<u>Week 9 Homework due on Friday Nov 9, 22:00 Hour</u>

<u>Group 5</u>

Wong Ann Yi (1004000)
Liu Bowen (1004028)
Tan Chin Leong Leonard (1004041)

<u>Exercise 1</u>

Suppose Alice and Bob are communicating using the secure channel described in FSK's Chapter 7. Eve is eavesdropping on the communications. What types of traffic analysis information could Eve learn by eavesdropping on the encrypted channel? Describe a situation in which information exposure via traffic analysis is a serious privacy problem.

Answers:

The types of traffic analysis information Eve could learn by eavesdropping are: timing and size of the messages, packet headers (which are not encrypted), who is communicating with whom, how much and when. In particular Eve can even insert, delete and modify the data. This eavesdropping can take place even in secure channels such as SSL/TLS, IPsec and SSH.

Traffic analysis is a serious privacy problem. For example, if an attacker wants to know a certain important person or VIP and to find out who he contacts with and what websites he visits. The attacker can over time form a communication and behavior pattern of the VIP person.

In another example, in a shared-medium nature such as in WIFI communication, it poses a great challenge on user privacy. Recent privacy preserving in WIFI networks has mainly focused on location privacy and user identification. From publicly available databases of WIFI networks, it is feasible to track users' location through the analysis of the log files. Furthermore, adversaries may adopt wireless signal strength in multiple monitoring locations to obtain an accurate estimation of a user's location and motion. (Extracted from http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1212&context=electricalengineeringfacpub).

<u>Exercise 2</u>

Compare the advantages and disadvantages among the different orders of applying encryption and authentication when creating a secure channel.

Answers:

There are three types of encryption and authentication. They are: encrypt-and-authenticate, authenticate-then-encrypt and encrypt-then-authenticate.

From the text book Cryptography Engineering, it is argued that theoretical results show that encrypt-first solution is more secure compared to the other methods. It is also more efficient as bogus messages will be discarded without being decrypted which will consumer CPU resources.

The text book also has a counter argument in favour of an authenticate first method. It is argued that in this method, an attacker only sees the ciphertext and the encrypted MAC value. The MAC input and MAC tag are hidden and making it difficult for the attack on the MAC function. The other argument in favor is that this method will not violate the Horton Principle as it will not authenticate the cipher text which will break the principle.

The encrypt-and-authenticate performs both encryption and authentication in parallel. It then transmits the ciphertext and the tag. The advantages are: it is efficient as both encryption and authentication can be performed in parallel. The disadvantages are: it is theoretically insecure, and the attacker can get more information of the tag of the initial message itself and can lead to privacy leak. Because MAC protects authenticity and not privacy, it leaks private information about the underlying message thereby compromising the privacy of the secure channel.

The authenticate-then-encrypt method authenticates a message then encrypts both the message and the tag. It then transmits the ciphertext. This method has the following advantages: the attacker cannot see the tag. The disadvantages are: it is not efficient because the receiver has to decrypt the message (even if it is bogus) first before he can check the authentication.

The encrypt-then-authenticate method encrypts a message and authenticates the ciphertext. It then transmits the ciphertext and the tag. This method has the following advantages: it is theoretically secure and more efficient because the receiver will never decrypt a bogus message thereby saving resources. The method also helps to make denial-of-service (DOS) attack difficult. The disadvantages are: the attacker can gain access to the valid (message, authtag) pairs. This method also violates the Horton Principle of “authenticate what it meant, not what is said). In other words, if a cipher text sent by the sender is modified by an attacker, it may be authenticated with a valid tag value and be accepted by the receiver.

Exercise 3

For your platform, language, and crypto library of choice, implement authenticated encryption in GCM (Galois/Counter Mode).

• K = 128-bit 1

• P = SUTD-MSSD-51.505*Foundations-CS*SUTD-MSSD-51.505

• IV = 128-bit 0

Answers:

We developed the codes in Python 2.7 and install a package “Cryptodome” to perform the authenticated encryption function of GCM.

The codes are as follows:

^^

```
from Cryptodome.Cipher import AES

IV_1 = '0000000000000000'

key_1 = '1111111111111111'

encryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

ciphertext, tag = encryptor.encrypt_and_digest('SUTD-MSSD-51.505*Foundations-CS*SUTD-MSSD-51.505')

print binascii.hexlify(ciphertext), binascii.hexlify(tag)
```

^^

The output is:

The ciphertext is: “16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b0f3578c16aecd9e4bbb0d8f52f0e4f0f”

The tag is: “68e4478b894f5535242216fd8983adea”.

a) What are the ciphertext C and authentication tag T?

