x
Return value	
Note	<code>*type = 0x05; // Business and Express</code> <code>*customer = 0x00; // Standard version</code> <code>date[0] = 18; //Year()</code> <code>date[1] = 12; //Month()</code> <code>date[2] = 28; //Day ()</code> <code>*version = 0x10; // Version V1.0</code>

i. Appendix I DES Algorithm Library Function Demo

```

uint32_t DES_test()
{
    uint32_t i,flag1,flag2,flag3,flag4;
    uint32_t ret;
    DES_PARM DES_Parm={0};
    /* If the test case needs to be modified,When the true value of the parameter is
    "0x0102030405060708" since u32 data is stored in small end order, when initializing and assigning
    the above parameters, please enter "0x04030201,0x08070605". If there is no special instruction,
    this demo parameter is set in this way */
    uint32_t in1 [16]={
    0x5FE2D4C0,0xAEAE3F30,0x692930A8,0x1DA69A51,0xDD34B34B,0xAF8D237A,0x2114F489,
    0xE461FF17,0x47C795FD,0x8FF62B49,0x62E9BD63,0x1AF52817,0xECB9DFD4,0xE04421C9,
    0x87B4B22E,0x9FF98759
    };
    uint32_t key1 [2]={0x946AB06B,0x2276E632};
    uint32_t iv1 [2]={0x482A8C66,0xC324FC78};
    uint32_t out[16];
    uint32_t
    DES_ECB_EN[16]={0x2FD8D31F,0xC3E2E705,0x4B6D1C4C,0x31EB4154,0xDA273EEC,
    0x8EED57DA,0x26FDE038,0x15B0D57D,0xBCE7464F,0x78D7997A,
    0x4F9917D7,0xAE9C1DA9,0x749FEAEE,0xDFE6A911,0x34D556D5,
    0xA32FA0A2};
    /*DES_ECB_EN=0x1FD3D82F05E7E2C34C1C6D4B5441EB31EC3E27DADA57ED8E38E0FD267
    DD5B0154F46E7BC7A99D778D717994FA91D9CAEEEEA9F7411A9E6DFD556D534A2A02FA3*/
    uint32_t
    DES_ECB_DE[16]={0xBD77D94A,0xCF5698BB,0xF113743F,0x0FCFC898,0x7DD21DA8,
    0x3908A674,0x65303E6C,0x56CB0E02,0xF0B14651,0x3BBB36AB,
    0x8C129CC3,0xC42D5DD0,0x74549F20,0x5A7E5029,0xE5334FE2,
    0xD5ED9CA8};
    /*DES_ECB_DE=0x4AD977BDBB9856CF3F7413F198C8CF0FA81DD27D74A608396C3E306502
    0ECB565146B1F0AB36BB3BC39C128CD05D2DC4209F547429507E5AE24F33E5A89CEDD5*/
    uint32_t DES_CBC_EN[16]={0x236813B0,0x14D3A0CA,0xDB57CA2F,0x073FADB0,0x83577985,
    0x7DEBA1CB,0xD5410854,0x2C0E74D8,0x8B8019BB,0xBAB789EF,
    0xF93DEC2E,0xD1BFE8F4,0xE061C81D,0x2F620219,0x662759FF,
    0x77CABBF6};
    /*DES_CBC_EN=0xB0136823CAA0D3142FCA57DBB0AD3F0785795783CBA1EB7D540841D5D8
    740E2CBB19808BEF89B7BA2EEC3DF9F4E8BFD11DC861E01902622FFF592766F6BBCA77*/
    uint32_t DES_CBC_DE[16]={0xF55D552C,0x0C7264C3,0xAEF1A0FF,0xA161F7A8,0x14FB2D00,
    0x24AE3C25,0xB8048D27,0xF9462D78,0xD1A5B2D8,0xDFDAC9BC,
    0xCBD5093E,0x4BDB7699,0x16BD2243,0x408B783E,0x098A9036,
    0x35A9BD61};
    /*DES_CBC_DE=0x2C555DF5C364720CFFA0F1AEA8F761A1002DFB14253CAE24278D04B8782
    D46F9D8B2A5D1BCC9DADF3E09D5CB9976DB4B4322BD163E788B4036908A0961BDA935*/
    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 Algorithm Library Function Demo

```

uint32_t TDES_2Key_test()
{
    uint32_t i,flag1,flag2,flag3,flag4;
    uint32_t ret;
    DES_PARM TDES_Parm={0};
    /* If the test case needs to be modified, when the true value of the parameter is
    "0x0102030405060708", since u32 data is stored in small end order, when initializing and
    assigning the above parameters, please enter "0x04030201,0x08070605". If there is no special
    instruction, this demo parameter is set in this way */
    uint32_t in1[16]={
    0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,
    0xED8A71D7,0xA66B2FAA,0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,
    0x05CDB663,0xF7D812E4,0x86341DEB,0xBA52B237
    };
    uint32_t key1[4]={0x81F08C18,0x5C6BE38C,0x4D6A6563,0xFF220031};
    uint32_t iv1[2]={0xB5CC3A62,0xC96EF050};
    uint32_t out[16];
    uint32_t
    TDES_ECB_EN[16]={0x42976179,0x3A15FDA5,0x278639E4,0x3F4D2DDD,0x987EAF74,
    0x17376CD5,0x9BE1CAB1,0x5501A0BA,0xD18D511B,0x11054F45,
    0x7EAC1828,0x375B9DAD,0x3823A312,0x8EE802FF,0xF2F00328,
    0x3F81CF19};
    /*TDES_ECB_EN=0x79619742A5FD153AE4398627DD2D4D3F74AF7E98D56C3717B1CAE19B
    BAA001551B518DD1454F05112818AC7EAD9D5B3712A32338FF02E88E2803F0F219CF813F*/
    uint32_t
    TDES_ECB_DE[16]={0x58AD407C,0x76B43ED7,0x23B44DDA,0x22EC376C,0x50311263,
    0xECC57D42,0x2FA5ADAA,0xE7A099A0,0x287DBD9B,0x3951FD62,
    0x530A3728,0x9AFAA2D3,0x0C41708F,0x5BFE1BCC,0x3B21EE97,
    0xE29E749A};
    /*TDES_ECB_DE=0x7C40AD58D73EB476DA4DB4236C37EC2263123150427DC5ECAAADA52F
    A099A0E79BBBD7D2862FD513928370A53D3A2AF9A8F70410CCC1BFE5B97EE213B9A749EE2*/
    uint32_t TDES_CBC_EN[16]={0x3723A485,0x3E2EEB10,0x9E5434C4,0x2692C8FD,0x978D5743,
    0x10CBCFD7,0x873A396C,0xD9CF6AEB,0x5C8953FC,0xD62F3744,
    0xDE2D0B60,0x1DA22B35,0x00793D6F,0x543CD424,0x833BE660,
    0x05703F52};
    /*TDES_CBC_EN=0x85A4233710EB2E3EC434549EFDC8922643578D97D7CF7CB106C393A87EB
    6ACFD9FC53895C44372FD6600B2DDE352BA21D6F3D790024D43C5460E63B83523F7005*/
    uint32_t
    TDES_CBC_DE[16]={0xED617A1E,0xBFDAACE87,0x1FCAFD57,0x8D3ECA85,0x72154F73,
    0xF84F987F,0xE8AABC7B,0xEFB3677F,0xC5F7CC4C,0x9F3AD2C8,
    0x40779B72,0x00D7F205,0xF1A8B424,0x9A389EA2,0x3EEC58F4,
    0x1546667E};
    /*TDES_CBC_DE=0x1E7A61ED87CEDABF57FDCA1F85CA3E8D734F15727F984FF87BBCAE8
    7F67B3EF4CCCF7C5C8D23A9F729B774005F2D70024B4A8F1A29E389AF458EC3E7E664615*/
    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;
}
}
uint32_t TDES_3Key_test()
{
    uint32_t i,flag1,flag2,flag3,flag4,ret=0;
    DES_PARM TDES_Parm={0};
    uint32_t in1[16]= {

```

