

# N32G45x&N32G4FR&N32WB452 系列算法库使用指南

**V1.0.0**

## 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用者在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用者承担，同时使用者应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。

## 注 意

这是国民技术不便于披露的文件，它包含一些保密的信息。在没有签订任何保密协议前或者在国民技术单方面要求的情况下请归还于国民技术。任何非国民技术委托人不得使用或者参考该文件。

如果你得到了这份文件，请注意：

- 不得公开文档内容
- 不得转载全部或部分文档内容
- 不得修改全部或部分文档内容

在以下情况这份文件必须销毁

- 国民技术已经提供更新的版本
- 未签订保密协议或者保密协议已经过期
- 受委托人离职

## 给我们的客户

我们一直在不断的改进我们的产品及说明文档的品质。我们努力保证这份文档的说明是准确的，但也可能存在一些我们未曾发现的失误。如果您发现了文档中有任何疑问或错失的地方请及时联系我们。您的理解及支持将使得这份文档更加完善。

## 版本历史

版本	日期	备注
V1.0	2020.04.24	新建文档

## 术语及缩略语

缩写	全拼
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

# 目 录

目 录.....	6 -
1. 概述.....	8 -
1.1. 支持的算法.....	8 -
1.2. 基本数据类型.....	8 -
2. DES/TDES 算法 API 说明.....	9 -
2.1. 算法库使用方法.....	9 -
2.2. 数据类型定义.....	9 -
2.3. 函数接口说明.....	10 -
2.3.1. DES/TDES 算法初始化.....	10 -
2.3.2. DES/TDES 算法加解密.....	11 -
2.3.3. DES/TDES 关闭.....	11 -
2.3.4. 获取 DES/TDES 库版本信息.....	12 -
3. AES 算法 API 说明.....	13 -
3.1. 算法库使用方法.....	13 -
3.2. 数据类型定义.....	13 -
3.3. 函数接口说明.....	14 -
3.3.1. AES 算法初始化.....	15 -
3.3.2. AES 算法加解密.....	15 -
3.3.3. 关闭 AES.....	15 -
3.3.4. 获取 AES 库版本信息.....	16 -
4. HASH 算法 API 说明.....	17 -
4.1. 算法库使用方法.....	17 -
4.2. 数据类型定义.....	17 -
4.3. 函数接口说明.....	19 -
4.3.1. HASH 初始化.....	19 -

4.3.2.	<i>HASH 启动运算</i> .....	- 20 -
4.3.3.	<i>HASH 分步处理数据</i> .....	- 20 -
4.3.4.	<i>HASH 完成并取结果</i> .....	- 21 -
4.3.5.	<i>HASH 运算关闭</i> .....	- 21 -
4.3.6.	<i>获取 HASH 库版本信息</i> .....	- 22 -
<b>5</b>	<b>RNG 算法 API 说明</b> .....	<b>- 23 -</b>
5.1	算法库使用方法.....	- 23 -
5.2	数据类型定义.....	- 23 -
5.3	函数接口说明.....	- 23 -
5.3.1	<i>伪随机生成函数</i> .....	- 23 -
5.3.2	<i>随机数生成函数</i> .....	- 24 -
5.3.3	<i>获取 RNG 库版本信息</i> .....	- 24 -
i.	附录一 DES 算法库函数调用例程.....	- 25 -
ii.	附录二 TDES 算法库函数调用例程.....	- 31 -
iii.	附录三 AES 算法库函数调用例程.....	- 41 -
iv.	附录四 HASH 算法库函数调用例程.....	- 64 -
v.	附录五 RNG 算法库调用例程.....	- 71 -

# 1. 概述

本文档适用于 N32G45x 系列、N32G4FR 系列、N32WB452 系列芯片，主要说明该类芯片中算法接口和使用方法。

对于 U32 数据类型参数，若采用 U8 强制转换 U32 形式，则需要确保 U8 地址按字对齐。

## 1.1. 支持的算法

提供的算法如下：

- DES: 加密/解密
- TDES: 加密/解密
- AES: 加密/解密 (AES-128/192/256)
- HASH: 获取摘要，支持 (SHA-1/SHA-224/SHA-256/MD5/SM3)
- RNG: 随机数生成

## 1.2. 基本数据类型

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



## 2. DES/TDES算法API说明

### 2.1. 算法库使用方法

算法库使用方法如下：

1. 将 n32g45x\_des.h 、 Type.h 、 n32g45x\_algo\_common.h 文件夹中；将 n32g45x\_algo\_common.lib、n32g45x\_des.lib 添加到工程中；
2. 按 2.3 节函数说明调用函数，例程见附录一、附录二提供的 demo。

