CSCI-GA.3205 Applied Cryptography & Network Security

Department of Computer Science
New York University

PRESENTED BY DR. MAZDAK ZAMANI
mazdak.zamani@NYU.edu

Message Integrity Authenticated Encryption

04

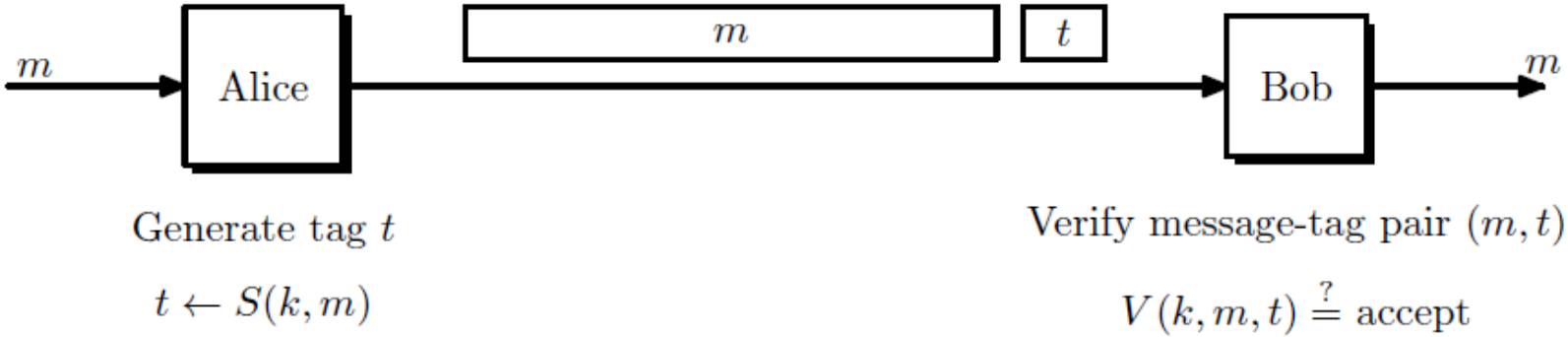
NYU

Message integrity

- In previous chapters we focused on security against an eavesdropping adversary.
- The adversary had the ability to eavesdrop on transmitted messages, but could not change messages en-route.
- In this chapter we turn our attention to active adversaries.
- We start with the basic question of *message integrity*: Bob receives a message m from Alice and wants to convince himself that the message was not modified en-route.

Short message integrity tag added to messages

- We will design a mechanism that lets Alice compute a short message integrity tag t for the message m and send the pair $(m; t)$ to Bob, as shown below.
- Upon receipt, Bob checks the tag t and rejects the message if the tag fails to verify. If the tag verifies then Bob is assured that the message was not modified in transmission.



Message secrecy and message integrity

- We emphasize that in this chapter the message itself need not be secret.
- Unlike previous chapters, our goal here is not to conceal the message. Instead, we only focus on message integrity.
- Later we will discuss the more general question of simultaneously providing message secrecy and message integrity.
- In this chapter, we will assume that both sender and receiver will share the secret key; later, this assumption will be relaxed.
- The TCP protocol uses a keyless 16-bit checksum which is embedded in every packet. We emphasize that these keyless integrity mechanisms are designed to detect random transmission errors, not malicious errors.

Message authentication code

Definition 6.1. *A **MAC** system $\mathcal{I} = (S, V)$ is a pair of efficient algorithms, S and V, where S is called a **signing algorithm** and V is called a **verification algorithm**. Algorithm S is used to generate tags and algorithm V is used to verify tags.*

- *S is a probabilistic algorithm that is invoked as $t \xleftarrow{R} S(k, m)$, where k is a key, m is a message, and the output t is called a **tag**.*
- *V is a deterministic algorithm that is invoked as $r \leftarrow V(k, m, t)$, where k is a key, m is a message, t is a tag, and the output r us either* accept *or* reject.
- *We require that tags generated by S are always accepted by V; that is, the MAC must satisfy the following **correctness property**: for all keys k and all messages m,*

$$\Pr[V(k,\ m,\ S(k,\ m)\ ) = \text{accept}] = 1.$$

Message authentication code

- Keys lie in some finite **key space K**, messages lie in a finite **message space M**, and tags lie in some finite **tag space T** .
- We say that I = (S; V) is defined over (K; M; T).
- Whenever algorithm V outputs "*accept*" for some message-tag pair (m; t), we say that t is a *valid tag* for m under key k, or that (m; t) is a *valid pair* under k.
- One of the MAC systems is one in which the signing algorithm S is deterministic, and the verification algorithm is defined as

$$V(k,m,t) = \begin{cases} \text{accept} & \text{if } S(k,m) = t, \\ \text{reject} & \text{otherwise.} \end{cases}$$

