

密码学引论实验——MbedTLS库的使用

 202200460104 网安1班 密语

1. 熟悉MbedTLS密码库：

Mbed TLS（前身为PolarSSL）是一个轻量级的密码库，专为嵌入式设备和受限环境而设计。它提供了一系列密码学功能，包括加密、解密、数字签名、密钥交换、随机数生成等，以及支持各种协议的实现，如SSL/TLS、DTLS、IPsec等。

- **轻量级和可移植性：** Mbed TLS是一个轻量级的密码库，设计用于嵌入式设备和受限环境。它的代码精简、可移植，易于集成到各种不同的硬件平台和操作系统中。
- **密码学功能：** Mbed TLS提供了一系列密码学功能，包括对称加密算法（如AES、DES、3DES）、非对称加密算法（如RSA、DSA、ECC）、哈希函数（如SHA-1、SHA-256）、消息认证码（如HMAC）等。
- **随机数生成：** Mbed TLS包含了用于生成随机数的功能，这对于安全协议和算法来说是至关重要的。它提供了多种随机数生成器（如CTR_DRBG、HMAC_DRBG），以满足不同的安全需求。
- **开源和可扩展：** Mbed TLS是开源的，遵循Apache License 2.0许可。它的源代码可以自由获取和修改，便于用户根据自己的需求进行定制和扩展。

2. 调用DES，加密学号：

- 函数 `int aes_test(mbedtls_cipher_type_t cipher_type)` 的参数是要加密的加密算法类型，此处我们在 `main函数` 中调用该函数，并将参数设置为 `MBEDTLS_CIPHER_DES_CBC`，使用DES加密算法，CBC分组加密算法。
- DES加密的初始向量iv和密钥key都应设置为8个字节：

```
1 uint8_t key[8] = {
2     0x06, 0xa9, 0x21, 0x40, 0x36, 0xb8, 0xa1, 0x5b
3 };
4
5 uint8_t iv[8] = {
6     0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30
7 };
```

- 修改finish cipher，DES的输出需要+olen

```
1 /* 7. finish cipher *///64分组DES输出需要+olen., (填充并生成最后一个分组
2     ret = mbedtls_cipher_finish(&ctx, output_buf+olen , &len);
3     if (ret != 0) {
4         goto exit;
5     }
6     olen += len;
```

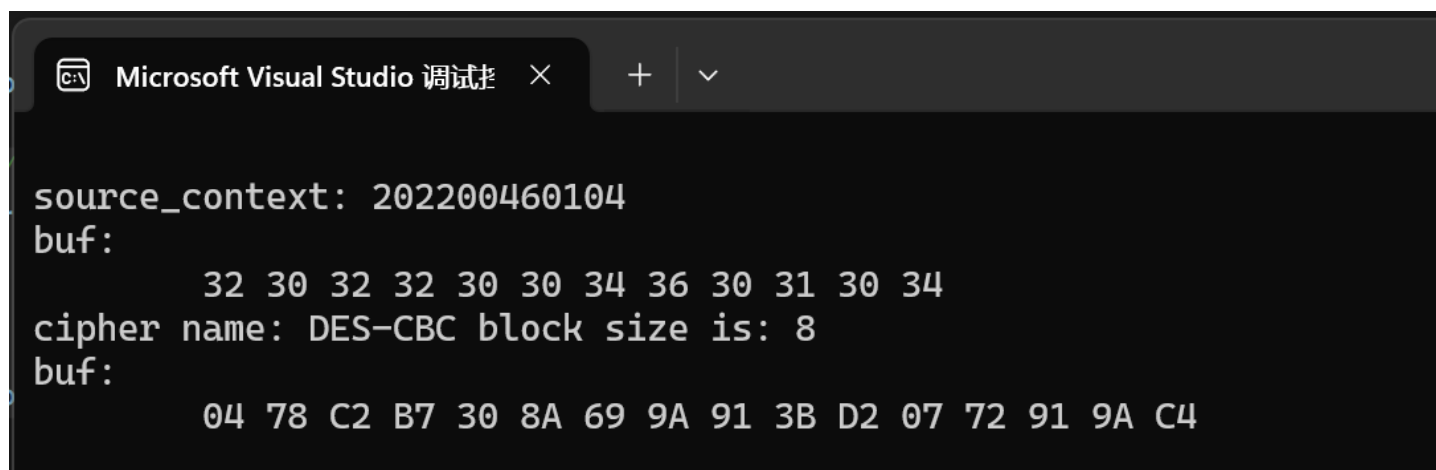
- 设置加密的明文为学号：

```
1 const char input[] = "202200460104";
```

