Message authentication codes (MACs)

Tanja Lange

Eindhoven University of Technology

2WF80: Introduction to Cryptology

Encryption and authentication

- Simplest case: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Encryption takes plaintext m and produces ciphertext c, decryption takes c and produces m so that $\text{Dec}(\text{Enc}(m)) = m$.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.
- Security goal #3: **Authenticity**, i.e., recognizing Eve impersonating.
- Decryption fails for invalid ciphertexts.
  (This needs a definition of what "invalid" means).

Encryption and authentication

- Simplest case: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Encryption takes plaintext m and produces ciphertext c, decryption takes c and produces m so that $\text{Dec}(\text{Enc}(m)) = m$.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.
- Security goal #3: **Authenticity**, i.e., recognizing Eve impersonating.
- Decryption fails for invalid ciphertexts.
  (This needs a definition of what "invalid" means).

Encryption and authentication

- Simplest case: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Encryption takes plaintext m and produces ciphertext c, decryption takes c and produces m so that $\mathrm{Dec}(\mathrm{Enc}(m)) = m$.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.
- Security goal #3: **Authenticity**, i.e., recognizing Eve impersonating.
- Decryption fails for invalid ciphertexts.
  (This needs a definition of what "invalid" means).

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  011001110011001010010010111001
+ 010111101100011010110100101001
--------------------------------
  001110011111010000100110010000
```

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  01100111001100101001001011100 1
+ 010111101100011010110100101001
-------------------------------
  001110011011010000100110010000
```

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  011001110111001010010010111001
+ 010111101100011010110100101001
--------------------------------
  001110011011010000100110010000
```

Flipping bit i in the ciphertext means flipping bit i in the plaintext.

Encryption requires integrity protection!

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  011001110111001010010010111001
+ 010111101100011010110100101001
--------------------------------
  001110011011010000100110010000
```

Flipping bit i in the ciphertext means flipping bit i in the plaintext.

Encryption requires integrity protection!

Solution: send $c, H(c)$. Checksum $H(c)$ detects changes in c.

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  01100111011100101001001011100 1
+ 0101111011000110101101001010 01
--------------------------------
  00111001101101000010011001000 0
```

Flipping bit i in the ciphertext means flipping bit i in the plaintext.

Encryption requires integrity protection!

~~Solution~~: send $c, H(c)$. Checksum $H(c)$ detects changes in c.

But Eve could change that to $c', H(c')$.

Motivation

Encryption using stream cipher (or OTP) as $c = m + s$, taken modulo 2.

```
  01100111011100101001001011100 1
+ 01011110110001101011010010100 1
---------------------------------
  00111001101101000010011001000 0
```

Flipping bit i in the ciphertext means flipping bit i in the plaintext.

Encryption requires integrity protection!

~~Solution~~: send $c, H(c)$. Checksum $H(c)$ detects changes in c.

But Eve could change that to $c', H(c')$.

Solution(?): Send $c, H(m)$.

This works if m is secret.
Not desirable to decrypt before checking validity.

Positive feature:
Bob is sure that Alice sent this, as sender must know encryption key.

Message authentication code (MAC)

A MAC is a cryptographic checksum ensuring integrity and authenticity.

It takes a message (plaintext or ciphertext) and a shared key and produces the authentication tag:

$$\mathsf{MAC} : \{0,1\}^* \times \{0,1\}^\ell \to \{0,1\}^n.$$

Like hash functions, MACs take blocks of bits; some padding rules apply.

Security requirements

- Computing a valid MAC without knowing k is hard.
- Given a valid pair $(c, \mathsf{MAC}(c,k))$ it is hard to produce a valid pair $(c', \mathsf{MAC}(c',k))$ for $c' \neq c$.
- Even given the power to request $\mathsf{MAC}(c_i,k)$ on chosen messages c_i it is hard to produce a valid pair $(c', \mathsf{MAC}(c',k))$ for new $c' \neq c_i$.

Note

- Alice and Bob typically share encryption key and authentication key. Key k here is the authentication key.
- A MAC convinces Bob that the message came from Alice; it cannot convince outsiders: Alice and Bob share key k.

Simple MAC

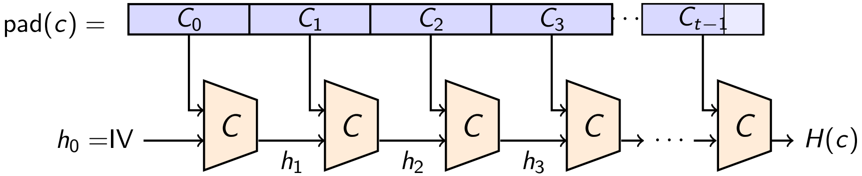
Define $\text{MAC}(c, k) = H(k\, c)$, for cryptographic hash function H.

Want c in the second position so that collisions in H cannot be turned into forged MACs. (This matters in the 3rd scenario where the attacker can request MACs on chosen messages).

Simple MAC

Define $\text{MAC}(c, k) = H(k\,c)$, for cryptographic hash function H.

Want c in the second position so that collisions in H cannot be turned into forged MACs. (This matters in the 3rd scenario where the attacker can request MACs on chosen messages).



Simple MAC is insecure if H uses the Merkle-Damgård construction:

$$H(k\,c\,C_t) = C(C_t, H(k\,c))$$

is computable from $\text{MAC}(c, k) = H(k\,c)$ without knowing k.

Patch by insisting on fixed padding at end of message or use other H.

General comment: We typically want to encrypt then MAC.

Image credit: adapted from Jérémy Jean.

HMAC

There are many MAC designs in the literature. (*MAC exists for almost all values of *.)

HMAC

There are many MAC designs in the literature. (*MAC exists for almost all values of *.)

HMAC (1996) by Bellare, Canetti, and Krawczyk.

HMAC deals with issues such as length-extension attacks (Merkle-Damgård) or collisions in H by putting k at beginning and end.

Also uses two different padding strings (ipad, opad) to tweak the key:
$\text{HMAC}(c, k) = H((k+\text{opad})\, H((k+\text{ipad})\, c))$
(some details to fit k into one block of H).

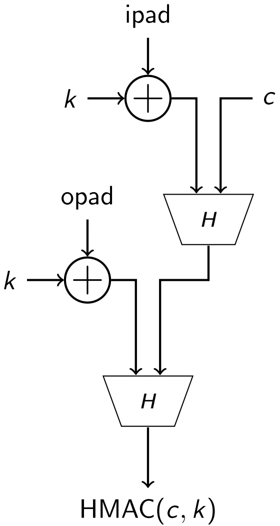


Image credit: adapted from Carl Richard Theodor Schneider.