UM-SJTU Joint Institute
Project One
Message Authentication Code
(Ve475)

Group 7
Message Authentication Code

Ma Siyin **517370910003**
Ming Xingyu **517370910224**
Zhang Liqin **517370910123**
Zhao Zhijie **517370910035**

Date: from 2020/5/30 to 2019/6/22

Abstract

It's quite exciting to start our self-investigation in the world of cryptography with the topic of Message Authentication Code (MAC). Before all, we need to know what is MAC. It is a designed method to verify whether the message comes from a stated sender. It also protects the message's data integrity and authenticity [7], while the former one means the message is not lost or altered during the transformation while the latter one means the message is from the write place written by the right people.

For this project, we are following the structure of "Cryptography and Network Security: Principle and Practics" [1]. We explain the theory, classification, security, and application of the MAC. In Message Authentication, we investigate what message authentication is and why we need that. We further dig into the concept of MAC and understand the basic method and ways to verify a message. Later we learn about two types of MAC, HMAC, and CMAC. Authenticated encryption is also deeply discussed in the third part. Finally, we consider the security and the application of the MAC. The former includes situations under different types of attacks while the latter includes the pseudo-random number generator and application in Transport Layer Security protocol.

Message Authentication Code is complicated, but based on the knowledge we learned in the lecture and our hard work, we gradually understand the concept of it. We still need to work hard to study in the field of Cryptography.

Contents

1 Message Authentication

1.1 WHY Message Authentication

Before diving into the world of Message Authentication Code (MAC), we need to be clear why MAC exists and what is MAC capable of. One primary goal of cryptography is to secure the communication and ensure the information integrity. Here, the 'secured communication' can be established by encrypting the message content using several ciphers. However, in most cases, it's very important for the receiver/sender to guarantee the message integrity (or message authentication) in the sense that they should be able to identify when a message it receives was sent by the party claiming to send it, and was not modified in transit. The flow chart shown below illustrates the broad idea to measure the integrity of communication.

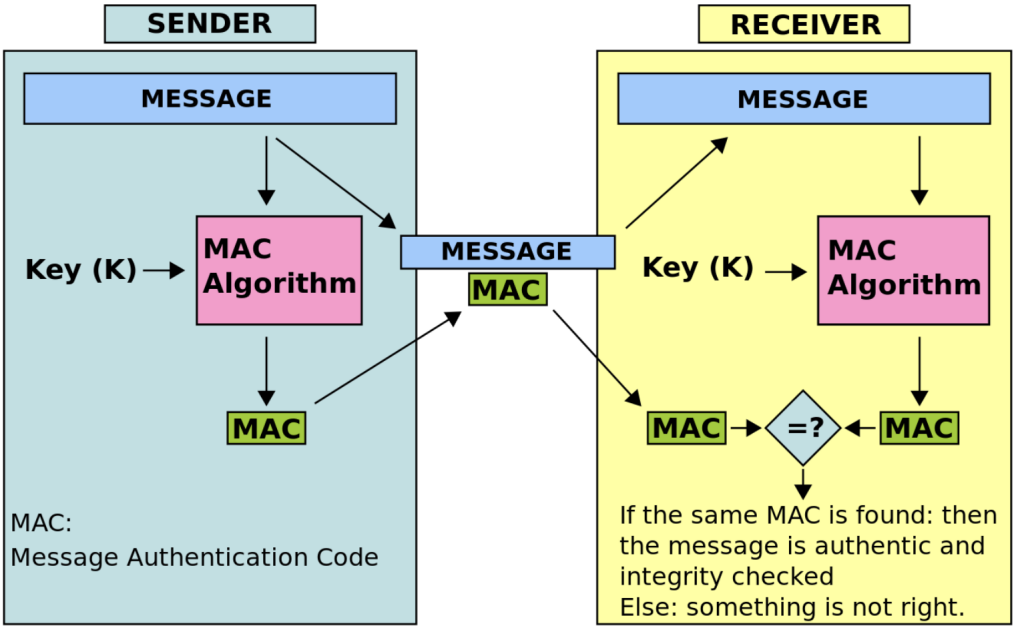

Figure 1: General Process of Applying MAC

Note that encryption is different from message integrity, without which could result in severe consequences, such as currency trading. We will introduce an additional mechanism to enable both sides of communication to know whether or not a message has been modified during the transition, and either of whom can verify the source of the message. MAC is usually used as a method to check for the information integrity based on a secret key shared by both sides of the communication channel and to authenticate further message travelled between them. Here, we only consider the private key setting, which means that the same secret key will be used for both sides.

