2020 Spring RSA Public-Key Encryption and Signature

实验要求

Task 1: Deriving the Private Key

Task 2: Encrypting a Message

Task 3: Decrypting a Message

Task 4: Signing a Message

Task 5: Verifying a Signature

Task 6: Manually Verifying an X.509 Certificate

实验代码

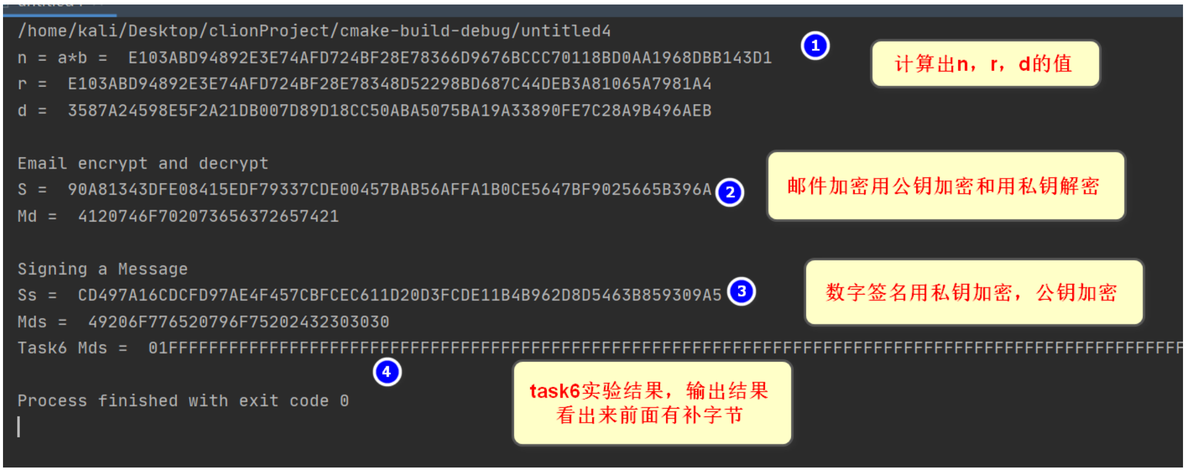
> 附录代码实验环境clion+kali 注意cmake文件里面配置link_libraries(crypto)

