

# Final Report

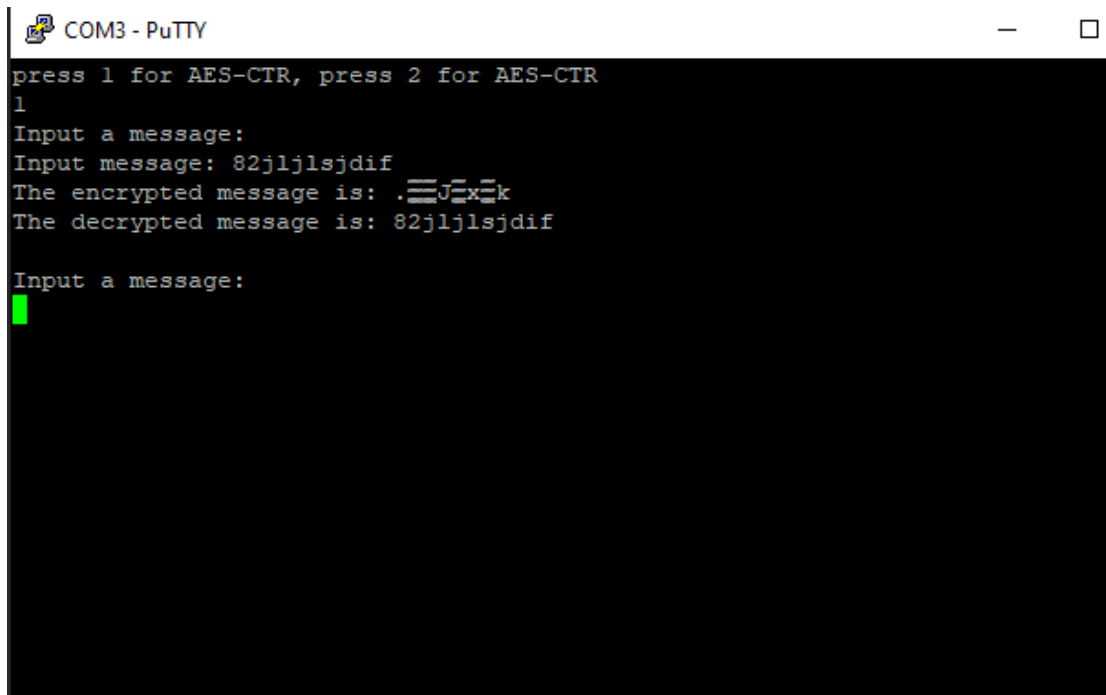
## Task 1: Encryption and Decryption with AES-CTR

- High Level Description
  - Use the True Random Number Generator to generate a 256-bit. After receiving the message, encrypt and decrypt it with AES-CTR. The original plain text, the cipher text, and the decrypted text should be shown. Also, the program should keep accepting new messages without system reset.
  - Flowchart in appendix.
- Low Level Description
  - 
  - First, to properly generate a random key. We use the True Random Number Generator.

```
for (int i =0; i<length; i++){  
    if ((i+1)%32 == 1){  
        rngNumber = HAL_RNG_GetRandomNumber(&rng);  
    }  
    secret_key[i] = rngNumber % 2;  
    rngNumber = rngNumber / 2;  
}
```

 is used to set the length of the key.
  - Then, we read the message and get the length of the message by using `lengths = uart_getline(&USB_UART,input,70);`.
  - To apply the key for encryption, `mbdctl_aes_setkey_enc(&ctx, secret_key, 256);` is used. If it did not generate the key successfully, it will `printf("Generate Key Failure!\r\n");`.
  - The plaintext will be encrypted by `mbdctl_aes_encrypt_ctr(&ctx, lengths, &nc_off, nonce_counter, stream_block, &input, &output);`.
  - For the decryption, `mbdctl_aes_encrypt_ctr(&ctx, lengths, &nc_off, nonce_counter, stream_block, &output, &backoutput);` is used. If the message is decrypted successfully, it will be shown by `printf("Decrypt the Plaintext Failure\r\n");`.
  - Finally, to clear all the arrays and the messages. We will use
  - ```
memset(input, 0, sizeof(input));  
memset(output, 0, sizeof(output));  
memset(backoutput, 0, sizeof(backoutput));  
memset(nonce_counter, 0, sizeof(nonce_counter));  
memset(stream_block, 0, sizeof(stream_block));  
nc_off = 0;  
printf("\r\n");
```
- Results and Analysis