NYU

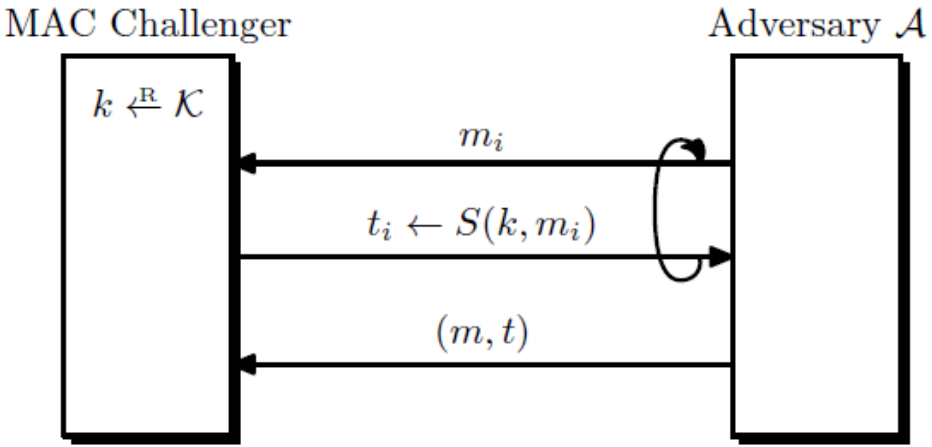
MAC attack game

Attack Game 6.1 (MAC security). For a given MAC system $\mathcal{I} = (S, V)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots$, the ith *signing query* is a message $m_i \in \mathcal{M}$. Given m_i, the challenger computes a tag $t_i \xleftarrow{R} S(k, m_i)$, and then gives t_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, t) \in \mathcal{M} \times \mathcal{T}$ that is not among the signed pairs, i.e.,

$$(m, t) \notin \{(m_1, t_1), (m_2, t_2), \ldots\}.$$

MAC attack game



- We say that a MAC system I is secure if for all efficient adversaries A, the value MACadv[A; I] is negligible.

Constructing MACs from PRFs

- In previous chapters we used pseudo random functions (PRFs) to build various encryption systems.
- Here we show that any secure PRF can be directly used to build a secure MAC.

Recall that a PRF is an algorithm F that takes two inputs, a key k and an input data block x, and outputs a value $y := F(k, x)$. As usual, we say that F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where keys are in $\mathcal{K}$, inputs are in $\mathcal{X}$, and outputs are in $\mathcal{Y}$. For a PRF F we define the **deterministic MAC system $\mathcal{I} = (S, V)$ derived from F** as:

$$S(k,\ m) := F(k,\ m);$$

$$V(k,\ m,\ t) := \begin{cases} \text{accept} & \text{if } F(k,m) = t, \\ \text{reject} & \text{otherwise.} \end{cases}$$

Prefix-free PRFs for long messages

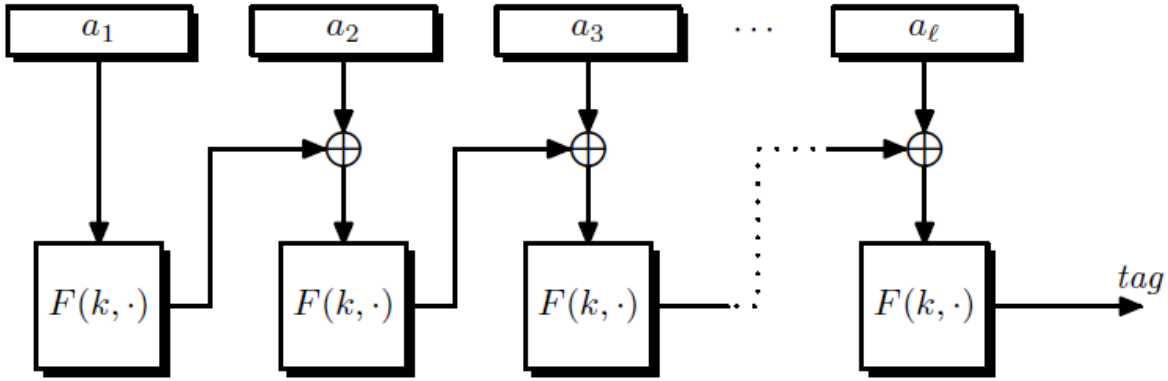
- We saw that any secure PRF is also a secure MAC.
- However, the concrete examples of PRFs only take short inputs and can therefore only be used to provide integrity for very short messages.
  - For example, viewing AES as a PRF gives a MAC for 128-bit messages.
- Clearly, we want to build MACs for much longer messages.
- All the MAC constructions in this chapter follow the same paradigm:
  - They start from a PRF for short inputs (like AES) and produce a PRF, and therefore a MAC, for much longer inputs.
- Hence, our goal for the remainder of the chapter is the following:
  - Given a secure PRF on short inputs construct a secure PRF on long inputs.