- main函数：

```
1 int main()
2 {
3     aes_test(MBEDTLS_CIPHER_DES_CBC);
4     return 1;
5 }
```

- 加密结果：



The screenshot shows the Microsoft Visual Studio debugger interface. The 'source_context' is set to '202200460104'. The 'buf' variable is displayed with its memory address and contents: '32 30 32 32 30 30 34 36 30 31 30 34'. The 'cipher name' is 'DES-CBC' and the 'block size is: 8'. The 'buf' variable is also displayed with its memory address and contents: '04 78 C2 B7 30 8A 69 9A 91 3B D2 07 72 91 9A C4'.

3. 调用RSA，加密学号：

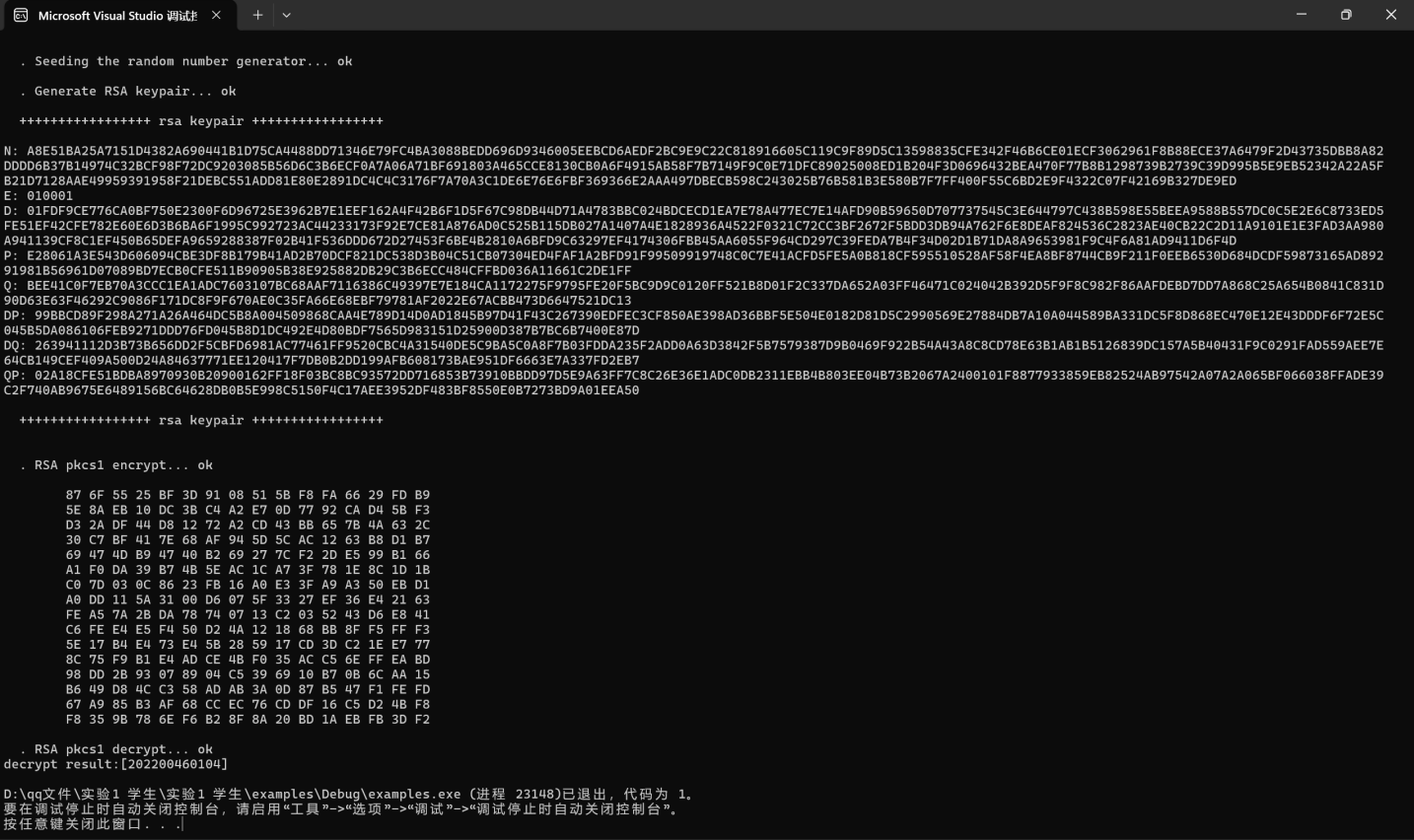
- 修改函数 `mbedtls_rsa_test(void)`，将待加密的明文改为学号：

```
1 const char* msg = "202200460104";
```