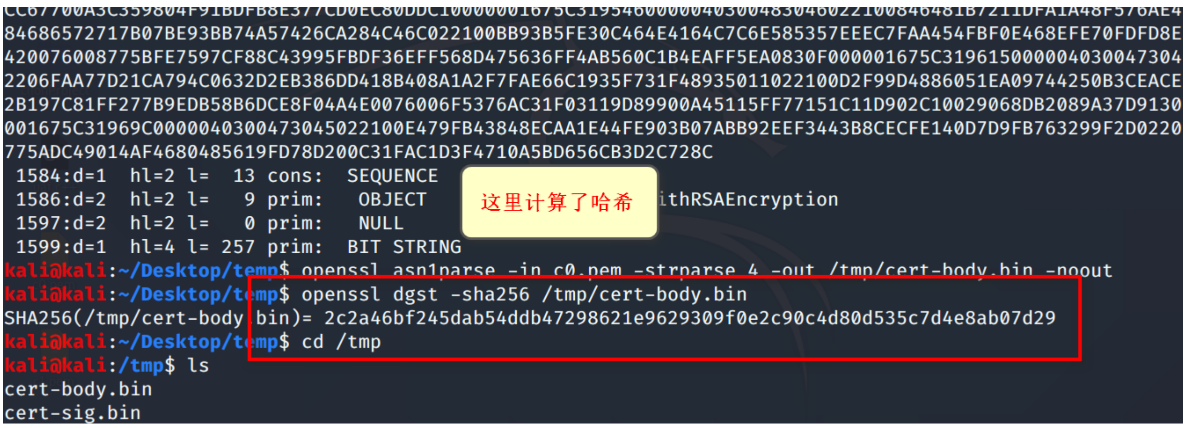
```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char* msg, BIGNUM *a){
    char* number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}
int main ()
{
    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* a = BN_new();
    BIGNUM* b = BN_new();

    BIGNUM* n = BN_new();
    BIGNUM* res = BN_new();

    /*example
    // Initialize a, b, n
    BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
    BN_dec2bn(&b, "27348946379683850184859276946719436926 8");
    BN_rand(n, NBITS, 0, 0);
    // res = a * b
    BN_mul(res, a, b, ctx);
    printBN("a*b = ", res);
    // res = a^b mod n
    BN_mod_exp(res, a, b, n, ctx);
    printBN("a^c mod n = ", res);
     */
```

```
30
31      //geneerate
32      BIGNUM* r = BN_new();
33      BIGNUM* e = BN_new();
34      BIGNUM* d = BN_new();
35      BIGNUM* a_1 = BN_new();
36      BIGNUM* b_1 = BN_new();
37      BIGNUM* temp = BN_new();
38
39
40      BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
41      BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");
42      BN_hex2bn(&e, "0D88C3");
43
44      BN_hex2bn(&a_1, "F7E75FDC469067FFDC4E847C51F452DE");
45      BN_hex2bn(&b_1, "E85CED54AF57E53E092113E62F436F4E");
46
47
48      BN_mul(n, a, b, ctx);
49      printBN("n = a*b = ", n);
50
51      BN_mul(r, a_1, b_1, ctx);
52      printBN("r = ", r);
53
54      BN_mod_inverse(d, e, r, ctx);
55      printBN("d = ", d);
56
57
58      //encrypt
59      printf("\nEmail encrypt and decrypt\n");
60      BIGNUM* M = BN_new();
61      BN_hex2bn(&M, "4120746f702073656372657421");
62
63      BIGNUM* S = BN_new();
64      BN_mod_exp(S, M, e, n, ctx);
65      printBN("S = ", S);
66
67
68      BIGNUM* Md = BN_new();
69      BN_mod_exp(Md, S, d, n, ctx);
70      printBN("Md = ", Md);
71
72      //签名,用私钥加密49206f776520796f75202432303030
73      printf("\nSigning a Message\n");
74
75      BIGNUM* Ms = BN_new();
76      BN_hex2bn(&Ms, "49206f776520796f75202432303030");
77      //BN_hex2bn(&Ms, "49206f776520796f75202433303030");
78
79      BIGNUM* Ss = BN_new();
80      BN_mod_exp(Ss, Ms, d, n, ctx);
81      printBN("Ss = ", Ss);
82
83      BIGNUM* Msd = BN_new();
84      BN_mod_exp(Msd, Ss, e, n, ctx);
85      printBN("Mds = ", Msd);
86
87      //task6验证
```

```c
    BN_hex2bn(&e, "10001");
    BN_hex2bn(&n,
"DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066D0
0A736E1198361764AF379DFDFA4184AFC7AF8CFE1A734DCF339790A2968753832BB9A675482
D1D56377BDA31321AD7ACAB06F4AA5D4BB74746DD2A93C3902E798080EF13046A143BB59B92
BEC207654EFCDAFCFF7AAEDC5C7E55310CE83907A4D7BE2FD30B6AD2B1DF5FFE5774533B358
0DDAE8E4498B39F0ED3DAE0D7F46B29AB44A74B58846D924B81C3DA738B129748900445751A
DD37319792E8CD540D3BE4C13F395E2EB8F35C7E108E8641008D456647B0A165CEA0AA29094
EF397EBE82EAB0F72A7300EFAC7F4FD1477C3A45B2857C2B3F982FDB745589B");
    BN_hex2bn(&M,
"737085Ef4041a76a43d5789c7b5548e6bc6b9986bafb0d038b78fe11f029a00ccd69140bc6
0478b2cef087d5019dc4597a71fef06e9ec1a0b0912d1fea3d55c533050ccdc13518b06a686
64cbf5621da5bd948b98c3521915ddc75d77a462c2227a66fd33a17ebbebd13c5122673c05d
a335896afb27d4ddaa74742e37e5013ba6d030b083d0a1c4752185b2e5fa670030a2bc53834
dbfd6a883bbbcd6ed1cb31ef1580382008e9cef90f21a5fa2a306da5dbe9fda5da6e62fde58
8018d3f1627ba6a39faea86972638165ae8283a3b5978a9b2051ff1a3f61401e48d06b38f9e
1fa17d8774a88e63d36244fef0ab99f70f38327f8cf2a057510a18a0a8088cd");
    BN_mod_exp(Msd, M, e, n, ctx);
    printBN("Task6 Mds = ", Msd);


    return 0;
}
```