Prefix-free PRFs

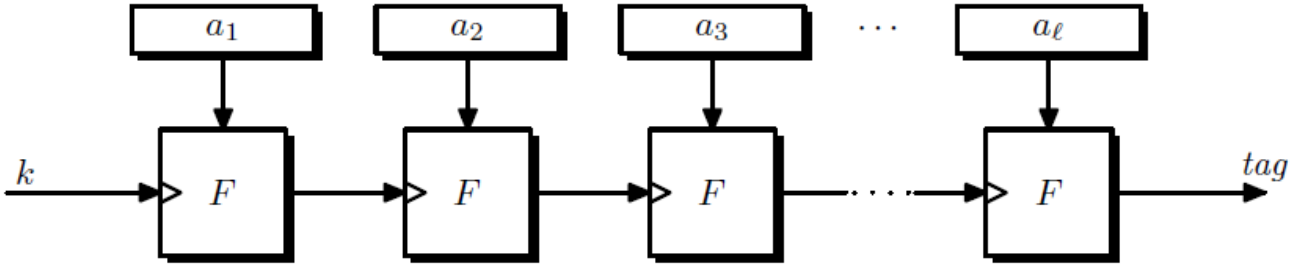
- We begin with two classic constructions for prefix-free secure PRFs.
  - The Cipher block chaining (CBC) construction is shown in Fig. a.
  - The cascade construction is shown in Fig. b.
- When the underlying F is a secure PRF, both CBC and cascade are prefix-free secure PRFs.

NYU

Prefix-free PRFs



(a) The CBC construction $F_{\text{CBC}}(k, m)$



(b) The cascade construction $F^*(k, m)$

Prefix-free PRFs

- A prefix-free secure PRF is only secure in a limited sense: prefix-free adversaries cannot distinguish the PRF from a random function.
- Note that CBC uses a *fixed key* k for all applications of F while cascade uses a *different key* in each round.
    - Since block ciphers are typically optimized to encrypt many blocks using the same key, the constant re-keying in cascade may result in worse performance than CBC. Hence, CBC is the more natural choice when using an off the shelf block cipher like AES.
- An advantage of cascade is that the cascade construction remains secure even if the underlying PRF has a small domain X. CBC, in contrast, is secure only when X is large. As a result, cascade can be used to convert a PRG into a PRF for large inputs while CBC cannot.

An attack on CBC

- We describe a simple MAC forger on the MAC derived from CBC. The forger works as follows:

1. pick an arbitrary $a_1 \in \mathcal{X}$;
2. request the tag t on the one-block message (a_1);
3. define $a_2 := a_1 \oplus t$ and output t as a MAC forgery for the two-block message $(a_1, a_2) \in \mathcal{X}^2$.

Observe that $t = F(k, a_1)$ and $a_1 = F(k, a_1) \oplus a_2$. By definition of CBC we have:

$$F_{\text{CBC}}\big(k,\ (a_1, a_2)\ \big) = F\big(k,\ \ F(k,\ a_1) \oplus a_2\ \big) = F(k,\ a_1) = t.$$

Hence, $\big((a_1, a_2),\ t\big)$ is an existential forgery for the MAC derived from CBC. Consequently, F_{CBC} cannot be a secure PRF. Note that the attack on the cascade MAC is far more devastating than on the CBC MAC. But in any case, these attacks show that neither CBC nor cascade should be used directly as MACs.