- 修改main函数：

```
1 int main()  
2 {  
3     mbedtls_rsa_test();  
4     return 1;  
5 }
```

- 加密结果：



可以发现RSA keypair中包括N,E,D,P,Q,DP,DQ,QP等加解密所用的参数，和加解密的结果。

4. 调用RSA，对学号进行签名：

- 修改函数 `mbedtls_rsa_sign_test()`，将待加密的明文改成学号：

```
1 const char* msg = "202200460104";
```

- 修改main函数为：

```

1 int main()
2 {
3     mbedtls_rsa_sign_test();
4     return 1;
5 }

```

• 签名结果：

```

Microsoft Visual Studio 调试
. Seeding the random number generator... ok
. Generate RSA keypair... ok

+++++ rsa keypair +++++

N: DD82BE968221D86B0247DD042EBB373BA6257C9B76A5993A6CAB1319A176E1E4E7AA57CA65B4B959E29245250F0C90A26FC31374BB4ACF41862057D1454ECA6EB3A8D24E47DD49DE4DF521E7E0BCBE05C4B2B388E88A13F
60C6468F1922E0B26CF7CED08D0590D4689FBE92922350C940E844DBF7482220F28A9C75EDA223FF6CC1F6958F9C19A41FE540B04AD2FB834D3A3F815D7953C98525A8058F2E0118C56592E9BD387895AC424EDE9E79B088D9
E5E4F268C7FA97FCC3136D0B989E8EAC72015002E2ECE2A6482D3B4BC564858032D30BB39471EE36B609D6AB956F4592406D044F9E59EE6668DE61A2FD3EB6B77343A64867047D9CC4755CF6E5
E: 010001
D: 21D1A53135CE0E982EFE376B99400E5A462B6CEC012A83A8B3D4C8BC261E74DA177F7CE88C411A17C3530E41801BDAD64C5E7E181EB959B82986E698395A0D63C00514DBF19880739E9EA578AAFC71F27AE2B870D9E8
33CD85F08CEB226048C8A80E07227F8D67E833C762C8F05C985780E997E1F112DAECE4CBE336DB619447746F5A17CC2AE0686CF705E951935CF7FF0884538851ECA24EFA6BA14BD8038EB41A1CD0AAE2E21768EE8B054EF94
E74FDA0CEC108FD03AE7B9D9BE072D24E814C70B88B80565F9A7FD0A439AB8C0596790264F256889C178FCAB649D55D9F9CFD454ADC093308EBF82D949AE0CE6174E0CE6348EF1456F1503006868686A1
P: F5DD34E4C088E5E149866224771D69EB680C8D1569C977CD83D0CF98358026C4E9F507EF1CAE4BA130A42459B9F7F337D45693E12CD4D29D5F1F7688649BE8806D27493FE07A039F83CCE62FD24930EBF263B7D
89449244A47657C528374A009238525E89494C477F32B63BF7333CEB0C004972901B44A0D126C07FE
Q: E6D071A8453A922CFB66DA89F58663FE897214E1EC680F098E8B1CC49694500E17868D1C550A124C63878F37795753ADD8B2CCF73130A474F2B618063536DC5066327997A494285C28FEDC62BFF87B984BFAE5A6076F29
580AE3F93ACAE9E37E4BAEA114D3AC1A788AC109F06022987E263D206A36252BDCA5484BB99A139B
DP: 18542585CE2C6F73A529D66201F7E1D899F827C75D6561B95D556C2891F012A5B81D8F38880211FEB11A0303A14F37E0FC151820D050242226918E371C493E888B8F0045D0F5678C5B516C8458FEC06B37A9115A26
B60AAE57CA374E9FBCA98451039D0178C81A572D3A7BF52667E9D8591FD7887D08B84D280DFC2CAD7
DQ: 7DFAA52FDC16898C61E157EE3D944EB6822654D5F8950F17395990E8F401196D9788A416C201ABDA62B4A540EFC4E6B4057A0ABF54A7E614AA1B1220D1088E4E90751A43D459C36C06328AAE97714E7F5B761BB8EC758
B6B5218D3C8107EE2D31718C26C4AD3ADDE1171C3AA95CA466D8F0D920C6D3A494BA608B7252AB117E1
QP: E571684EBD9127566C328199A3A537543E79B2F3FC8717C8EC2890165320AEA9276126402ECD84066AB259EEC8B70571C38E29313C80CFFAE3D026D00A96481795EDC5E0B0D29F2E60B426BB8135B25C65B0416C7B389B
80EB447F078F68F30B47A5828561CB6FE8A4814C9FFB8E485947BD367C02D259A369E93DF6ED46D879

+++++ rsa keypair +++++

. RSA pkcs1 sign... ok

63 DE 2C 77 C7 84 7E F3 79 57 2F FF 41 EC C0 5F
68 66 71 A2 6D C8 A1 3A 1C 5B 41 A8 05 C2 72 14
2E E0 21 F8 C1 8F 2D EB B6 C6 79 5B B0 8F 0D 4B
68 FE 22 46 8F 27 F1 A7 6C 2F D0 FB 6A 08 BE DD
1F 7F A0 82 EF EA 1B F7 4E 23 99 E8 1D AC EA E3
9F 33 75 4F EC 97 71 67 D0 91 F0 F3 EC 3C 69 EC
4C 2D 55 1F CB 8C E9 79 0C B5 65 59 D8 47 FF 60
7C 45 47 04 C2 C2 60 7B 7B 23 71 30 4B 73 A4 A2
DC B5 F3 C9 A6 DC 42 89 2D 22 FA 6C 65 84 06 B6
BA 2F B4 DC 4C 9F 35 89 C3 3D 1B C2 FA 67 C1 C3
CD 05 09 F0 12 EE B0 E4 34 2E D3 99 1A D2 E4 26
B4 E4 15 2D CC FF 8D CA C4 EC 54 38 61 64 DB 86
2F CE 6A 7B 36 5F 22 54 F4 FE 2B 2A 99 D3 70 6A
85 CA FD 27 0E 91 2B A5 FF DA 86 4D 8B 04 CE 0C
4C 47 12 A9 C4 DD 5A FD FA 22 50 A1 47 EC 0A 4D
83 42 09 A6 D7 A6 33 77 41 93 F5 18 9C 2F D9 B2

. RSA pkcs1 verify... ok

D:\qq文件\实验1 学生\实验1 学生\examples\Debug\examples.exe (进程 21972)已退出，代码为 1。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