1.2 Message Authentication Functions

As we mentioned above, one important functionality of the message authentication system is to generate an authenticator, which can assess whether a message is authenticated by providing a score, this allows the receiver to verify the authenticity later on. Here are some example functions that can serve as authenticator generators:

1. **Hash function**: An injective function that maps any message with all length into a fixed-length hash value.
2. **Message encryption**: The encrypted message, or the ciphertext can be used as an authenticator.
3. **MAC**: A function that takes the message and some secret key as input, and output a fixed-length value.

1.3 Message Authentication Requirements

The message authentication is generated to identify the following attacks during the process of network communication.

1. **Camouflage**: Insertion of messages from fraud sources in the creation or receipt of messages.
2. **Content modification**: Modification on the message contents.
3. **Sequence modification**: Changes like reordering, insertion and deletion to a set of messages.
4. **Timing modification**: Modification based on the timeline, like delaying or replaying some of messages during the conversation.

In conclusion, we apply message authentication to verify if the message is sent from the declared party and the content has not been modified in any ways.

2 Message Authentication Codes

2.1 Definition of MAC

As a message authentication function we introduced above, MAC takes any message with arbitrary length a secret key as input and produces an fixed length authentication code as output. The person/party who gets the authentication code has the ability to examine the integrity of the message. Here is a former definition of MAC from [1].

Definition 1 *A message authentication code (or* MAC*) consists of three probabilistic polynomial-time algorithms* (Gen, Mac, Vrfy) *such that:*

1. *The key-generation algorithm* Gen *takes as input the security parameter 1n and outputs a key k with* $|k| \geq n$.

2. *The tag-generation algorithm* Mac *takes as input a key k and a message* $m \in \{0,1\}$*, and outputs a tag t. Since this algorithm may be randomized, we write this as* $t \longleftarrow \text{Mac}_k(m)$.

3. *The deterministic verification algorithm* Vrfy *takes as input a key k, a message m, and a tag t. It outputs a bit b, with b = 1 meaning valid and b = 0 meaning invalid. We write this as* $b := \text{Vrfy}_k(m,t)$.

Note that for every positive integer n, and every code k given by $\text{Gen}(1^n)$*, and every* $m \in \{0,1\}$*, it always holds true that* $\text{Vrfy}_k(m, \text{Mac}_k(m)) = 1$. *This is the goal of the message authentication.*

2.2 Message Authentication Codes Requirements

To access the security of the MAC function, we have to set requirements to defend the attacks. Assuming an attacker knows the MAC function, but he has no knowledge of K, then the MAC function generated in the form of $T = \text{MAC}(K, M)$ should meet the following three requirements.[1]

1. If M and $\text{MAC}(K, M)$ are known to an attacker, a new M' that satisfies the following equation should be hard to construct.

$$\text{MAC}\,(K, M') = \text{MAC}(K, M)$$

2. $\text{MAC}(K, M)$ should be uniformly distributed, where the probability of two randomly selected messages are identical with a probability of 2^{-n}, where n represents the number of bits in a tag.

3. Let $M' = \text{f}(M)$, as a transformation on M. The following requirement should be met.

$$\Pr\left[\text{MAC}(K, M) = \text{MAC}\,(K, M')\right] = 2^{-n}$$

For the first requirement, it prevents people from constructing a new message instead of figuring out the original one to supply for a given tag.
The second one defend the brute-force attack, which is also a chosen plaintext attack. It protect the MAC function from continuously attempts of various M to match a given tag without knowing K. Here the attack is $O(2^{n-1})$.
Finally, the authentication algorithm should make sure that it is infeasible for an opponent to create a new message M' based on the known old M to match the old tag.[1]

3 MAC Classification

There are two ways of constructing a MAC, one way is to establish a hash table with secret key, another is to use a symmetric block cipher such that it will produce a fixed length output. We will cover both two types of MAC in this section.

3.1 HMAC - MAC using Hash Table

3.1.1 Introduction

HMAC, Keyed-hash message authentication code, defines a MAC that apply a hash function with a secret key. The main goal behind the construction are[4]

1. We can apply the hash function without modification, which is convenient for construction when there already exist many hash functions used in software.
2. We can maintain the efficiency of the original hash function.
3. We can obtain a cryptographic analysis of the strength of the hash-based authentication mechanism.
4. We can simply handle and apply keys.
5. We can replace the hash function with a safer or faster one if necessary.