From prefix-free secure PRF to fully secure PRF: CMAC

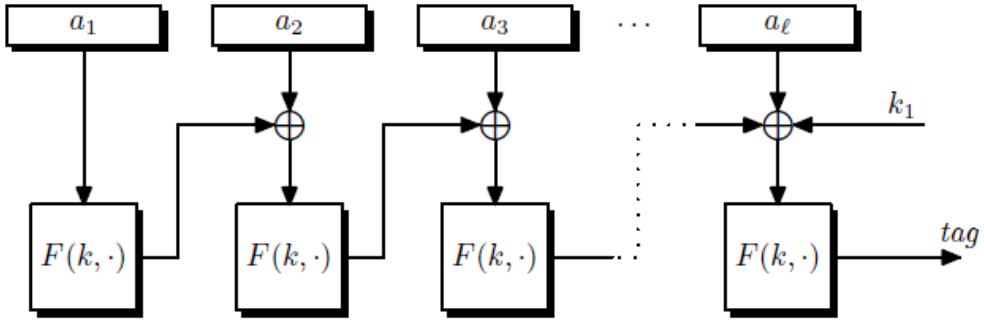
Using *rpf*. Let PF be a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$ and $rpf : \mathcal{K}_1 \times \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ be a randomized prefix-free encoding. Define the derived PRF F as

$$F\big((k, k_1),\ m\big) := PF\big(k, rpf(k_1, m)\big). \tag{6.21}$$

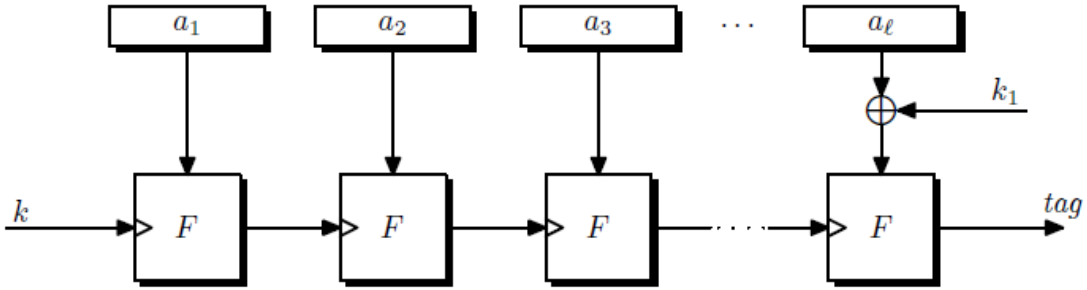
Then F is defined over $(\mathcal{K} \times \mathcal{K}_1, \mathcal{M}, \mathcal{Y})$. We obtain the following theorem, which is analogous to Theorem 6.8.

Theorem 6.9. *If PF is a prefix-free secure PRF, ϵ is negligible, and rpf a randomized ϵ-prefix-free encoding, then F defined in (6.21) is a secure PRF.*

Secure PRFs using random prefix-free encodings



(a) *rpf* applied to CBC



(b) *rpf* applied to cascade

Message integrity from universal hashing

- We describe a general paradigm for constructing MACs using hash functions.
- By a hash function we generally mean a function H that maps inputs in some large set M to short outputs in T .
- Elements in T are often called message digests or just digests.
- Keyed hash functions, used throughout this chapter, also take as input a key k.

Universal hash functions (UHFs)

- We begin our discussion by defining a keyed hash function - a widely used tool in cryptography.
- A keyed hash function H takes two inputs: a key k and a message m. It outputs a short digest t := H(k;m).
- The key k can be thought of as a hash function selector: for every k we obtain a specific function H(k; .) from messages to digests.
- More precisely, keyed hash functions are defined as follows:

Keyed hash function

Definition 7.1 (Keyed hash functions). *A **keyed hash function** H is a deterministic algorithm that takes two inputs, a **key** k and a **message** m; its output $t := H(k, x)$ is called a **digest**. As usual, there are associated spaces: the keyspace $\mathcal{K}$, in which k lies, a message space $\mathcal{M}$, in which m lies, and the digest space $\mathcal{T}$, in which t lies. We say that the hash function H is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$.*

