



SHANDONG UNIVERSITY

---

## 密码学引论实验报告

---

谢子洋 202100460116

2023 年 3 月 10 日

# 目录

<b>1 题目 2 调用 DES 进行加密</b>	<b>2</b>
<b>2 题目 3 调用 RSA 进行加密</b>	<b>2</b>
<b>3 题目 4 调用 RSA 进行签名</b>	<b>3</b>
<b>4 题目 5 调用 DHM、SHA512、ARIA 等多个算法</b>	<b>4</b>
4.1 调用 DHM 密钥交换协议 . . . . .	4
4.2 调用 SHA512 哈希函数 . . . . .	5
4.3 调用 ARIA 分组加密 . . . . .	6
4.4 连续调用三种算法 . . . . .	7
<b>5 题目 6 调用密码库实现 AES 加、解密协议</b>	<b>8</b>
<b>参考文献</b>	<b>10</b>

## 1 题目 2 调用 DES 进行加密

DES 加密理论上需要 58bit 的密钥,64bit 的明文,输出 64bit 的密文.使用时应根据不同明文长度选择 DES 的不同模式.本题中需要加密字符串"202100460116",因为一个字符为 1Byte 大小,12 个字符为 96bit,不符合 ECB 模式中明文长度为 64bit 的整数倍的要求,因此此题中选择 CBC 模式进行加密.

所需参数除 DES 模式外还有:明文 const char\* input 和加密密钥 uint8\_t \* key,无需用到向量 iv.

```
1 void DES_encrypt()
2 {
3     const uint8_t input[] = "202100460116";
4     uint8_t key[8] = {0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b };
5     uint8_t iv[8] = {0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30 };
6     DES(MBEDTLS_CIPHER_DES_CBC, MBEDTLS_ENCRYPT, input, key, sizeof(key), iv,
7                                               sizeof(iv));
8 }
9 int DES(mbedtls_cipher_type_t cipher_type, mbedtls_operation_t mode,
10         uint8_t* input, uint8_t* key, int keyLength, uint8_t* iv, int ivLength)
11 {
12     mbedtls_cipher_context_t ctx;
13     const mbedtls_cipher_info_t* info;
14     int ret;
15     size_t len;
16     int olen = 0;
17     //1. 初始化
18     mbedtls_cipher_init(&ctx);
19     info = mbedtls_cipher_info_from_type(cipher_type);
20     ret = mbedtls_cipher_setup(&ctx, info);
21     //2. 设置密钥
22     ret = mbedtls_cipher_setkey(&ctx, key, 8 * 8, MBEDTLS_ENCRYPT);
23     //3. 设置向量iv(如果需要)
24     ret = mbedtls_cipher_set_iv(&ctx, iv, 8);
25     //4. 更新
26     ret = mbedtls_cipher_update(&ctx, input, strlen(input), output_buf, &len);
27     olen += len;
28     //5.完成加密
29     ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
30     olen += len;
31     //输出结果
32     dump_buf(output_buf, olen);
33     exit:
34     mbedtls_cipher_free(&ctx);
35     return ret;
36 }
```