```

5. 利用 DH 密钥协商协议协商一个密钥，并利用 SHA对协商的秘密信息做哈希，得到一个密钥，利用该密钥和 ARIA 密码算法对你的学号进行加密、解密

5.1 修改DH密钥协商函数：

为函数 `mbedtls_dhm_test` 添加一个返回值，用于返回协商的共享秘密。

添加一个 `return_buf` 函数，接收计算出的共享秘密，并保存在char*指针中，用作返回值：

```

1 //mbedtls_dhm_test.c
2 static char *return_buf(uint8_t* buf, uint32_t len)
3 {
4     char* result = (char*)malloc((len * 2 + 1) * sizeof(char));
5     int j;
6     for (j = 0; j < len; j++) {

```

```

7         sprintf(&result[j * 2], "%02X", buf[j]); // 将每个字节转换成两个十六进制字
    符
8     }
9     result[len * 2] = '\0'; // 添加字符串终止符
10
11     return result;
12 }

```

在主函数中使用一个char*指针接收Shared_Secret:

```

1 //main.c
2 char* Shared_Secret=NULL;
3 Shared_Secret=mbedtls_dhm_test();

```

5.2 修改哈希函数:

为函数 `mbedtls_shax_test` 添加一个接收参数:

```

1 char* mbedtls_shax_test(mbedtls_md_type_t md_type, char*message)