3.1.2 Definition of HMAC

Symbols	Definitions
H	An embedded hash function.
B	Number of bytes in a block of the input of H.
L	Number of bytes in a lock of the output of H.
ipad	The byte x‘36’ (00110110) repeated B times.
K	The secrest key, and pad it with zeros on the left to achieve length 8B.
K_0	The K after hashing with B byte if necessary.
opad	The byte x‘5c’(01011100) repeated B times.
text	The input message of HMAC without the padded key.
	The length of text is n bits, where $0 < n < 2^B - 8B$
$\|$	Concatenation.
x ‘N’	Hexadecimal N.
$\oplus$	XOR operation.

Table 1: HMAC Parameters and Symbols

HMAC uses a key, K, with certain security strength, as specified in NIST Special Publication (SP) 800-107. When an application uses a K longer than B-bytes, then it shall first use H to hash K. Then we will get a L-byte string, which is used as the key K_0; details can

be found in Table 1.[5]
For other two conditions, we briefly explain why we do not choose the keys that are either longer or shorter than L. Keys less than L bytes would be not secure enough for the HMAC function. Keys' length greater than L is redundant to enhance the security of the funtion, since the bits beyond L don't have strong effect on the security of the function.[4]

If we have to calculate MAC with the input text, we should strictly follow the steps listed below. It includes all the condition of the key length with different relationships with size of the block B. The computation procedure is quite straightforward. We just need to construct the following equation and we will get the result of MAC.

$$MAC(text) = HMAC(K, text) = H\left((K_0 \oplus opad) \| H\left((K_0 \oplus ipad) \| text\right)\right)$$

The steps are depicted in figure 2

1. If $len(K) = B : K_0 = K$. Go to step 4.
2. If $len(K) > B$: Use hash function on K to get an L byte string, then append $(B - L)$ zeros on the left to create a B-byte string. Go to step 4.
3. If $len(K) < B$: append zeros to K and create a B byte string K_0
4. XOR K_0 with *ipad*.
5. Append *text* to the string in step 4
6. Apply the hash function to the stream in step 5
7. XOR K_0 with *opad*.
8. Append the result from step 6 and step 7.
9. Apply hash function to the stream in step 8.

3.2 DAA AND CMAC - MAC using Symmetric Block Ciphers

DAA and CMAC are two kind of MACs that are based on the use of block cipher. To begin with, we will introduce the Data Authentication Algorithm (DAA) which has a longer history and is now abandoned. After that, we will introduce CMAC, which is designed as an improvement of DAA.

3.2.1 Definition of DAA

DAA(Figure 3) is based on DES. However, its weakness in security makes it replaced by stronger algorithms. DAA algorithm basically uses the cipher block chaining of DES,

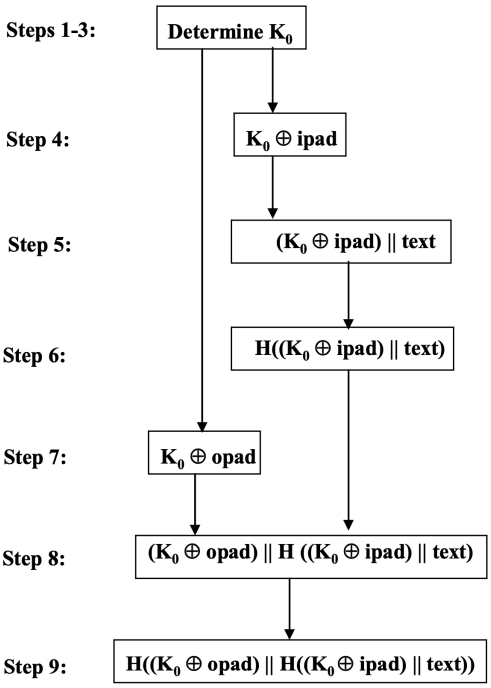

Figure 2: Illustration of the HMAC Construction

and it has an initialization vector (IV) of zeros. First, we can divide our message to be authenticated into many fixed 64-bit-sized blocks, let's denote them: D_1, D_2, ..., D_n. When the message length cannot be fully divided into 64, we will leave the final block filled with zeroes on the right and thus to fill a 64-bit block fully. Given DES encryption algorithm E and a secret key K, the DAC is calculated as

$$\begin{aligned} O_1 &= E(K, D_1) \\ O_2 &= E(K, D_2 \oplus O_1) \\ O_3 &= E(K, D_3 \oplus O_2) \\ &\vdots \\ O_n &= E(K, D_n \oplus O_{n-1}) \end{aligned}$$

According to the flow graph, we could obtain the DAC by reading 16 to 64 leftmost bits of the block O_N.

