SHANDONG UNIVERSITY

密码学引论实验报告

谢子洋 202100460116

2023 年 3 月 10 日

目录

1 题目 2 调用 DES 进行加密 **2**

2 题目 3 调用 RSA 进行加密 **2**

3 题目 4 调用 RSA 进行签名 **3**

4 题目 5 调用 DHM、SHA512、ARIA 等多个算法 **4**

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

5 题目 6 调用密码库实现 AES 加、解密协议 **8**

参考文献 **10**

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.

```
void DES_encrypt()
{
    const uint8_t input[] = "202100460116";
    uint8_t key[8] = {0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b };
    uint8_t iv[8] =  {0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30 };
    DES(MBEDTLS_CIPHER_DES_CBC, MBEDTLS_ENCRYPT, input, key, sizeof(key), iv,
                                                                sizeof(iv));
}
int DES(mbedtls_cipher_type_t cipher_type, mbedtls_operation_t mode,
        uint8_t* input, uint8_t* key, int keyLength, uint8_t* iv, int ivLengthe)
{
    mbedtls_cipher_context_t ctx;
    const mbedtls_cipher_info_t* info;
    int ret;
    size_t len;
    int olen = 0;
    //1. 初始化
    mbedtls_cipher_init(&ctx);
    info = mbedtls_cipher_info_from_type(cipher_type);
    ret = mbedtls_cipher_setup(&ctx, info);
    //2. 设置密钥
    ret = mbedtls_cipher_setkey(&ctx, key, 8 * 8, MBEDTLS_ENCRYPT);
    //3. 设置向量iv(如果需要)
    ret = mbedtls_cipher_set_iv(&ctx, iv, 8);
    //4. 更新
    ret = mbedtls_cipher_update(&ctx, input, strlen(input), output_buf, &len);
    olen += len;
    //5.完成加密
    ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
    olen += len;
    //输出结果
    dump_buf(output_buf, olen);
exit:
    mbedtls_cipher_free(&ctx);
    return ret;
}
```

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.

```c
uint8_t output_buf[2048/8];
int RSA_encrypt()
{
    const uint8_t msg[] = "202100460116";
    const uint8_t* pers = "rand_seed";
    RSA_encrypt(msg, pers);
}
int RSA(const char*msg,const char*pers)
{
    int ret;
    size_t olen;
    uint8_t decrypt_buf[20];
    mbedtls_entropy_context entropy;
    mbedtls_ctr_drbg_context ctr_drbg;
    mbedtls_rsa_context ctx;

    mbedtls_entropy_init(&entropy);
    mbedtls_ctr_drbg_init(&ctr_drbg);
    mbedtls_rsa_init(&ctx, MBEDTLS_RSA_PKCS_V21, MBEDTLS_MD_SHA256);
    //1. 初始化
    ret = mbedtls_ctr_drbg_seed( &ctr_drbg, mbedtls_entropy_func, &entropy,
                                (const unsigned char *) pers,strlen(pers));
    //2. 生成公钥对和私钥对
    ret = mbedtls_rsa_gen_key(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg, 2048, 65537);
    //3. 加密
    ret = mbedtls_rsa_pkcs1_encrypt(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
                                    MBEDTLS_RSA_PUBLIC,
                                    strlen(msg), (uint8_t *)msg, output_buf);
    printf("RSA encrypt result:");
    dump_buf(output_buf, sizeof(output_buf));
  exit:
    mbedtls_ctr_drbg_free(&ctr_drbg);
    mbedtls_entropy_free(&entropy);
    mbedtls_rsa_free(&ctx);
    return ret;
}
```

