Cryptography

Corso di Laurea Magistrale in Informatica

Private-Key Authentication Schemes

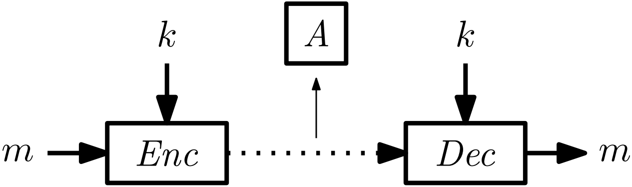
Ugo Dal Lago

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

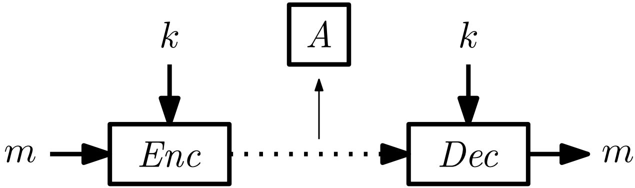
informatiques mathématiques Inria

Academic Year 2023-2024

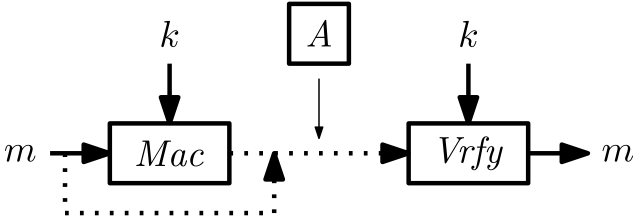
Secrecy



Secrecy



Authentication



Authentication

- We want to guarantee that the receiver is sure of the **integrity** and **authenticity** of the received message.
- Authentication and secrecy are distinct properties: solving the problem of secrecy does not necessarily automatically solve the problem of authentication.
  - If we use Π^G, where G is a pseudorandom generator, a hypothetical adversary could easily construct, from $Enc(k, m) = G(k) \oplus m$, the ciphertext for a message exactly similar to m.
  - Attacks similar to the previous one also affect CBC (where a modification of IV affects the first block) and also CTR (where a change to block i only affects block i).

Message Authentication Codes

- The same way ciphers are the tool to guarantee secrecy, *message authentication codes* (or *MAC*) are the tool to solve the authentication problem.
- A MAC is a triple $\Pi = (\mathit{Gen}, \mathit{Mac}, \mathit{Vrfy})$ of PPT algorithms such that
  - Gen takes as input a string in the form 1^n, to be interpreted as the security parameter, and outputs a key k, which can be such that $|k| \geq n$.
  - The Mac algorithm takes as input a key k and a message m, and outputs a *tag* t.
  - The Vrfy algorithm takes as input a key k, a message m and a tag t, and outputs a boolean b.
- A MAC $\Pi = (\mathit{Gen}, \mathit{Mac}, \mathit{Vrfy})$ is *correct* when $\mathit{Vrfy}(k, m, \mathit{Mac}(k, m)) = m$.

Defining Security of a MAC

- The question, as in the case of secrecy, is: what do we want an adversary **not to be able** to do?

Defining Security of a MAC

- The question, as in the case of secrecy, is: what do we want an adversary **not to be able** to do?
- The answer here is quite simple: we don't want the adversary to be able to *forge* (i.e. to produce a valid tag for) a message of his choice m, without knowing the key k.
- In this case, we are very pessimistic from the beginning and assume that:
  - The adversary has access to an oracle for $Mac_k(\cdot)$.
  - Instead, a pair (m, t) obtained through access to the oracle is not to be considered a correct forging.
  - The adversary has to be, as usual, a PPT algorithm.

