Report on Project E

Tran Thi Hong Hanh

04^{th}, May, 2019

Abstract

Project E includes 2 practical works. The former one focuses on encrypt/decrypt/MAC on AES, while the latter is about key generation, encrypt/decrypt and sign/verify on RSA using Standard Mode and CRT mode. The report will cover the implementation and the results, answering all questions from these practical works.

1 Encrypt/Decrypt/MAC on AES

1.1 AES encryption/decryption on ECB

The exercise includes 2 C program which, respectively, ciphers and deciphers a file in ECB chaining mode. Both programs are executed as following:

1.1.1 Encryption

- Command:
  - Build: gcc -o aes-encrypt-ecb aes-encrypt-ecb.c aes.c
  - Execute: ./aes-encrypt-ecb logo.png aes-e-ecb
- Result: aes-e-ecb

1.1.2 Decryption

- Command:
  - Build: gcc -o aes-decrypt-ecb aes-decrypt-ecb.c aes.c
  - Execute: ./aes-decrypt-ecb aes-e-ecb aes-d-ecb
- Result: aes-d-ecb (exactly the same logo.png)

1.2 AES encryption/decryption on CBC

The exercise includes 2 C program which, respectively, ciphers and deciphers a file in CBC chaining mode. Both programs are executed as following:

1.2.1 Encryption

- Command:
  - Build: gcc -o aes-encrypt-cbc aes-encrypt-cbc.c aes.c
  - Execute: ./aes-encrypt-cbc logo.png aes-e-cbc
- Result: aes-e-cbc

1.2.2 Decryption

- Command:
  - Build: gcc -o aes-decrypt-cbc aes-decrypt-cbc.c aes.c
  - Execute: ./aes-decrypt-cbc aes-e-cbc aes-d-cbc
- Result: aes-d-cbc (exactly the same logo.png)

1.3 MAC AES CBC

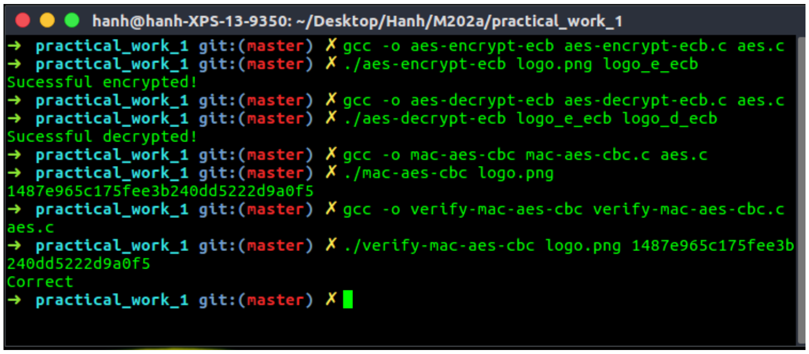
The program computes the MAC AES CBC message authentication code of a file, and outputs it in hexadecimal form. The program are executed as following:

- Command:
  - Build: gcc -o mac-aes-cbc mac-aes-cbc.c aes.c
  - Execute: ./mac-aes-cbc logo.png
- Result: MAC value (e.g: 1487e965c175fee3b240dd5222d9a0f5)

1.4 Verify MAC AES CBC

The program verifies the consistency between a file and a MAC value given as a string made of only hexadecimal digits. The output is a message informing whether the MAC is correctly verified for this file or not.