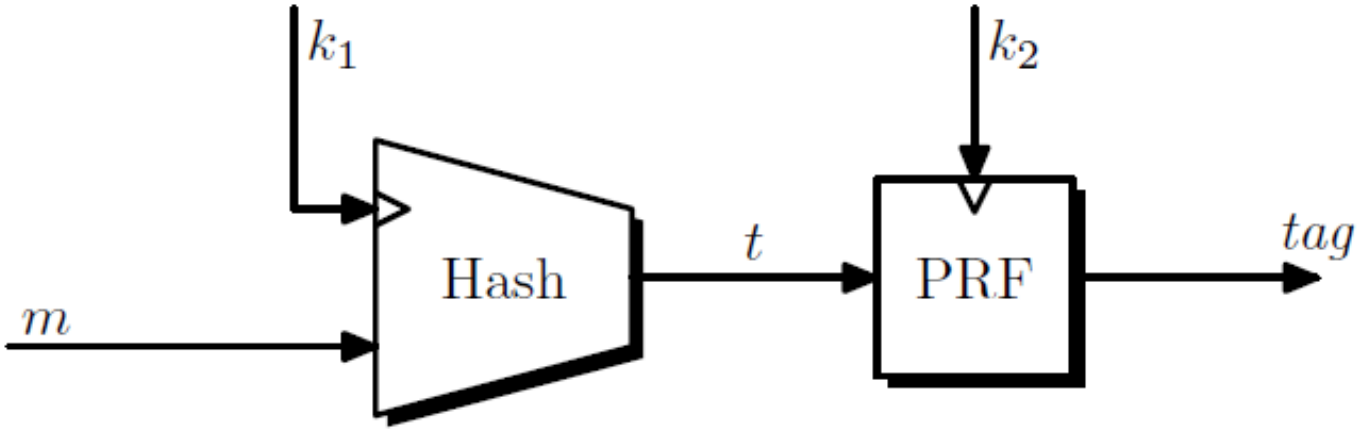
We note that the output digest $t \in \mathcal{T}$ can be much shorter than the input message m. Typically digests will have some fixed size, say 128 or 256 bits, independent of the input message length. A hash function $H(k, \cdot)$ can map gigabyte long messages into just 256-bit digests.

We say that two messages $m_0, m_1 \in \mathcal{M}$ form a **collision for H under key $k \in \mathcal{K}$** if

$$H(k,\ m_0) = H(k,\ m_1) \qquad \text{and} \qquad m_0 \neq m_1.$$

NYU

The hash-then-PRF paradigm



Universal hash functions (UHFs)

- We focus on the weakest formulation of this collision resistance property, in which the adversary must find a collision with no information about the key at all.
  - On the one hand, this property is weak enough that we can actually build very efficient hash functions that satisfy this property without making any assumptions at all on the computational power of the adversary.
  - On the other hand, this property is strong enough to ensure that the hash-then-PRF paradigm yields a secure MAC.
- Hash functions that satisfy this very weak collision resistance property are called universal hash functions, or UHFs.

Attack Game 7.1 (universal hash function)

Attack Game 7.1 (universal hash function). For a keyed hash function H defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$ and keeps k to itself.
- $\mathcal{A}$ outputs two distinct messages $m_0, m_1 \in \mathcal{M}$.

We say that $\mathcal{A}$ wins the above game if $H(k, m_0) = H(k, m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\mathsf{UHFadv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

Constructing UHFs using polynomials

We start with a UHF construction using polynomials modulo a prime. Let ℓ be a (poly-bounded) length parameter and let p be a prime. We define a hash function H_{poly} that hashes a message $m \in \mathbb{Z}_p^{\leq \ell}$ to a single element $t \in \mathbb{Z}_p$. The key space is $\mathcal{K} := \mathbb{Z}_p$.

Let m be a message, so $m = (a_1, a_2, \ldots, a_v) \in \mathbb{Z}_p^{\leq \ell}$ for some $0 \leq v \leq \ell$. Let $k \in \mathbb{Z}_p$ be a key. The hash function $H_{\text{poly}}(k, m)$ is defined as follows:

$$H_{\text{poly}}\big(k,\ (a_1, \ldots, a_v)\big) := k^v + a_1 k^{v-1} + a_2 k^{v-2} + \cdots + a_{v-1} k + a_v \in \mathbb{Z}_p \tag{7.3}$$

Message integrity from collision resistant hashing

- UHFs are keyed hash functions for which finding collisions is difficult, as long as the key is kept secret.
- A keyless function is an efficiently computable function whose description is fully public. There are no secret keys and anyone can evaluate the function.

Message integrity from collision resistant hashing

- Let H be a keyless hash function from some large message space M into a small digest space T .
- As in the previous section, we say that two messages $m_0, m_1 \in \mathbf{M}$ are a **collision** for the function H if

$$H(m_0) = H(m_1) \text{ and } m_0 \neq m_1 :$$

- Informally, we say that the function H is **collision resistant** if finding a collision for H is difficult.