   The ciphertext is: “16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b0f3578c16aecd9e4bbb0d8f52f0e4f0f”

   The tag is: “68e4478b894f5535242216fd8983adea”.

b) Swap the the first and third blocks of C, what is the outcome of tag verification and decryption? (For decryption, ignore tag verification).

We swap the first and third block and the resulting ciphertext is:

“0f3578c16aecd9e4bbb0d8f52f0e4f0fce8aaa8490d12a99e538f499eabab47b16042f8a8df1c09dee68e56a1a1d9157”

We then developed the below codes:

^^

decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

print decryptor.decrypt_and_verify(binascii.unhexlify('0f3578c16aecd9e4bbb0d8f52f0e4f0fce8aaa8490d12a99e538f499eabab47b16042f8a8df1c09dee68e56a1a1d9157'), tag)

^^

The output is: ValueError: “MAC check failed”. The GCM performed the tag verification first and then the decryption. The change in ciphertext will result in the failure in the authentication operation as the “integrity” of the ciphertext is breached.

If we now change the decrypt_and_verify() function into decrypt() function, we can obtain the decrypted plain text. The codes and output are below:

^^^

decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

print decryptor.decrypt(binascii.unhexlify('0f3578c16aecd9e4bbb0d8f52f0e4f0fce8aaa8490d12a99e538f499eabab47b16042f8a8df1c09dee68e56a1a1d9157'))

^^^

The output is shown below:

```
Jd �PJ*� &�m*Foundations-CS*Jd �PJ*� &�m
```

c) Remove the third block of C, what is the outcome of tag verification and decryption? (For decryption, ignore tag verification)

After we delete the last block, we got the ciphertext is:
“16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b”.

The codes are:

^^

decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

print
decryptor.decrypt_and_verify(binascii.unhexlify('16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b'), tag)

^^^

The result is still “MAC check failed”. The reason is the same as in part b).

We then changed the decrypt_and_verify() into decrypt() and the results are as follows:

^^^

decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

print
decryptor.decrypt(binascii.unhexlify('16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b'))

^^^

The result and output is:

```
SUTD-MSSD-51.505*Foundations-CS*
```

d) Change the last bit of C, what is the outcome of tag verification and decryption? (For decryption, ignore tag verification).

   We then changed the last bit of block “f” into “e”. We then run the below codes:

   ^^
   decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)
   print
   decryptor.decrypt_and_verify(binascii.unhexlify('16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b0f3578c16aecd9e4bbb0d8f52f0e4f0e'), tag)

   ^^

   The result is still “MAC check failed”. The reason is similar to part b) and c).

   We then changed the decrypt_and_verify() into decrypt() and the result is:

^^^

decryptor = AES.new(key_1, AES.MODE_GCM, IV_1)

print
decryptor.decrypt(binascii.unhexlify('16042f8a8df1c09dee68e56a1a1d9157ce8aaa8490d12a99e538f499eabab47b0f3578c16aecd9e4bbb0d8f52f0e4f0e'))

^^^

The output and result is:

```
SUTD-MSSD-51.505*Foundations-CS*SUTD-MSSD-51.504
```

The last part of the plaintext is changed from “51.505” into “51.504”.

e) Discuss the security features of GCM based on the results of b) - d).

Based on the b)- d), we know that the authentication tag in the GCM mode is bound to the ciphertext. When a receiver obtained the messages, he would first authenticate the MAC tag value before decrypting the ciphertext. Only when the tag verification is successful, then the decryption function will proceed. If tag verification failed, the decryption process will not be performed.

When we alter the ciphertext in b) c) and d), the integrity of the ciphertext is breached and therefore the tag verification cannot be passed. For just decrypt() process, we decrypt the ciphertext and no tag verification and we obtained the modified plaintext. AES GCM mode encrypts the message one block at a time and if we swap first and third block we cannot get the correct plaintext in the correct order. Lastly, if we change the last bit of the ciphertext, we will notice the final message is also not correct (last portion is changed from “505” into “504”). Any change in ciphertext will cause the wrong decryption results.