### 2.2. 数据类型定义

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

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

*typedef struct*

```

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

## 2.3. 函数接口说明

DES 算法库包含的函数列表如下：

表 2-1 DES/TDES 算法库函数表

函数	描述
u32 DES_Init(DES_PARM *parm);	DES/TDES 初始化函数
u32 DES_Crypto(DES_PARM *parm)	DES/TDES 加解密
void DES_Close(void)	DES/TDES 关闭
void DES_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	DES 版本获取函数

### 2.3.1. DES/TDES算法初始化

#### **DES\_Init**

DES/TDES 算法初始化

函数原型

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

参数说明

parm 输入，指向 DES\_PARM 结构体的指针

返回值

DES\_Init\_OK：初始化成功 其他：初始化错误

## 注意事项

1. 若是 ECB 模式，则参数 iv 可直接用 NULL 替换。

### 2.3.2. DES/TDES 算法加解密

<b>DES_Crypto</b>	DES/TDES 算法初始化，加解密
函数原型	u32 DES_Crypto(DES_PARM *parm)
参数说明	parm 输入，指向 DES_PARM 结构体的指针
返回值	DES_Crypto_OK: 运算正确 其他: 运算错误
注意事项	<p>在调用本函数前，若还未初始化或已切换到其他算法，先调用 DES_Init 函数；</p> <ol style="list-style-type: none"><li>1. 若是 ECB 模式，则参数 iv1 可直接用 NULL 替换。</li><li>2. 大量数据作为一整体但分多块进行 CBC 加密时，需注意： 第 X 块数据 (X&gt;1) 调用本函数进行加密，使用的初始向量 IV (IV = iv1) 一定要更新为第 X-1 块数据调用本函数进行加密得到的密文的最后一个分组 (8 字节)。</li><li>3. 大量数据作为一整体但分多块进行 CBC 解密时，需注意： 第 X 块数据 (X&gt;1) 调用本函数进行解密，使用的初始向量 IV (IV = iv1) 一定要更新为第 X-1 块数据的最后一个分组 (8 字节)。</li><li>4. 调用方式请参考附录一和附录二。</li></ol>

### 2.3.3. DES/TDES 关闭

<b>DES_Close</b>	关闭 DES/TDES 算法时钟和系统时钟
函数原型	void DES_Close(void)
参数说明	
返回值	

## 2.3.4. 获取DES/TDES库版本信息

### DES\_Version

获取 DES/TDES 库版本信息

---

函数原型

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

参数说明

type        商业或快速版本  
customer    标准或定制版本  
date        年, 月, 日  
version     //版本 x.x

返回值

注意事项

```
*type = 0x05; // 商业和快速版  
*customer = 0x00; // 标准版本  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; //表示版本 1.0
```

## 3. AES算法API说明

### 3.1. 算法库使用方法

算法库使用方法如下：

1. 将 n32g45x\_aes.h、Type.h、n32g45x\_algo\_common.h 中；将 n32g45x\_algo\_common.lib、n32g45x\_aes.lib 程中；
2. 按 3.3 节函数说明调用函数，例程见附录三提供的 demo

### 3.2. 数据类型定义

```
#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
    //if Working mode is CTR,the length of input message cannot be
    //be 4 times and cannot be zero;
    //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. 函数接口说明

AES 算法库包含的函数列表如下：

表 3-1 AES 算法库函数表

函数	描述
u32 AES_Init(AES_PARM *parm)	AES 初始化
u32 AES_Crypto(AES_PARM *parm)	AES 加解密函数
void AES_Close(void)	AES 关闭函数
void AES_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	AES 版本获取函数

### 3.3.1. AES算法初始化

#### **AES\_Init**

#### AES 算法初始化

---

函数原型	u32 AES_Init(AES_PARM *parm)
参数说明	parm 输入，指向 AES_PARM 结构体的指针
返回值	AES_Init_OK: 运算正确 其他: 运算错误
注意事项	1.调用方式请参考附录三。

### 3.3.2. AES算法加解密

#### **AES\_Crypto**

#### AES 算法加解密

---

函数原型	u32 AES_Crypto(AES_PARM *parm)
参数说明	parm 输入，指向 AES_PARM 结构体的指针
返回值	AES_Crypto_OK: 运算正确 其他: 运算错误
注意事项	在调用本函数前，若还未初始化或已切换到其他算法，先调用 AES_Init 函数； 1.调用方式请参考附录三。

### 3.3.3. 关闭AES

#### **AES\_Close**

#### 关闭 AES 算法时钟和系统时钟

---

函数原型	void AES_Close(void)
参数说明	
返回值	

### 3.3.4 获取AES库版本信息

#### **AES\_Version**

获取 AES 库版本信息

---

函数原型

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

参数说明

type        商业或快速版本  
customer    标准或定制版本  
date        年, 月, 日  
version     //版本 x.x

返回值

注意事项

```
*type = 0x05; // 商业和快速版  
*customer = 0x00; // 标准版本  
date[0] = 18; //Year()  
date[1] = 12; //Month()  
date[2] = 28; //Day ()  
*version = 0x10; //表示版本 1.0
```



## 4. HASH算法API说明

包括 SHA1/SHA224/SHA256/MD5/SM3 算法库。

### 4.1. 算法库使用方法

数据输入及输出均采用字节大端顺序。算法库使用方法如下：

1. 将 Type.h 、 n32g45x\_hash.h 、 n32g45x\_algo\_common.h 加入头文件夹中，将 n32g45x\_algo\_common.lib、n32g45x\_hash.lib 添加到工程中；
2. 按 4.3 节函数说明调用函数，例程见附录四提供的 demo

### 4.2. 数据类型定义

*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. 函数接口说明

HASH 算法库包含的函数列表如下：

表 4-1 HASH 算法库函数表

函数	描述
u32 HASH_Init(HASH_CTX *ctx)	HASH 初始化函数
u32 HASH_Start(HASH_CTX *ctx)	HASH 分步杂凑开始运算函数
u32 HASH_Update(HASH_CTX *ctx, u8 *in, u32 byteLen)	HASH 分步杂凑处理函数
u32 HASH_Complete(HASH_CTX *ctx, u8 *out)	HASH 分步杂凑完成函数
u32 HASH_Close(void)	关闭 HASH 函数
void HASH_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	获取 HASH 算法库版本

#### 4.3.1. HASH初始化

##### **HASH\_Init**

HASH 初始化

函数原型

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

参数说明

ctx 输入，指向 HASH\_CTX 结构体的指针

返回值

HASH\_Init\_OK：运算正确 其他值：运算错误

注意事项

1. ctx 必须指向 RAM 区，且指向的内容不可更改(为杂凑计算的中间状态和临时内容存储)，下同

2. 分步计算一段消息的杂凑值时，必须先调用本函数

### 4.3.2.HASH启动运算

#### **HASH\_Start**

#### HASH 启动运算

---

函数原型

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

参数说明

ctx            输入，指向 HASH\_CTX 结构体的指针

返回值

HASH\_Start\_OK: 运算正确      其他值: 运算错误

注意事项

1. 若需要 HASH 运算过程中支持中断，将 ctx->sequence 置为 *HASH\_SEQUENCE\_TRUE*，在中断结束后需要重新调用 HASH\_Init 函数，然后再调用 HASH\_Update 函数；否则，置为 *HASH\_SEQUENCE\_FALSE*。
- 2.调用方式请参考附录四。