## 2 题目 3 调用 RSA 进行加密

调用库函数实现 RSA 加密主要需要提供如下参数:明文字符串,随机数生成所需种子.计算机中随机数大多是伪随机,合适的随机数种子能有效提高算法安全性.

```

1  uint8_t output_buf[2048/8];
2  int RSA_encrypt()
3  {
4      const uint8_t msg[] = "202100460116";
5      const uint8_t* pers = "rand_seed";
6      RSA_encrypt(msg, pers);
7  }
8  int RSA(const char*msg,const char*pers)
9  {
10     int ret;
11     size_t olen;
12     uint8_t decrypt_buf[20];
13     mbedtls_entropy_context entropy;
14     mbedtls_ctr_drbg_context ctr_drbg;
15     mbedtls_rsa_context ctx;
16
17     mbedtls_entropy_init(&entropy);
18     mbedtls_ctr_drbg_init(&ctr_drbg);
19     mbedtls_rsa_init(&ctx, MBEDTLS_RSA_PKCS_V21, MBEDTLS_MD_SHA256);
20     //1. 初始化
21     ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
22                                (const unsigned char *) pers,strlen(pers));
23     //2. 生成公钥对和私钥对
24     ret = mbedtls_rsa_gen_key(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg, 2048, 65537);
25     //3. 加密
26     ret = mbedtls_rsa_pkcs1_encrypt(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
27                                    MBEDTLS_RSA_PUBLIC,
28                                    strlen(msg), (uint8_t *)msg, output_buf);
29     printf("RSA encrypt result:");
30     dump_buf(output_buf, sizeof(output_buf));
31     exit:
32     mbedtls_ctr_drbg_free(&ctr_drbg);
33     mbedtls_entropy_free(&entropy);
34     mbedtls_rsa_free(&ctx);
35     return ret;
36 }

```

### 3 题目 4 调用 RSA 进行签名

RSA 签名调用过程基本与 RSA 加密一致;

```

1  int RSA_sign_test()
2  {
3      const uint8_t msg[] = "202100460116";
4      const uint8_t* pers = "rsa_test";
5      RSA_sign(msg, pers);
6  }
7  int RSA_sign(const char* msg, const char* pers)
8  {
9      int ret;
10     mbedtls_entropy_context entropy;
11     mbedtls_ctr_drbg_context ctr_drbg;
12     mbedtls_rsa_context ctx;
13     //1. 初始化

```

```

14     mbedtls_entropy_init(&entropy);
15     mbedtls_ctr_drbg_init(&ctr_drbg);
16     mbedtls_rsa_init(&ctx, MBEDTLS_RSA_PKCS_V21, MBEDTLS_MD_SHA256);
17     ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
18                                (const unsigned char *) pers,
19                                strlen(pers));
20     //2. 生成key
21     ret = mbedtls_rsa_gen_key(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg, 2048, 65537);
22     dump_rsa_key(&ctx);
23     //3. sign
24     ret = mbedtls_rsa_pkcs1_sign(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
25                                  MBEDTLS_RSA_PRIVATE, MBEDTLS_MD_SHA256,
26                                  strlen(msg), (uint8_t *)msg, output_buf);
27     dump_buf(output_buf, sizeof(output_buf));
28     //4. 验证
29     ret = mbedtls_rsa_pkcs1_verify(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
30                                    MBEDTLS_RSA_PUBLIC, MBEDTLS_MD_SHA256,
31                                    strlen(msg), (uint8_t *)msg, output_buf);
32     printf("\nsuccess verify !\n");
33     exit:
34     mbedtls_ctr_drbg_free(&ctr_drbg);
35     mbedtls_entropy_free(&entropy);
36     mbedtls_rsa_free(&ctx);
37     return ret;
38 }

```

## 4 题目5调用 DHM、SHA512、ARIA 等多个算法

### 4.1 调用 DHM 密钥交换协议

Diffie-Hellman 密钥交换协议为解决密钥配送问题的通常方式,其可以通过公共通道安全地交换加密密钥. DH 密钥交换协议的通信过程中,只有原根 G、大素数 P、公钥 A、公钥 B 会在网络中进行传输,而私钥 a、b 是不会通过网络进行传输.