Defining Security of a MAC

- Let us proceed, as usual, by giving a suitable notion of experiment:

$$
\begin{array}{l}
\mathsf{MacForge}_{A,\Pi}(n): \\
k \leftarrow Gen(1^n); \\
(m,t) \leftarrow A(1^n, Mac_k(\cdot)); \\
\mathbb{Q} \leftarrow \{m \mid A \text{ queries } Mac_k(\cdot) \text{ on } m\}; \\
\textbf{Result: } (m \notin \mathbb{Q} \wedge Vrfy(k,m,t) = 1)
\end{array}
$$

Defining Security of a MAC

- Let us proceed, as usual, by giving a suitable notion of experiment:

$$\begin{array}{l}\mathsf{MacForge}_{A,\Pi}(n): \\ k \leftarrow Gen(1^n); \\ (m,t) \leftarrow A(1^n, Mac_k(\cdot)); \\ \mathbb{Q} \leftarrow \{m \mid A \text{ queries } Mac_k(\cdot) \text{ on } m\}; \\ \textbf{Result: } (m \notin \mathbb{Q} \wedge Vrfy(k,m,t) = 1)\end{array}$$

Definition

A MAC Π is *secure* iff for every PPT adversary A there exists a negligible function $\varepsilon \in \mathcal{NGL}$ such that

$$Pr(\mathsf{MacForge}_{\Pi,A}(n) = 1) = \varepsilon(n)$$

Some Comments on the Definition

- From a certain point of view, the definition is **very strong**:
  - Concretely, an adversary is interested in forging *meaningful* messages.
  - Here we consider it problematical that the adversary forges any message.
- From another point of view, the definition would seem a bit **weak**:
  - There is a class of attacks that our definition does not take into account, namely the *replay attacks*.
  - In such attacks, the same pair (m, t) is sent multiple times over the channel, the first time by a legitimate user, the others by the adversary.
  - In practice, replay attacks are handled at a higher level in the stack, through mechanisms such as *sequence numbers* or *timestamps*.

Constructing a Secure MAC

- The first example of Secure MAC that we will give is based on a notion we are already familiar with, namely that of Pseudorandom Function.

Constructing a Secure MAC

- The first example of Secure MAC that we will give is based on a notion we are already familiar with, namely that of Pseudorandom Function.

Definition (PRF induced MAC)

Given a PRF F, the MAC $\Pi^F = (Gen, Mac, Vrfy)$ is defined as follows:

- The algorithm Gen for input 1^n outputs every string long n with same probability, i.e. $\frac{1}{2^n}$.
- $Mac(k, m) = F_k(m)$.
- $Vrfy(k, m, t) = (F_k(m) \stackrel{?}{=} t)$.

Constructing a Secure MAC

- The first example of Secure MAC that we will give is based on a notion we are already familiar with, namely that of Pseudorandom Function.

Definition (PRF induced MAC)

Given a PRF F, the MAC $\Pi^F = (Gen, Mac, Vrfy)$ is defined as follows:

- The algorithm Gen for input 1^n outputs every string long n with same probability, i.e. $\frac{1}{2^n}$.
- $Mac(k, m) = F_k(m)$.
- $Vrfy(k, m, t) = (F_k(m) \stackrel{?}{=} t)$.

Theorem

If F is pseudorandom, then the MAC Π^F is secure.

Handling Variable-Length Messages

- The Π^F construction allows messages as long as the keys to be handled.
  - Once again, therefore, we have a scheme with great security properties, but not very useful.
- Given a message $m = m_1 || \cdots || m_n$, we could try to proceed as follows:
  1. Authenticate $\oplus_{i=1}^{n} m_i$. In this case, however, an adversary would easily succeed in forging a message $p = p_1 || \cdots || p_n$ where $\oplus_{i=1}^{n} m_i = \oplus_{i=1}^{n} p_i$.

Handling Variable-Length Messages

- The Π^F construction allows messages as long as the keys to be handled.
  - Once again, therefore, we have a scheme with great security properties, but not very useful.
- Given a message $m = m_1 || \cdots || m_n$, we could try to proceed as follows:
  1. Authenticate $\oplus_{i=1}^n m_i$. In this case, however, an adversary would easily succeed in forging a message $p = p_1 || \cdots || p_n$ where $\oplus_{i=1}^n m_i = \oplus_{i=1}^n p_i$.
  2. Authenticate *each block* m_i *separately*, and then take the xor: $Mac(k, m) = \oplus_{i=1}^n Mac(k, m_i)$. In this case, however, an adversary could easily forge the message $p = m_{\pi(1)} \cdots m_{\pi(n)}$, where π is a permutation.