### 4.3.3.HASH分步处理数据

#### **HASH\_Update**

#### HASH 分步处理数据

---

函数原型

u32 HASH\_Update(HASH\_CTX \*ctx, u8 \*in, u32 byteLen)

参数说明

ctx            输入，指向 HASH\_CTX 结构体的指针

in             输入，指要杂凑的信息

byteLen       输入，指杂凑信息的字节长度

返回值

HASH\_Update\_OK: 运算正确      其他值: 运算错误

注意事项

- 在调用本函数前，若还未初始化或已切换到其他算法，先调用 HASH\_Init 和 HASH\_Start 函数：
1. 调用此函数前必须先调用初始化函数 HASH\_Init 和 HASH\_Start
  2. ctx 必须指向 RAM 区，且指向的内容不可更改(为杂凑计算的中间状态和临时内容存储)。
  3. in 内容可指向 RAM 或 Flash 区，in 可以是 NULL，计算结果为 NULL 的摘要值。
  4. byteLen 可以是 0 或者 NULL，计算结果为 NULL 的摘要值

5. 初始化后, 对一整块消息可任意分割成多小块, 对每一小块消息可依次调用此函数, 最后调用 HASH\_Complete 函数, 即可得到这一整块消息的杂凑结果。

6. 若需要级联应用, 需要将  $ctx->sequence = HASH\_SEQUENCE\_TRUE$ , 把外部的 IV 拷贝到  $ctx->IV$ , 并且把已 Update 的数据长度 len 用  $ctx->hashAlg->ByteLenPlus(ctx->msgByteLen, len)$  加到  $ctx->msgByteLen$ , 然后调用 HASH\_Update 函数, 才能级联成功。

7. 调用方式请参考附录四。

#### 4.3.4. HASH完成并取结果

##### **HASH\_Complete**

##### HASH 完成并取结果

---

函数原型

u32 HASH\_Complete(HASH\_CTX \*ctx, u8 \*out)

参数说明

ctx 输入, 指向 HASH\_CTX 结构体的指针

out 输出, 指向 HASH 结果的指针

返回值

HASH\_Complete\_OK: 运算正确 其他值: 运算错误

注意事项

在调用本函数前, 若还未初始化或已切换到其他算法, 先调用 HASH\_Init 和 HASH\_Start 函数;

1. 消息输入完毕, 调用此函数才能获得最终结果,

2. ctx 必须指向 RAM 区, 且指向的内容不可更改(为杂凑计算的中间状态和临时内容存储)。

3. 调用方式请参考附录四。

#### 4.3.5. HASH运算关闭

##### **HASH\_Close**

##### HASH 运算关闭

---

函数原型

u32 HASH\_Close(void)

参数说明

返回值

HASH\_Close\_OK: 运算正确

## 注意事项

### 4.3.6. 获取HASH库版本信息

**HASH\_Version**获取 HASH 库版本信息

---

## 函数原型

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

## 参数说明

```
type      商业或快速版本
customer  标准或定制版本
date      年, 月, 日
version   //版本 x.x
```

## 返回值

## 注意事项

```
*type = 0x05; // 商业和快速版
*customer = 0x00; // 标准版本
date[0] = 18; //Year()
date[1] = 12; //Month()
date[2] = 28; //Day ()
*version = 0x10; //表示版本 1.0
```

## 5 RNG算法API说明

### 5.1 算法库使用方法

算法库使用方法如下：

- 1、将 Type.h、n32g45x\_rng.h、n32g45x\_algo\_common\_1B.h 加入头文件夹中，将 n32g45x\_algo\_common.lib 、n32g45x\_rng.lib 添加到工程中；
- 2、按 7.3 节函数说明调用函数。

### 5.2 数据类型定义

*enum*

*RNG\_OK = 0x5a5a5a5a,*

*LENError = 0x311ECF50, //RNG generation of key length error*

*ADDRNULL = 0x7A9DB86C, // This address is empty*

*};*

### 5.3 函数接口说明

RNG 算法库包含的函数列表如下：

表 7-1 RNG 算法库函数表

函数	描述
u32 GetPseudoRand_U32(u32 *rand, u32 wordLen,u32 seed[2])	伪随机数按 word 生成函数
u32 GetTrueRand_U32(u32 *rand, u32 wordLen)	真随机数按字生成函数
void RNG_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)	获取 RNG 库版本信息

#### 5.3.1 伪随机生成函数

**GetPseudoRand\_U32**

伪随机数按 word 生成函数

函数原型	u32 GetPseudoRand_U32(u32 *rand, u32 wordLen, u32 seed[2])
参数说明	rand 指针，指向生成的随机数 wordlen: 拟获取伪随机数word长 seed[2] 输入，伪随机种子变量数组
返回值	RNG_OK 成功；其他 生成伪随机数出错
说明	按word生成伪随机数
注意事项	1. 用户可输入种子数组，如果用户输入seed为NULL，则内部自动生成种子；
例程	

### 5.3.2 随机数生成函数

#### **GetTrueRand\_U32**真随机数生成函数

---

函数原型	u32 GetTrueRand_U32(u32 *rand, u32 wordLen)
参数说明	rand: 指针，指向生成的随机数某内存地址 wordLen: 拟获取真随机数的字长度
返回值	RNG_OK 成功；其他: 生成真随机数出错，详见枚举类型值定义
注意事项	