3.2.2 Definition of CMAC

As we've discussed above that DAA has one following limitations: the message to be authenticated only has a fix length bits are processed, this length can be expressed as the

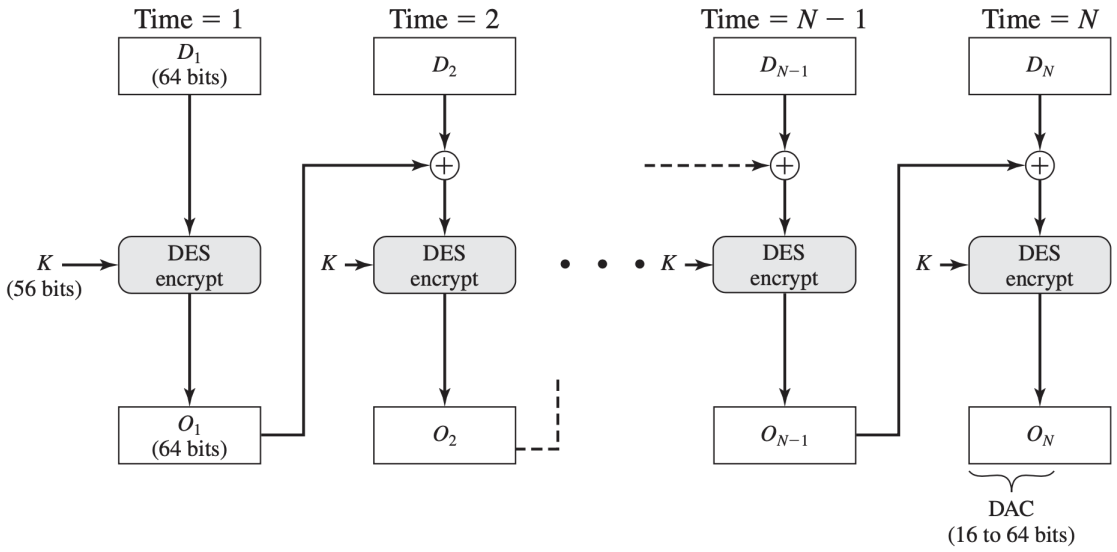

Figure 3: Data Authentication Algorithm

Symbols	Definitions
T	message authentication code, also referred to as the tag
Tlen	bit length of T
$MSB_s(X)$	the s leftmost bits of the bit string X

Table 2: CMAC Parameters and Symbols

product of an positive integer and the size of cipher block.

Black and Rogaway [2] demonstrated that this limitation could be overcome using three keys K_0, K_1 and K_2. One of which can be set length k, the same size as the encryption key, and to be used at each step of the cipher block chaining. The other two keys are set at size n, which is the size of each message block. This idea was further generalized by Iwata and Kurosawa in the way that the two keys of size n could be derived from the encryption key, rather than being provided separately [3]. What we have from the refine is called the Cipher-based Message Authentication Code (CMAC), which could use with AES and triple DES.

The CMAC algorithm has the following procedure. First, the initial setting to apply CMAC is when the message is an integer multiple n of the cipher block length b. And we set for AES, $b = 128$, and for triple DES, $b = 64$. As is with DMAC, the message is divided into n blocks, let's denote (M_1, M_2, …, M_n). The algorithm uses a k-bit encryption key K and an n-bit constant K_1. The key size k for AES can be 128, 192, or 256 bits. The key size for triple DES is 112 or 168 bits. And then we could calculate CMAC as

$$
\begin{aligned}
C_1 &= E(K, M_1) \\
C_2 &= E(K, M_2 \oplus C_1) \\
C_3 &= E(K, M_3 \oplus C_2) \\
&\vdots \\
C_n &= E(K, D_n \oplus C_{n-1} \oplus K_1) \\
T &= MSB_{Tlen}(C_n)
\end{aligned}
$$