Handling Variable-Length Messages

- The Π^F construction allows messages as long as the keys to be handled.
  - Once again, therefore, we have a scheme with great security properties, but not very useful.
- Given a message $m = m_1 || \cdots || m_n$, we could try to proceed as follows:
  1. Authenticate $\oplus_{i=1}^n m_i$. In this case, however, an adversary would easily succeed in forging a message $p = p_1 || \cdots || p_n$ where $\oplus_{i=1}^n m_i = \oplus_{i=1}^n p_i$.
  2. Authenticate *each block* m_i *separately*, and then take the xor: $Mac(k, m) = \oplus_{i=1}^n Mac(k, m_i)$. In this case, however, an adversary could easily forge the message $p = m_{\pi(1)} \cdots m_{\pi(n)}$, where π is a permutation.
  3. Authenticate each block m_i separately *with* the sequence number i, i.e. $Mac(k, m) = \oplus_{i=1}^n Mac(k, m_i || i)$. Even in this case, however, the adversary could easily forge other messages.

Handling Variable-Length Messages

- Different ideas need to be put together, i.e block division, sequence numbers and randomization.
- Given a MAC $\Pi = (Gen, Mac, Vrfy)$ for fixed-length messages, we can construct a new MAC $\Pi^* = (Gen, Mac^*, Vrfy^*)$ for variable-length messages as follows:

$Mac^*(k, m)$:
$m_1 || \cdots || m_d \leftarrow m$;
`/* such that` $|m_i| = \frac{n}{4}$ `*/`
$\ell \leftarrow |m|$;
$r \leftarrow \{0,1\}^{\frac{n}{4}}$;
for $i \leftarrow 1$ **to** d **do**
  $t_i \leftarrow Mac(k, r||\ell||i||m_i)$
Result: $(r, t_1, \ldots, t_d)$

$Vrfy^*(k, m, (r, t_1, \ldots, t_d))$:
$m_1 || \cdots || m_d \leftarrow m$;
`/* such that` $|m_i| = \frac{n}{4}$ `*/`
$\ell \leftarrow |m|$;
for $i \leftarrow 1$ **to** d **do**
  if $Vrfy(k, r||\ell||i||m_i) = 0$ **then**
    Result: 0
Result: 1

Handling Variable-Length Messages

- Different ideas need to be put together, i.e block division, sequence numbers and randomization.
- Given a MAC $\Pi = (Gen, Mac, Vrfy)$ for fixed-length messages, we can construct a new MAC $\Pi^* = (Gen, Mac^*, Vrfy^*)$ for variable-length messages as follows:

$Mac^*(k, m)$:
$m_1 || \cdots || m_d \leftarrow m$;
`/* such that` $|m_i| = \frac{n}{4}$ `*/`
$\ell \leftarrow |m|$;
$r \leftarrow \{0,1\}^{\frac{n}{4}}$;
for $i \leftarrow 1$ **to** d **do**
  $t_i \leftarrow Mac(k, r||\ell||i||m_i)$
Result: $(r, t_1, \ldots, t_d)$

$Vrfy^*(k, m, (r, t_1, \ldots, t_d))$:
$m_1 || \cdots || m_d \leftarrow m$;
`/* such that` $|m_i| = \frac{n}{4}$ `*/`
$\ell \leftarrow |m|$;
for $i \leftarrow 1$ **to** d **do**
  if $Vrfy(k, r||\ell||i||m_i) = 0$ **then**
    Result: 0
Result: 1

Theorem

If Π is secure, then Π^* is also secure.

CBC-MAC Construction

- The construction we have just seen, when applied to a PRF F, calls the latter $4d$ times, while the tag is $4dn$ long.
- This can be avoided by the CBC-MAC construction, also based on a PRF F, but defined differently and parameterized on ℓ.

$Mac^{CBC}(k, m)$:
$\ell \leftarrow \ell(|k|)$;
$m_1 || \cdots || m_\ell \leftarrow m$;
`/* such that` $|m_i| = |k|$ `*/`
$t_0 \leftarrow 0^n$;
for $i \leftarrow 1$ **to** ℓ **do**
  $t_i \leftarrow F_k(t_{i-1} \oplus m_i)$
Result: t_ℓ

$Vrfy^{CBC}(k, m, t)$:
if $\ell(|k|) \cdot |k| \neq |m|$ **then**
  Result: 0
Result: $t \stackrel{?}{=} Mac^{CBC}(k, m)$

CBC-MAC Construction

- The construction we have just seen, when applied to a PRF F, calls the latter $4d$ times, while the tag is $4dn$ long.
- This can be avoided by the CBC-MAC construction, also based on a PRF F, but defined differently and parameterized on ℓ.