- Command:
  - Build: gcc -o verify-mac-aes-cbc verify-mac-aes-cbc.c aes.c
  - Execute: ./verify-mac-aes-cbc logo.png 1487e965c175fee3b240dd5222d9a0f5
- Result: Display ”Correct” if they are the same MAC.


hanh@hanh-XPS-13-9350: ~/Desktop/Hanh/M202a/practical_work_1

```
→ practical_work_1 git:(master) ✗ gcc -o aes-encrypt-ecb aes-encrypt-ecb.c aes.c
→ practical_work_1 git:(master) ✗ ./aes-encrypt-ecb logo.png logo_e_ecb
Sucessful encrypted!
→ practical_work_1 git:(master) ✗ gcc -o aes-decrypt-ecb aes-decrypt-ecb.c aes.c
→ practical_work_1 git:(master) ✗ ./aes-decrypt-ecb logo_e_ecb logo_d_ecb
Sucessful decrypted!
→ practical_work_1 git:(master) ✗ gcc -o mac-aes-cbc mac-aes-cbc.c aes.c
→ practical_work_1 git:(master) ✗ ./mac-aes-cbc logo.png
1487e965c175fee3b240dd5222d9a0f5
→ practical_work_1 git:(master) ✗ gcc -o verify-mac-aes-cbc verify-mac-aes-cbc.c
aes.c
→ practical_work_1 git:(master) ✗ ./verify-mac-aes-cbc logo.png 1487e965c175fee3b
240dd5222d9a0f5
Correct
→ practical_work_1 git:(master) ✗
```

Figure 1: The commands of the practical work 1

2 Key generation, Encrypt/Decrypt, Sign/Verify on RSA

2.1 RSA Cryptosystem (standard mode)

2.1.1 Key generation

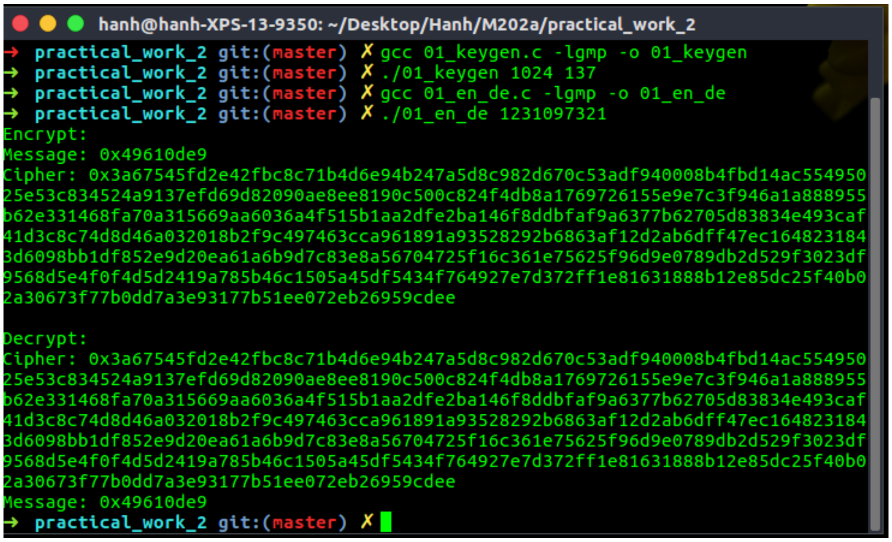
The program takes as inputs two integers k and e, and generates a (standard mode) RSA key of size k bits with e as its public exponent.The modulus n is computed as the product of two primes p and q of same bit-size and n bit-size itself is exactly k.

- Command:
  - Build: gcc 01_keygen.c -lgmp -o 01_keygen
  - Execute: ./01_keygen 1024 137
- Result: The generated key on standard output in hexadecimal form, saved in key.txt.

2.1.2 Encryption and Decryption

The program contains an encryption function; a decryption function that allow to encrypt and/or decrypt a small text. The key elements are to be read from a key file created at key generation part.