- The program worked just as described in the Lab instruction. When we start the program, it will automatically ask for the message. And when the user typed in the message, it will then encrypt the message and show both the plain text and the ciphertext. After that, it will decrypt the message and then print out the recovered plaintext.
- What's new in this lab is how to use AES-CTR to encrypt and decrypt the messages. Also, how to make sure the program can work properly even when the length of the message is not evenly divisible by 16.
- Here is the result:



```
COM3 - PuTTY
press 1 for AES-CTR, press 2 for AES-CTR
1
Input a message:
Input message: 82jljlsjdif
The encrypted message is: .≡J≡x≡k
The decrypted message is: 82jljlsjdif
Input a message:
█
```

Figure 1: result of task1 AES-CTR

## Task 2: Password-Based key Derivation and AES-GCM

- (a) *Introduction and High-Level Description*
  - The purpose of this task is encrypt a message using a password and AES-GCM with an authentication tag.
  - With a correct password, the DISCO board can decrypt the message from encrypt data. If the password is incorrect, it will not decrypt the correct message.
  - The key of this task is get from the password. Using PBKDF2 with a hash function.

Flowchart in the appendix

- (b) *Low Level Description*
  - The password which student input need to be processed so that it can be used as a key for encryption. The method to get a key in this task is to use PBKDF2 to derive a 256-bit key with a random salt. The salt will decide what output we get. Here student create a 128-bit random salt.
  - A SHA-256 hashing function is used here to get the random key. The hash process can be done multiple times. Students here hash it for 5 times.

```
correct =  
mbedtls_pkcs5_pbkdf2_hmac(&ctx1,input,lengths,hash_key,128,5,256,midkey);
```

- The PBKDF2 function need to be initialized. The code is:

```
extern const mbedtls_md_info_t mbedtls_sha256_info;
```

```
int resultCode = mbedtls_md_setup(&ctx1, &mbedtls_sha256_info, 1);
```

- After the key is generated, it is set as a key which can be used by AES-GCM.

```
mbedtls_gcm_setkey(&ctx_gcm, MBEDTLS_CIPHER_ID_AES, midkey, 256)
```

- The program will read the message from USART 1, putty from the computer, the encrypt of the message is:

```
mbedtls_gcm_crypt_and_tag(&ctx_gcm, MBEDTLS_GCM_ENCRYPT, lengths, iv, 12, addition, 4, inputMessage, messageEc, 16, tagBuf)
```

- Here **iv** should be a random data 12 bytes long data buffer. However, student set a fixed data buffer, which may cause some concern in consider of safety.