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;

```c
int RSA_sign_test()
{
    const uint8_t msg[] = "202100460116";
    const uint8_t* pers = "rsa_test";
    RSA_sign(msg, pers);
}
int RSA_sign(const char* msg, const char* pers)
{
    int ret;
    mbedtls_entropy_context entropy;
    mbedtls_ctr_drbg_context ctr_drbg;
    mbedtls_rsa_context ctx;
    //1. 初始化
```

```
    mbedtls_entropy_init(&entropy);
    mbedtls_ctr_drbg_init(&ctr_drbg);
    mbedtls_rsa_init(&ctx, MBEDTLS_RSA_PKCS_V21, MBEDTLS_MD_SHA256);
    ret = mbedtls_ctr_drbg_seed( &ctr_drbg, mbedtls_entropy_func, &entropy,
                                 (const unsigned char *) pers,
                                 strlen(pers));
    //2. 生成key
    ret = mbedtls_rsa_gen_key(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg, 2048, 65537);
    dump_rsa_key(&ctx);
    //3. sign
    ret = mbedtls_rsa_pkcs1_sign(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
                                 MBEDTLS_RSA_PRIVATE, MBEDTLS_MD_SHA256,
                                 strlen(msg), (uint8_t *)msg, output_buf);
    dump_buf(output_buf, sizeof(output_buf));
    //4. 验证
    ret = mbedtls_rsa_pkcs1_verify(&ctx, mbedtls_ctr_drbg_random, &ctr_drbg,
                                 MBEDTLS_RSA_PUBLIC, MBEDTLS_MD_SHA256,
                                 strlen(msg), (uint8_t *)msg, output_buf);
    printf("\nsucess verify !\n");
  exit:
    mbedtls_ctr_drbg_free(&ctr_drbg);
    mbedtls_entropy_free(&entropy);
    mbedtls_rsa_free(&ctx);
    return ret;
}
```

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:

```
uint8_t* dhm(const char* pers)
{
    int ret;
    size_t olen;
    mbedtls_entropy_context entropy;
    mbedtls_ctr_drbg_context ctr_drbg;
    mbedtls_dhm_context dhm_server, dhm_client;
```

```
    //1. 初始化
    mbedtls_entropy_init(&entropy);
    mbedtls_ctr_drbg_init(&ctr_drbg);
    mbedtls_dhm_init(&dhm_server);
    mbedtls_dhm_init(&dhm_client);
    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
        (const unsigned char*)pers,
        strlen(pers));

    // 2.. 生成 2048 bit prime(G, P)
    mbedtls_mpi_read_string(&dhm_server.P, 16, T_P);
    mbedtls_mpi_read_string(&dhm_server.G, 10, GENERATOR);
    dhm_server.len = mbedtls_mpi_size(&dhm_server.P);

    mbedtls_mpi_read_string(&dhm_client.P, 16, T_P);
    mbedtls_mpi_read_string(&dhm_client.G, 10, GENERATOR);
    dhm_client.len = mbedtls_mpi_size(&dhm_client.P);

    //3. 生成公钥
    ret = mbedtls_dhm_make_public(&dhm_server, 256, server_pub, sizeof(server_pub),
        mbedtls_ctr_drbg_random, &ctr_drbg);
    ret = mbedtls_dhm_make_public(&dhm_client, 256, client_pub, sizeof(client_pub),
        mbedtls_ctr_drbg_random, &ctr_drbg);

    //4. 读取公钥
    ret = mbedtls_dhm_read_public(&dhm_server, client_pub, sizeof(client_pub));
    ret = mbedtls_dhm_read_public(&dhm_client, server_pub, sizeof(server_pub));

    //5. 计算共享密钥
    ret = mbedtls_dhm_calc_secret(&dhm_server, server_secret, sizeof(server_secret),
        &olen, mbedtls_ctr_drbg_random, &ctr_drbg);
    ret = mbedtls_dhm_calc_secret(&dhm_client, client_secret, sizeof(client_secret),
        &olen, mbedtls_ctr_drbg_random, &ctr_drbg);
    //dump_buf(client_secret, sizeof(client_secret));
exit:
    mbedtls_ctr_drbg_free(&ctr_drbg);
    mbedtls_entropy_free(&entropy);
    mbedtls_dhm_free(&dhm_server);
    mbedtls_dhm_free(&dhm_client);
    return client_secret;
}
```