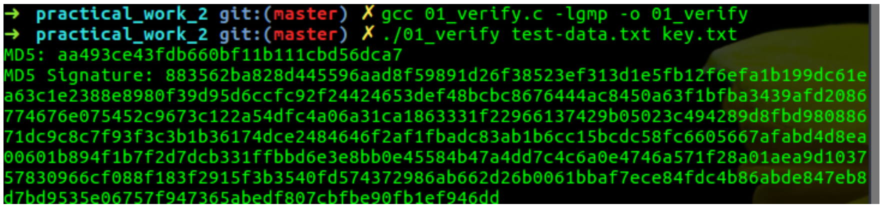
- Command:
  - Build: gcc 01_en_de.c -lgmp -o 01_en_de
  - Execute: ./01_en_de 1231097321
- Result are illustrated as Figure 2.



```
hanh@hanh-XPS-13-9350: ~/Desktop/Hanh/M202a/practical_work_2
→ practical_work_2 git:(master) ✗ gcc 01_keygen.c -lgmp -o 01_keygen
→ practical_work_2 git:(master) ✗ ./01_keygen 1024 137
→ practical_work_2 git:(master) ✗ gcc 01_en_de.c -lgmp -o 01_en_de
→ practical_work_2 git:(master) ✗ ./01_en_de 1231097321
Encrypt:
Message: 0x49610de9
Cipher: 0x3a67545fd2e42fbc8c71b4d6e94b247a5d8c982d670c53adf940008b4fbd14ac554950
25e53c834524a9137efd69d82090ae8ee8190c500c824f4db8a1769726155e9e7c3f946a1a888955
b62e331468fa70a315669aa6036a4f515b1aa2dfe2ba146f8ddbfaf9a6377b62705d83834e493caf
41d3c8c74d8d46a032018b2f9c497463cca961891a93528292b6863af12d2ab6dff47ec164823184
3d6098bb1df852e9d20ea61a6b9d7c83e8a56704725f16c361e75625f96d9e0789db2d529f3023df
9568d5e4f0f4d5d2419a785b46c1505a45df5434f764927e7d372ff1e81631888b12e85dc25f40b0
2a30673f77b0dd7a3e93177b51ee072eb26959cdee

Decrypt:
Cipher: 0x3a67545fd2e42fbc8c71b4d6e94b247a5d8c982d670c53adf940008b4fbd14ac554950
25e53c834524a9137efd69d82090ae8ee8190c500c824f4db8a1769726155e9e7c3f946a1a888955
b62e331468fa70a315669aa6036a4f515b1aa2dfe2ba146f8ddbfaf9a6377b62705d83834e493caf
41d3c8c74d8d46a032018b2f9c497463cca961891a93528292b6863af12d2ab6dff47ec164823184
3d6098bb1df852e9d20ea61a6b9d7c83e8a56704725f16c361e75625f96d9e0789db2d529f3023df
9568d5e4f0f4d5d2419a785b46c1505a45df5434f764927e7d372ff1e81631888b12e85dc25f40b0
2a30673f77b0dd7a3e93177b51ee072eb26959cdee
Message: 0x49610de9
→ practical_work_2 git:(master) ✗
```

Figure 2: The commands of the key generation and encryption/ decryption on standard mode



```
→ practical_work_2 git:(master) ✗ gcc 01_verify.c -lgmp -o 01_verify
→ practical_work_2 git:(master) ✗ ./01_verify test-data.txt key.txt
MD5: aa493ce43fdb660bf11b111cbd56dca7
MD5 Signature: 883562ba828d445596aad8f59891d26f38523ef313d1e5fb12f6efa1b199dc61e
a63c1e2388e8980f39d95d6ccfc92f24424653def48bcbc8676444ac8450a63f1bfba3439afd2086
774676e075452c9673c122a54dfc4a06a31ca1863331f22966137429b05023c494289d8fbd980886
71dc9c8c7f93f3c3b1b36174dce2484646f2af1fbadc83ab1b6cc15bcdc58fc6605667afabd4d8ea
00601b894f1b7f2d7dcb331ffbbd6e3e8bb0e45584b47a4dd7c4c6a0e4746a571f28a01aea9d1037
57830966cf088f183f2915f3b3540fd574372986ab662d26b0061bbaf7ece84fdc4b86abde847eb8
d7bd9535e06757f947365abedf807cbfbe90fb1ef946dd
```

Figure 3: The commands of the signature and verification on standard mode

2.1.3 Signature and Verification

The program takes a filename and an RSA private key as inputs, and computes the signature of the file.