```

0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,0xED8A71D
7,0xA66B2FAA,0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,0x05CDB663,0xF7D812
E4,0x86341DEB,0xBA52B237
};
uint32_t key1[6]={0x675BE5D2,0x1641A6AD,0x14531A6B,0xEBFA006E,0x90DFD0CD,
0x2D029B93};
uint32_t iv1[2]={0xB5CC3A62,0xC96EF050};
uint32_t out[16];
uint32_t TDES_ECB_EN[16]={0x5D6C633C,0x8EDFC4C7,0x3D02A02C,0x97431789,
0x83EF4C36,0xFF591C67,0xE869DB08,0xAB82D05B,0x11771439,0xDC6F79BB,0x5B46D128,0x
F52114F5,0x2C758CB4,0x1A4D1A6A,0x0DC3FBCA,0x82222BB2};
uint32_t TDES_ECB_DE[16]={0x6780A75A,0x62EC1AC8,0xD0341FF5,0x2260C44E,
0xF2720589,0xB0EBBBE0,0xBFE0991D,0x1EA78C1C,0xBAB53D00,0xE3FA25D6,0x9430DEF,0x
C465511C,0xEE9D2DFB,0x9796AADC,0x4FFFEF58,0x172D00A2};
uint32_t TDES_CBC_EN[16]={0x048BD8AD,0xF98F2C51,0x5F6FD563,0xA26A1038,
0x8017FC81,0xBBD5AF4C,0x0A7AEFFF,0xB7D428A1,0x316E31F7,0xD8F283E1,0xDDD4395,0x
8076C2D0,0x0434D1E9,0xD1A94D4D,0xFF3E3B5E,0x77C93116};
uint32_t TDES_CBC_DE[16]={0xD24C9D38,0xAB82EA98,0xEC4AAF78,0x8DB239A7,
0xD0565899,0xA4615EDD,0x78EF88CC,0x16B472C3,0x573F4CD7,0x45910A7C,0x874D72AE,0x
5E1D01CA,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 Demo

```

uint32_t AES_128_test()
{
    uint32_t flag1,flag2,flag3,flag4,flag5,flag6;
    uint32_t ret;
    AES_PARM AES_Parm={0};
    /* If the test case needs to be modified, when the true value of the parameter is
    "0x0102030405060708", since u32 data is stored in small end order, when initializing and assigning
    the above parameters, please enter "0x04030201,0x08070605". If there is no special instruction, this
    demo parameter is set in this way */
    uint32_t 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=0xA570874A98DAC273D1522DF5464A885FD572CF8D7D200F2ACEF579749EBEB53FFE98
    793D6D58597C4B29E1309077E1B3BD0C08CA1379B42A03B8093BE70F411BEF3742E65EBE763
    5F6AAD7E4B05F4919B1C32D817A9F33DD5F496FBE3A80B08C60970DCDD4D6C0A4BB1D389
    810CA699799DD673B1A5A334CC8EFD48500D7AA9B*/
    uint32_t key[4]={0x7FDDA35D,0x7D5C725B,0x1960F327,0x4FD9DDA2};
    /*key=0x5DA3DD7F5B725C7D27F36019A2DDD94F*/
    uint32_t iv[4]={0x7B00FE39,0xD3E06638,0xD52BC983,0x38E98017};
    /*iv=0x39FE007B3866E0D383C92BD51780E938*/
    uint32_t out[32];
    uint32_t 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=0x38544EB203A345017FA250C470EEDE2A4E316F90AD2942B20E361213228B
    9C9402BCC1E29E236019E5D5D2CAE27DC58DE19C42137608FCE8DB8145CA50900108F84229
    4B623C07D648B63F1127CC671989990B25E08011860C0E451AAF27D7818E6079B66916D35399
    1E071DDBB6CE4205420944681633D098B704279C7E346E*/
    uint32_t AES_ECB_DE[32]={0x818D1AFD,0xEC4B4F8E,0x69D9F9FF,0x5567B549,
    0x42DD5C4B,0x3BCA1DD3,0xF318E616,0x89297FEC,0x2A3E0A06,0xFDA90D61,
    0x93DCAE5D,0xCF1AFEAE,0x3CF5A889,0x4CFFFEFE3,0xB2C42607,
    0x37D43F8A,0x9C1CD1D8,0x2FE878E8,0x22D941C3,0x239B9D2D,
    0xD9FEB719,0xA4F9E01C,0xC9C39FE8,0x336B01FA,0xFD12E415,
    0x2B6A0006,0x4A35AFBC,0xA7942FAB,0x09DF0A3A,0x9545521B,

```

```

0x7E009336,0x030A5DA5};
/*AES_ECB_DE=0xFD1A8D818E4F4BECFFF9D96949B567554B5CDD42D31DCA3B16E618F3E
C7F2989060A3E2A610DA9FD5DAEDC93AEFE1ACF89A8F53CE3EFFF4C0726C4B28A3FD437D
8D11C9CE878E82FC341D9222D9D9B2319B7FED91CE0F9A4E89FC3C9FA016B3315E412FD06
006A2BBCAF354AAB2F94A73A0ADF091B5245953693007EA55D0A03*/
uint32_tAES_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=0x06E0838A10B63AACCBBC4D20CA92AF2213C3E966154DE2F992385407EFF
6192742D8059E13C7E80BC67AF161C294657E7253257730D6088EEBBE04F328CA4267470E4A
98E0FBEB14D2E9CC49D5BA041AD43917C9EF25BF5155D19E7F4461E9E2D09988EB7D031F
8F80A89F1129CFFAB0CE6FCE649A5CD5661F78EE3FF1DAF6249*/
uint32_tAES_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};
uint32_tAES_CTR_EN[32]={0xF14C3DA0,0xA74E1089,0x81480939,0x5C8D4E8D,
0x655E20AB,0x6D797028,0x1E355F48,0x58184929,0x52B1495A,0xC15EB91D,0xFBD499AB,
0xF59B39FE,0x96DAE1C3,0x6ECC9CDA,0xDA1FB535,0xAA1C74B2,0xA3F19C5E,
0x9944E1A6,0xDAA05E9A,0xB96278E3,0x1E4915FC,0xB77FBBD2,0x92BA80B9,
0xCA97857E,0x509D0365,0x78A6FD99,0xB56F5B3C,0xFBEFF5B2,0xF9E928C6,
0xBC28AE3A,0xD8B82D7A,0xA99BF98D};
uint32_tAES_CTR_DE[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,
0x8DCF72D5,0x2A0F207D,0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0x
B3E17790,0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,
0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,
0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,
0x85D4EFC8,0x9BAAD700};
/*AES_CBC_DE=0xC4E48DFAB629AB3F7C30F2BC5E358E6DEE2C5A084BC70848C7B43506AA
35A1D6D378F1A71C2DA6D7935BA5E73040AFF077308C018EB7A6304C0F25821A48358465DD1
456FB015C05C0F9D019CA92DA38F680BC3F425E8F911E35142D4A5E222AA4273F7C7C9F59F
6E3E65AF491AF242B5A9DD2C4CF8485318D8E38E6B5976394*/
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;
}
}