### 5.3.3 获取RNG库版本信息

#### **RNG\_Version** 获取 RNG 库版本信息

---

函数原型	void RNG_Version(u8 *type, u8 *customer, u8 date[3], u8 *version)
参数说明	type 商业或快速版本 customer 标准或定制版本 date 年，月，日 version //版本 x.x
返回值	
注意事项	*type = 0x05; // 商业和快速版 *customer = 0x00; // 标准版本



```
date[0] = 18; //Year()
date[1] = 12; //Month()
date[2] = 28; //Day ()
*version = 0x10; //表示版本 1.0
```

## i. 附录一 DES 算法库函数调用例程

```
u32 DES_test()
{
    u32 i,flag1,flag2,flag3,flag4;
    u32 ret;
    DES_PARM DES_Parm={0};
    /*若需要修改测试实例，当参数的真实值为“0x0102030405060708”时，由于 u32 数据是字节小
    端序存储，在对以上参数进行初始化赋值时，请输入“0x04030201,0x08070605”.若无特殊说明，本
    例程参数都以这种方式设置*/
    u32 in1 [16]={
        0x5FE2D4C0,0xAEAE3F30,0x692930A8,0x1DA69A51,0xDD34B34B,0xAF8D237A,0x2114F489,
        0xE461FF17,0x47C795FD,0x8FF62B49,0x62E9BD63,0x1AF52817,0xECB9DFD4,0xE04421C9,
        0x87B4B22E,0x9FF98759
    };
    u32 key1 [2]={0x946AB06B,0x2276E632};
    u32 iv1 [2]={0x482A8C66,0xC324FC78};
    u32 out[16];
    u32 DES_ECB_EN[16]={0x2FD8D31F,0xC3E2E705,0x4B6D1C4C,0x31EB4154,0xDA273EEC,
```

```
0x8EED57DA,0x26FDE038,0x15B0D57D,0xBCE7464F,0x78D7997A,  
0x4F9917D7,0xAE9C1DA9,0x749FEAEE,0xDFE6A911,0x34D556D5,  
0xA32FA0A2};
```

```
/*DES_ECB_EN=0x1FD3D82F05E7E2C34C1C6D4B5441EB31EC3E27DADA57ED8E38E0FD26  
7DD5B0154F46E7BC7A99D778D717994FA91D9CAEEEEEA9F7411A9E6DFD556D534A2A02FA3*/
```

```
u32 DES_ECB_DE[16]={0xBD77D94A,0xCF5698BB,0xF113743F,0x0FCFC898,0x7DD21DA8,  
0x3908A674,0x65303E6C,0x56CB0E02,0xF0B14651,0x3BBB36AB,  
0x8C129CC3,0xC42D5DD0,0x74549F20,0x5A7E5029,0xE5334FE2,  
0xD5ED9CA8};
```

```
/*DES_ECB_DE=0x4AD977BDBB9856CF3F7413F198C8CF0FA81DD27D74A608396C3E30650  
20ECB565146B1F0AB36BB3BC39C128CD05D2DC4209F547429507E5AE24F33E5A89CEDD5*/
```

```
u32 DES_CBC_EN[16]={0x236813B0,0x14D3A0CA,0xDB57CA2F,0x073FADB0,0x83577985,  
0x7DEBA1CB,0xD5410854,0x2C0E74D8,0x8B8019BB,0xBAB789EF,  
0xF93DEC2E,0xD1BFE8F4,0xE061C81D,0x2F620219,0x662759FF,  
0x77CABBF6};
```

```
/*DES_CBC_EN=0xB0136823CAA0D3142FCA57DBB0AD3F0785795783CBA1EB7D540841D5  
D8740E2CBB19808BEF89B7BA2EEC3DF9F4E8BFD11DC861E01902622FFF592766F6BBCA77*/
```

```
u32 DES_CBC_DE[16]={0xF55D552C,0x0C7264C3,0xAEF1A0FF,0xA161F7A8,0x14FB2D00,  
0x24AE3C25,0xB8048D27,0xF9462D78,0xD1A5B2D8,0xDFDAC9BC,  
0xCBD5093E,0x4BDB7699,0x16BD2243,0x408B783E,0x098A9036,  
0x35A9BD61};
```

```
/*DES_CBC_DE=0x2C555DF5C364720CFFA0F1AEA8F761A1002DFB14253CAE24278D04B87  
82D46F9D8B2A5D1BCC9DADF3E09D5CB9976DB4B4322BD163E788B4036908A0961BDA935*/
```

```
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.附录二 TDES算法库函数调用例程

```
u32 TDES_2Key_test()
{
    u32 i,flag1,flag2,flag3,flag4;
    u32 ret;
    DES_PARM TDES_Parm={0};
    /*若需要修改测试实例，当参数的真实值为“0x0102030405060708”时，由于 u32 数据是字节小
    端序存储，在对以上参数进行初始化赋值时，请输入“0x04030201,0x08070605”.若无特殊说明，本
    例程参数都以这种方式设置*/
    u32 in1[16]={
        0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,
        0xED8A71D7,0xA66B2FAA,0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,
        0x05CDB663,0xF7D812E4,0x86341DEB,0xBA52B237
    };

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

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

    u32 out[16];

    u32 TDES_ECB_EN[16]={0x42976179,0x3A15FDA5,0x278639E4,0x3F4D2DDD,0x987EAF74,
    0x17376CD5,0x9BE1CAB1,0x5501A0BA,0xD18D511B,0x11054F45,
    0x7EAC1828,0x375B9DAD,0x3823A312,0x8EE802FF,0xF2F00328,
    0x3F81CF19};
    /*TDES_ECB_EN=0x79619742A5FD153AE4398627DD2D4D3F74AF7E98D56C3717B1CAE19B
    BAA001551B518DD1454F05112818AC7EAD9D5B3712A32338FF02E88E2803F0F219CF813F*/
```