- Command:
  - Build: gcc 01_verify.c -lgmp -o 01_verify
  - Execute: ./01_verify test-data.txt key.txt
- Result are illustrated as Figure 3.

2.2 RSA Cryptosystem (CRT mode)

The programs in CRT mode are similar to standard mode in encryption part, however, it is necessary to modify specific part on decryption in standard mode.

In both mode, e and n are used as the public key. Meanwhile, for private key, the standard mode use d and the CRT mode use more parameters, including p, q, dp, dq and qi. Therefore, we need to change private key generation, decryption and signing while we remain the encryption.

2.2.1 Key generation

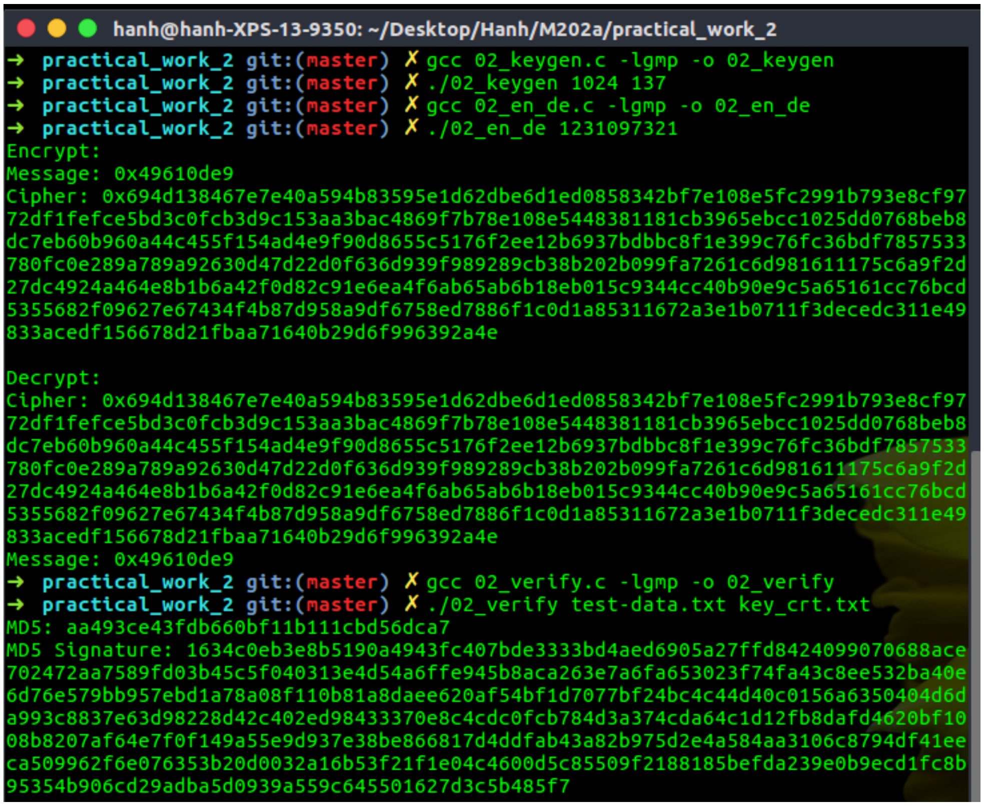
- Command:
  - Build: gcc 02_keygen.c -lgmp -o 02_keygen
  - Execute:./02_keygen 1024 137
- Result: The generated key on standard output in hexadecimal form, saved in key.txt.

2.2.2 Encryption and Decryption

- Command:
  - Build: gcc 02_en_de.c -lgmp -o 02_en_de
  - Execute: ./02_en_de 1231097321
- Result are illustrated as Figure 4.