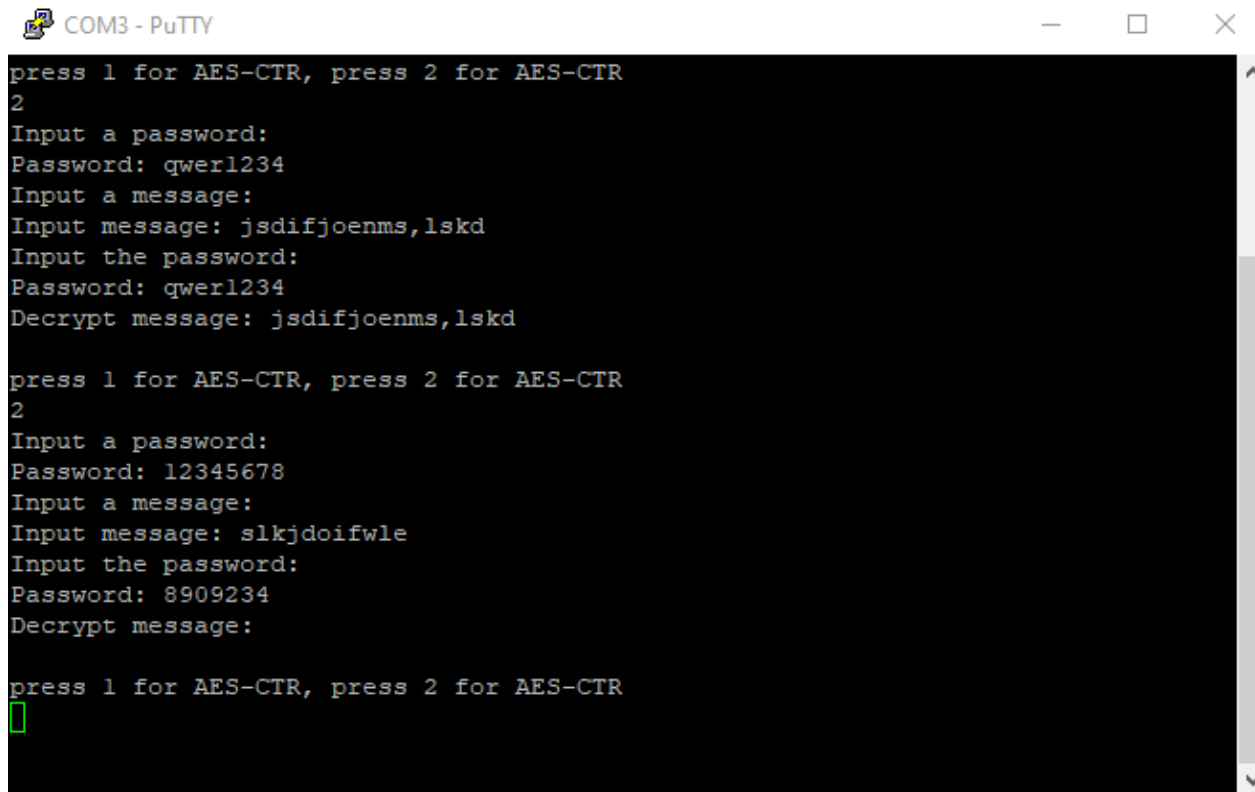
**Addition** data buffer is for the increment of processing data. Students make the **tagBuf** to be empty, this will make the encrypt function work.

- Then student will request the password again. Use the same procedure to generate the key. And then the program will decrypt the encrypted message use this key. If the password is correct, the message will be decrypted correct, otherwise it will not be decrypted.

```
mbedtls_gcm_auth_decrypt(&ctx_gcm, messageLen, iv, 12, addition, 4, tagBuf, 16, messageEc, backoutput);
```

- (c) *Results and Analysis:*
  - The program worked as expected

- As talked to TA, the iv in this part should be random generated 12-byte data buffer. Student set a fixed data buffer, which may course some issue about safety.



```
COM3 - PuTTY
press 1 for AES-CTR, press 2 for AES-CTR
2
Input a password:
Password: qwer1234
Input a message:
Input message: jsdifjoenms,lskd
Input the password:
Password: qwer1234
Decrypt message: jsdifjoenms,lskd

press 1 for AES-CTR, press 2 for AES-CTR
2
Input a password:
Password: 12345678
Input a message:
Input message: slkjdoifwle
Input the password:
Password: 8909234
Decrypt message:

press 1 for AES-CTR, press 2 for AES-CTR
█
```

Figure 2: Task 2 AEC-GCM result

## Task 3: Public Key Exchange

- High Level Description
  - Use Curve 25519 to generate an ECDH keypair.
  - Transmit the public key to the Secure Communication Peer via USART6.
  - The salt used with HKDF is the XOR of the salts provided in the public key messages.
- Low Level Description
  - We use

```
privateKey [32]byte
for i := range privateKey[:] {
    privateKey[i] = byte(rand.Intn(256))
} and curve25519.ScalarBaseMult(&publicKey, &privateKey);
```

 to generate the public key and the private key. Use `printf("Private key is %x\r\r", privateKey);` and `printf("Public key is %x\r\n", pubickey);` to print out both private key and public key in the terminal.
  - Using the same process, we can generate the private key and the public key for the second user.
  - To generate the shared key, we can use `curve25519.ScalarMult(&out1, &privateKey, &publicKey2);` and `curve25519.ScalarMult(&out2, &privateKey2, &publicKey);`. The product of these two functions should be the same.

## Conclusion

In this lab, student learned how to use Advanced Encryption Standard to encrypt message and decrypt the message. This encrypt operation involves the probability, logic to make the message encrypted and cannot be read.