```
u32 TDES_ECB_DE[16]={0x58AD407C,0x76B43ED7,0x23B44DDA,0x22EC376C,0x50311263,  
0xECC57D42,0x2FA5ADAA,0xE7A099A0,0x287DBD9B,0x3951FD62,  
0x530A3728,0x9AAFA2D3,0x0C41708F,0x5BFE1BCC,0x3B21EE97,  
0xE29E749A};
```

```
/*TDES_ECB_DE=0x7C40AD58D73EB476DA4DB4236C37EC2263123150427DC5ECAADA5  
2FA099A0E79BBD7D2862FD513928370A53D3A2AF9A8F70410CCC1BFE5B97EE213B9A749EE2*/
```

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

```
/*TDES_CBC_EN=0x85A4233710EB2E3EC434549EFDC8922643578D97D7CFCB106C393A87E  
B6ACFD9FC53895C44372FD6600B2DDE352BA21D6F3D790024D43C5460E63B83523F7005*/
```

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

```
/*TDES_CBC_DE=0x1E7A61ED87CEDABF57FDCA1F85CA3E8D734F15727F984FF87BBCAA  
E87F67B3EF4CCCF7C5C8D23A9F729B774005F2D70024B4A8F1A29E389AF458EC3E7E664615*/
```

```
TDES_Parm.in = in1;
```

```
TDES_Parm.key = key1;
```

```
TDES_Parm.out = out;
```

```
TDES_Parm.inWordLen = 16;
```

```
TDES_Parm.keyMode = TDES_2KEY;
```

```
TDES_Parm.Mode = DES_ECB;
```

```
TDES_Parm.En_De = DES_ENC;
```



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

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

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

```
{

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

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

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

```
    else
    {
        flag3=0;
    }
}

TDES_Parm.iv = iv1;
TDES_Parm.En_De = DES_DEC;
ret = DES_Init(&TDES_Parm);
ret=(DES_Crypto(&TDES_Parm));
DES_Close();
if (ret!= DES_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
{

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

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

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

u32 TDES\_3Key\_test()

```
{  
    u32 i,flag1,flag2,flag3,flag4,ret=0;  
    DES_PARM TDES_Parm={0};  
    u32 in1[16]= {  
  
        0x3C7EB08D,0xAFD2FDE9,0x22245D10,0x148AE53D,0xC70F11D1,0x0813FEDF,0xED8A71D7,  
0xA66B2FAA,  
  
        0x137DAC5A,0x9A7850D6,0xFDE9C4AB,0xC1C6856E,0x05CDB663,0xF7D812E4,0x86341DEB  
,0xBA52B237  
    };  
    u32 key1[6]={0x675BE5D2,0x1641A6AD,0x14531A6B,0xEBFA006E,0x90DFD0CD,0x2D029B93};  
  
    u32 iv1[2]={0xB5CC3A62,0xC96EF050};  
  
    u32 out[16];
```

u32

TDES\_ECB\_EN[16]={0x5D6C633C,0x8EDFC4C7,0x3D02A02C,0x97431789,0x83EF4C36,0xFF591C  
67,0xE869DB08,0xAB82D05B,

0x11771439,0xDC6F79BB,0x5B46D128,0xF52114F5,0x2C758CB4,0x1A4D1A6A,0x0DC3FBCA,0x82  
222BB2};

u32

TDES\_ECB\_DE[16]={0x6780A75A,0x62EC1AC8,0xD0341FF5,0x2260C44E,0xF2720589,0xB0EBBB  
E0,0xBFE0991D,0x1EA78C1C,

0xBAB53D00,0xE3FA25D6,0x9430DEF4,0xC465511C,0xEE9D2DFB,0x9796AADC,0x4FFFEF58,0x1  
72D00A2};

u32

TDES\_CBC\_EN[16]={0x048BD8AD,0xF98F2C51,0x5F6FD563,0xA26A1038,0x8017FC81,0xBBD5A  
F4C,0x0A7AEFF,0xB7D428A1,

0x316E31F7,0xD8F283E1,0xDDD4395F,0x8076C2D0,0x0434D1E9,0xD1A94D4D,0xFF3E3B5E,0x77  
C93116};

u32

TDES\_CBC\_DE[16]={0xD24C9D38,0xAB82EA98,0xEC4AAF78,0x8DB239A7,0xD0565899,0xA4615  
EDD,0x78EF88CC,0x16B472C3,

0x573F4CD7,0x45910A7C,0x874D72AE,0x5E1D01CA,0x1374E950,0x56502FB2,0x4A32593B,0xE0F  
51246};

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.附录三 AES算法库函数调用例程

```
u32 AES_128_test()
{
    u32 flag1,flag2,flag3,flag4,flag5,flag6;
    u32 ret;
    AES_PARM AES_Parm={0};
    /*若需要修改测试实例，当参数的真实值为“0x0102030405060708”时，由于 u32 数据是字节小
    端序存储，在对以上参数进行初始化赋值时，请输入“0x04030201,0x08070605”.若无特殊说明，本
    例程参数都以这种方式设置*/
    u32 in[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207D,
    0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,
    0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,
    0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,
    0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,
    0x85D4EFC8,0x9BAAD700};
    /*in=0xA570874A98DAC273D1522DF5464A885FD572CF8D7D200F2ACEF579749EBEB53FFE9
    8793D6D58597C4B29E1309077E1B3BD0C08CA1379B42A03B8093BE70F411BEF3742E65EBE7635
    F6AAD7E4B05F4919B1C32D817A9F33DD5F496FBE3A80B08C60970DCDD4D6C0A4BB1D389810
    CA699799DD673B1A5A334CC8EFD48500D7AA9B*/
    u32 key[4]={0x7FDDA35D,0x7D5C725B,0x1960F327,0x4FD9DDA2};
    /*key=0x5DA3DD7F5B725C7D27F36019A2DDD94F*/
    u32 iv[4]={0x7B00FE39,0xD3E06638,0xD52BC983,0x38E98017};
    /*iv=0x39FE007B3866E0D383C92BD51780E938*/
    u32 out[32];
```

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

