Week 8 Homework due on November 9, 18:59 Hour

Group 5

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

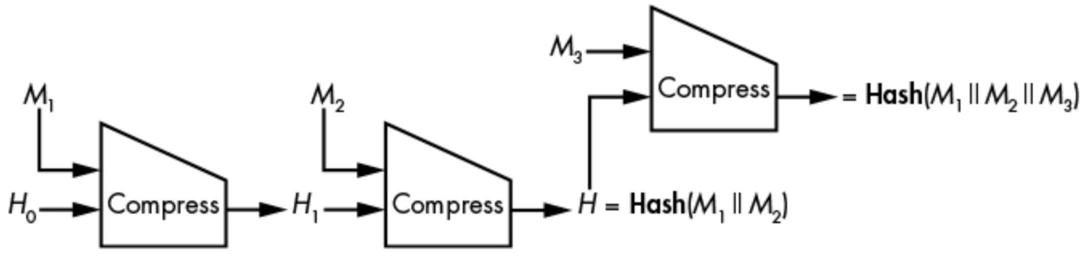
Exercise 1

Design a new mechanism to defend the length extension attack against MD-based hash functions.

Answers:

The length-extension attack is a threat to the Merkle–Damgård (MD) based hash functions. construction. Basically, if we know Hash(M) for a message, M which is composed of blocks M_1 and M_2 (after padding), we can determine Hash($M_1 \| M_2 \| M_3$) for any block, M_3. Because the hash of $M_1 \| M_2$ is the chaining value that follows immediately after M_2, we can add another block, M_3, to the hashed message, even though we do not know the data that was hashed. This threat generalizes to any number of blocks in the unknown message ($M_1 \| M_2 \|$ here) or in the suffix (M_3) as shown in the below figure.

1) H(M3, H(M1llM2) != H(M3llM1llM2) if H is not xor operation. Your proposal of HMAC is not a new one.



One mechanism which can resolve this threat is to make the last compression function call different from the previous ones. This can be done by adding an extra parameter to each compression function. We can differentiate the last function call by taking a 1 bit value while the previous function calls take a 0 bit. This serves as a flag to indicate the processing of the last block of the message.

The lecture notes have introduced two fixes: using the interactive hash computations and truncate the output by using the first n-s bits as the has value. These will not be discussed here as we are instructed to use techniques outside of the lecture notes.

The other mechanism to defend against MD-based hash functions is to use HMAC together with MD based hash functions, MD5. By using the HMAC which act as a shared secret key which is known to both sender and receiver will protect the integrity of the message against length extension attack.

The sender will hash the message and the secret key to form a HMAC. The HMAC and the message is sent to the receiver. The receiver will use its own secret key to calculate its own HMAC via the received message and compare the two hash values so if it is equal then the message is not tampered.

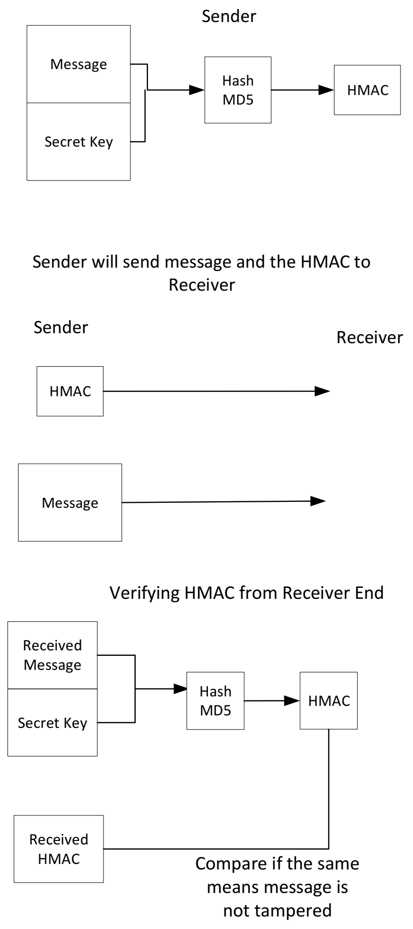
<u>HMAC Algorithm</u>

$\text{HMAC}\,[K,m] = H\,[\,K \oplus 0x5C5C\ldots \,\|\, H\,[\,K \oplus 0x3636.. \,\|m]]$

Where K is a secret key, m is the message to be hashed

H is a hash function (e.g. MD5)

Below is an illustration of HMAC


Sender
Message
Secret Key
Hash MD5
HMAC
Sender will send message and the HMAC to Receiver
Sender
Receiver
HMAC
Message
Verifying HMAC from Receiver End
Received Message
Secret Key
Hash MD5
HMAC
Received HMAC
Compare if the same means message is not tampered

<u>Exercise 2</u>

Find two messages that produce the same tag for AES-based CBC-MAC. Show code to demonstrate that.

Answers:

To construct the two messages with the same tag for CBC-MAC, we consider a scenario when given message a and its CBC-MAC tag $MAC_k(a)$. We can construct message $a \| b$ where $b = a \oplus MAC_k(a)$. After that, we can get the same CBC-MAC tag output $MAC_k(a \| b) = MAC_k(a)$.