To make the message not be read by hackers, it is necessary to generate a key to encrypt the message, and the key cannot to be tried out. The first task use a random generated key. In the second task, hash function will make the password to become a key. This is a more safe option because the repeat hashing can prevent brute method to tryout the key.

The application of the AES can keep the message transfer between microprocessor safe. The data is more difficult to be stolen. The protection of data has been used in many data transfer procedure today.

## Appendix

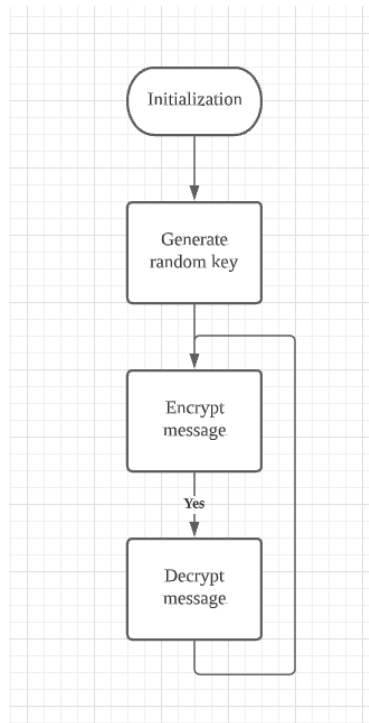


Figure 3: Task 1 flowchart



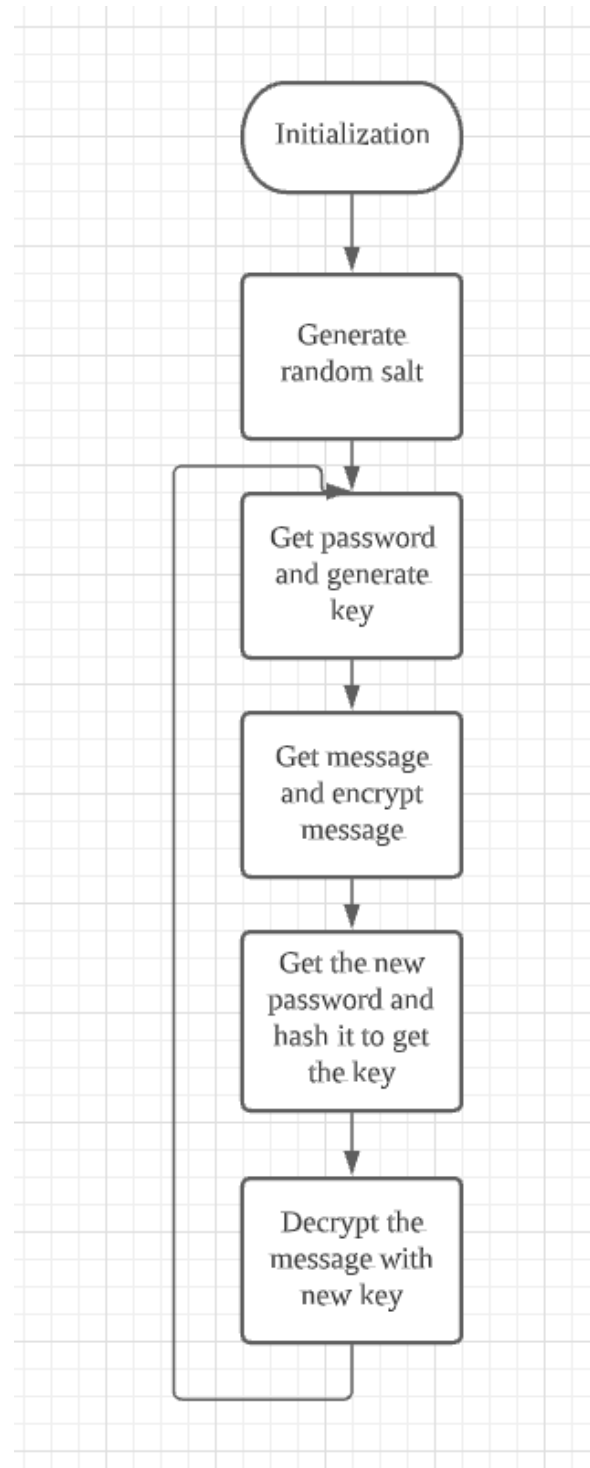


Figure 4 : Task 2 flowchart