

Cryptography

Message Authentication Codes

Prof. Dr. Heiko Knospe

TH Köln – University of Applied Sciences

May 14, 2025

Message Authentication Codes

A message authentication code (MAC) is a cryptographic *tag*, which protects the *integrity* and the *origin* of a message. A correct tag shows that the data has not been tampered with by an adversary or modified by accidental errors. MACs are widely used to protect mass data, for example in the network security protocols TLS and IPsec.

Message authentication codes use a *symmetric secret key* for tag generation and verification. This constitutes a major difference to *signatures*, where messages are signed with a *private key* and verification is done using a *public key*.

Definition of a MAC

Definition

A *message authentication code* (MAC) is given by the following spaces and polynomial-time algorithms:

- A message space \mathcal{M} ,
- A key space \mathcal{K} ,
- A key generation algorithm $Gen(1^n)$ that takes a security parameter 1^n as input and outputs a key k ,
- A tag generation algorithm, which may be randomized. It takes a message m and a key k as input and outputs a tag $MAC_k(m)$,
- A deterministic verification algorithm that takes a key k , a message m and a tag t and outputs 1 if the tag is valid, or otherwise 0. *Canonical verification* means to re-compute $MAC_k(m)$ and to compare the result with the given tag.

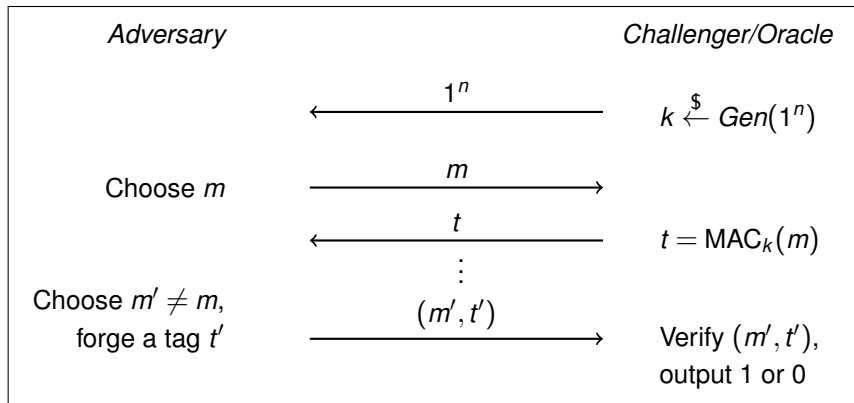
Security of a MAC

A message authentication tag is usually short (similar to a hash value) and does not include the message. Therefore, verification requires the *message, the tag and the secret key*.

The security of message authentication codes is determined by the difficulty to *forge a valid tag* of a message without knowing the key.

We assume that an adversary can choose messages and obtains the corresponding MACs (*chosen message attack*). A MAC is *EUFCMA secure* (existentially unforgeable under an adaptive chosen message attack), if the *probability* that a polynomial-time adversary generates a message, which he did not query previously, and a *valid tag* is *negligible*.

MAC Forgery Experiment



Secure and Strongly Secure MACs

A MAC is *secure* if polynomial-time adversaries can only generate a valid tag *of a new message* with negligible probability.

A MAC is *strongly secure* if the probability of successfully forging a MAC is negligible in a modified experiment that might increase the odds of an adversary: a tag is accepted if the message or the tag (or both) are new. So he might generate a new (and valid) tag of a previously seen message. Moreover, one has the following statement:

Theorem

A secure MAC with canonical verification is strongly secure.

CBC MAC

We want to construct a MAC for messages of arbitrary length based on a *block cipher*.

A popular construction is the *CBC MAC* (Cipher Block Chaining Message Authentication Code). The message is encrypted in CBC mode (with zero IV), and the *last ciphertext block* is used as authentication tag.

Unfortunately, the basic CBC MAC is not secure for messages of variable length, since an adversary can extend a message and predict the tag without knowing the key. This problem can be fixed with the CMAC construction (see below).