2.2.3 Signature and Verification

- Command:
  - Build: gcc 02_verify.c -lgmp -o 02_verify
  - Execute: ./02_verify test-data.txt key_crt.txt
- Result are illustrated as Figure 4.



```
hanh@hanh-XPS-13-9350: ~/Desktop/Hanh/M202a/practical_work_2
→  practical_work_2 git:(master) ✗ gcc 02_keygen.c -lgmp -o 02_keygen
→  practical_work_2 git:(master) ✗ ./02_keygen 1024 137
→  practical_work_2 git:(master) ✗ gcc 02_en_de.c -lgmp -o 02_en_de
→  practical_work_2 git:(master) ✗ ./02_en_de 1231097321
Encrypt:
Message: 0x49610de9
Cipher: 0x694d138467e7e40a594b83595e1d62dbe6d1ed0858342bf7e108e5fc2991b793e8cf97
72df1fefce5bd3c0fcb3d9c153aa3bac4869f7b78e108e5448381181cb3965ebcc1025dd0768beb8
dc7eb60b960a44c455f154ad4e9f90d8655c5176f2ee12b6937bdbbc8f1e399c76fc36bdf7857533
780fc0e289a789a92630d47d22d0f636d939f989289cb38b202b099fa7261c6d981611175c6a9f2d
27dc4924a464e8b1b6a42f0d82c91e6ea4f6ab65ab6b18eb015c9344cc40b90e9c5a65161cc76bcd
5355682f09627e67434f4b87d958a9df6758ed7886f1c0d1a85311672a3e1b0711f3decedc311e49
833acedf156678d21fbaa71640b29d6f996392a4e

Decrypt:
Cipher: 0x694d138467e7e40a594b83595e1d62dbe6d1ed0858342bf7e108e5fc2991b793e8cf97
72df1fefce5bd3c0fcb3d9c153aa3bac4869f7b78e108e5448381181cb3965ebcc1025dd0768beb8
dc7eb60b960a44c455f154ad4e9f90d8655c5176f2ee12b6937bdbbc8f1e399c76fc36bdf7857533
780fc0e289a789a92630d47d22d0f636d939f989289cb38b202b099fa7261c6d981611175c6a9f2d
27dc4924a464e8b1b6a42f0d82c91e6ea4f6ab65ab6b18eb015c9344cc40b90e9c5a65161cc76bcd
5355682f09627e67434f4b87d958a9df6758ed7886f1c0d1a85311672a3e1b0711f3decedc311e49
833acedf156678d21fbaa71640b29d6f996392a4e
Message: 0x49610de9
→  practical_work_2 git:(master) ✗ gcc 02_verify.c -lgmp -o 02_verify
→  practical_work_2 git:(master) ✗ ./02_verify test-data.txt key_crt.txt
MD5: aa493ce43fdb660bf11b111cbd56dca7
MD5 Signature: 1634c0eb3e8b5190a4943fc407bde3333bd4aed6905a27ffd8424099070688ace
702472aa7589fd03b45c5f040313e4d54a6ffe945b8aca263e7a6fa653023f74fa43c8ee532ba40e
6d76e579bb957ebd1a78a08f110b81a8daee620af54bf1d7077bf24bc4c44d40c0156a6350404d6d
a993c8837e63d98228d42c402ed98433370e8c4cdc0fcb784d3a374cda64c1d12fb8dafd4620bf10
08b8207af64e7f0f149a55e9d937e38be866817d4ddfab43a82b975d2e4a584aa3106c8794df41ee
ca509962f6e076353b20d0032a16b53f21f1e04c4600d5c85509f2188185befda239e0b9ecd1fc8b
95354b906cd29adba5d0939a559c645501627d3c5b485f7
```

Figure 4: The commands of RSA ecosystem on CRT mode