本题中调用库函数生成了 256 字节长度的共享密钥,并且主要使用到了以下函数:

mbedtls\_dhm\_init() 进行初始化

mbedtls\_dhm\_make\_public() 用户端和服务端各自生成公私密钥对

mbedtls\_dhm\_read\_public() 用户端和服务端各自读取对方发送公钥

mbedtls\_dhm\_calc\_secret() 用户端和服务端各自计算出共享密钥

mbedtls\_dhm\_free() 清空

函数具体代码如下:

```

1  uint8_t* dhm(const char* pers)
2  {
3      int ret;
4      size_t olen;
5      mbedtls_entropy_context entropy;
6      mbedtls_ctr_drbg_context ctr_drbg;
7      mbedtls_dhm_context dhm_server, dhm_client;

```

```

8 //1. 初始化
9 mbedtls_entropy_init(&entropy);
10 mbedtls_ctr_drbg_init(&ctr_drbg);
11 mbedtls_dhm_init(&dhm_server);
12 mbedtls_dhm_init(&dhm_client);
13 ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
14     (const unsigned char*)pers,
15     strlen(pers));
16
17 // 2.. 生成 2048 bit prime(G, P)
18 mbedtls_mpi_read_string(&dhm_server.P, 16, T_P);
19 mbedtls_mpi_read_string(&dhm_server.G, 10, GENERATOR);
20 dhm_server.len = mbedtls_mpi_size(&dhm_server.P);
21
22 mbedtls_mpi_read_string(&dhm_client.P, 16, T_P);
23 mbedtls_mpi_read_string(&dhm_client.G, 10, GENERATOR);
24 dhm_client.len = mbedtls_mpi_size(&dhm_client.P);
25
26 //3. 生成公钥
27 ret = mbedtls_dhm_make_public(&dhm_server, 256, server_pub, sizeof(server_pub),
28     mbedtls_ctr_drbg_random, &ctr_drbg);
29 ret = mbedtls_dhm_make_public(&dhm_client, 256, client_pub, sizeof(client_pub),
30     mbedtls_ctr_drbg_random, &ctr_drbg);
31
32 //4. 读取公钥
33 ret = mbedtls_dhm_read_public(&dhm_server, client_pub, sizeof(client_pub));
34 ret = mbedtls_dhm_read_public(&dhm_client, server_pub, sizeof(server_pub));
35
36 //5. 计算共享密钥
37 ret = mbedtls_dhm_calc_secret(&dhm_server, server_secret, sizeof(server_secret),
38     &olen, mbedtls_ctr_drbg_random, &ctr_drbg);
39 ret = mbedtls_dhm_calc_secret(&dhm_client, client_secret, sizeof(client_secret),
40     &olen, mbedtls_ctr_drbg_random, &ctr_drbg);
41 //dump_buf(client_secret, sizeof(client_secret));
42 exit:
43 mbedtls_ctr_drbg_free(&ctr_drbg);
44 mbedtls_entropy_free(&entropy);
45 mbedtls_dhm_free(&dhm_server);
46 mbedtls_dhm_free(&dhm_client);
47 return client_secret;
48 }

```

## 4.2 调用 SHA512 哈希函数

SHA-5126 可以用来散列处理长度为  $L$  位的消息, 其中  $0 < L \leq 2^{128}$ 。首先将消息填充至 1024bit 的整数倍长度, 之后再将消息分为 1024 位的消息分组, 对每个分组进行一系列操作最终计算出哈希值。

调用库函数实现代码如下:

```

1 unsigned char* sha_x(mbedtls_md_type_t md_type, uint8_t*message)
2 {
3     int len, i;
4     int ret;
5     mbedtls_md_context_t ctx;

```

```

6     const mbedtls_md_info_t* info;
7     printf("SHA512 source is:\r\n");
8     dump_buf(message,256);
9
10    mbedtls_md_init(&ctx);
11    info = mbedtls_md_info_from_type(md_type);
12    ret = mbedtls_md_setup(&ctx, info, 0);
13    ret = mbedtls_md_starts(&ctx);
14    ret = mbedtls_md_update(&ctx, (unsigned char*)message, strlen(message));
15    ret = mbedtls_md_finish(&ctx, digest);
16    len = mbedtls_md_get_size(info);
17 exit:
18    mbedtls_md_free(&ctx);
19    return digest;
20 }

```

### 4.3 调用 ARIA 分组加密

ARIA 属于分组加密算法, 故同样有多种模式. `ecb` 模式只能加密 16 字节长度的明文; `cbc` 模式可加密长度为 16 字节整数倍的明文; `cfb128` 模式可加密不定长度字节的明文, 并且使用相同密钥进行加解密;

因为要待加密明文 "202100460116" 长度不是 16 或 16 的整数倍, 所以此处本文可选择 `cfb128` 模式进行加密. 主要用到以下函数:

`mbedtls_aria_init()` 函数初始化指定的 ARIA 上下文, 它必须使用前调用的第一个 API.