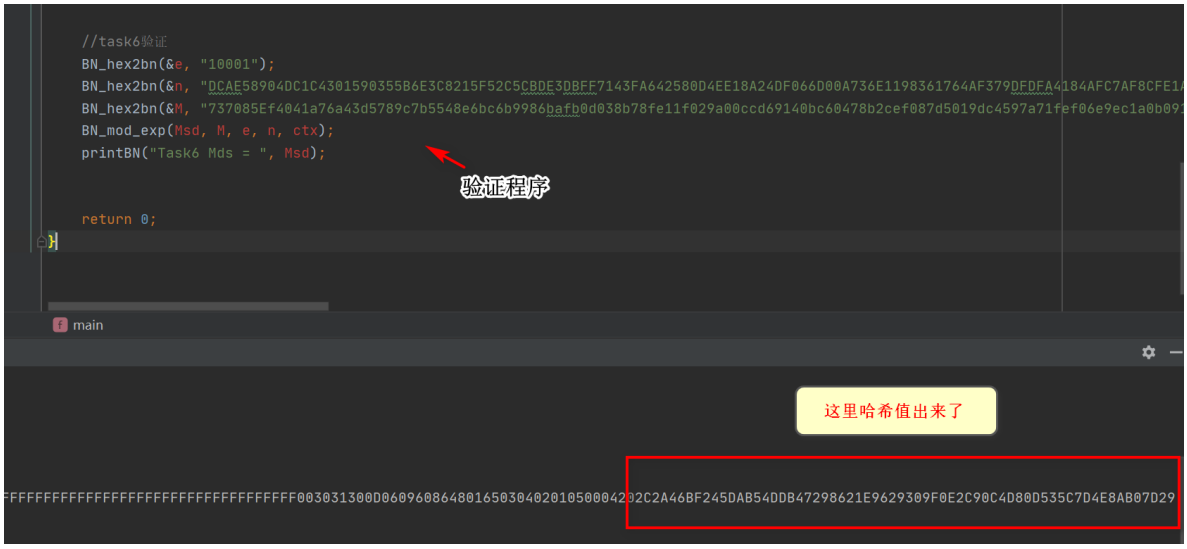
实验截图






D6:56:CB:3D:2C:72:8C
Signature Algorithm: sha256WithRSAEncryption
73:70:85:ef:40:41:a7:6a:43:d5:78:9c:7b:55:48:e6:bc:6b:
99:86:ba:fb:0d:03:8b:78:fe:11:f0:29:a0:0c:cd:69:14:0b:
c6:04:78:b2:ce:f0:87:d5:01:9d:c4:59:7a:71:fe:f0:6e:9e:
c1:a0:b0:91:2d:1f:ea:3d:55:c5:33:05:0c:cd:c1:35:18:b0:
6a:68:66:4c:bf:56:21:da:5b:d9:48:b9:8c:35:21:91:5d:dc:
75:d7:7a:46:2c:22:27:a6:6f:d3:3a:17:eb:be:bd:13:c5:12:
26:73:c0:5d:a3:35:89:6a:fb:27:d4:dd:aa:74:74:2e:37:e5:
01:3b:a6:d0:30:b0:83:d0:a1:c4:75:21:85:b2:e5:fa:67:00:
30:a2:bc:53:83:4d:bf:d6:a8:83:bb:bc:d6:ed:1c:b3:1e:f1:
58:03:82:00:8e:9c:ef:90:f2:1a:5f:a2:a3:06:da:5d:be:9f:
da:5d:a6:e6:2f:de:58:80:18:d3:f1:62:7b:a6:a3:9f:ae:a8:
69:72:63:81:65:ae:82:83:a3:b5:97:8a:9b:20:51:ff:1a:3f:
61:40:1e:48:d0:6b:38:f9:e1:fa:17:d8:77:4a:88:e6:3d:36:
24:4f:ef:0a:b9:9f:70:f3:83:27:f8:cf:2a:05:75:10:a1:8a:
0a:80:88:cd
kali@kali:~/Desktop/temp$ SIGNATURE_HEX=$(openssl x509 -in c0.pem -text -noout -certopt ca_default -certopt no_validity -certopt no_serial -certopt no_subject -certopt no_extensions -certopt no_signame | grep -v 'Signature Algorithm' | tr -d '[:space:]:')
kali@kali:~/Desktop/temp$ echo ${SIGNATURE_HEX} | xxd -r -p > /tmp/cert-sig.bin
kali@kali:~/Desktop/temp$ openssl rsautl -verify -inkey /tmp/issuer-pub.pem -in /tmp/cert-sig.bin -pubin > /tmp/cert-sig-decrypted.bin
kali@kali:~/Desktop/temp$ openssl asn1parse -inform der -in /tmp/cert-sig-decrypted.bin
数字签名