```

```
uint32_t AES_192_test()
```

```

{
    uint32_t flag1,flag2,flag3,flag4,flag5,flag6,ret=0;
    AES_PARM AES_Parm={0};
    uint32_t in[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,
    0xDE77B9E1,0xE1A33077,0xE3AEA215,0x2670CBEB,0x160CA5C2,0x86808BEA,
    0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,0xC4BD0D05,0xAB21B81D,0x7477E171,0x
    DE6FFE4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,0x9C3A69BA,0x6F7CD7A9,0x2
    BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,0x03D4B04F,0xE0EAAAC56,0x73CC7E9C};
    uint32_t key[6]={0xA1148977,0xCFA42A1F,0x9D983F36,0x521C1313,0xDAD2CB6F,
    0xC6254819};
    uint32_t iv[4]={0xFCAA7077,0x44DB6BB5,0xDC74178D,0xA91A44D6};
    uint32_t out[32];
    uint32_t AES_ECB_EN[32]={0x9FCB396D,0xF9A6B55C,0x4CCE7669,0x917CAF2F,
    0x71F8907D,0xC6893936,0x5ABA1DFB,0xA933FF81,0xBD33847F,0x0F1B2F6C,0x1B4AACA7,
    0xE555E2EE,0x0CBD4683,0x76ECD138,0x7BFE81E8,0xE05FE788,0xAF688124,0xED29ACF2,
    0xCE424458,0x8E304A1C,0xE5A21E6C,0x3C7D433A,0x32DC028D,0x697F9624,0xB451070E,0x
    F82A4488,0x33D99F4C,0x7FBCC3E,0x8BB01E57,0x0C1EE01B,0x6D96FF7F,0xDEC84BD8};
    uint32_t AES_ECB_DE[32]={0x41F29D18,0x13C52105,0xB24DBDDD,0x46B6BAB9,
    0x95F63F1A,0x28B24F73,0xAA774293,0xA086E548,0xD446667D,0xF8D67CCE,
    0x7AC5BD02,0xE43EE791,0x25B857B4,0x30A3D7FB,0x8DB4C416,0xAE6B0B0C,
    0x0F7E89E1,0xBA900B96,0x516EC69B,0xBED1D082,0x3590FD32,0x878C5EE5,0x91B71430,0x
    6A005A7F,0x0627EF04,0x28D96A77,0xF8DCDCFC,0x790D0304,0x02149E37,0xDC8E518D,0x8
    0D75D77,0x80670408};
    uint32_t AES_CBC_EN[32]={0xE5682F2E,0x07A087E9,0x37D60ED6,0x41262C81,
    0xD69A23B5,0x1800A3FD,0xAC50301D,0xB12F3C5E,0x568A1F62,0xC1057524,0x7E7D09BC,
    0x26F42541,0x5C2FB09B,0x12C68EFC,0xE03B2AF8,0x6E2C9934,0xD805445F,
    0x3876A6E4,0xCA85688F,0xD1116501,0x2DE18902,0xCBFE9B2,0x57911796,
    0x0719A673,0x3915B680,0x3B760C23,0x23F715DE,0x6D3425B9,0x9C339EF5,0x6C91D7B0,
    0x050E91DA,0x286AB477};
    uint32_t AES_CBC_DE[32]={0xBD58ED6F,0x571E4AB0,0x6E39AA50,0xEFACFE6F,
    0xCFB4F836,0x21432C5A,0x43CA36B8,0x148505B7,0x6E05BE79,0x26A1C52F,0x9B668D75,0x
    07904584,0x03C89C5F,0x26AF7239,0x0B344FFC,0x9311957F,0xBE10E141,0xA875B40E,
    0xDB762AC4,0x7A6CDD87,0x9EB1452F,0xF3FBBF94,0x4FD8EAC4,0xD20B3287,0xA288EAA5
    ,0x34AE4EED,0x4A1074FA,0xE5376ABE,0x6D68499E,0xF757B012,0xF8634844,0xAF390CFF};
    uint32_t AES_CTR_EN[32]={0xF4EB3E15,0xCEC90E4B,0x1708E770,0x6A1297BB,
    0x045A69FD,0x7FC870A7,0x56BE6A22,0x5A912CEA,0xC22E6811,0x37177967,0x68D08A6A,0x
    CECA04AE,0x30EA7217,0x16992F79,0xF0DD4DAD,0x4710126B,0xCC06BD7F,
    0x03093EE5,0x596D2B9B,0xD9844F7C,0x130D4E24,0xD6C87ABF,0xE1745614,0xEF260225,