`mbedtls_aria_setkey_enc()` 函数设置加密密钥. 密钥必须为 128,192,256 bits 长.

`mbedtls_aria_crypt_cfb128()` 函数进行加解密. 其中 `iv_off` 参数为 `iv` 中的偏移量, 其值不大于 15. 初始化向量 `iv` 为大小为 16 字节的可读缓冲区。

调用库函数实现中要注意每次加解密后可变变量 `iv` 和 `iv_off` 都会被修改.

`mbedtls_aria_free()` 函数释放并清除指定的 ARIA 上下文.

具体函数实现代码如下:

```

1 void ARIA_EncryAndDecey(uint8_t* input, int inputLength, const uint8_t* key)
2 {
3     uint8_t iv[16] = "1234567887654321";
4     uint8_t iv02[16] = "1234567887654321";
5     ARIA_encry(MBEDTLS_CIPHER_ARIA_128_CBC, input, key, inputLength, iv, iv_off_value);
6     ARIA_decry(MBEDTLS_CIPHER_ARIA_128_CBC, cipher, key, inputLength, iv02, ...
7         iv_off_value);
8 }
9 int ARIA_encry(mbedtls_cipher_type_t cipher_type, uint8_t* input, uint8_t* key, int ...
10     inputLength, uint8_t* iv, int iv_off)
11 {
12     mbedtls_aria_context ctx; //上下文: 密钥状态,
13     mbedtls_aria_init(&ctx);
14     mbedtls_aria_setkey_enc(&ctx, key, keybits);
15     mbedtls_aria_crypt_cfb128(&ctx, MBEDTLS_ARIA_ENCRYPT, inputLength, &iv_off, iv, ...
16         input, cipher);
17     mbedtls_aria_free(&ctx);
18 }

```

```

15     printf("ARIA cipher:\n");
16     dump_buf(cipher, inputLength);
17     return 1;
18 }
19
20 int ARIA_decry(mbedtls_cipher_type_t cipher_type, uint8_t* input, uint8_t* key, int ...
    inputLength, uint8_t* iv, int iv_off)
21 {
22     mbedtls_aria_context ctx; //上下文:密钥状态,
23     mbedtls_aria_init(&ctx);
24     mbedtls_aria_setkey_dec(&ctx, key, keybits);
25     mbedtls_aria_crypt_cfb128(&ctx, MBEDTLS_ARIA_DECRYPT, inputLength, &iv_off, iv, ...
        input, plain);
26     mbedtls_aria_free(&ctx);
27     printf("ARIA plaintext:\n");
28     dump_buf(plain, inputLength);
29     return 1;
30 }

```

#### 4.4 连续调用三种算法

首先调用 Diffie-Hellman 密钥交换协议, 计算出 256 字节的共享密钥, 作为秘密信息. 该信息经过 SHA-512 哈希算法计算出一 512bit 的哈希值, 令该哈希值作为 ARIA 算法的密钥. 但因为 ARIA 分组加密算法只能接受长度为 128、192、256bit 长度的密钥, 因此本文先对 SHA512 哈希值进行截断, 取 512bit(64Byte) 的前 128bit(16Byte) 数据作为密钥, 对明文"202100460116"进行 ARIA 加密.