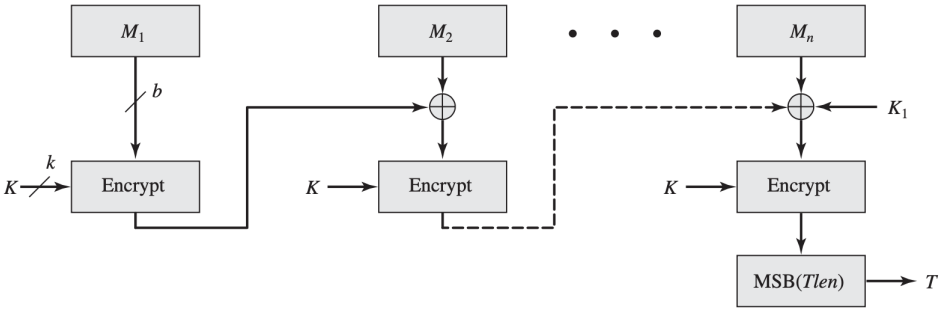

Figure 4: Message length is integer multiple of block size

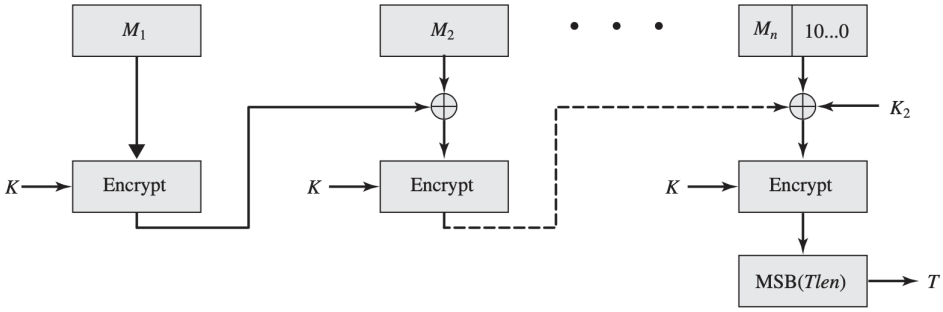

Figure 5: Message length is not integer multiple of block size

We notice that when the message cannot be fully divided by the length of a integer sized cipher block, the the final block is padded to the right with the first letter 1 and following by a series of 0s. Therefore, the final block can be increased to the length b. After this operation, the CMAC could proceeds as its algorithm, notice here the K_1 is replaced with another n-bit size key K_2, which is calculated as

$$L = E(K, 0^n)$$
$$K_1 = L \cdot x$$
$$K_2 = L \cdot x^2$$

The operation happens in the finite field $GF(2^n)$ and x and x^2 are first and second order polynomials which belongs to $GF(2^n)$.

3.3 Authenticated Encryption

Authenticated encryption (AE) is often used to describe encryption systems that protect both security and authenticity of the communication information. The importance of both properties has been covered in previous sections, and there are a lot of protocols that also requires both securities. Nowadays, there are mainly 4 techniques to providing these two properties for a message.

1. HtE (Hash then Encrypt): First compute $h = H(M)$ and then encrypt $W = E(K, (M \mid\mid h))$.
2. MtE (MAC then Encrypt): First compute $m = MAC(K_1, M)$ and then encrypt $W = E(K_2, (M \mid\mid m))$.
3. EtM (Encrypt then MAC): First encrypt $w = E(K_1, M)$ and then compute $W = MAC(K_2, w)$.
4. E&M (Encrypt and MAC):Encrypt $w = E(K_1, M)$ and compute $m = MAC(K_2, M)$ and use these two values to authenticate.

4 MAC Security

The attacks addressed on MACs can be categorized into two categories, brute-force attacks, and cryptanalysis. Between these two kinds of attacks, the brute-force attacks are generally used while there is far less work has been done on developing such attacks [1]. Hence, in this part we mainly focus on the brute-force attack.

4.1 Generalized Brute-force attack

First, we can state the expected security property of a MAC algorithm with K be the secret key and x be the message input:

- Computation resistance: Given one or more text-MAC pairs $[x_i, MAC(K, x_i)]$, it is computationally infeasible to compute any text-MAC pair $[x, MAC(K, x)]$ for any new input $x \neq x_i$[1].

More specifically, the attacker needs to compute a valid message-tag pair or find the key. That is to say the attacker has two approaches to do this: attack the key space or attack the MAC value.

4.1.1 Attack the key space

The key of the MAC algorithm is vital to the security and cannot be revealed. If the attacker can find the key, then the attacker is very much likely to generate a valid MAC code for any input message x.

Now, suppose the attacker has the information about one message-tag pair and the key size is k bits, which means the key space has 2^k possible keys. Then, to implement the attack, the attacker can compute all the possible tags corresponding to all the possible keys in the key space, we can have the pseudo-code of this algorithm as follows

```
Input:
  K:the key space x:the input message t:the tag correspond to x
  function KeySpaceAttack(K, x, t)
      keys ← {}
      for i = 1 to size(K) do
          k ← K[i]
          tag ← MAC[k, x]
          if tag == t then keys.append(k)
          end if
      end for
      return keys
  end function
```

Since the attacker goes through the entire key space, there must exist one right key that can provide the correct tag.