```

```
0x0F90C354,0x7557E159,0x4CBC3789,0xDB0552F8,0x28F27315,0x046363A6,0xAF1F0089,
0x29AC2CC1};
```

```
uint32_t AES_CTR_DE[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,
0xBA43D804,0xDE77B9E1,0xE1A33077,0xE3AEA215,0x2670CBEB,0x160CA5C2,0x86808BEA,
0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,0xC4BD0D05,0xAB21B81D,
0x7477E171,0xDE6FFEF4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,
0x9C3A69BA,0x6F7CD7A9,0x2BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,
0x03D4B04F,0xE0EAAAC56,0x73CC7E9C};
```

```
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;
    }
}
uint32_t AES_256_test()
{
    uint32_t flag1,flag2,flag3,flag4,flag5,flag6,ret=0;
    AES_PARM AES_Parm={0};
    uint32_t 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,0x39B678E4,0x84986F66,0x94
BB95FA,0x976D9797};
    uint32_t key[8]={0xB2591B82,0xD25676DB,0x2546F076,0xC8D01753,0xB4A620E7,
0x4AADD91D,0x2E5EDF9B,0x596C1146};
    uint32_t iv[4]={0xF0E72786,0xD272F169,0x0ECED17B,0x29D34319};
    uint32_t out[32];

    uint32_t AES_ECB_EN[32]={0x5766DACC,0x50DBB1F9,0x58720E73,0x2182AA3E,
0x7D5A6D4D,0xA07EF43D,0x5A533E1E,0x34816CF3,0xBA23F9CD,0x99A7BD14,0x6789D933,0x
D14B2F0D,0xAF53E19E,0xB88DA31F,0xEFB0472,0x03F077B1,0x4489E477,0x97161707,
0x6C24CB62,0x0FF361DC,0x60BBD2CF,0xEB7AB0C1,0xFA3421E5,0x2F5DB80E,
0x2D61A7CD,0x22988E98,0x51B195AF,0x22C8A4C0,0x7F8E90C3,0x6690789A,0x48AF0FAF,
0xAC16F7A6};
    uint32_t AES_ECB_DE[32]={0x0ADBD493,0x93C512ED,0x6A99A60B,0x0A1841B5,
0x135E685D,0xB9ADC987,0x6262573F,0x9090A7D3,0x2B7DDAA3,0x7370FB9D,0xE7E739C6,
0xCA013CA6,0x3509E08F,0x74A21641,0x3D2C9527,0xF8DF90F0,0xED8209E9,0x9DD57975,0x0
A506603,0x7C2EFD3B,0x0937237E,0x2828BAAF,0x245E9D40,0xF3BB882A,
0x66E82B24,0xF3E778E7,0x386802D1,0xD74C7057,0xEF8525C8,0x1EB7AA48,0x362EACDD,
0x8AA0F286};
    uint32_t AES_CBC_EN[32]={0x39AD6F3A,0xF8E3E1DD,0x2209A14B,0x241642CC,
0x83FA4820,0xD82816B3,0xEF66B17A,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};
uint32_t AES_CBC_DE[32]={0xFA3CFD15,0x41B7E384,0x64577770,0x23CB02AC,
0x95811940,0x0069DBAA,0x7154DC12,0xC335689C,0x9682708F,0xC7A4485D,0x6C5E4570,
0x53EB3740,0xBE3A6E92,0x8AB25C5D,0x733F40C4,0x505915DF,0x8AD021A8,0x00CA8C94,0x
E5EDA5A0,0xDBEC8452,0x0D42E557,0xFCC3A85F,0x612E2967,0x0A92ED3C,
0x3E1FDF82,0xD97A448C,0x5D4E5630,0x94CD75A1,0x77EAA401,0x7D28FBFA,0x95383C5F,0x
E675A58A};
uint32_t AES_CTR_EN[32]={0x85F1DD33,0xAE808F2F,0x26A40960,0xB2020DF8,
0xB6C2006E,0xA22A35F6,0x33BB584A,0xBFEA7F68,0x73E54E78,0xF3EB0368,0x80816676,
0x6109DE39,0xE0001920,0x8D2B18B8,0x0E46A012,0xE43F1DD1,0x3CA4BC36,0xD5101452,
0x83020170,0x4B752F62,0x3D27A004,0x3C18B5DB,0x99DA9032,0xEA59B340,0x79BBD087,
0x2EF8CB3D,0xDC32D3CA,0x30F577EA,0x56774C66,0xC33DA1F8,0x0288B1D6, 0x091C9666};
uint32_t AES_CTR_DE[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,
0xBDFFAA2C,0xB4D4B3C0,0x8BB97CB6,0x99EA0BE6,0x8B338E1D,0xFE104A1C,
0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBC3A3,0xA7C27969,
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 Algorithm Library Function Demo

```

uint32_t MD5_fixed_steps_test(void)
{
    uint8_t out[16];
    char in[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    uint8_t 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, (uint8_t*)in, 28);
    HASH_Update(ctx, ((uint8_t*)in)+ 28, 28);
    HASH_Update(ctx, ((uint8_t*)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 case
uint32_t SM3_test(void)
{
    uint8_t out[32];
    //SM3 Fixed step hash
    // Step by step message
    uint8_t 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 summary
uint8_t 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
};
uint8_t i,byteLen=48;
HASH_CTX ctx[1];
// Set to SM3 operation
// ctx->hashAlg Different HASH operations can be selected,
//for example 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 by step SM3 test failed
    printf("SM3-FIX-Test fail\r\n");
    return HASH_ATTACK;
}
else
{
    // Step by step SM3 test succeeded
    printf("SM3-FIX-Test success\r\n");
}
return SM3_Hash_OK;
}

```

// This function routine performs single-step hash operation on hash sha1/224/256 respectively

```
uint32_t HASH_test(void)
{
    uint32_t TEST_BUF[200];
    uint8_t 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,0x84,0xA3,0xFC,0x39,0xDE,0x8C,0xD6,0xFC,0x2F,0xC8,0x88
    };
    uint8_t in2[10] = {0x1C,0x61,0xAD,0x6C,0x05,0xF3,0x98,0xA4,0x4C,0xFD};
    uint8_t out[64];
    uint8_t sha1_out[20]=
    {
        0x0E,0xEC,0x49,0xC5,0x36,0xBB,0xD7,0x87,0xD2,0xE2,0x0C,0x97,0xC4,0xF8,0x65,0x7C,
        0xCC,0x74,0x8D,0x1E
    };
    uint8_t sha224_out[28]=
    {
        0xC1,0x44,0x4F,0xD0,0xB8,0xA9,0xA3,0xD9,0xE8,0x04,0xA0,0xD1,0x9E,0x38,0xF3,0x5E,0x8
        5,0xB4,0x0F,0x10,0x5A,0x1C,0x48,0xC4,0xF2,0x40,0x10,0x48
    };
    uint8_t 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
    };
    uint8_t SM3_out[32]=
    {
        0xBD,0x77,0x63,0x33,0x0A,0x71,0x19,0x5C,0x5D,0x26,0xE7,0x99,0x7B,0x41,0x22,0xB0,
        0xBC,0xB0,0xBE,0x52,0x3E,0xDA,0x0F,0xBE,0xE6,0xA4,0x33,0x96,0xB8,0x83,0x76, 0xD4
    };
    uint32_t 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 SM7 Algorithm Library Function Demo

```

uint32_t SM7_test(void)
{
    uint32_t flag1,flag2,flag3,flag4;
    uint32_t ret;
    SM7_PARM SM7_Parm={0};
    /* If the test case needs to be modified, when the true value of the parameter is
    "0x0102030405060708", since u32 data is stored in small end order, when initializing and
    assigning the above parameters, please enter "0x04030201,0x08070605". If there is no special
    instruction, this demo parameter is set in this way */
    uint32_t 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
    };
    uint32_t key1[4]={0x84853E30,0xB3D3154D,0x9A887F49,0xDC65910A};
    uint32_t iv1[2]={0x2FA6B65A,0x1D0EC205};
    uint32_t out[32];
    uint32_t SM7_ECB_EN[32]=
    {
        0xFDC0A8D5,0x92728D71,0x5A804C88,0x430AB48D,0x9D20E77A,0x74ADB168,
        0xD4848355,0x92C8EE23,0x5B0C32C4,0x3D612420,0x8B42878A,0x6D3B380E,
        0x21CD8165,0x66013D3D,0xD7BD0FB4,0xA6666999,0x30588D82,0xB105C519,
        0xEF0A9B40,0xDBC36099,0x0AF7F6AF,0x51FFE183,0xC7A983D3,0xB766EA14,
        0xFBFA1269,0x6AAF5BFA,0xD23E3184,0x05AA9FB6,0xD3270BB1,0x1B146200,
        0xD5A3E2FC,0x1348DE04
    };
    uint32_t SM7_ECB_DE[32]=
    {
        0x42FE0F12,0x13C56F4B,0x924BB950,0x8BB82615,0xB3F69779,0x91A42F3E,
        0x8165114F,0xCE65AE5F,0x397E1BE4,0xD987776C,0x4B4FC3A0,0x2ADFD517,0x34A36717
        ,0x94D408B5,0xFF72D19C,0xBAAC265E,0x19DD3CD9,0xB50E3835,0x76307CBE,0x9125D4
        79,0x3A900FF2,0xD97B7140,0x24783470,0xDE5D0187,0xA192A61E,0x0D56C887,0x82FB9C
        72,0x714D123B,0xF7C4876E,0xEF1D331A,0x387E2635,0x5A7343CA
    };
    uint32_t SM7_CBC_EN[32]=
    {
        0x969843B8,0x1E7267D5,0x40640F93,0xC04D7107,0xE94FC3DA,0x3B47318F,
        0xEA6FE714,0xC4046E8C,0xDF311550,0xF4A404FD,0xFF591CCE,0xDDDA6AAA,
        0xC9AE0C6C,0x4CF1AE42,0x90F03ED4,0xC061F7D2,0xF10DA467,0xEB896034,
        0x53A0FBC9,0x9A1059D2,0x7FAE69C6,0xB664E266,0xF101AE3E,0x003864EE,
    }
}