实验代码如下:

```

1 void t5() {
2
3     const uint8_t input[] = "202100460116";
4     const uint8_t* pers = "rsa_test";
5     //5.1
6     uint8_t* client_secret = dhm(pers);
7     printf("DH协商密钥");
8     dump_buf(client_secret, 256);
9     //5.2
10    uint8_t* digest = sha_x(MBEDTLS_MD_SHA512, client_secret);
11    printf("Hash值");
12    dump_buf(digest, 64);
13    //5.3
14    uint8_t key[16];
15    for (int i = 0; i < 16; i++){
16        key[i] = digest[i];
17    }
18    printf("ARIA密钥");
19    dump_buf(key, 16);
20    ARIA_EncryAndDecey(input, sizeof(input) - 1, key);
21 }

```

实验结果如图:



```

DH协商密钥
CC D2 F8 98 17 53 08 53 D8 03 46 A2 B7 61 D7 6B
16 3B C0 31 F5 55 05 2D 59 8B 7D 97 33 FA 1D BD
63 89 CA FB 58 BF 10 5E 29 20 46 F9 3F 7B 3F 53
2F F1 47 76 76 14 38 AF 41 B0 55 A1 69 81 CA 4A
F6 DC 75 DD D3 86 03 A5 C2 4A 20 87 57 B5 3E A2
5F 55 65 20 D1 61 41 D3 80 71 AD 12 02 6B C7 AC
74 04 E8 88 C4 79 3E A2 0F 8D 28 60 A8 61 94 0B
F6 57 78 8C 7A 3F 70 14 CF 75 09 DD 9B 46 D0 53
AE B9 15 8D 81 D9 CE D2 E2 73 CB 04 FF 4D 0C E6
E2 01 A1 14 55 AC AB 37 28 B4 D7 D9 B7 D0 9E 49
5B 03 98 11 2F EA F6 CA 0A D1 8E E9 61 E2 49 B4
6B 58 47 B2 72 B7 D2 6D 66 4D 38 A7 A0 50 62 68
56 80 27 94 43 7A A2 B0 55 F2 4A 9E A8 39 FA 46
C5 85 3C C3 C7 3B 3C 31 94 AD 37 F6 5C 3F 1E 91
3C 50 47 FF D9 40 A7 58 58 E2 3A 93 ED 18 C8 37
89 22 AD 9F A8 FA 1F 84 F4 19 87 3D 30 43 78 FD
SHA512 source is:
CC D2 F8 98 17 53 08 53 D8 03 46 A2 B7 61 D7 6B
16 3B C0 31 F5 55 05 2D 59 8B 7D 97 33 FA 1D BD
63 89 CA FB 58 BF 10 5E 29 20 46 F9 3F 7B 3F 53
2F F1 47 76 76 14 38 AF 41 B0 55 A1 69 81 CA 4A
F6 DC 75 DD D3 86 03 A5 C2 4A 20 87 57 B5 3E A2
5F 55 65 20 D1 61 41 D3 80 71 AD 12 02 6B C7 AC
74 04 E8 88 C4 79 3E A2 0F 8D 28 60 A8 61 94 0B
F6 57 78 8C 7A 3F 70 14 CF 75 09 DD 9B 46 D0 53
AE B9 15 8D 81 D9 CE D2 E2 73 CB 04 FF 4D 0C E6
E2 01 A1 14 55 AC AB 37 28 B4 D7 D9 B7 D0 9E 49
5B 03 98 11 2F EA F6 CA 0A D1 8E E9 61 E2 49 B4
6B 58 47 B2 72 B7 D2 6D 66 4D 38 A7 A0 50 62 68
56 80 27 94 43 7A A2 B0 55 F2 4A 9E A8 39 FA 46
C5 85 3C C3 C7 3B 3C 31 94 AD 37 F6 5C 3F 1E 91
3C 50 47 FF D9 40 A7 58 58 E2 3A 93 ED 18 C8 37
89 22 AD 9F A8 FA 1F 84 F4 19 87 3D 30 43 78 FD
Hash值
F3 87 F3 32 8F 52 97 8E CE EC B6 98 AB C0 B7 D5
EF 59 AF 56 B4 B4 60 E9 2A 27 24 0B C4 D2 75 EC
1F 2A EB DE 44 63 55 14 0D 02 9D 39 2C 21 8F 0A
F7 F7 BB D0 B9 09 77 7C 6A 5B 0F 34 1A CE 23 7D
ARIA密钥
F3 87 F3 32 8F 52 97 8E CE EC B6 98 AB C0 B7 D5
ARIA cipher:
01 04 07 07 07 08 0C 01 06 04 05 05
ARIA plaintext:
32 30 32 31 30 30 34 36 30 31 31 36