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:

```
unsigned char* sha_x(mbedtls_md_type_t md_type,uint8_t*message)
{
    int len, i;
    int ret;
    mbedtls_md_context_t ctx;
```

```
    const mbedtls_md_info_t* info;
    printf("SHA512 source is:\r\n");
    dump_buf(message,256);

    mbedtls_md_init(&ctx);
    info = mbedtls_md_info_from_type(md_type);
    ret = mbedtls_md_setup(&ctx, info, 0);
    ret = mbedtls_md_starts(&ctx);
    ret = mbedtls_md_update(&ctx, (unsigned char*)message, strlen(message));
    ret = mbedtls_md_finish(&ctx, digest);
    len = mbedtls_md_get_size(info);
exit:
    mbedtls_md_free(&ctx);
    return digest;
}
```

4.3 调用 ARIA 分组加密

ARIA 属于分组加密加密算法, 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, 它必须是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:

```
void ARIA_EncryAndDecey(uint8_t* input, int inputLength, const uint8_t* key)
{
    uint8_t iv[16] = "1234567887654321";
    uint8_t iv02[16] = "1234567887654321";
    ARIA_encry(MBEDTLS_CIPHER_ARIA_128_CBC, input, key, inputLength, iv, iv_off_value);
    ARIA_decry(MBEDTLS_CIPHER_ARIA_128_CBC, cipher, key, inputLength, iv02, ...
        iv_off_value);
}
int ARIA_encry(mbedtls_cipher_type_t cipher_type, uint8_t* input, uint8_t* key, int ...
    inputLength, uint8_t* iv, int iv_off)
{
    mbedtls_aria_context ctx;//上下文:密钥状态,
    mbedtls_aria_init(&ctx);
    mbedtls_aria_setkey_enc(&ctx, key, keybits);
    mbedtls_aria_crypt_cfb128(&ctx, MBEDTLS_ARIA_ENCRYPT, inputLength, &iv_off, iv, ...
        input, cipher);
    mbedtls_aria_free(&ctx);
```

```
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:

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,
```

```
    0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41 };
    //加密
    AES(MBEDTLS_CIPHER_AES_128_CBC, MBEDTLS_ENCRYPT,input,key,
                                    sizeof(key),iv,sizeof(iv));
    uint8_t cipher[17];
    for (int i = 0; i < 16;i++){cipher[i] = output_buf[i];}
    cipher[16] = 0x00;
    //解密
    AES(MBEDTLS_CIPHER_AES_128_CBC, MBEDTLS_DECRYPT,cipher,key,
                                   sizeof(key),iv02,sizeof(iv02));
}
int AES(mbedtls_cipher_type_t cipher_type, mbedtls_operation_t mode,
          uint8_t*input,uint8_t*key,int keyLength,uint8_t*iv,int ivLength)
{
    int ret;
    size_t len;
    int olen = 0;
    mbedtls_cipher_context_t ctx;//上下文:密钥状态,
    const mbedtls_cipher_info_t *info;//密码算法类型

    mbedtls_cipher_init(&ctx);
    info = mbedtls_cipher_info_from_type(cipher_type);
    ret = mbedtls_cipher_setup(&ctx, info);

    ret = mbedtls_cipher_setkey(&ctx, key, keyLength*8, mode);
    ret = mbedtls_cipher_set_iv(&ctx, iv, ivLength);

    int a = strlen(input);
    ret = mbedtls_cipher_update(&ctx, input, strlen(input), output_buf, &len);
    olen += len;
    ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
    olen += len;

    dump_buf(output_buf, olen);
  exit:
    mbedtls_cipher_free(&ctx);
    return ret;
}
```

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轜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.