```

```

0xEC4A469B,0x85840724,0xF3D7D05D,0x8B1B7B50,0xC6B4E78D,0xE4F104E5,
0xB405AB34,0xD799B659
};
uint32_t SM7_CBC_DE[32]=
{
    0x6D58B948,0xECBAD4E,0xD91EA520,0x5EF58015,0x09545D06,0x9B1E43E6,
    0x16D98C32,0x6300A917,0x3CEEEAA7,0xAB0F8A63,0xD590D3A5,0x9D0B7310,
    0xBA77E7DE,0xA02944EC,0x68BBF9F2,0x6A0E1E09,0x03631CFF,0xD46D6FBF,
    0x83CBD30A,0xE3FEA5CE,0x1BB17BC3,0x21C53F8A,0x93452468,0x728E7995,
    0x9E6338F9,0x41C3BD39,0x7300B4EC,0x42245328,0x79013C7E,0xD400CC45,
    0x91A88390,0xE3FED302
};
Cpy_U32(out, in1,32);
SM7_Parm.in = out;
SM7_Parm.key = key1;
SM7_Parm.out = out;
SM7_Parm.Mode = SM7_ECB;
SM7_Parm.inWordLen = 32;
SM7_Parm.En_De = SM7_ENC;
ret=SM7_Init(&SM7_Parm);
ret=(SM7_Crypto(&SM7_Parm));
SM7_Close();
if (ret!=SM7_Crypto_OK)
{
    flag1=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM7_ECB_EN, 32, out, 32))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }
}
Cpy_U32(out, in1,32);
SM7_Parm.En_De = SM7_DEC;
ret=SM7_Init(&SM7_Parm);
ret=(SM7_Crypto(&SM7_Parm));
SM7_Close();
if (ret!=SM7_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM7_ECB_DE, 32, out, 32))
    {

```

```

        flag2=0x5A5A5A5A;
    }
    else
    {
        flag2=0;
    }
}
Cpy_U32(out, in1,32);
SM7_Parm.iv = iv1;
SM7_Parm.Mode = SM7_CBC;
SM7_Parm.En_De = SM7_ENC;
ret=SM7_Init(&SM7_Parm);
ret=(SM7_Crypto(&SM7_Parm));
SM7_Close();
if (ret!=SM7_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM7_CBC_EN, 32, out, 32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}
Cpy_U32(out, in1,32);
SM7_Parm.iv = iv1;
SM7_Parm.En_De = SM7_DEC;
ret=SM7_Init(&SM7_Parm);
ret=(SM7_Crypto(&SM7_Parm));
SM7_Close();
if (ret!=SM7_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM7_CBC_DE, 32, out, 32))
    {
        flag4=0x5A5A5A5A;
    }
    else
    {
        flag4=0;
    }
}
}

```

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

vi. Appendix VI SM4 Algorithm Library Function Demo

```

uint32_t SM4_test(void)
{
    uint32_t flag1,flag2,flag3,flag4;
    uint32_t ret;
    SM4_PARM SM4_Parm={0};
    /* If the test case needs to be modified, when the true value of the parameter is
    "0x0102030405060708", since u32 data is stored in small end order, when initializing and
    assigning the above parameters, please enter "0x04030201,0x08070605". If there is no special
    instruction. this demo parameter is set in this way */
    uint32_t 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
    };
    uint32_t key1[4]={0x84853E30,0xB3D3154D,0x9A887F49,0xDC65910A};
    uint32_t iv1[4]={0x2FA6B65A,0x1D0EC205,0xB90B8620,0x42E74F58};
    uint32_t out[32];
    uint32_t 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=0x9C381AD6ADA036E17E6B62BD73F17742825E3EAFDF846D877B5B067AA8F
    FBB1CDC317CC5FC6AD85BEFEA2508A4620160AC87473E7925B358BF35913A7CA106B8C4F454
    988FD25C061FF2FD684C2CA69C6EA72F5B2B3A69ECF6AD28F018EDA2FAB1B4956369009B7A
    4CE0559D3FC2CDA5926CC57FA199F189E1D928F20A0505D7*/

    uint32_t 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=0xA0DF07310A3DEEC1D5F9259081C0AC900AF9727ACEF181645054DF76CF
2A26CD3B3C8ECE90738B2026F5F8C9CCFF731A6FE2B60A4A542BA002D60C76A40C256D67F
F77249EC3CB44CCF5EC844406F37D1CD44687ECB942CB8C5975E9416C752870C8C364B3CB
A89E8EA92FBA7BBA101BA0E8501CFD97E61ED5DDE2A412D929BB*/
```

```
uint32_t 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=0x3C1C4E309D64DA10BEB5BC5E84AD642956975918D2AA0621244B36842DE
6A9573BB060D1743A29589F3857EEC2698E395909FD63FD84455BBEE8A64D01458E579B15B0
7404860E5749DBE2387E3828E08449DECD9F7E716B98D616E55E022065A787D1C83F37086E66
2647C3410D4A65953B367FD2B58EAD2AF1F0015AD66981*/
uint32_t 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=0xFA69A11E0FFFE0DCF57F2E29D98F4BD27AE52731CE57CCB12F9E7DCC1
7469CC746A13259D874EE8D650468CCC302FB686AF269944DF2FF17CB56D8F8FD40D85909D
7BEB3C9FB6994EAD5529ECE51901CA87BBBD725BC899B9BD2D54C88B22CBD068D8FED3A1B
37B326937DE85C5CF85573EC0ABEDEED68F2DC566272A4D263480*/
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;
}
}
```