CBC MAC

Definition

Let $E : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ be a block cipher. Fix a number N of input blocks and set $\mathcal{M} = \{0, 1\}^{N \cdot l}$. Let $k \xleftarrow{\$} \{0, 1\}^n$ be a key. The *basic CBC MAC* of a message $m = m_1 \parallel m_2 \parallel \dots \parallel m_N$ of fixed length $N \cdot l$ is defined by encrypting m in CBC mode and outputting the last ciphertext block:

$$c_0 = 0^l$$

$$c_i = E_k(m_i \oplus c_{i-1}) \text{ for } i = 1, 2, \dots, N$$

$$\text{MAC}_k(m) = c_N$$

The CBC MAC is deterministic and verification is canonical.

CMAC

Definition

Let E be a block cipher with 128-bit block length, k a secret key and $m = m_1 \| m_2 \| \dots \| m_N$ a sequence of 128-bit message blocks, where the last block m_N may be shorter. Then the CMAC of m is defined as the CBC MAC of $m' = m_1 \| m_2 \| \dots \| m_{N-1} \| m'_N$, where the last block is tweaked in the following way: first, define 128-bit subkeys k_1 and k_2 : $k_1 = E_k(0^{128}) \cdot (0^{126}10)$ and $k_2 = E_k(0^{128}) \cdot (0^{125}100)$. Multiplication of 128-bit blocks is defined via the field $GF(2^{128})$. Then put:

$$m'_N = \begin{cases} m_N \oplus k_1 & \text{if } |m_N| = 128 \\ (m_N \| 10 \dots 0) \oplus k_2 & \text{if } |m_N| < 128 \end{cases}$$

$$\text{CMAC}_k(m) = \text{CBC-MAC}(m_1 \| m_2 \| \dots \| m_{N-1} \| m'_N) = c_N$$

Verification of the CMAC is canonical.

Security of CMAC

An adversary cannot produce a valid CMAC without knowing k_1 or k_2 , and these subkeys depend on the secret MAC key k . Furthermore, k_1 or k_2 cannot be recovered from valid tags, since m'_N is protected by CBC mode encryption.

Theorem

If E is a pseudorandom permutation or function, then the CMAC construction defines an EUF-CMA secure MAC for messages of variable length.

Hash-based MAC

Another widely used MAC construction is based on *hash functions*.

However, the obvious *prefix* construction $H_k(m) = H(k||m)$ is *insecure* for messages of variable length, if H is a Merkle- Damgård hash function (*length extension attack*). Note that the SHA-3 family is not vulnerable to this attack.

The *Hash-based Message Authentication Code* (HMAC) uses *two nested hashing* operations, where the input to the outer hash has fixed length. This protects against *length extension attacks*. HMAC is widely used in practice, not only as message authentication code, but also as a pseudorandom function (*prf*) and as a building block in (password-based) key derivation functions such as PBKDF2.

HMAC

Definition

Let H be a Merkle-Damgård hash function. Let b be the input block length in bytes of the underlying compression function, e.g., $b = 64$.

The message space is $\mathcal{M} = \{0, 1\}^*$ and HMAC keys $k \xleftarrow{\$} \{0, 1\}^n$ are chosen uniformly at random. Define ipad and opad strings by repeating the bytes 36 and $5C$, respectively, b times. The key k is padded by zeros, so that the byte length of $\bar{k} = (k \parallel 0 \dots 0)$ is b . Then the *HMAC* message authentication tag of a message m is defined as

$$\text{HMAC}(k, m) = H(\bar{k} \oplus \text{opad} \parallel H(\bar{k} \oplus \text{ipad} \parallel m)).$$

The verification of HMAC is canonical.

HMAC Security

The following Theorem reduces the security of HMAC to the pseudorandomness of the compression function f and its dual function \bar{f} .

Theorem

Let $f : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^n$ be a compression function and let $\bar{f} : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^n$ be the dual function with the same values as f , but keyed via the second component. Let H be the Merkle-Damgård hash function associated with f .

If f is a prf and \bar{f} is a prf under restricted related-key attacks, then HMAC is a pseudorandom function and an EUF-CMA secure MAC for messages of arbitrary length.

Unforgeable Encryption

In practice, encryption and message authentication are often combined in order to achieve *confidentiality* and *integrity protection*. *Tampering with encrypted messages* should be impossible, and the scheme should be secure under a *chosen ciphertext attack*.