$Mac^{CBC}(k, m)$:
$\ell \leftarrow \ell(|k|)$;
$m_1 || \cdots || m_\ell \leftarrow m$;
`/* such that` $|m_i| = |k|$ `*/`
$t_0 \leftarrow 0^n$;
for $i \leftarrow 1$ **to** ℓ **do**
  $t_i \leftarrow F_k(t_{i-1} \oplus m_i)$
Result: t_ℓ

$Vrfy^{CBC}(k, m, t)$:
if $\ell(|k|) \cdot |k| \neq |m|$ **then**
  Result: 0
Result: $t \stackrel{?}{=} Mac^{CBC}(k, m)$

Theorem

If ℓ is a polynomial and F is a PRF, then Π^{CBC} is a secure MAC.

Hash Functions

- *Hash functions* are functions that compress long strings into shorter ones, so that there are as few collisions as possible.
  - A collision for a hash function H is a pair (x, y) such that $H(x) = H(y)$ with $x \neq y$.
  - Hash functions are used for the design of data structures, but also in cryptography

Hash Functions

- *Hash functions* are functions that compress long strings into shorter ones, so that there are as few collisions as possible.
  - A collision for a hash function H is a pair (x, y) such that $H(x) = H(y)$ with $x \neq y$.
  - Hash functions are used for the design of data structures, but also in cryptography
- In cryptography, we require something more from hash functions, namely that:
  1. Collisions are not only *as few as* possible, but in some way *impossible* to determine.
  2. The impossibility of determining collisions must also be valid for adversaries built specifically to find them.
- More specifically, a hash function is seen as a function $H : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that:
  - The first parameter is a key s, which however is made public.
  - The second is a string x, which the function H tries to compress.

Hash Functions

Definition

A *hash function* is a pair of PPT algorithms (Gen, H) such that:

- Gen is an algorithm that takes as input a security parameter 1^n and returns a key s (from which n can be efficiently calculated).
- There exists a polynomial ℓ such that $H(s, x)$ returns a string of length $\ell(n)$ (where n is the implicit parameter in s).

- If there exists a polynomial p such that $p(n) > \ell(n)$, for every n, and $H(s, x)$ is defined only when $|x| = p(n)$ (and s is the parameter implicit in s), then H is called a *fixed-length* hash function.
- As usual, we denote H^s the function that, for input x, returns $H(s, x)$

Collision-Resistant Hash Functions

- Hash functions can be defined to be secure in different ways. The strongest, most restrictive notion is based on the following experiment:

$\mathsf{HashColl}_{A,\Pi}(n)$:
$s \leftarrow Gen(1^n)$;
$(x, y) \leftarrow A(s)$;
Result: $(x \neq y) \wedge (H(x) = H(y))$

Definition

A hash function $\Pi = (Gen, H)$ is *collision-resistant* iff for every adversary PPT A there exists a negligible function ε such that

$$Pr(\mathsf{HashColl}_{A,\Pi}(n) = 1) \leq \varepsilon(n)$$

Weaker Notions of Security

- **Second Preimage Resistance**
  - Given s and x, it must be impossible for A to construct $y \neq x$ such that $H^s(x) = H^s(y)$.
- **Preimage Resistance**
  - Given s and $z = H^s(x)$, it must be impossible for A to construct y such that $H^s(y) = z$.
- The three notions of security we have just seen are progressively weaker.
  - A few proofs by reduction are sufficient to realize this.

Birthday Attacks

- The question is how to attack "brute force" a hash function, and what is the complexity of this attack.
- Let us consider a situation in which the key s is already fixed and in which the adversary A wants to find a collision in $H^s : \{0,1\}^* \to \{0,1\}^\ell$.