vii. Appendix VII RNG Algorithm Library Function Demo

```

#define POKER_RAND_BYTE 40 //320bit
uint32_t TrueRand_Poker_Test(void)
{
    u16 count[16] = {0};
    uint32_t sum = 0;
    uint8_t rand[POKER_RAND_BYTE];
    uint8_t i, j, k, tmp;

    GetTrueRand_U32((uint32_t*)rand, POKER_RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER_RAND_BYTE);
    //GetPseudoRand_U32((uint32_t*)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 += ((uint32_t)count[i]) * count[i];
    }

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

uint32_t PseudoRand_Poker_Test(void)
{
    u16 count[16] = {0};
    uint32_t sum = 0;
    uint8_t rand[POKER_RAND_BYTE];
    uint8_t i, j, k, tmp;
    //GetTrueRand_U32((uint32_t*)rand, POKER_RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER_RAND_BYTE);
    GetPseudoRand_U32((uint32_t*)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 += ((uint32_t)count[i]) * count[i];
}
if(405 < sum && sum < 687)
    return 0;
else
    return 1;
}
```

viii. Appendix VIII SM1 Algorithm Library Function Demo

```

uint32_t SM1_test(void)
{
    uint32_t flag1,flag2,flag3,flag4;
    uint32_t ret;
    SM1_PARM SM1_Parm={0};
    /* If the test case needs to be modified, when the true value of the parameter is
    "0x0102030405060708", since u32 data is stored in small end order, when initializing and assigning
    the above parameters, please enter "0x04030201,0x08070605". If there is no special instruction, this
    demo parameter is set in this way */
    uint32_t 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
    };
    uint32_t key1[4]={0x84853E30,0xB3D3154D,0x9A887F49,0xDC65910A};
    uint32_t iv1[4]={0x2FA6B65A,0x1D0EC205,0xB90B8620,0x42E74F58};
    uint32_t out[32];
    uint32_tSM1_ECB_EN[32]={
        0x3E244A82,0x5E5BFFB7,0x6C09BB78,0x1D528A72,0xD71DD7D3,
        0xB2C63572,0xCAB798B7,0xE98B7E7E,0x31B74EA0,0x2B0F7AFF,
        0xD2B67660,0xF2F95230,0x1ABB0C33,0x453DD692,0xC18728A9,
        0xB0A8C5A8,0x216F18A1,0x8956499A,0x6D1A2E36,0x0A90F9DE,
        0x977AC571,0x44126188,0x889801FA,0x264B1879,0x14B71EC3,
        0x62249397,0xF99B8C04,0x1154D5D8,0x1B16B017,0x4477C020,
        0xB1D85955,0xB006BCB7};

    /*SM1_ECB_EN=0x824A243EB7FF5B5E78BB096C728A521DD3D71DD77235C6B2B798B7CA7
    E7E8BE9A04EB731FF7A0F2B6076B6D23052F9F2330CBB1A92D63D45A92887C1A8C5A8B0A1
    186F219A495689362E1A6DDEF9900A71C57A9788611244FA01988879184B26C31EB714979324
    62048C9BF9D8D5541117B0161B20C077445559D8B1B7BC06B0*/

    uint32_tSM1_ECB_DE[32]={
        0xA357F33D,0xEB1489B1,0x71D862D4,0x4B512067,
        0xFFC1DE29,0xE4BA99AA,0x269DC310,0xA9FFFD6C,0x114C4507,0x7CE06089,
        0x693512EF,0x9574B52F,0xBB222811,0xA84EEC57,0xEF6E3FDC,
        0x11F5714C,0x6EB451F4,0xD818C7B6,0x2C003C47,0x281183A1,
        0x20E2D39F,0x4F868F49,0x28223D38,0xDB20BBC0,0x486B8235,
        0x5DE53208,0x9BEE9D24,0x2787CF79,0xAC6CA11D,0xA4BBBDC2,
        0x961D4845,0x239BC8E1};

```

```
/*SMI_ECB_DE=0x3DF357A3B18914EBD462D8716720514B29DEC1FFAA99BAE410C39D266
CFDFFA907454C118960E07CEF1235692FB57495112822BB57EC4EA8DC3F6EEF4C71F511F4
51B46EB6C718D8473C002CA18311289FD3E220498F864F383D2228C0BB20DB35826B480832E
55D249DEE9B79CF87271DA16CACC2BDBBA445481D96E1C89B23*/
```

```
uint32_t SMI_CBC_EN[32]={
    0x56583277,0x048D4BFE,0xD505B83B,0x69D8F23F,0x1FC9D047,
    0x09522EC1,0xC77CFE03,0x59CB89D2,0x01E97431,0xF981C0FB,
    0x887184D0,0x33716293,0x2886538C,0xC0961363,0x9DCBF1FA,
    0x9BCBF5AF,0x9E9519C9,0x102FD1E9,0x8B54747D,0x283C5E40,
    0xBFA30847,0xB0752EC2,0xD21F7B3C,0x4559D420,0xBC7CD8E9,
    0xB6CC72ED,0x4E8F1B1B,0x4FACCAF2,0x3F14A032,0x70A79877,
    0xE6C1F6DA,0xEFE11EEC};
```

```
/*SMI_CBC_EN=0x77325856FE4B8D043BB805D53FF2D86947D0C91FC12E520903FE7CC7D289
CB593174E901FBC081F9D0847188936271338C538628631396C0FAF1CB9DAFF5CB9BC919959E
E9D12F107D74548B405E3C284708A3BFC22E75B03C7B1FD220D45945E9D87CBCED72CCB61B
1B8F4EF2CAAC4F32A0143F7798A770DAF6C1E6EC1EE1EF*/
```

```
uint32_t SMI_CBC_DE[32]={0x8CF14567,0xF61A4BB4,0xC8D3E4F4,0x09B66F3F,
    0xB494C259,0x31F73FAA,0x9C3F096F,0xA34591B4,0x86F0D87A,0xD18567C1,
    0x6CA5E3AC,0xE7FC4820,0x25FD3814,0x1F9A4A50,0x61BABF15,
    0x25083D15,0xF97D799A,0x08BAFFE1,0x36BE1C61,0x4972D42B,
    0xD5197C2B,0x3D5DFEFE,0x09034909,0x239EF50A,0xFF56922D,
    0xF1364A1A,0xA41F03C3,0x6B12BAC7,0x5D978983,0x97D2FCD1,
    0x18D8F355,0x188637BE};
/*SMI_CBC_DE=0x6745F18CB44B1AF6F4E4D3C83F6FB60959C294B4AA3FF7316F093F9CB4
914A37AD8F086C16785D1ACE3A56C2048FCE71438FD25504A9A1F15BFBA61153D08259A797
DF9E1FFBA08611CBE362BD472492B7C19D5FEFE5D3D094903090AF59E232D9256FF1A4A36
F1C3031FA4C7BA126B8389975DD1FCD29755F3D818BE378618*/
```

```
Cpy_U32(out, in1, 32);
SMI_Parm.in = out;
SMI_Parm.key = key1;
SMI_Parm.out = out;
SMI_Parm.Mode = SMI_ECB;
SMI_Parm.inWordLen = 32;
SMI_Parm.En_De = SMI_ENC;
ret=SMI_Init(&SMI_Parm);
ret=(SMI_Crypto(&SMI_Parm));
SMI_Close();
if (ret!=SMI_Crypto_OK)
{
    flag1=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SMI_ECB_EN, 32, out, 32))
    {
        flag1=0x5A5A5A5A;
    }
}
```

```

    else
    {
        flag1=0;
    }
}
Cpy_U32(out, in1,32);
SMI_Parm.En_De = SMI_DEC;
ret=SMI_Init(&SMI_Parm);
ret=(SMI_Crypto(&SMI_Parm));
SMI_Close();  if (ret!=SMI_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SMI_ECB_DE, 32, out, 32))
    {
        flag2=0x5A5A5A5A;
    }
    else
    {
        flag2=0;
    }
}
Cpy_U32(out, in1,32);
SMI_Parm.iv = iv1;
SMI_Parm.Mode = SMI_CBC;
SMI_Parm.En_De = SMI_ENC;
ret=SMI_Init(&SMI_Parm);
ret=(SMI_Crypto(&SMI_Parm));
SMI_Close();
if (ret!=SMI_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SMI_CBC_EN, 32, out, 32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}
Cpy_U32(out, in1,32);
SMI_Parm.iv = iv1;
SMI_Parm.En_De = SMI_DEC;
ret=SMI_Init(&SMI_Parm);