Definition

An encryption scheme is called *unforgeable* if an adversary, who has access to an encryption oracle, cannot produce a new and valid ciphertext in polynomial time.

An encryption scheme is called an *authenticated encryption scheme*, if it is CCA2-secure and unforgeable.

Note: CBC and CTR mode ciphertexts are forgeable.

Authenticated Encryption

An obvious construction is to combine an encryption scheme and a message authentication code.

Theorem

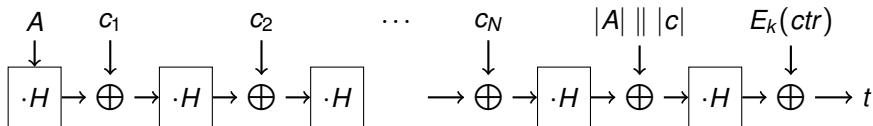
Consider an encryption scheme and a message authentication code. Suppose the encryption scheme is CPA-secure and the message authentication code is strongly secure, for example a secure MAC with canonical verification. Then the encrypt-then-authenticate construction defines an authenticated encryption scheme.

Note: the *encrypt-then-authenticate* construction *first* encrypts the plaintext and *then* computes the MAC of the ciphertext.

GCM Mode

The *Galois Counter Mode* (GCM) extends the CTR mode and provides encryption and message authentication in one pass (for block ciphers with $l = 128$). GCM can also authenticate *additional plaintext data* A .

First, the ciphertext $c = IV \parallel c_1 \parallel \dots \parallel c_N$ is computed in CTR mode, where $IV \xleftarrow{\$} \{0, 1\}^{96}$ and $ctr = IV \parallel 0^{31} \parallel 1$ is the initial counter value. Then the GCM tag t is derived from the ciphertext c and additional data A . Encryption and message authentication use the same key k .



$H = E_k(0^{128})$ is called *hash key*, and multiplication $(\cdot H)$ is defined in the field $GF(2^{128}) = GF(2)[x]/(x^{128} + x^7 + x^2 + x + 1)$.

Authenticated Encryption with GCM

The authenticated ciphertext is given by

$$(c, t, A).$$

For decryption, the tag is re-computed using c and A . If t is valid, then the plaintext m is computed by decrypting c in counter mode.

Otherwise, an error symbol \perp is output.

Note that ciphertext manipulations are no longer possible in GCM mode, since they would invalidate the tag.

Key Derivation

Message authentication codes are also used as *Key Derivation Functions* (KDF). In contrast to pseudorandom functions, key derivation functions also take a *non-uniform input key*, which is produced by some entropy pool. Nevertheless, the derived keys should look uniform random.

The NIST publication [SP800-56Cr2](#) recommends *one-step key derivation functions* (using an approved hash function, HMAC or KMAC), and also *two-step extract-then-expand key derivation functions*. The first step uses either HMAC or AES-CMAC and there are two inputs: a salt, which serves as a MAC key and an input key, which serves as the message. The second step uses either HMAC, CMAC or KMAC.

HKDF

The HMAC-based two-step extract-and-expand Key Derivation Function (HKDF) is standardized in [RFC 5869](#).

- 1 The first stage takes the input keying material and *extracts* from it a fixed-length pseudorandom key *PRK*. The goal of the extract stage is to concentrate the input keying material IKM (which is not necessarily uniformly distributed) into a short, but cryptographically strong, pseudorandom key. To this end, a single HMAC operation is performed.
- 2 The second stage *expands* the key *PRK* into several additional pseudorandom keys (the output of the KDF). This stage may use several HMAC operations.

HKDF Steps

HKDF can operate with or without random salts. However, the use of salt adds significantly to the strength of HKDF.

- 1 The output PRK is recursively calculated as follows:

$$PRK = HMAC - Hash(salt, IKM)$$

- 2 The output keying material $T(1) || T(2) || \dots$ of the required length is calculated as follows. Let $T(0)$ be empty and *info* optional context data. Define

$$T(n) = HMAC - Hash(PRK, T(n-1) || info || 0x n), \quad n = 1, 2, \dots$$

Question: Would you use a HKDF as a *password-based* key derivation function?