Birthday Attacks

- The question is how to attack "brute force" a hash function, and what is the complexity of this attack.
- Let us consider a situation in which the key s is already fixed and in which the adversary A wants to find a collision in $H^s : \{0,1\}^* \to \{0,1\}^\ell$.
- The adversary could simply randomly choose q strings in $\{0,1\}^*$ and test H^s on each of them, hoping to find two for which H^s returns the same value.
  - If $q > 2^\ell$, a collision is assured.
  - What if, on the other hand, q is much smaller 2^ℓ?

Theorem (Birthday Theorem)

Given q uniformly chosen random values in a finite set of cardinality N, the probability that two of them are identical is $\Theta(\frac{q^2}{N})$.

Birthday Attacks

- If we assume that the behaviour of H^s is as close as possible to that of a random function (i.e. if we are in the worst case), we can then conclude that a birthday attack where $q = \Theta(2^{\frac{\ell}{2}})$ will have probability of success

$$\Theta\left(\frac{(2^{\frac{\ell}{2}})^2}{2^\ell}\right) = \Theta\left(\frac{2^\ell}{2^\ell}\right) = \Theta(1)$$

  i.e. constant probability, independent of q and ℓ.
- Birthday attacks can be improved in two ways:
  - **Reducing Complexity in Space**
  - **Making the Attack Context-Dependent**

The Merkle-Damgård Transform

- Is it possible for a fixed-length hash function to handle arbitrary-length messages?
  - ... while remaining collision-resistant.
- The answer is yes, and is called **Merkle-Damgård transformation**.
- Suppose that (Gen, H) is a hash function for messages of length $p(n) = 2n$ (and the output is n long) and construct from it (Gen, H^{MD}) as follows

$H^{MD}(s, x)$:
$B \leftarrow \lceil |x|/n \rceil$;
$x_1 || \cdots || x_B \leftarrow x$;
`/* such that` $|x_i| = n$ `*/`
$x_{B+1} \leftarrow |x|$;
$z_0 \leftarrow 0^n$;
for $i \leftarrow 1$ **to** $B + 1$ **do**
  $z_i \leftarrow H(s, z_{i-1} || x_i)$
Result: z_{B+1}

Theorem

If (Gen, H) is collision-resistant, then so is (Gen, H^{MD}).

Hash-and-Mac

- How can hash functions be useful in the construction of MACs?
- A very interesting way of using (collision-resistant) hash functions is to construct a MAC that is secure for variable-length messages from one which is only secure for fixed-length messages.

Hash-and-Mac

- How can hash functions be useful in the construction of MACs?
- A very interesting way of using (collision-resistant) hash functions is to construct a MAC that is secure for variable-length messages from one which is only secure for fixed-length messages.
- Given a MAC $\Pi = (Gen, Mac, Vrfy)$ and a hash function (Gen', H), we define the MAC $\Pi^H = (Gen^H, Mac^H, Vrfy^H)$ as follows
  - Gen^H on input 1^n outputs (the encoding of) a pair (s, k) where s is the result of $Gen'(1^n)$ and k is the result of $Gen(1^n)$.
  - $Mac^H((s, k), m)$ outputs $Mac_k(H^s(m))$.
  - As usual, $Vrfy((s, k), m, t)$ outputs 1 iff $Mac^H((s, k), m) = t$.

Hash-and-Mac

- How can hash functions be useful in the construction of MACs?
- A very interesting way of using (collision-resistant) hash functions is to construct a MAC that is secure for variable-length messages from one which is only secure for fixed-length messages.
- Given a MAC $\Pi = (Gen, Mac, Vrfy)$ and a hash function (Gen', H), we define the MAC $\Pi^H = (Gen^H, Mac^H, Vrfy^H)$ as follows
  - Gen^H on input 1^n outputs (the encoding of) a pair (s, k) where s is the result of $Gen'(1^n)$ and k is the result of $Gen(1^n)$.
  - $Mac^H((s, k), m)$ outputs $Mac_k(H^s(m))$.
  - As usual, $Vrfy((s, k), m, t)$ outputs 1 iff $Mac^H((s, k), m) = t$.

Theorem

If Π is a secure MAC and (Gen', H) is collision-resistant, then Π^H is secure.