MAC(Message authentication codes)

Sarthak Mahajan, 2018111005

March 7, 2022

1 Definition of MAC:

MAC or Message authentication codes are used to guarantee the integrity of a message. Its most basic use is to protect against the cases where an adversary can change the ciphertext even if it can't decrypt it. On the sender's end a MAC algorithm generates a tag t for a message m such that

$$t = MAC_k(m)$$

, where k is a key generated by a generation algorithm: GEN. The receiver receives (m,t), and uses

$$b = VERIFY_k(m, t)$$

, if b= True, the message is accepted by the receiver otherwise its rejected. Thus MAC consists of

$$(GEN, MAC_k(.), VERIFY_k(.))$$

2 Security of MAC:

2.1 MAC Setup:

Let the Adversary have access to an oracle producing MAC, and Q be the set of all queries of the adversary to the oracle, then if in the MAC algorithm, $VERIFY_k(m, t) = 1$, and m sent by the adversary is not in Q, then this setup(Mac-Setup) outputs 1.

2.2 Condition for security:

A message authentication code (Gen, MAC, Verify) is secure if for all probabilistic polynomial-time adversaries A:

$$\Pr[\text{ Mac-Setup } (n) = 1] \leq \text{ negl } (n)$$

3 Fixed length MAC using PRF:

- Gen (1^n) chooses k to be a random n-bit string
- $MAC_k(m) = F_k(m) = t$ (the tag)
- Verify $_k(m, t) = Accept$, if and only if $t = F_k(m)$

3.1 Explanation:

This works because PRF is supposed to generate a number akin to a random number, so its very unlikely for the PRF to return the same output for different seeds.

4 Variable length MAC using fixed length MAC:

Let $\Pi' = \left(\text{Gen}', \text{Mac}', \text{Vrfy}'\right)$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Gen: this is identical to Gen'.
- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of length $\ell < 2^{\frac{n}{4}}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0 s if necessary.) Next, choose a random identifier $r \leftarrow \{0,1\}^{n/4}$. For $i = 1, \ldots, d$, compute $t_i \leftarrow \text{Mac}'_k\left(r \| \ell \| i \| m_i\right)$, where i and ℓ are uniquely encoded as strings of length $n/4$.[4] Finally, output the tag $t := \langle r, t_1, \ldots, t_d \rangle$
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{\frac{n}{4}}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0 s if necessary.) Output 1 if and only if $d' = d$ and $\text{Vrfy}'_k\left(r \| \ell \| i \| m_i, t_i\right) = 1$ for $1 \leq i \leq d$.

5 Variable length MAC using CBC-MAC:

5.1 $MAC_k(m)$ generation

Let the length of the key $k = n$, assume that the length of the message $m = n_2$ and $n_2 >= n$, We start breaking m into chunks of size n, starting from the MSB, in case the last chunk cannot be of size n, we pad it with zeros on the left to make it of size n. Let m_i donate the i^{th} message chunk. Let each "block" of the CBC-MAC be a PRF $F_k(.)$, with key k. The input of i^{th} block$(i \neq 1)$ is:

$$m_{i-1} \oplus o_{i-1}$$

, where o_{i-1} is the output of the i^{th} block. The input for the 1st block when $i - 1 = 0$ is n_2. The output of the 1st block is: $\text{o}_1 = \text{F}_k(n_2)$, while output for i^{th} block$(i \neq 1)$ is:

$$o_i = F_k(o_{i-1} \oplus m_{i-1})$$

The $MAC_k(m)$ is the output of the final block of the CBC-MAC.

5.2 Verification:

On the receiver's end they obtain the message (m, t), the receiver sends the m to the same CBC-MAC described above to obtain $MAC_k(m)$. If $MAC_k(m) = t$, then message is authenticated, otherwise its rejected.

5.3 Explanation:

CBC-MAC appends length first and then the message chunks so that the adversary cannot pass off a subset of a valid message a valid message too.Since the CBC-MAC, applies PRF on each chunk and then feeds it into the next block serially, a message with changed permutation or mixture of another valid message(in an interleaved manner) won't be authenticated by this scheme.