```

图 1: 第 5 题结果

## 5 题目 6 调用密码库实现 AES 加、解密协议

本题中调用库函数实现了 AES 加密及解密. 对 AES() 函数的 mode 参数传入不同值实现不同功能, 传入 MBEDTLS\_ENCRYPT 代表加密, 传入 MBEDTLS\_DECRYPT 代表解密.

AES\_EncryptAndDecrypt() 函数测试了加解密协议的功能.

```

1 void AES_EncryptAndDecrypt()
2 {
3     uint8_t* input = "202100460116";
4     uint8_t key[16] = {
5         0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b,
6         0x51, 0x2e, 0x03, 0xd5, 0x34, 0x12, 0x00, 0x06};
7     uint8_t iv[16] = {
8         0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,
9         0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41};
10    uint8_t iv02[16] = {
11        0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,

```

```

12     0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41 };
13     //加密
14     AES(MBEDTLS_CIPHER_AES_128_CBC, MBEDTLS_ENCRYPT, input, key,
15         sizeof(key), iv, sizeof(iv));
16     uint8_t cipher[17];
17     for (int i = 0; i < 16; i++) { cipher[i] = output_buf[i]; }
18     cipher[16] = 0x00;
19     //解密
20     AES(MBEDTLS_CIPHER_AES_128_CBC, MBEDTLS_DECRYPT, cipher, key,
21         sizeof(key), iv02, sizeof(iv02));
22 }
23 int AES(mbedtls_cipher_type_t cipher_type, mbedtls_operation_t mode,
24     uint8_t*input, uint8_t*key, int keyLength, uint8_t*iv, int ivLength)
25 {
26     int ret;
27     size_t len;
28     int olen = 0;
29     mbedtls_cipher_context_t ctx; //上下文:密钥状态,
30     const mbedtls_cipher_info_t *info; //密码算法类型
31
32     mbedtls_cipher_init(&ctx);
33     info = mbedtls_cipher_info_from_type(cipher_type);
34     ret = mbedtls_cipher_setup(&ctx, info);
35
36     ret = mbedtls_cipher_setkey(&ctx, key, keyLength*8, mode);
37     ret = mbedtls_cipher_set_iv(&ctx, iv, ivLength);
38
39     int a = strlen(input);
40     ret = mbedtls_cipher_update(&ctx, input, strlen(input), output_buf, &len);
41     olen += len;
42     ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
43     olen += len;
44
45     dump_buf(output_buf, olen);
46     exit:
47     mbedtls_cipher_free(&ctx);
48     return ret;
49 }

```

结果如图:

```

cipher name: AES-128-CBC block size is: 16
source_context: 202100460116
before:
    32 30 32 31 30 30 34 36 30 31 31 36
after:
    BF A2 F5 3A 6E 5C 43 71 12 BD 2E 57 DE 57 7A 84

cipher name: AES-128-CBC block size is: 16
source_context: 竣?n\Cq?W轹z?
before:
    BF A2 F5 3A 6E 5C 43 71 12 BD 2E 57 DE 57 7A 84
after:
    32 30 32 31 30 30 34 36 30 31 31 36

```

图 2: 第 6 题结果

## 参考文献

- [1] Dougals R.Stinson. 密码学原理与实践: 第三版. 北京: 电子工业出版社,2009.7.