```

并使用一个uint_t类型的指针返回一个哈希值的数组，下面给出完整函数代码:

```

1 char* mbedtls_shax_test(mbedtls_md_type_t md_type, char*message)
2 {
3     int len, i;
4     int ret;
5     uint8_t * result = NULL;
6     //const char *message = "HelloWorld";
7     unsigned char digest[32];
8
9     mbedtls_md_context_t ctx;
10    const mbedtls_md_info_t *info;
11
12    printf("\033[31mmessage:\033[0m%s\r\n\n", message);
13
14    /* 1. init mbedtls_md_context_t structure */
15    mbedtls_md_init(&ctx);
16
17    /* 2. get md info structure pointer */
18    info = mbedtls_md_info_from_type(md_type);
19

```

```

20  /* 3. setup md info structure */
21  ret = mbedtls_md_setup(&ctx, info, 0);
22  if (ret != 0) {
23      goto exit;
24  }
25
26  /* 4. start */
27  ret = mbedtls_md_starts(&ctx);
28  if (ret != 0) {
29      goto exit;
30  }
31
32  /* 5. update */
33  ret = mbedtls_md_update(&ctx, (unsigned char *)message, strlen(message));
34  if (ret != 0) {
35      goto exit;
36  }
37
38  /* 6. finish */
39  ret = mbedtls_md_finish(&ctx, digest);
40  if (ret != 0) {
41      goto exit;
42  }
43
44  /* show */
45  printf("%s digest context is:", mbedtls_md_get_name(info));
46  len= mbedtls_md_get_size(info);
47  result = (int*)malloc(len * sizeof(int));
48  for (i = 0; i < len; i++) {
49      printf("%02x", digest[i]);
50      result[i] = (int)digest[i];
51  }
52  printf("]\r\n");
53  printf("\n");
54  exit:
55  /* 7. free */
56  mbedtls_md_free(&ctx);
57
58  return result;
59 }

```

同时主函数中也使用一个uint8_t的指针接收返回值，并指定Hash函数的类型为

MBEDTLS_MD_SHA256：

```
1 uint8_t* Key=NULL;
```

```
2 Key= mbedtls_shax_test(MBEDTLS_MD_SHA256,Shared_Secret);
```

5.3 编写ARIA加密函数：

函数的参数定义如下：

```
1 int aria_encrypt(mbedtls_cipher_type_t cipher_type,uint8_t*Key)
```

接收一个密钥，返回一个加密后的密文数组。

完整函数如下：

```
1 int aria_encrypt(mbedtls_cipher_type_t cipher_type,uint8_t*Key) //可选对称密码算
   法
2 {
3     uint8_t iv[16] = {
4         0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,
5         0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
6     };
7     uint8_t key[32];
8     uint8_t* result = NULL;
9     for (int i = 0; i < 32; i++) {
10         key[i] = Key[i];
11     }
12     printf("\033[31mKey: \033[0m\n");
13     dump_buf(key, 32);
14     int ret;
15     size_t len;
16     int olen = 0;
17     uint8_t output_buf[64];
18     //const char input[] =
19     "HelloWorld123456781231321321123132132132132113213";
20     const char input[] = "202200460104";
21     //const char input[] = "HelloWorld123456";
22     //const char input[] = {'m', 'c', 'u', 'l', 'o', 'v', 'e', 'r', '6', '6',
23     '6', ' ', 'i', 's'};
24
25     mbedtls_cipher_context_t ctx;
26     const mbedtls_cipher_info_t *info;
27
28     /* 1. init cipher structuer */
29     mbedtls_cipher_init(&ctx);
```

```

29  /* 2. get info structuer from type */
30  info = mbedtls_cipher_info_from_type(cipher_type);
31
32  /* 3. setup cipher structuer */
33  ret = mbedtls_cipher_setup(&ctx, info);
34  if (ret != 0) {
35      goto exit;
36  }
37
38  /* 4. set key */
39  ret = mbedtls_cipher_setkey(&ctx, key, sizeof(key) * 8, MBEDTLS_ENCRYPT);
40  if (ret != 0) {
41      goto exit;
42  }
43
44  /* 5. set iv */
45  ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
46  if (ret != 0) {
47      goto exit;
48  }
49
50  /* 6. update cipher */
51  ret = mbedtls_cipher_update(&ctx, (unsigned char*)input, strlen(input),
output_buf, &len);
52  if (ret != 0) {
53      goto exit;
54  }
55  olen += len;
56
57  /* 7. finish cipher */ /* 64分组DES输出需要+olen., (填充并生成最后一个分组
58  ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
59  if (ret != 0) {
60      goto exit;
61  }
62  olen += len;
63
64  /* show */
65  printf("\r\nsource_context: %s\r\n", input);
66  dump_buf((uint8_t *)input, strlen(input));
67  len = mbedtls_md_get_size(info);
68  result = (int*)malloc(16*sizeof(int));
69  for (int i = 0; i < 16; i++) {
70      result[i] = output_buf[i];
71  }
72  printf("\n");
73  printf("cipher name: %s block size is: %d\r\n",
mbedtls_cipher_get_name(&ctx), mbedtls_cipher_get_block_size(&ctx));