However, we know that the MAC is a many-to-one function. According to the *Pigeonhole Principle*, it is very likely to have several keys that can provide the correct tag. Hence, the keys need to be tested.

To summarize, the effort taken for this kind of attack is roughly 2^k, both in time and in space, which is positively related to the length of key.

4.1.2 Attack the MAC value

In this method, the attacker needs either to generate a valid tag for a chosen message or to find a message that matches a given tag without recovering the key [6]. This kind of attack is similar to attack the one-way or weak collision-resistant property of a hash code. Moreover, to complete the attack, the attackers need some more information such as the approaches to choose text-tag pairs or some knowledge or information of the key. One example of this kind of attack will be shown in the coming part. Hence, to attack the MAC value, the attackers can do it with the level of effort as 2^n, where n is the length of tags.

To put in a nutshell, the level of complexity of a brute-force attack on MAC can be expressed as $min(2^k, 2^n)$. Therefore, in order to make the MAC algorithm safe, we need to make the $min(k, n) \geq N$, where N is perhaps in the range of 128 bits[6].

4.2 Brute-force attack on CBC-MAC

Here the attack on CBC-MAC is, be more accurate is the birthday attack. We assume the attacker can have access to a huge amount of message-tag pairs. If the attackers can find one collision, then they can formulate a new message and its corresponding MAC. In this way, the attackers can eventually create new messages and forge the MAC. The following is the detail of this attack.

4.2.1 Set up

Let $n \in \mathbb{Z}$, $n \geq 3$, $x_3, x_4, ..., x_n$ be the bit strings with length of t. Then, according to D.Stinson[6], randomly generate $q \approx 1.17 \times 2^{t/2}$, different bit strings $x_1^1, ..., x_1^q$ with length of t along with $x_2^1, ..., x_2^q$ with length of t. For $1 \leq i \leq q$ and $3 \leq k \leq n$, make $x_k^i = x_k$, and construct

$$x^i = x_1^i || ... || x_n^i$$

4.2.2 Attack

As an assumption, the attacker can have the MAC of $x^1, x^2, ..., x^q$ in which y_n^i being the MAC of x^i. Since we have enough pairs of amount of message-tag pairs, let's say $y_n^i = y_n^j$, which means the MAC for x^i and x^j are the same. Noticing that $y_n^i = y_n^j$ if and only if $y_2^i = y_2^j$, which can further indicate that

$$y_1^i \oplus x_2^i = y_1^j \oplus x_2^j$$

That is to say, as long as the attack find such collision, they can have much more freedom on the content of x_2^i and x_2^j.

$$v = x_1^i || (x_2^i \oplus x_\delta) || ... || x_n^i$$

and

$$w = x_1^j || (x_2^j \oplus x_\delta) || ... || x_n^j$$

if the attacker only has the MAC for one of v and w, without loss of generation, we assume the attack has the access to the MAC of v, the MAC of w is clearly the same as v's. Hence, the attacker successfully forges the message-tag pair without knowing the key.

4.3 Brute-force attack on HMAC

As stated before, the HMAC is consist of two Hash functions,

$$\text{HMAC}(K, M) = H_o((K \oplus opad) || H_i((K \oplus ipad) || M))$$

The attacks are focus on the entire HMAC or the two Hash functions(for convenience we labelled them with i and o). The principle of this attack is similar to the attacks taken on the single hash functions. For simplicity, we use $H(\cdot)$ and HMAC$(\cdot)$ in this part.

- **For the attack on HMAC**
  The attackers can choose $x = M$ and find $z = \text{HMAC}(M)$ accordingly. Their goal is to generate some unchosen x' and z', such that $z' = \text{HMAC}(x)$.
- **For the attack on the inside Hash function**
  The attackers can choose $x = M$ and find $y = H_i(M)$ accordingly. Their goal is to find some x' and x'', such that $x' \neq x''$ with $y' = y''$.

- **For the attack on the outside Hash function**
  The attackers can choose $y = M$ and find $z = H_o(M)$ accordingly. Their goal is to generate some unchosen y' and z', such that $z' = H_o(y)$.

The procedure can be shown as follows.