```
/*AES_ECB_EN=0x38544EB203A345017FA250C470EEDE2A4E316F90AD2942B20E361213228  
B9C9402BCC1E29E236019E5D5D2CAE27DC58DE19C42137608FCE8DB8145CA50900108F842294  
B623C07D648B63F1127CC671989990B25E08011860C0E451AAF27D7818E6079B66916D353991E07  
1DDBB6CE4205420944681633D098B704279C7E346E*/
```

```
u32 AES_ECB_DE[32]={0x818D1AFD,0xEC4B4F8E,0x69D9F9FF,0x5567B549,0x42DD5C4B,  
0x3BCA1DD3,0xF318E616,0x89297FEC,0x2A3E0A06,0xFDA90D61,  
0x93DCAE5D,0xCF1AFEAE,0x3CF5A889,0x4CFEFE3,0xB2C42607,  
0x37D43F8A,0x9C1CD1D8,0x2FE878E8,0x22D941C3,0x239B9D2D,  
0xD9FEB719,0xA4F9E01C,0xC9C39FE8,0x336B01FA,0xFD12E415,  
0x2B6A0006,0x4A35AFBC,0xA7942FAB,0x09DF0A3A,0x9545521B,  
0x7E009336,0x030A5DA5};
```

```
/*AES_ECB_DE=0xFD1A8D818E4F4BECFFF9D96949B567554B5CDD42D31DCA3B16E618F3  
EC7F2989060A3E2A610DA9FD5DAEDC93AEFE1ACF89A8F53CE3EFFF4C0726C4B28A3FD437D  
8D11C9CE878E82FC341D9222D9D9B2319B7FED91CE0F9A4E89FC3C9FA016B3315E412FD06006  
A2BBCAF354AAB2F94A73A0ADF091B5245953693007EA55D0A03*/
```

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

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

```
/*AES_CBC_EN=0x06E0838A10B63AACBC4D20CA92AF2213C3E966154DE2F992385407EF  
F6192742D8059E13C7E80BC67AF161C294657E7253257730D6088EEBBE04F328CA4267470E4A98  
E0FBEB14D2E9CC49D5BA041AD43917C9EF25BF5155D19E7F4461E9E2D09988EB7D031F8F80A  
89F1129CFAB0CE6FCE649A5CD5661F78EE3FF1DAF6249*/
```

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

u32 AES\_CTR\_EN[32]={0xF14C3DA0,0xA74E1089,0x81480939,0x5C8D4E8D,0x655E20AB,  
0x6D797028,0x1E355F48,0x58184929,0x52B1495A,0xC15EB91D,0xFBD499AB,  
0xF59B39FE,0x96DAE1C3,0x6ECC9CDA,0xDA1FB535,0xAA1C74B2,0xA3F19C5E,  
0x9944E1A6,0xDAA05E9A,0xB96278E3,0x1E4915FC,0xB77FBBD2,0x92BA80B9,  
0xCA97857E,0x509D0365,0x78A6FD99,0xB56F5B3C,0xFBEBFF5B2,0xF9E928C6,  
0xBC28AE3A,0xD8B82D7A,0xA99BF98D};

u32

AES\_CTR\_DE[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207  
D,  
0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,  
0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,  
0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,  
0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,  
0x85D4EFC8,0x9BAAD700};

```
/*AES_CBC_DE=0xC4E48DFAB629AB3F7C30F2BC5E358E6DEE2C5A084BC70848C7B43506  
AA35A1D6D378F1A71C2DA6D7935BA5E73040AFF077308C018EB7A6304C0F25821A48358465D  
D1456FB015C05C0F9D019CA92DA38F680BC3F425E8F911E35142D4A5E222AA4273F7C7C9F59F  
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;
}

}

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

    u32
in[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9E1,0xE1A330
77,0xE3AEA215,

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

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

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

    u32
key[6]={0xA1148977,0xCFA42A1F,0x9D983F36,0x521C1313,0xDAD2CB6F,0xC6254819};

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

u32 out[32];

u32

AES\_ECB\_EN[32]={0x9FCB396D,0xF9A6B55C,0x4CCE7669,0x917CAF2F,0x71F8907D,0xC689393  
6,0x5ABA1DFB,0xA933FF81,

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

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

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

u32

AES\_ECB\_DE[32]={0x41F29D18,0x13C52105,0xB24DBDDD,0x46B6BAB9,0x95F63F1A,0x28B24F  
73,0xAA774293,0xA086E548,

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

0x0F7E89E1,0xBA900B96,0x516EC69B,0xBED1D082,0x3590FD32,0x878C5EE5,0x91B71430,0x  
6A005A7F,

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

u32

AES\_CBC\_EN[32]={0xE5682F2E,0x07A087E9,0x37D60ED6,0x41262C81,0xD69A23B5,0x1800A3F  
D,0xAC50301D,0xB12F3C5E,

0x568A1F62,0xC1057524,0x7E7D09BC,0x26F42541,0x5C2FB09B,0x12C68EFC,0xE03B2AF8,0x  
6E2C9934,

0xD805445F,0x3876A6E4,0xCA85688F,0xD1116501,0x2DE18902,0xCBFD9E9B2,0x57911796,0x0  
719A673,

0x3915B680,0x3B760C23,0x23F715DE,0x6D3425B9,0x9C339EF5,0x6C91D7B0,0x050E91DA,0x  
286AB477};

u32

AES\_CBC\_DE[32]={0xBD58ED6F,0x571E4AB0,0x6E39AA50,0xEFACFE6F,0xCFB4F836,0x21432C  
5A,0x43CA36B8,0x148505B7,

0x6E05BE79,0x26A1C52F,0x9B668D75,0x07904584,0x03C89C5F,0x26AF7239,0x0B344FFC,0x9  
311957F,