```



```

74     printf("\033[31mencrypt_result\033[0m: \n");
75     dump_buf(output_buf, olen);
76     printf("\n");
77     exit:
78     /* 8. free cipher structure */
79     mbedtls_cipher_free(&ctx);
80
81
82     return result;
83 }

```

5.4 编写ARIA解密函数：

函数的定义如下：

```

1 int aria_test(uint8_t *cipher , uint8_t *key)

```

接收密文和密钥，解密后输出对应的明文。

完整的解密函数如下：

```

1 int aria_test(uint8_t *cipher , uint8_t *key)
2 {
3     uint8_t iv[16] = {
4         0x3d, 0xaf, 0xba, 0x42, 0x9d, 0x9e, 0xb4, 0x30,
5         0xb4, 0x22, 0xda, 0x80, 0x2c, 0x9f, 0xac, 0x41
6     };
7     uint8_t input[16];
8     uint8_t Key[32];
9     for (int i = 0; i < 16; i++) {
10         input[i] = cipher[i];
11     }
12     for (int i = 0; i < 32; i++) {
13         Key[i] = key[i];
14     }
15     int ret;
16     size_t len;
17     uint8_t output_buf[64];
18
19     mbedtls_cipher_context_t ctx;
20     const mbedtls_cipher_info_t* info;
21
22     mbedtls_cipher_init(&ctx);

```

```
23
24     info = mbedtls_cipher_info_from_type(MBEDTLS_CIPHER_ARIA_256_CBC);
25
26     ret = mbedtls_cipher_setup(&ctx, info);
27     if (ret != 0) {
28         goto exit;
29     }
30
31     ret = mbedtls_cipher_setkey(&ctx, Key, sizeof(Key) * 8, MBEDTLS_DECRYPT);
32     if (ret != 0) {
33         goto exit;
34     }
35
36     ret = mbedtls_cipher_set_iv(&ctx, iv, sizeof(iv));
37     if (ret != 0) {
38         goto exit;
39     }
40
41     ret = mbedtls_cipher_reset(&ctx);
42     if (ret != 0) {
43         goto exit;
44     }
45
46     ret = mbedtls_cipher_update(&ctx, input, sizeof(input), output_buf, &len);
47     if (ret != 0) {
48         goto exit;
49     }
50     ret = mbedtls_cipher_finish(&ctx, output_buf, &len);
51     if (ret != 0) {
52         goto exit;
53     }
54
55
56     printf("\033[31mdump_buf: \033[0m\n");
57     dump_buf(output_buf, len);
58     printf("\033[31mdecrypt_result: \033[0m\n");
59     for (int i = 0; i < 16; i++) {
60         printf("%c", output_buf[i]);
61     }
62
63 exit:
64     mbedtls_cipher_free(&ctx);
65
66     return ret;
67 }
```

5.5 main函数：

通过给函数增加接收参数和添加返回值，从而实现了全流程的自动化。

main函数如下：

```
1 int main()
2 {
3     char* Shared_Secret=NULL;
4     uint8_t* Key=NULL;
5     uint8_t* cipher = NULL;
6     printf("\033[32m[+]DES Encrypt:\033[0m\n");
7     des_test(MBEDTLS_CIPHER_DES_CBC); //DES加密
8     printf("\033[32m[+]RSA Encrypt:\033[0m\n");
9     mbedtls_rsa_test(); //RSA加密
10    printf("\033[32m[+]DES Sign:\033[0m\n");
11    mbedtls_rsa_sign_test(); //RSA签名
12    printf("\033[32m[+]DH密钥协商: \033[0m\n");
13    Shared_Secret=mbedtls_dhm_test(); //DH密钥协商
14    printf("\n\033[31mshared_secret:\033[0m%s\n\n",Shared_Secret);
15    printf("\033[32m[+]SHA256: \033[0m\n");
16    Key= mbedtls_shax_test(MBEDTLS_MD_SHA256,Shared_Secret); //SHA256
17    printf("\033[32m[+]ARIA Encrypt: \033[0m\n");
18    cipher=aria_encrypt(MBEDTLS_CIPHER_ARIA_256_CBC,Key); //ARIA加密
19    printf("\033[32m[+]ARIA Decrypt: \033[0m\n");
20    aria_test(cipher,Key); //ARIA解密
21    free(Shared_Secret); //释放指针
22    free(Key);
23    free(cipher);
24    return 1;
25 }
```