```



```
ret=(SM1_Crypto(&SM1_Parm));
SM1_Close();
if (ret!=SM1_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM1_CBC_DE, 32, out, 32))
    {
        flag4=0x5A5A5A5A;
    }
    else
    {
        flag4=0;
    }
}
if (flag1|flag2|flag3|flag4)
{
    return 0x5A5A5A5A;
}
else
{
    return 0;
}
}
```

ix. Appendix IX SM2 Algorithm Library Function Demo

```

void SM2_test(void)
{
    uint8_t ID[20] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x00, 0x11, 0x22, 0x33, 0x44};
    uint8_t msg[40] =
    {
        0x0D,0x3D,0x6B,0x96,0x88,0x33,0x3F,0xF1,0x93,0xF0,0xF6,
        0xDE,0x6E,0xC0,0x0D,0x3D,0x6B,0x96,0x88,0x33,0x3F,0xF1,
        0x93,0xF0,0xF6
    };
    //uint8_t priKey[32] = {0};
    uint8_t priKey[32] = {
        0x44,0x13,0xFB,0xED,0xFE,0x4D,0x61,0x09,
        0x9A,0x33,0x4C,0xAA,0xB9,0x80,0x64,0x7B,
        0x63,0xD9,0x75,0xAB,0xC5,0x9D,0x83,0xDA,
        0xC8,0x88,0x7E,0xB0,0x08,0xCB,0x49,0xE1
    };
    uint8_t pubKey[65] = {
        0x04,0x75,0xF7,0xFF,0x18,0x74,0x40,0x69,0x6D,
        0x5B,0x7C,0x62,0x34,0x40,0xFA,0x02,0x99,
        0x18,0x67,0x2C,0xFD,0x48,0x9E,0xFE,0x9D,
        0x5D,0xF5,0xA3,0xB4,0x89,0x5B,0xEB,0x38,
        0xD2,0x9A,0x3F,0x9F,0xCF,0x63,0xC1,0xF1,
        0xEB,0x64,0x9A,0xCB,0x25,0x9F,0x93,0x83,
        0x0E,0x88,0x6A,0x4E,0xEE,0xDD,0x83,0x06,
        0xCE,0x5D,0xFF,0x6F,0xEF,0x19,0xBF,0xEE};
    uint8_t r[32] =
    {0x85, 0xB1, 0x14, 0xC7, 0x6F, 0x02, 0xB3, 0xFA, 0xE0,
    0x69, 0x23, 0xE6, 0xDE, 0xF4, 0x4D, 0xC3, 0x2F, 0x3A,
    0x43, 0xD2, 0xF9, 0xB7, 0xB6, 0x99, 0xBB, 0xDE, 0x7B, 0x25, 0x10,
    0xAC, 0x46, 0x7B},
    s[32] = {
        0x06, 0x06, 0xE1, 0xF9, 0x10, 0xA3, 0x3D, 0x27, 0x6B,
        0x48, 0x62, 0xAD, 0x44, 0xA3, 0xCA, 0xEA, 0xB5, 0x6A, 0x4F, 0x67,
        0x2B, 0xF2, 0xA0, 0x41, 0x3B, 0x13, 0xB1, 0x96, 0x7D, 0x08, 0x9A, 0x8D};
    uint8_t Z[32];
    uint8_t E[32];
    uint32_t ret;
    uint8_t pubKey1[65]={0};
    uint8_t C[140] = {0};
    uint32_t CByteLen = 0, MByteLen = 0;

```

```

uint8_t M[40] = {0};
uint32_t flag1;
uint32_t kByteLen=33;
uint8_t KA1[33],S11[32],SA1[32],KA2[33],S12[32],SA2[32];
uint8_t role1=1, role0=0;
uint8_t IDA[16]={0xBB,0xD2,0xB0,0xAB,0xFB,0xF2,0xCF,0x5D,0x6C,0x78,0x55,0xFA,0xA9,
0x64,0xBB,0x7A};
/*IDA=0xBBD2B0ABFBF2CF5D6C7855FAA964BB7A*/

uint8_t IDB[16]={0xAA,0xF0,0x0F,0x68,0x73,0xD2,0x55,0x60,0x72,0x8E,0x36,
0xBA,0x2C,0x1A,0x1C,0x28};
/*IDB=0xA AF00F6873D25560728E36BA2C1A1C28*/
uint16 IDByteLen=16,IDBByteLen=16;
uint8_t dA[32]={0x5A,0x86,0x38,0xA7,0x92,0x09,0x8E,0x99,0xE7,0x65,0x30,0x41,
0x7E,0xA7,0xE5,0x68,0x74,0xF8,0xBC,0x21,0x5A,0xE6,0x89,0x9E,0x4F,0xE7,0x05,
0x05,0xD6,0x3C,0x37,0xAC};
/*dA=0x5A8638A792098E99E76530417EA7E56874F8BC215AE6899E4FE70505D63C37AC*/
uint8_t PA[65]={0x04,0x75,0x0F,0xF5,0x33,0x8E,0xD8,0xF6,0xCD,0x8D,
0x8E,0x5B,0xF8,0x6D,0x07,0xB2,0xFF,0xFD,0xEE,0x0A,0xEC,0xDF,0x76,0xB2,0xE7,
0xE0,0xE3,0x67,0x82,0x5C,0xFD,0x0F,0x5F,0x12,0xC2,0xFF,0x52,0x23,0xED,0x06,
0xCC,0x18,0x0E,0x94,0x19,0x81,0xD2,0xC2,0xBC,0x58,0xA5,0x9A,0xA7,0xD0,0xC9,
0x0C,0xA3,0x88,0xD2,0xDD,0x3E,0x54,0x78,0xDF,0x3C};
/*PA=(0x750FF5338ED8F6CD8D8E5BF86D07B2FFFDEE0AECDF76B2E7E0E367825CFD0F5
F,0x12C2FF5223ED06CC180E941981D2C2BC58A59AA7D0C90CA388D2DD3E5478DF3C)*/
uint8_t dB[32]={0x4F,0x8E,0x22,0xFE,0x49,0xE5,0x71,0xAD,0x94,0x37,0xBC,
0x8C,0x95,0x20,0x71,0x0E,0xB4,0x49,0x7C,0xD0,0xFA,0x37,0x2D,0x05,0xA4,0xDF,
0x0D,0x33,0x96,0xDB,0x34,0xC4};
/*dB=0x4F8E22FE49E571AD9437BC8C9520710EB4497CD0FA372D05A4DF0D3396DB34C4*/
uint8_t PB[65]={0x04,0xBA,0x15,0x5C,0x6C,0x6F,0x65,0x73,0x27,0xA2,0x47,0xA7,
0x18,0xEE,0xE5,0x70,0x58,0x19,0xB6,0x61,0xD4,0x67,0x64,0xE9,0x8E,0xBD,0x48,
0xE1,0x08,0xD1,0x4A,0x07,0xBB,0x1F,0xC3,0x8D,0xD3,0x2A,0x7C,0x64,0xA7,
0x17,0x30,0xF8,0x42,0x91,0x11,0x44,0x96,0x6F,0xDA,0xC8,0xBA,0x6F,0xDB,
0x47,0x3E,0x64,0x19,0x4F,0x73,0x55,0x47,0x87,0xE1};
/*PB=(0xBA155C6C6F657327A247A718EEE5705819B661D46764E98EBD48E108D14A07BB,
0x1FC38DD32A7C64A71730F842911144966FDAC8BA6FDB473E64194F73554787E1)*/
uint8_t rA[32]={0x1C,0x34,0xFB,0x8A,0xDB,0x04,0x38,0xF2,0x75,0xAD,0x59,0x22,
0xFF,0x39,0xD4,0xFB,0xC4,0xAD,0x6A,0x1C,0xC0,0x66,0xDB,0x04,0x9F,0x07,
0x58,0xDA,0x38,0xDA,0x32,0x22};
/*rA=0x1C34FB8ADB0438F275AD5922FF39D4FBC4AD6A1CC066DB049F0758DA38DA3222*/
uint8_t RA[65]={0x04,0x7D,0xEE,0xEC,0x05,0xE7,0xB4,0x9B,0x0E,0xCF,0x1D,0x6A,
0xFA,0xE5,0x07,0x7B,0xF0,0xF3,0x3A,0x21,0x12,0xCB,0x8D,0x66,0x4C,0x88,0x90,0x26,0x51,
0x20,0x74,0x95,0x83,0xEE,0x5A,0x68,0x91,0x82,0x1E,0xC8,0x29,0x2B,0x8D,0x41,
0xE1,0x7D,0x49,0x61,0x3E,0xEA,0xF4,0x1C,0x36,0xD0,0xCE,0x42,0x57,0xD1,
0xD6,0x76,0xC5,0x36,0x97,0x83,0xF9};

