Message Authentication Codes (MACs)

Based on *Cryptography Engineering* by Schneier, Ferguson, Kohno, Chapter 6

Lecture notes of Alexander Wood
awood@jjay.cuny.edu

John Jay College of Criminal Justice

MACs

A **message authentication code (MAC)** is a key-dependent one-way hash function.

They satisfy the same properties as one-way hash functions.

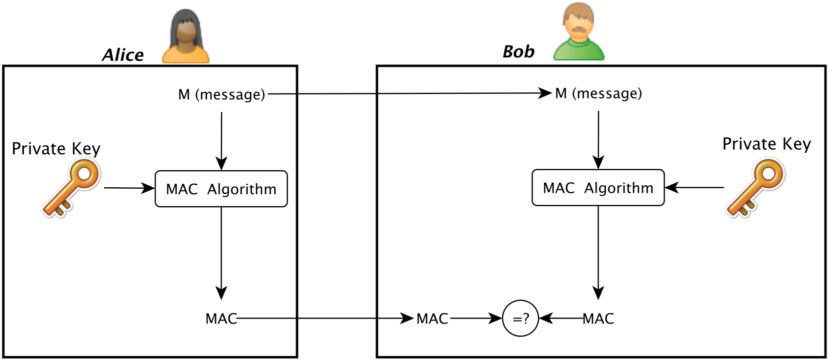
In addition they have a **key**.

MACs for Authentication

MACs are used to **authenticate** files between users. It checks its **authenticity** (confirms the sender) as well as its **integrity** (it has not been tampered with).

Sometimes a MAC is referred to as a Message Integrity Code (MIC), espcially in applications where MAC already stands for Media Access Control.

MAC Visualization



MAC Algorithms

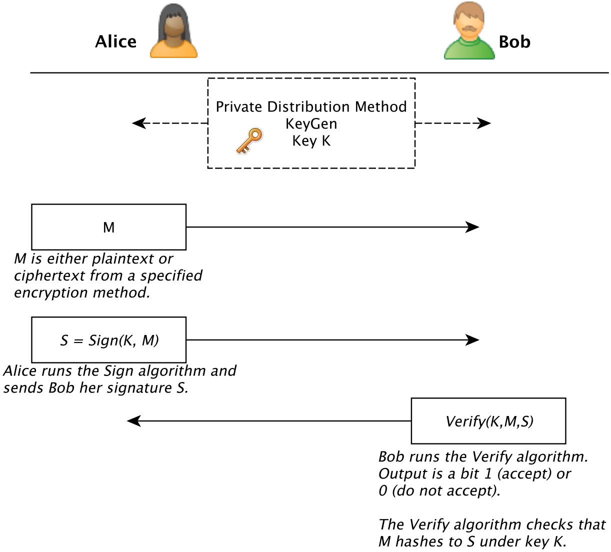
- `KeyGen` - generates a key 1^n uniformly at random.
- `Sign` - Alice inputs her key k and message M, receives output t (tag).
- `Verify` - Bob verifies the authenticity of Alice's message.

MACs

A one-way hash function can be turned into a MAC by encrypting the hash function with a symmetric algorithm.

Conversely, a MAC algorithm can be turned into a one-way hash function by changing the private key to a public key.

MAC Algorithms Visualization



MAC Security

Again we take our security definition by *Cryptography Engineering* by Ferguson, Schneier, and Kohno (Chapter 6).

Definition

*An **ideal MAC function** is a random mapping from all possible inputs (key, message pairs) to all possible n-bit outputs.*

The attacker should *not* know the entirety of the key.

Attack on a MAC

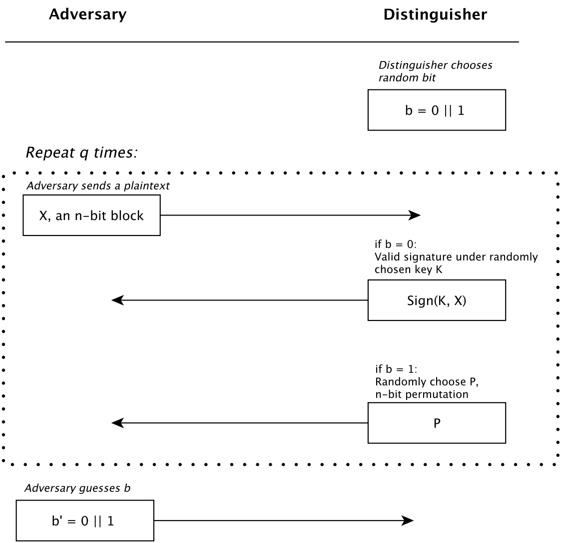
While other security definitions exist, we will use the following broad one in this course.

Definition

*An **attack on a MAC** is a non-generic method of distinguishing the MAC from an ideal MAC function.*

Recall that this is called a **distinguishing attack.** The idea is that the attacker should not be able to distinguish a valid signature from a random bitstring. Where have we seen this sort of attack before?

Distinguishing Attack Visualization



CBC-MAC

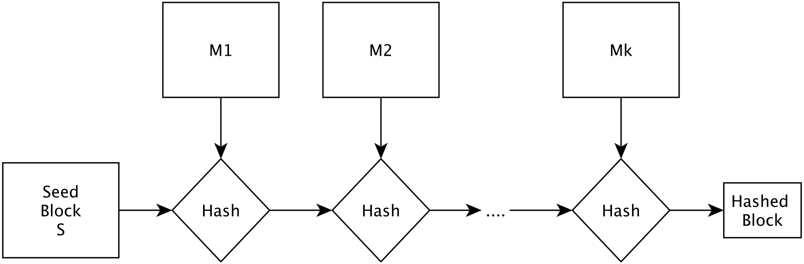
CBC-MAC (cipher block chaining message authentication code) is a method of turning a block cipher into a MAC under a key K. The message is encrypted using cipher block chaining mode in order to create a construction where each block depends on the previous block.