Explain the reason: $MAC_k(a \| b) = Enc(key, b \oplus MACk(a)) = Enc(key, a \oplus MACk(a) \oplus MACk(a)) = MACk(a)$

The demonstration code is as below: (in Python 2.7)

^^

```
from Crypto.Cipher import AES

import hashlib

import hmac

key = ('00000000000000000000000000000000').decode('hex')

IV = ('00000000000000000000000000000000').decode('hex')

msg1 = ('a0a1a2a3a4a5a6a7a8a9aaabacadaeaf').decode('hex')

cipher1 = AES.new(key, AES.MODE_CBC,IV)

tag1 = cipher1.encrypt(msg1).encode('hex')

print tag1

msg1_to_int = int("a0a1a2a3a4a5a6a7a8a9aaabacadaeaf", 16)

tag1_to_int = int(tag1, 16)

msg2 = format((msg1_to_int ^ tag1_to_int), 'x').decode('hex')

cipher2 = AES.new(key, AES.MODE_CBC,IV)

tag2 = cipher2.encrypt(msg1+msg2).encode('hex')[32:64]

print tag2
```

^^

The output is: `11d4d0fb8b52063651ac08f1a593e3fa` for tag1 and `11d4d0fb8b52063651ac08f1a593e3fa` for tag 2. Both tags are of the same value.

At the beginning, we initialize the AES CBC mode with 128-bit-zero key and 128-bit-zero IV. For this case the message *a* is msg1 with the value of "'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf". For the CBC-MAC tag of the msg1, we can get the tag1 equals to "11d4d0fb8b52063651ac08f1a593e3fa".

After we get msg1 and MAC_k(msg1), we construct new message that equals msg1 || msg2 where msg2 = msg1 ⊕ MAC_k(msg1). As the new message has two block size, the CBC-MAC tag for new message is the second half of ciphertext. The running result is also "11d4d0fb8b52063651ac08f1a593e3fa" which means that the MAC_k(msg1) = MAC_k(msg1|| msg2).

Exercise 3

Let's assume that CBC-MAC is used as a MAC scheme. Suppose c is one block long, a and b are strings that are a multiple of the block length, and MAC_K(a||c) = MAC_K(b||c). Then MAC_K(a||d) = MAC_K(b||d) for any block d. Explain why this claim is true.

Answers:

From the definition, we obtain the equation MAC_k(a || c) = Enc(key, c ⊕ MAC_k(a)) and MAC_k(b || c) = Enc(key, c ⊕ MAC_k(b)).

As MAC_k(a || c) = MAC_k(b || c) and c is one block size, we can get the equation MAC_k(a) = MAC_k(b).

Therefore, we can get MAC_k(a || d) = Enc(key, d ⊕ MAC_k(a)) = Enc(key, d ⊕ MAC_k(b)) = MAC_k(b || d) for any block d. An attacker then gets the sender to authenticate a || d and he can replace message with b || d without changing the MAC value.

Exercise 4

Suppose message *a* is one block long. Suppose that an attacker has received the MAC t for *a* using CBC-MAC under some random key unknown to the attacker. Explain how to forge the MAC for a two block message of your choice. What is the two-block message that you chose? What is the tag that you chose? Why is your chosen tag a valid tag for your two-block message?

Answers:

A CBC-MAC is computed as follows:

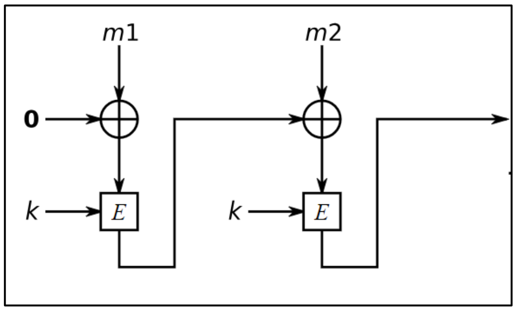
H_0 := IV (which is typically fixed at 0)
$H_i := E_K(P_i \oplus H_{i-1})$
MAC := H_k

In this case, the message *a* is only 1 block. Therefore, the MAC or M(a) = t = $H_1 := E_K(a \oplus H_0) = E_K(a \oplus 0)$.

Suppose we construct a 2 block message comprising of a || (a ⊕ t), the resulting MAC will be M(a) which is t. It will work for subsequent block(s) as long as they are (a ⊕ t). CBC-MAC operates as shown in the below figure. Assuming m1 = a and m2 = (a ⊕ t) or (a ⊕ M(a))

In the first step, we encrypt *a* ⊕ 0 = a with the block cipher E and key k and obtain M(a) or t.
In the second step, we compute M(a) XOR m2 = (a ⊕ t) or (a ⊕ M(a))
Therefore, m2 ⊕ M(a) = a ⊕ M(a) ⊕ M(a) = a ⊕ 0 = a
Therefore, at the end of step 2, the encryption of a E(a) will result in M(a) or t again.

In summary, the 2 block message is made up of *a || (a ⊕ t)*. The valid tag of the message is t.



We have also developed an alternative solution as follows.

We use the birthday paradox to find another message block b which has the same MAC value as message a. Therefore, M(a) = M(b) = t. Referring to the above figure, we assume a 2 block message with m1 and m2. We also assume m1 = a and m2 = (b ⊕ t).

In the first step, we encrypt $a \oplus 0 = a$ with the block cipher E and key k and obtain M(a) or t. In the second step, we compute M(a) XOR with m2 = $(t \oplus b \oplus t) = b$. The encryption of b will also yield t. This is because M(a) = M(b).

Therefore, the 2 block message can be *a || (b ⊕ t)* where M(a) = M (b) = t and the valid tag is t.