kr7CB2VO/Nr8/3qu3Fx+VTEM6DkHpNe+L9MLatKx31/+V3RTOzWA3a6ORJiznw7T
2uDX9Gspq0SnS1iEbZJLgcPac4sSl0iQBEV1Gt03MZeS6M1UDTvkwT85Xi6481x+
EI6GQQCNRWZHsKFlzqCqKQlO85fr6C6rD3KnMA76x/T9FHfDpFsoV8Kz+YL9t0VY
mwIDAQAB
-----END PUBLIC KEY-----
RSA Public-Key: (2048 bit)
Modulus:
00:dc:ae:58:90:4d:c1:c4:30:15:90:35:5b:6e:3c:
82:15:f5:2c:5c:bd:e3:db:ff:71:43:fa:64:25:80:
d4:ee:18:a2:4d:f0:66:d0:0a:73:6e:11:98:36:17:
64:af:37:9d:fd:fa:41:84:af:c7:af:8c:fe:1a:73:
4d:cf:33:97:90:a2:96:87:53:83:2b:b9:a6:75:48:
2d:1d:56:37:7b:da:31:32:1a:d7:ac:ab:06:f4:aa:
5d:4b:b7:47:46:dd:2a:93:c3:90:2e:79:80:80:ef:
13:04:6a:14:3b:b5:9b:92:be:c2:07:65:4e:fc:da:
fc:ff:7a:ae:dc:5c:7e:55:31:0c:e8:39:07:a4:d7:
be:2f:d3:0b:6a:d2:b1:df:5f:fe:57:74:53:3b:35:
80:dd:ae:8e:44:98:b3:9f:0e:d3:da:e0:d7:f4:6b:
29:ab:44:a7:4b:58:84:6d:92:4b:81:c3:da:73:8b:
12:97:48:90:04:45:75:1a:dd:37:31:97:92:e8:cd:
54:0d:3b:e4:c1:3f:39:5e:2e:b8:f3:5c:7e:10:8e:
86:41:00:8d:45:66:47:b0:a1:65:ce:a0:aa:29:09:
4e:f3:97:eb:e8:2e:ab:0f:72:a7:30:0e:fa:c7:f4:
fd:14:77:c3:a4:5b:28:57:c2:b3:f9:82:fd:b7:45:
58:9b
Exponent: 65537 (0x10001)
kali@kali:/tmp$ echo ${SIGNATURE_HEX}
737085ef4041a76a43d5789c7b5548e6bc6b9986bafb0d038b78fe11f029a00ccd69140bc60478b2cef087d5019dc4597a71fef06
e9ec1a0b0912d1fea3d55c533050ccdc13518b06a68664cbf5621da5bd948b98c3521915ddc75d77a462c2227a66fd33a17ebbebd
13c5122673c05da335896afb27d4ddaa74742e37e5013ba6d030b083d0a1c4752185b2e5fa670030a2bc53834dbfd6a883bbbcd6e
d1cb31ef1580382008e9cef90f21a5fa2a306da5dbe9fda5da6e62fde588018d3f1627ba6a39faea86972638165ae8283a3b5978a
9b2051ff1a3f61401e48d06b38f9e1fa17d8774a88e63d36244fef0ab99f70f38327f8cf2a057510a18a0a8088cd
kali@kali:/tmp$
1
2
3
n模数
e的值
签名


//task6验证
BN_hex2bn(&e, "10001");
BN_hex2bn(&n, "DCAE58904DC1C4301590355B6E3C8215F52C5CBDE3DBFF7143FA642580D4EE18A24DF066D00A736E1198361764AF379DFDFA4184AFC7AF8CFE1A
BN_hex2bn(&M, "737085Ef4041a76a43d5789c7b5548e6bc6b9986bafb0d038b78fe11f029a00ccd69140bc60478b2cef087d5019dc4597a71fef06e9ec1a0b091
BN_mod_exp(Msd, M, e, n, ctx);
printBN("Task6 Mds = ", Msd);
return 0;
}
main
验证程序
这里哈希值出来了
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D060960864801650304020105000420C2A46BF245DAB54DDB47298621E9629309F0E2C90C4D80D535C7D4E8AB07D29

task6 根据证书计算出。

实验总结

在task6中一开始按照始终解决不了计算出来的结果和hash不多。有学习https://linuxctl.com/2017/02/x509-certificate-manual-signature-verification/这篇文章的解决思路。这篇文章虽说是手动检查证书，但是最后也是没有使用公私钥去解决问题。而是用openssl命令`openssl rsautl -verify -inkey /tmp/issuer-pub.pem -in /tmp/cert-sig.bin -pubin > /tmp/cert-sig-decrypted.bin`

再`openssl asn1parse -inform der -in /tmp/cert-sig-decrypted.bin`计算出hash。最后还有一个疑问，最后证书签名过程中到底还用了什么数据，如上图，在计算出来的hash前面还有一串位未知意义的字符。