SHA256

- To give an example of a collision resistant function we mention a US federal standard called the Secure Hash Algorithm standard or SHA for short.
- The SHA standard describes a number of hash functions over varying degrees of collision resistance.
    - For example, SHA256 is a function that hashes long messages into 256-bit digests. It is believed that finding collisions for SHA256 is difficult.

Definition of collision resistant hashing

A **(keyless) hash function** $H : \mathcal{M} \to \mathcal{T}$ is an efficiently computable function from some (large) message space $\mathcal{M}$ into a (small) digest space $\mathcal{T}$. We say that H is defined over $(\mathcal{M}, \mathcal{T})$. We define collision resistance of H using the following (degenerate) game:

Attack Game 8.1 (Collision Resistance). For a given hash function H defined over $(\mathcal{M}, \mathcal{T})$ and adversary $\mathcal{A}$, the adversary takes no input and outputs two messages m_0 and m_1 in $\mathcal{M}$.

We say that $\mathcal{A}$ wins the game if the pair m_0, m_1 is a collision for H, namely $m_0 \neq m_1$ and $H(m_0) = H(m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\mathsf{CRadv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. Adversary $\mathcal{A}$ is called a **collision finder**. □

Definition 8.1. *We say that a hash function H over $(\mathcal{M}, \mathcal{T})$ is **collision resistant** if for all efficient adversaries $\mathcal{A}$, the quantity $\mathsf{CRadv}[\mathcal{A}, H]$ is negligible.*

Building a MAC for large messages

To exercise the definition of collision resistance, we begin with an easy application described in the introduction — extending the message space of a MAC. Suppose we are given a secure MAC $\mathcal{I} = (S, V)$ for short messages. Our goal is to build a new secure MAC $\mathcal{I}'$ for much longer messages. We do so using a collision resistant hash function: $\mathcal{I}'$ computes a tag for a long message m by first hashing m to a short digest and then applying $\mathcal{I}$ to the digest, as shown in Fig. 8.1.

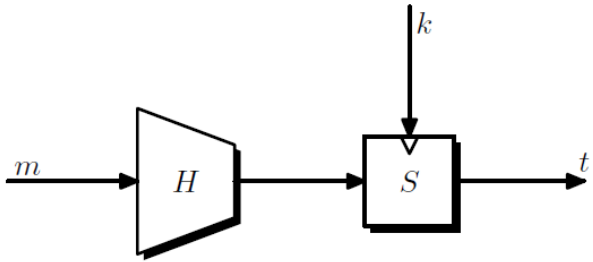


Figure 8.1: Hash-then-MAC construction

Hash-then-MAC construction

More precisely, let H be a hash function that hashes long messages in $\mathcal{M}$ to short digests in $\mathcal{T}_H$. Suppose $\mathcal{I}$ is defined over $(\mathcal{K}, \mathcal{T}_H, \mathcal{T})$. Define $\mathcal{I}' = (S', V')$ for long messages as follows:

$$S'(k, m) := S\big(k,\ H(m)\big) \quad \text{and} \quad V'(k, m, t) := V\big(k,\ H(m),\ t\big) \tag{8.1}$$

Then $\mathcal{I}'$ authenticates long messages in $\mathcal{M}$. The following easy theorem shows that $\mathcal{I}'$ is secure, assuming H is collision resistant.

Theorem 8.1. *Suppose the MAC system $\mathcal{I}$ is a secure MAC and the hash function H is collision resistant. Then the derived MAC system $\mathcal{I}' = (S', V')$ defined in (8.1) is a secure MAC.*

The Merkle-Damgard paradigm

Let $h : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$ be a hash function. We shall assume that $\mathcal{Y}$ is of the form $\{0,1\}^\ell$ for some ℓ. While it is not necessary, typically $\mathcal{X}$ is of the form $\{0,1\}^n$ for some n. The **Merkle-Damgård function derived from** h, denoted H_{MD} and shown in Fig. 8.5, is a hash function defined over $(\{0,1\}^{\leq L},\ \mathcal{X})$ that works as follows (the pad PB is defined below):

input: $M \in \{0,1\}^{\leq L}$
output: a tag in $\mathcal{X}$

$\hat{M} \leftarrow M \,\|\, \text{PB}$ // *pad with PB to ensure that the length of* M *is a multiple of* ℓ *bits*
partition $\hat{M}$ into consecutive ℓ-bit blocks so that

$$\hat{M} = m_1 \,\|\, m_2 \,\|\, \cdots \,\|\, m_s \quad \text{where} \quad m_1, \ldots, m_s \in \{0,1\}^\ell$$

$t_0 \leftarrow \text{IV} \in \mathcal{X}$
for $i = 1$ to s do:
    $t_i \leftarrow h(t_{i-1}, m_i)$
output t_s

The function SHA256 is a Merkle-Damgård function where $\ell = 512$ and $n = 256$.