1. Find q pairs of $x_1, \ldots, x_q$ and corresponding $z_1, \ldots, z_q$, and one pair of unknown message-tag pair (x, z).
2. Use $x_1, \ldots, x_q, x$ to find corresponding $y_1, \ldots, y_q, y$.
3. If $\exists i,\ y = y_i$, then the collision (x, x_i) is found.
4. Else if $y \notin \{y_1, \ldots, y_q\}$, it is then very likely for (y, z) to be a valid pair of H_o.

5 MAC Application

5.1 Pseudo-random Number Generation Using Hash Functions and MACS

We have learned the BBS generator in class, which is a pseudo-random bits generator (PRNG). Similar to many method we have learned in class, we need two large prime numbers to be the key of PRNG. We denote them as p and q. And the base of BBS, the Quadratic Residuosity is also the base of many encryption algorithms. We can see that, as we want our cipher text to be similar to a totally random sequence, the encryption algorithm is likely to serve as a basis of PRNG. Therefore, MACS or even the primary version - a hash function can also be used to build a PRNG.

The structure of the PRNG is quite simple, repeatedly using MAC to generate a small block with a private key K, while the "plaintext" of HMAC can be dealt in multiple ways that involve a seed V. Combining the blocks together will end in a pseudo-random bits generation. The Basic structure is shown below:

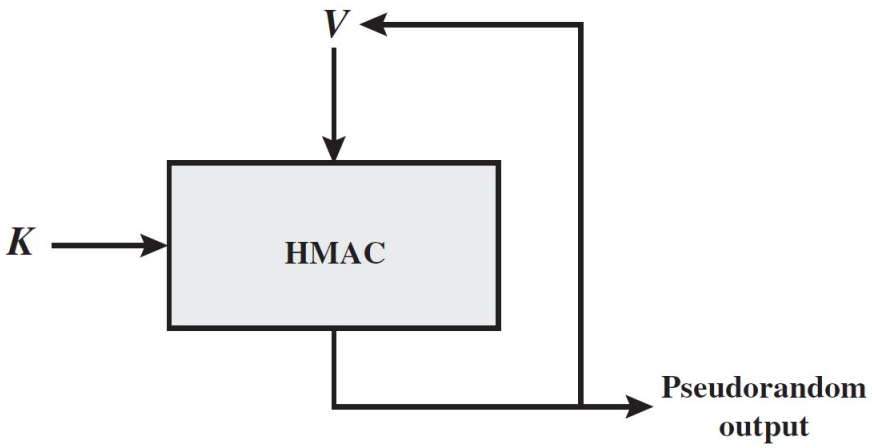

Figure 6: General Structure of PRNG using HMAC [1].

As we have proven that MAC is quite secure, using MAC to construct PRNG can provide a high degree of confidence. In real life, HMAC is often used to construct PRNG as it is widely used in many protocols and applications. Although the complicated HMAC will require more time to execute, it's worth it to have a more secure random number.

Next, we will introduce three different PRNGs based on HMAC. We defined the length of the output of the MAC function as L.

Algorithm NIST SP 800-90 [1]

Input: a seed V and a private key K

Output: a pseudo random number sequence with length n

$j = \lceil n/L \rceil$
$u_0 = V$
$U =$ null string
for $i = 1$ to j **do**
    $u_i = MAC(K, u_{i-1})$
    $U = U||u_i$
end for
return $U[0 : n-1]$

Algorithm IEEE 802.11i [1]

Input: a seed V and a private key K

Output: a pseudo random number sequence with length n

$j = \lceil n/L \rceil$
$U =$ null string
for $i = 1$ to j **do**
  $u_i = MAC(K, (V||i))$
  $U = U||u_i$
end for
return $U[0 : n-1]$

Algorithm TLS/WTLS [1]

Input: a seed V and a private key K

Output: a pseudo random number sequence with length n

$j = \lceil n/L \rceil$
$v(0) = V$
$W =$ null string
for $i = 1$ to j **do**
  $v(i) = MAC(K, v(i-1))$
  $u_i = MAC(K, (A(i)||V))$
  $U = U||u_i$
end for
return $U[0 : n-1]$

The SP 800-90 most follows the basic structure and encrypt the next block using the key and the previous block while the first block is the seed. While IEEE 802.11i uses the seed more flexibly, and combines the seed with a counter. While TLS/WTLS repeatedly using the MAC in order to create a better random number. Even if the opponent can find the input and output pair of the algorithm, based on the security of HMAC, he can not recover the key and predict the future pseudo-random bits. The figure below can show you the intuitively how HMAC is performed in TLS.