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

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

u32

AES\_CTR\_EN[32]={0xF4EB3E15,0xCEC90E4B,0x1708E770,0x6A1297BB,0x045A69FD,0x7FC870A  
7,0x56BE6A22,0x5A912CEA,

```
0xC22E6811,0x37177967,0x68D08A6A,0xCECA04AE,0x30EA7217,0x16992F79,0xF0DD4DAD,0x47  
10126B,0xCC06BD7F,
```

```
0x03093EE5,0x596D2B9B,0xD9844F7C,0x130D4E24,0xD6C87ABF,0xE1745614,0xEF260225,0x0F90  
C354,0x7557E159,
```

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

u32

```
AES_CTR_DE[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9  
E1,0xE1A33077,0xE3AEA215,
```

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

```
0xAB21B81D,0x7477E171,0xDE6FFE4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,  
0x9C3A69BA,
```

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

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

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

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

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

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

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

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

if(Cmp_U32(AES_ECB_EN, 32, out, 32))
{
    flag1=0x5A5A5A5A;
}
else
{
    flag1=0;
}

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

if(Cmp_U32(AES_ECB_DE, 32, out, 32))
{
    flag2=0x5A5A5A5A;
}
else
{
    flag2=0;
```

```
    }  
  
    //cbc  
  
    AES_Parm.Mode = AES_CBC;  
  
    AES_Parm.En_De = AES_ENC;  
  
    ret =AES_Init(&AES_Parm);  
    ret =AES_Crypto(&AES_Parm);  
    AES_Close();  
  
    if(Cmp_U32(AES_CBC_EN, 32, out, 32))  
    {  
        flag3=0x5A5A5A5A;  
    }  
    else  
    {  
        flag3=0;  
    }  
    AES_Parm.En_De = AES_DEC;  
    ret =AES_Init(&AES_Parm);  
    ret =AES_Crypto(&AES_Parm);  
    AES_Close();  
  
    if(Cmp_U32(AES_CBC_DE, 32, out, 32))  
    {  
        flag4=0x5A5A5A5A;  
    }  
    else  
    {  
        flag4=0;  
    }  
}
```

```
//ctr

AES_Parm.Mode = AES_CTR;

AES_Parm.En_De = AES_ENC;

ret =AES_Init(&AES_Parm);

ret =AES_Crypto(&AES_Parm);

AES_Close();

if(Cmp_U32(AES_CTR_EN, 32, out, 32))
{
    flag5=0x5A5A5A5A;
}
else
{
    flag5=0;
}

AES_Parm.in = AES_CTR_EN;

AES_Parm.En_De = AES_DEC;

ret =AES_Init(&AES_Parm);

ret =AES_Crypto(&AES_Parm);

AES_Close();

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

```
    }

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

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

    u32
in[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B3C0,0x8BB9
7CB6,0x99EA0BE6,

    0x8B338E1D,0xFE104A1C,0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBD3A3,0
xA7C27969,

    0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x4
38105F6,
```



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

u32

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

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

u32 out[32];

u32

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

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

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

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

u32

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

0x2B7DDAA3,0x7370FB9D,0xE7E739C6,0xCA013CA6,0x3509E08F,0x74A21641,0x3D2C9527,0xF8DF90F0,

0xED8209E9,0x9DD57975,0x0A506603,0x7C2EFD3B,0x0937237E,0x2828BAAF,0x245E9D40,0x  
F3BB882A,

0x66E82B24,0xF3E778E7,0x386802D1,0xD74C7057,0xEF8525C8,0x1EB7AA48,0x362EACDD,0  
x8AA0F286};

u32

AES\_CBC\_EN[32]={0x39AD6F3A,0xF8E3E1DD,0x2209A14B,0x241642CC,0x83FA4820,0xD82816  
B3,0xEF66B17A,0xB5B49FCC,

0xA7540FD7,0xCC11801C,0xC6126D93,0x8E6C259A,0x626135EB,0x3FEA411B,0x45FF91A3,0  
x1B91B51A,

0x9169DD4C,0x2F42A1E6,0x4299E687,0xEB9FBAA4,0x3B667902,0xDCB4117A,0x45B78A05,0  
x5FECBFA7,

0x54C54A81,0xBDF538B1,0xF2D5804D,0x568910A8,0x41655B32,0xD47D533B,0x5A82D212,0x  
63C07B46};

u32

AES\_CBC\_DE[32]={0xFA3CFD15,0x41B7E384,0x64577770,0x23CB02AC,0x95811940,0x0069DBA  
A,0x7154DC12,0xC335689C,

0x9682708F,0xC7A4485D,0x6C5E4570,0x53EB3740,0xBE3A6E92,0x8AB25C5D,0x733F40C4,0x  
505915DF,

0x8AD021A8,0x00CA8C94,0xE5EDA5A0,0xDBEC8452,0x0D42E557,0xFCC3A85F,0x612E2967,  
0x0A92ED3C,

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

u32

AES\_CTR\_EN[32]={0x85F1DD33,0xAE808F2F,0x26A40960,0xB2020DF8,0xB6C2006E,0xA22A35F  
6,0x33BB584A,0xBFEA7F68,

0x73E54E78,0xF3EB0368,0x80816676,0x6109DE39,0xE0001920,0x8D2B18B8,0x0E46A012,0xE4  
3F1DD1,0x3CA4BC36,

0xD5101452,0x83020170,0x4B752F62,0x3D27A004,0x3C18B5DB,0x99DA9032,0xEA59B340,0x  
79BBD087,0x2EF8CB3D,

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

u32

AES\_CTR\_DE[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B  
3C0,0x8BB97CB6,0x99EA0BE6,

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

0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x4  
38105F6,

0x986F81C9,0x639F51B2,0xA3169082,0x6CD5570C,0x39B678E4,0x84986F66,0x94BB95FA,0x9  
76D9797};

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.附录四 HASH算法库函数调用例程

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

    HASH_Init(ctx);