The Merkle-Damgard paradigm

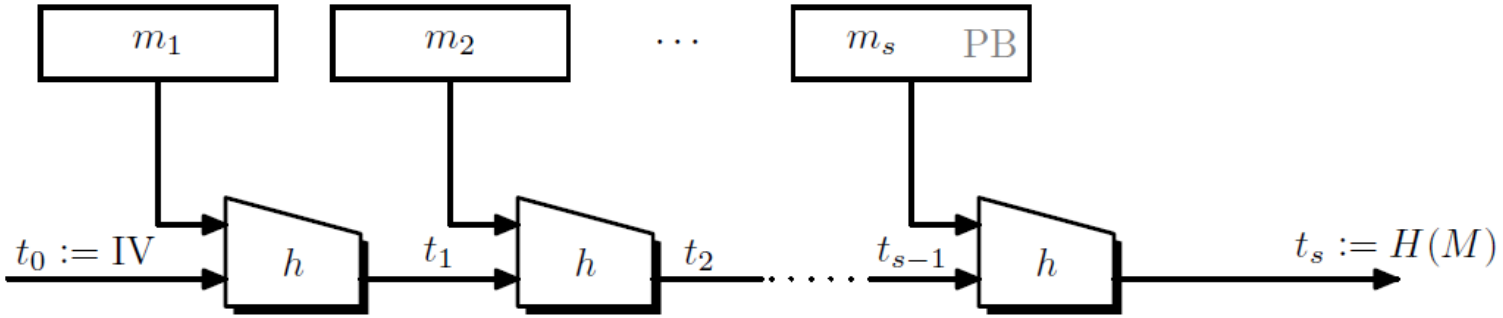


Figure 8.5: The Merkle-Damgård iterated hash function

Theorem 8.3 (Merkle-Damgård). *Let L be a poly-bounded length parameter and let h be a collision resistant hash function defined over $(\mathcal{X} \times \mathcal{Y},\ \mathcal{X})$. Then the Merkle-Damgård hash function H_{MD} derived from h, defined over $(\{0,1\}^{\leq L}, \mathcal{X})$, is collision resistant.*

NYU

Introduction of the elements in Fig. 8.5

- The hash function h is called the **compression function** of H.
- The constant IV is called the **initial value** and is fixed to some pre-specified value. One could take IV $= 0^n$, but usually the IV is set to some complicated string. For example, SHA256 uses a 256-bit IV whose value in hex is

$$\text{IV} := \text{6A09E667 BB67AE85 3C6EF372 A54FF53A 510E527F 9B05688C 1F83D9AB 5BE0CD19.}$$

- The variables $m_1, \ldots, m_s$ are called message blocks.
- The variables $t_0, t_1, \ldots, t_s \in \mathcal{X}$ are called **chaining variables**.
- The string PB is called the **padding block**. It is appended to the message to ensure that the message length is a multiple of ℓ bits.

The padding block PB must contain an encoding of the input message length. We will use this in the proof of security below. A standard format for PB is as follows:

$$\text{PB} := \boxed{100\ldots00 \parallel \langle s \rangle}$$

where $\langle s \rangle$ is a fixed-length bit string that encodes, in binary, the number of ℓ-bit blocks in M.

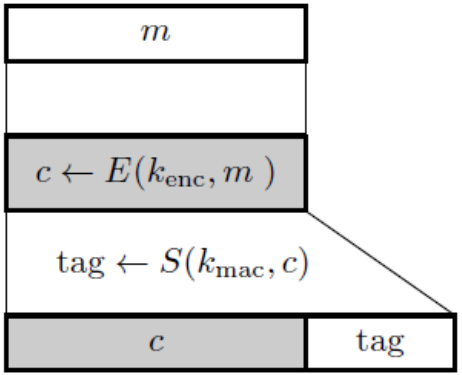
Authenticated Encryption

- Here we construct systems that ensure both data secrecy (confidentiality) and data integrity, even against very aggressive attackers that can interact maliciously with both the sender and the receiver.
- Such systems are said to provide authenticated encryption or are simply said to be AE-secure.