5.2 MAC used in Transport Layer Security protocol

Transport Layer Security protocol is a cryptographic protocol that is designed to make computer network more secure when passing information. TLS uses the pseudo-random function, PRF in another way. It is used for key generation or validation because it can produce secured data with arbitrary length without using a long secret value. While we have a secret key S and a seed V, we can use produce the secured data. The formula P_hash is combination of HMAC process. P_hash$(S, V) = H(1)||H(2)||H(3)...$ where $H(n) =$ HMAC_hash$(S, A(n)||V)$, $A(n) =$ HMAC_hash$(S, A(n-1))$ and $A(0) = V[1]$.

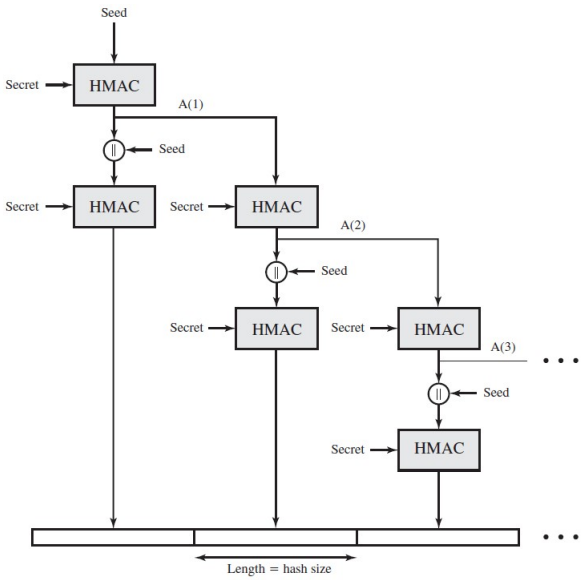

Figure 7: TLS Function P_hash [1]

We can iterate the P_{hash} many times. The number of iterations is gives us freedom to control the length of the outcome data. Four iterations can be used to generate 4 bytes of data, 64 bytes of the 80 generated bytes will be used. In brief, PRF is a method to produce a secure output with an arbitrary length given a secret value and a seed value while the seed value can be a combination of an identifying label and the real seed. This is used in the TLS..

6 Conclusion and Discussion

6.1 Discussion

1. In the attack on key space, we can have several keys that provides the correct tag, so we need to test the keys. However, we did not find the corresponding methods to do the test. We guess this may need some further information to complete the attack, for example, another pair of message-tag pair with the same key.

2. In the setup part of the brute-force attack on CBC-MAC, we have the number of messages generated being $1.17 \times 2^{t/2}$, we could not find how to calculate this value. But we guess this is the minimum value for us to find one collision.

3. In the brute-force attack on HMAC, we only have the strategy of attacking, however, by searching on the internet we cannot find the exact time complexity of this attacking. But, as we discussed before in the general analysis of the attacking method, we think the time complexity should be very similar to one of the attacks the MAC value.

6.2 Conclusion

The investigation of the concept of the Message Authentication Code is very interesting. From why we need it to how to used it, we have experienced multiple periods of understanding it. It's the first time that we turn our focus from how to prevent our key from being known by the others to how to prevent others from altering the concept of our message. Later we realize, the knowledge we have learned, the block ciphers, the security based on math and computational difficulty are all integrated into this small method. MAC itself is an application of what we have learned. Though we haven't learned about hash function in cryptology in class, it is also investigated during our research. In the end, the MAC is no longer something new, something different from AES or DES, they are all cryptology, a way to make information secure. By investigating this topic, we have a deeper understanding of what we have learned, and we have a clearer idea of what we need to do next and how the principles in cryptology are used in the real world.

References

[1] William Stallings, *Cryptography and Network Security: Principles and Practice,* 5th ed. Pearson Education Inc, 2011, pp. 362-394.

[2] Black, J., and Rogaway, P.; and Shrimpton, T. " CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions." Advances in Cryptology – CRYPTO '00, 2000.

[3] Iwata, T., and Kurosawa, K. "OMAC: One-Key CBC MAC." Proceedings, Fast Software Encryption, FSE '03, 2003.

[4] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[5] Quynh H. Dang, "The Keyed-Hash Message Authentication Code (HMAC)", Federal Inf. Process. Stds. (NIST FIPS) - 198-1, July 2008.

[6] D.Stinson, *Cryptography: Theory and Practice,* 3rd ed. CRC Press, 2009.

[7] Wikipedia Contributors. "Message authentication code." Wikipedia, Wikimedia Foundation, 16 June 2020, https://en.wikipedia.org/wiki/Message_authentication_code. Accessed accessed 21 June 2020.