5.6 运行效果：

```
shared_secret:7477E281C250EFEB778F4DA0B1FB8F4F719DA09D364D1C7DD2B8C2F73C3CB3ADADE6F2EDDC3715633581B40216EA2586A74D0EEED64EDF21ACE05708B7F38C38E8D61D44F634F655BCF5BFBE3321E2C3A355CF537C703BD6342139E3421533415FF1164C0E71165420F6AF03A2C66C9D6B0DC034DDEFB3C5173E134817EEF245A7D61CE20D603605685F3AB53F0E82694118F9C07829DAB772CD73A3112464B6BF8FEEFEDCB0F17805D914F975D4CE04FF9734044996D85611E4E9AE490560B142A6BA541F66817F776581B57B927F09D4D6116435C83CAA735532F5812FD0E9353460077F18C9788D9264548B98A0CD538DB75477FFDC2E16D29F41ECDB84B

[+]SHA256:
message:7477E281C250EFEB778F4DA0B1FB8F4F719DA09D364D1C7DD2B8C2F73C3CB3ADADE6F2EDDC3715633581B40216EA2586A74D0EEED64EDF21ACE05708B7F38C38E8D61D44F634F655BCF5BFBE3321E2C3A355CF537C703BD6342139E3421533415FF1164C0E71165420F6AF03A2C66C9D6B0DC034DDEFB3C5173E134817EEF245A7D61CE20D603605685F3AB53F0E82694118F9C07829DAB772CD73A3112464B6BF8FEEFEDCB0F17805D914F975D4CE04FF9734044996D85611E4E9AE490560B142A6BA541F66817F776581B57B927F09D4D6116435C83CAA735532F5812FD0E9353460077F18C9788D9264548B98A0CD538DB75477FFDC2E16D29F41ECDB84B

SHA256 digest context is:[6f4bdd3db4ef769472a4c746ee064e68081a8004041b0b60a083a6665bd81ffe]

[+]ARIA Encrypt:
Key:
buf:
    6F 4B DD 3D B4 EF 76 94 72 A4 C7 46 EE 06 4E 68
    08 1A 80 04 04 1B 0B 60 A0 83 A6 66 5B D8 1F FE

source_context: 202200460104
buf:
    32 30 32 32 30 30 34 36 30 31 30 34

cipher name: ARIA-256-CBC block size is: 16
encrypt_result:
buf:
    D7 1F F9 5D 47 AA 65 64 7F 6A C1 C8 93 BA C6 42

[+]ARIA Decrypt:
dump_buf:
buf:
    32 30 32 32 30 30 34 36 30 31 30 34
decrypt_result:
202200460104
```

可以清楚的展示出明文，密文，密钥的字节数，同时解密的结果也是正确的。

还通过网站验证了该sha256的哈希值是正确的。

SHA256加密在线工具

1

7477E281C250EFEB778F4DA0B1FB8F4F719DA09D364D1C7DD2B8C2F73C3CB3ADADE6F2EDDC3715633581B40216EA2586A74D0EEED64EDF21ACE05708B7F38C38E8D61D44F634F655BCF5BFBE3321E2C3A355CF537C703BD6342139E3421533415FF1164C0E71165420F6AF03A2C66C9D6B0DC034DDEFB3C5173E134817EEF245A7D61CE20D603605685F3AB53F0E82694118F9C07829DAB772CD73A3112464B6BF8FEEFEDCB0F17805D914F975D4CE04FF9734044996D85611E4E9AE490560B142A6BA541F66817F776581B57B927F09D4D6116435C83CAA735532F5812FD0E9353460077F18C9788D9264548B98A0CD538DB75477FFDC2E16D29F41ECDB84B

字母大小写

小写

清空

SHA256加密

复制结果

1

6f4bdd3db4ef769472a4c746ee064e68081a8004041b0b60a083a6665bd81ffe