    HASH_Start(ctx);
    HASH_Update(ctx, (u8*)in, 28);
    HASH_Update(ctx, ((u8*)in)+ 28, 28);
    HASH_Update(ctx, ((u8*)in)+ 56, 6);
    HASH_Complete(ctx, out);
    HASH_Close();
    if(memcmp(out,MD5_fixout,16))
    {

        //printf("MD5-FIX-Test fail\r\n");
        return 0x5a5a5a5a;
    }
}
```



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

```
        0x9C,0xC2,0x29,0x05
    };
//正确的消息摘要
u8 SM3_fixout[32]=
    {
        0xC7,0x8B,0xF5,0x97,0x52,0xCD,0xFE,0x9F,0x70,0x21,
        0x4F,0x5D,0x88,0x92,0x2E,0x60,0x35,0x22,0x3B,0x66,
        0x94,0xFD,0x08,0x96,0x5E,0x26,0x44,0xF9,0x72,0xFE,
        0xE2,0xB2
    };
u8 i,byteLen=48;
HASH_CTX ctx[1];
//设置为 SM3 运算
// ctx->hashAlg 可以选择不同 HASH 运算,
//如 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))
```

```
{
    //分步 SM3 测试失败
    printf("SM3-FIX-Test fail\r\n");
    return HASH_ATTACK;
}
else
{
    //分步 SM3 测试成功
    printf("SM3-FIX-Test success\r\n");
}
return SM3_Hash_OK;
}
```

//此函数例程分别对哈希 sha1/224/256 进行了单步哈希运算

u32 HASH\_test(void)

```
{
    u32 TEST_BUF[200];
    u8 in[48]=
    {
        0x1C,0xBB,0x9F,0x4A,0x43,0x6A,0xAD,0x81,0xFE,0x4F,0x52,0x4A,0x0A,0x76,0x22,0xC8,0
        x4F,0x90,0x18,0x30,0xA4,0xD2,0x8C,0x6A,0xC3,0x40,0xA0,0xBD,0x0A,0x6A,0x37,0x18,0x
        8D,0x19,0x9D,0xE5,0xCB,0x84,0xA3,0xFC,0x39,0xDE,0x8C,0xD6,0xFC,0x2F,0xC8,0x88
    };
    u8 in2[10] = {0x1C,0x61,0xAD,0x6C,0x05,0xF3,0x98,0xA4,0x4C,0xFD};
    u8 out[64];
    u8 sha1_out[20]=
    {
        0x0E,0xEC,0x49,0xC5,0x36,0xBB,0xD7,0x87,0xD2,0xE2,0x0C,0x97,0xC4,0xF8,0x65,0x7C,0x
        CC,0x74,0x8D,0x1E
    }
}
```

```
};  
u8 sha224_out[28]=  
{  
  
    0xC1,0x44,0x4F,0xD0,0xB8,0xA9,0xA3,0xD9,0xE8,0x04,0xA0,0xD1,0x9E,0x38,0xF3,0x5E,  
    0x85,0xB4,0x0F,0x10,0x5A,0x1C,0x48,0xC4,0xF2,0x40,0x10,0x48  
  
};  
u8 sha256_out[32]=  
{  
  
    0xE2,0xE4,0x2C,0x8A,0x01,0x1A,0xE7,0x98,0x67,0x74,0x93,0xAF,0x9D,0x65,0x99,0xB3,0  
    xA1,0x68,0x8B,0x5A,0xF1,0x32,0x3D,0x5B,0xFF,0xFB,0x12,0x30,0x94,0xE4,0x81,0xDD  
  
};  
  
u8 SM3_out[32]=  
{  
  
    0xBD,0x77,0x63,0x33,0x0A,0x71,0x19,0x5C,0x5D,0x26,0xE7,0x99,0x7B,0x41,0x22,0xB0,0  
    xBC,0xB0,0xBE,0x52,0x3E,0xDA,0x0F,0xBE,0xE6,0xA4,0x33,0x96,0xB8,0x83,0x76,0xD4  
  
};  
u32 ret=0x5123;  
  
#if 1  
  
    HASH_CTX *ctx;  
  
    ctx = (HASH_CTX*)(TEST_BUF);  
  
    ctx->hashAlg = HASH_ALG_SHA1;  
  
    ctx->sequence = HASH_SEQUENCE_FALSE;  
  
    HASH_Init(ctx);  
  
    HASH_Start(ctx);
```

```
HASH_Update(ctx, in, 48);
ret=HASH_Complete(ctx, out);
HASH_Close();
if (memcmp(out,sha1_out,20))
{
    return 0x5a5a5a5a;
}
else
{
    printf("SHA1-Test success\r\n");
}
ctx->hashAlg = HASH_ALG_SHA224;
ctx->sequence = HASH_SEQUENCE_FALSE;
HASH_Init(ctx);
HASH_Start(ctx);
HASH_Update(ctx, in, 48);
//HASH_Update(ctx, in2, 10);
ret=HASH_Complete(ctx, out);
HASH_Close();
if (memcmp(out,sha224_out,28))
{
    return 0x5a5a5a5a;
}
else
{
    printf("SHA224-Test success\r\n");
}

ctx->hashAlg = HASH_ALG_SHA256;
```

```
ctx->sequence = HASH_SEQUENCE_FALSE;
HASH_Init(ctx);
HASH_Start(ctx);
HASH_Update(ctx, in, 48);
ret=HASH_Complete(ctx, out);
HASH_Close();
if(memcmp(out,sha256_out,32))
{
    return 0x5a5a5a5a;
}
else
{
    printf("SHA256-Test success\r\n");
}
#endif
return 0;
}
```

## v.附录五 RNG算法库调用例程

```
#define POKER_RAND_BYTE 40 //320bit

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

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

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

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



```
    sum += ((u32)count[i]) * count[i];
}

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