It is constucted as follows, where M is split into length n blocks $M_1, \ldots, M_\ell$ and E_k is encryption under a key K.

$$\begin{aligned} H_0 &:= 0^n \\ H_i &:= E_K(M_i \oplus H_{i-1}) \\ MAC &:= H_\ell \end{aligned}$$

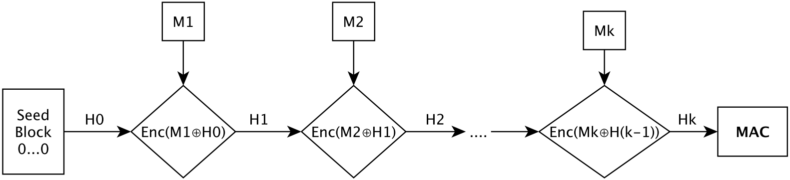
CBC-MAC

We saw something similar to this when constructing hash functions. Each hash block H_i depends on the previous hash block H_{i-1}, constructed with a different block in the message.



CBC-MAC

The difference now in **CBC-mode** is that we also must include a **key**. This requires a slightly different, though similar, construction.



Note On Keys

Do NOT use the same key for encryption and authentication!

CBC-MAC

CBC-MAC is secure for fixed-length messages *if the underlying block cipher used is secure*.

CBC-MAC: Collision Attack

Let M be a CBC-MAC function. Assume $M(a) = M(b)$. Then, $M(a\|c) = M(b\|c)$ where

$$M(a\|c) = E_k(c \oplus M(a))$$
$$M(b\|c) = E_k(c \oplus M(b))$$

There are two stages to the attack.

CBC-MAC: Collision Attack, Step 1

An attacker, Eve, collects MAC values until she finds a collision, ie, messages a and b such that $M(a) = M(b)$.

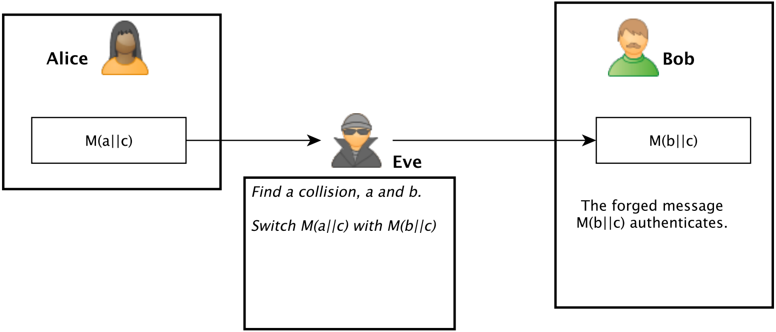
According to the Birthday paradox, if we are using a `128`-bit block cipher, this will take 2^{64} steps.

CBC-MAC: Collision Attack, Step 2

Eve gets the sender, Alice, to authenticate the message $a \| c$. She can now replace this message with $b \| c$ without changing the MAC value.

This means that the receiver, Bob, will accept the forged message $b \| c$.

CBC-MAC: Collision Attack



CBC-MAC: Implementation

If you wish to use CBC-MAC, follow the following steps.

1. Concatenate ℓ and m to construct a string s, where m is the message and ℓ is the length of m encoded in a fixed-length format.
2. Pad s until it is a multiple of the needed block size.
3. Apply CBC-MAC to the padded s.
4. Output the last ciphertext block. Do not output any intermediate values.

CMAC

CMAC, also known as One-Key MAC and OMAC, is based off of CBC-MAC and was standardized by NIST in 2005.

It fixes security deficiencies of CBC-MAC and is easier to implement with existing libraries in Python and Ruby.

CMAC

CMAC works like CBC-MAC except on the last block.

It derives two special values from its key and chooses one based on whether the length of the message is a multiple of the block length or not.

It `XOR`s one of two special values into the last block prior to the last block cipher encryption.

HMAC

HMAC (keyed-hash message authentication code) uses a hash function to construct a MAC.

HMACs are less affected by collision attacks than the hash functions which they are based off of. Hence, while MD5 is not colliion-resistant, HMAC-MD5 is considered to be reasonably safe as a MAC (as of 2011).

HMAC

HMAC computes:

$$h((K \oplus a) \| h((K \oplus b) \| m))$$

for constants a and b, message m, key K, and hash function h. Note that $\|$ denotes concatenation.

HMAC

Note that:

- The message m is only hashed once
- The output is then hashed again with the key K
- This HMAC construction works with any of the hash functions we discussed!

Using A MAC

Be careful when using a MAC! It is difficult to do properly.

Using a MAC

Say Bob received $\texttt{Sign}_K(m)$. Bob now knows that *someone with the key K approved the message m*. He knows nothing more than this.

For example: Eve recorded a message signed by Alice and sent to Bob, then sent that same message to Bob at a later time. Bob then verified this message as being from Alice.

Using a MAC

Suppose Alice and Bob want to authenticate a message m along with data d containing information such as the message number used, the source and destination of the message, etc.

Frequently d will be used as a header and concatenated to m. Then, $d \| m$ is signed and verified.

Using a MAC: The Horton Principle

The Horton Principle: Authenticate what is *meant*, not what is *said*.

Using a MAC: The Horton Principle

What is said is the bytes sent, and what is meant is the interpretation of the message.

Thus the MAC should authenticate both the message *m* as well as instructions on how to parse the message *m*. This can include a protocol identifier, version number, sizes for fields, et cetera.

References

- *Applied Cryptography* By Schneier, Chapter 18
- *Cryptography Engineering* by Schneier, Ferguson, Kohno, Chapter 6