```

```

/*RA=(0x7DEEEEC05E7B49B0ECF1D6AFAE5077BF0F33A2112CB8D664C8890265120749583,
0xEE5A6891821EC8292B8D41E17D49613EEAF41C36D0CE4257D1D676C5369783F9)*/
uint8_trB[32]={0xD2,0x17,0xC2,0x13,0x1A,0xE7,0xDD,0xC0,0xBC,0x9E,0x9E,0x7C,
0x9C,0x33,0xBA,0xDB,0x5E,0x45,0xA6,0xD5,0x61,0x70,0x79,0x46,0x37,0xED,0xA8,
0x40,0xF2,0x37,0xDF,0x74};
/*rB=0xD217C2131AE7DDC0BC9E9E7C9C33BADB5E45A6D56170794637EDA840F237DF74*/
uint8_tRB[65]={0x04,0xB1,0xF0,0xFC,0x13,0x30,0x30,0x3A,0x95,0x32,0x7D,0x49,
0x61,0xAB,0x56,0x22,0xD8,0x56,0x6C,0x02,0xE7,0x0F,0x2B,0x13,0x4B,0x0D,0xA1,0xFC,
0x37,0x91,0x00,0x9A,0x18,0xC2,0xEC,0x89,0x7E,0x4F,0x59,0xAC,0x38,0xDA,0xA3,0xEE,
0x0A,0x68,0x69,0x1A,0x60,0x27,0xC4,0xD2,0x65,0xCA,0x30,0x14,0x5E,0xE8,0x94,0xF4,
0xDA,0x74,0x2A,0xAE,0xA4};
SM2_getPubKey(priKey, pubKey1);
SM2_getKey(priKey, pubKey);
ret=SM2_PointIsOnCrv(pubKey);
if(ret != SM2_isCurve_Ok)
{
    printf("\r\nSM2_PointIsOnCrv is failed");
}
else
{
    printf("\r\nSM2_PointIsOnCrv is Sucessed");
}
SM2_getZ(ID, 16, pubKey, Z);
SM2_GetE(msg, 25, Z, E);
ret = SM2_sign(E, priKey, r, s);
if(ret != SM2_Sign_Ok)
{
    printf("\r\nSM2_sign is failed");
}
else
{
    printf("\r\nSM2_sign is Sucessed");
}
ret = SM2_verify(E, pubKey, r, s);
if(ret != SM2_Verif_Ok)
{
    printf("\r\nSM2_verify is failed");
}
else
{
    printf("\r\nSM2_verify is Sucessed");
}
ret = SM2_encrypt(msg, 40, pubKey, C, &CByteLen);
if(ret !=SM2_En_Ok)

```

```

{
    printf("\r\nSM2_encrypt is Failed!");
}
else
{
    printf("\r\nSM2_encrypt is Sucessed!");
}
ret = SM2_decrypt(C, CByteLen, priKey, M, &MByteLen);
if( ret ==SM2_En_Ok &&
    (Cmp_U8(msg,40,M,40) ==Cmp_EQUAL))
{
    printf("\r\nSM2_decrypt is Sucessed!");
}
else
{
    printf("\r\nSM2_decrypt is Failed!");
}
SM2_ExchangeKey(role1, IDA, IDAByteLen, IDB, IDBByteLen, dA, PA, PB, rA, RA, RB, kByteLen,
KA1, S11, SA1);
ret=SM2_ExchangeKey(role0, IDB, IDBByteLen, IDA, IDAByteLen, dB, PB, PA, rB, RB, RA,
kByteLen, KA2, S12, SA2);
if(ret != SM2_SUCCESS)
{
    printf("\r\nSM2 key exchange fail\n");
    //return 0x5a5a5a5a;
}
else
{
    flag1=0;
    if ((memcmp(KA1,KA2,33)))
    {
        flag1=1;
    }
    if ((memcmp(S11,SA2,32)))
    {
        flag1=1;
    }
    if ((memcmp(S12,SA1,32)))
    {
        flag1=1;
    }
}
if (!flag1)
{

```

```
printf("\r\nSM2 key exchange success\n");  
}  
else  
{  
    printf("\r\nSM2 key exchange fail\n");  
    //return 0x5a5a5a5a;  
}  
}
```

10 Version History

Version	Date	Changes
V1.0	2023.03.14	Initial release

11 Disclaimer

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 Nations Technologies Inc. and Nations 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 NATIONS has attempted to provide accurate and reliable information, NATIONS 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 NATIONS 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.

NATIONS 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 NATIONS and hold NATIONS 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 NATIONS, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.