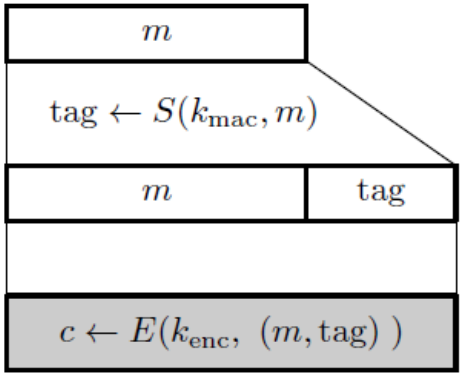
Two methods to combine encryption and MAC

Encrypt-then-MAC Encrypt the message, $c \xleftarrow{R} E(k_{enc}, m)$, then MAC the ciphertext, tag $\xleftarrow{R} S(k_{mac}, c)$; the result is the ciphertext-tag pair (c, tag). This method is supported in the TLS 1.2 protocol and later versions as well as in the IPsec protocol and in a widely-used NIST standard called GCM (see Section 9.7).

MAC-then-encrypt MAC the message, tag $\xleftarrow{R} S(k_{mac}, m)$, then encrypt the message-tag pair, $c \xleftarrow{R} E\big(k_{enc},\ (m, t)\big)$; the result is the ciphertext c. This method is used in older versions of TLS (e.g., SSL 3.0 and its successor called TLS 1.0) and in the 802.11i WiFi encryption protocol.



encrypt-then-mac



mac-then-encrypt

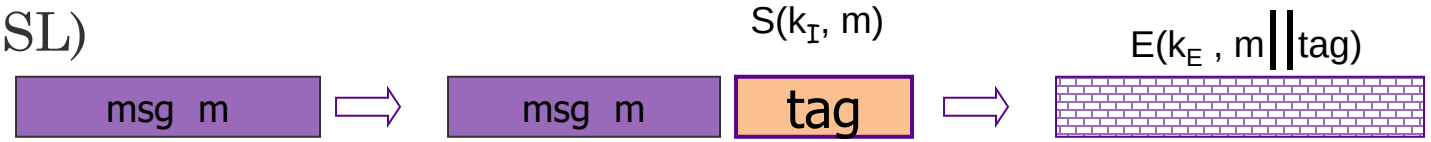
Two methods to combine encryption and MAC

- As it turns out, only the first method is secure for every combination of CPA-secure cipher and secure MAC. The intuition is that the MAC on the ciphertext prevents any tampering with the ciphertext.
- The second method can be insecure the MAC- and cipher can interact badly and cause the resulting system to not be AE-secure. This has led to many attacks on widely deployed systems.

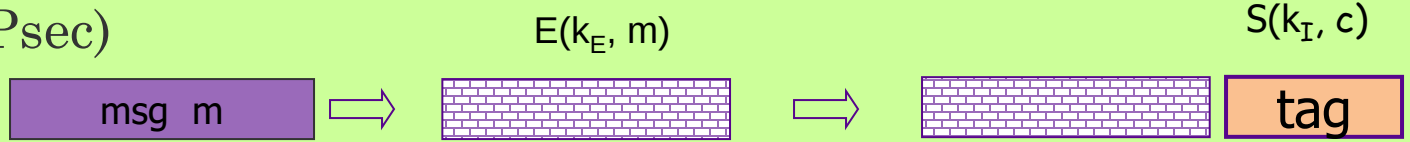
Combining MAC and Encryption

Encryption key k_E MAC key = k_I

Option 1: (SSL)

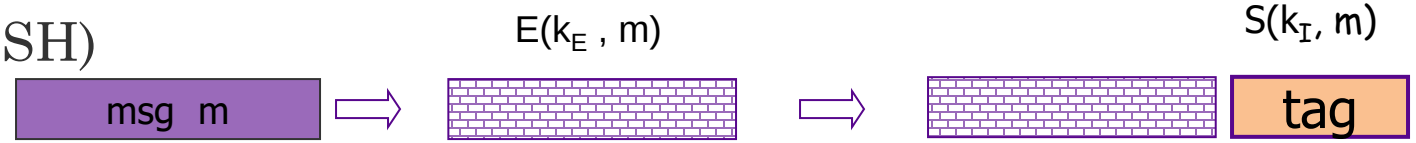


Option 2: (IPsec)



Always correct

Option 3: (SSH)



NYU