Introduction à la Sécurité
Cryptologie symétrique 2/2

Charles Bouillaguet
(slides de Damien Vergnaud)

6 février 2020
Contents

1 AES
   - Origins and Structure
   - Description

2 Hash Functions
   - Definitions and Generic Attacks
   - Merkle-Damgaard
   - MD5 and SHA-?

3 Message Authentication Codes (MAC)
   - Definitions
   - CBC-MAC
   - HMAC
AES Origins

- a **replacement** for DES was needed
  - theoretical attacks that can break it
  - exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks

- US NIST issued call for ciphers in 1997
  - **Block size:** 128 bits (possibly 64, 256, ...)
  - **Key size:** 128, 192, 256 bits

- 15 candidates accepted in June 98
- 5 were shortlisted in August 99
- **Rijndael** was selected as the AES in October 2000
- issued as FIPS PUB 197 standard in November 2001
AES Origins

- a **replacement** for DES was needed
  - theoretical attacks that can break it
  - exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks

- **US NIST** issued call for ciphers in 1997
  - **Block size:** 128 bits (possibly 64, 256, ...)
  - **Key size:** 128, 192, 256 bits

- 15 candidates accepted in June 98
- 5 were shortlisted in August 99
- **Rijndael** was selected as the AES in October 2000
- issued as FIPS PUB 197 standard in November 2001
AES Origins

- a **replacement** for DES was needed
  - theoretical attacks that can break it
  - exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks

US NIST issued call for ciphers in 1997
- **Block size**: 128 bits (possibly 64, 256, ...)
- **Key size**: 128, 192, 256 bits

- 15 candidates accepted in June 98
- 5 were shortlisted in August 99
- **Rijndael** was selected as the AES in October 2000
- issued as FIPS PUB 197 standard in November 2001
Rijndael — the Advanced Encryption Standard

- Designed by Rijmen and Daemen
- Winner of AES competition in 2001
- One of the **most widely used** encryption primitive

**AES basic structures**

- Substitution-Permutation network
- Block size: 128 bits
- Key lengths: **128**, 192 or 256 bits
- 10 rounds for the 128-bit version

Resistance against known attacks, Speed and code compactness on many CPUs, Design simplicity.
Substitution-Permutation Network

- to provide **Confusion** and **Diffusion** (Shannon)

**Substitution:** S-boxes substitute a small block of input bits into output bits
  - invertible, non-linear
  - changing one input bit $\rightarrow$ change about half of the output bits

**Permutation:** P-boxes permute bits for the next-round S-box inputs
  - output bits of an S-box distributed to as many S-box inputs as possible.

**Key:** in each round using group operation ($\oplus$)

- one S-box/P-box produces a *limited* amount of confusion/diffusion
- enough *rounds* $\rightarrow$ every input bit is diffused across every output bit
Algebraic Structure in the AES

- **Data block:** 128 bits $\rightsquigarrow$ 16 bytes in a $4 \times 4$ matrix

  
  \[
  \begin{array}{cccc}
  1 & 2 & 3 & 4 \\
  5 & 6 & 7 & 8 \\
  9 & 10 & 11 & 12 \\
  13 & 14 & 15 & 16 \\
  \end{array}
  \]

  A byte $b_7b_6b_5b_4b_3b_2b_1b_0$ is represented by a polynomial

  \[
  b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0
  \]

  with $b_i \in \{0, 1\} = \mathbb{F}_2$.

- **Example:** $5A = 01011010$

  $\rightsquigarrow x^6 + x^4 + x^3 + x^1$

  Bytes are identified with elements of the **finite field** $\mathbb{F}_{256} = \mathbb{F}_2[x]/(m(x))$ with

  \[
  m(x) = x^8 + x^4 + x^3 + x + 1
  \]
Algebraic Structure in the AES

- **Data block:** 128 bits $\mapsto$ 16 bytes in a $4 \times 4$ matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

- A byte $b_7b_6b_5b_4b_3b_2b_1b_0$ is represented by a polynomial

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

with $b_i \in \{0, 1\} = \mathbb{F}_2$.

- **Example:** $5A = 01011010$

  $\mapsto x^6 + x^4 + x^3 + x^1$

- Bytes are identified with elements of the **finite field** $\mathbb{F}_{256} = \mathbb{F}_2[x]/(m(x))$

  with

  $$m(x) = x^8 + x^4 + x^3 + x + 1$$
Description of the AES

AES Round

$x_i$  $k_i$  $x_{i+1}$
Description of the AES

\[ x_i \rightarrow S \rightarrow x_{i+1} \]
Description of the AES

\[ x_i \rightarrow [\text{S-box}] \rightarrow x_{i+1} \]

Where \( k_i \) is the key.
Description of the AES
Description of the AES

\[ x_i \times M \rightarrow x_{i+1} \]

\[ k_i \]
Description of the AES

\[ x_i \xrightarrow{ } \text{AES} \xrightarrow{ } x_{i+1} \]
Description of the AES

Confusion

Diffusion

\[ x_i \]

\[ k_i \]

\[ x_{i+1} \]
AES Structure

Message Block

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 \\
\end{array} \]

SubBytes

\[ \begin{array}{cccc}
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\end{array} \]

ShiftRows

\[ \begin{array}{cccc}
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\end{array} \]

MixColumns

\[ \begin{array}{cccc}
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ \\
\end{array} \]

AddRoundKey

\[ K_0 \]

\[ K_i \]

\[ \text{no MixColumns in the last round} \]
SubBytes

- S-box defined algebraically over $\mathbb{F}_{256}$
- First invert the byte (interpreted as an element of $\mathbb{F}_{256}$):
  \[ a \mapsto \begin{cases} 
  a^{-1} & \text{if } a \neq 0 \\
  0 & \text{otherwise}
  \end{cases} \]
- Then apply affine transformation:

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  y_5 \\
  y_6 \\
  y_7 \\
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
  1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
  1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
  1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
  1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
  0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
  0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
  0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6 \\
  x_7 \\
\end{bmatrix} + \begin{bmatrix}
  1 \\
  1 \\
  0 \\
  0 \\
  0 \\
  0 \\
  1 \\
  0 \\
\end{bmatrix}\]
the column is determined by the least significant nibble,
the row is determined by the most significant nibble.

Example: $S(9A) = B8$
### ShiftRows

<table>
<thead>
<tr>
<th>No change</th>
<th>Shift 1</th>
<th>Shift 2</th>
<th>Shift 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{0,0}</td>
<td>a_{1,0}</td>
<td>a_{2,0}</td>
<td>a_{3,0}</td>
</tr>
<tr>
<td>a_{0,1}</td>
<td>a_{1,1}</td>
<td>a_{2,1}</td>
<td>a_{3,1}</td>
</tr>
<tr>
<td>a_{0,2}</td>
<td>a_{1,2}</td>
<td>a_{2,2}</td>
<td>a_{3,2}</td>
</tr>
<tr>
<td>a_{0,3}</td>
<td>a_{1,3}</td>
<td>a_{2,3}</td>
<td>a_{3,3}</td>
</tr>
</tbody>
</table>

**ShiftRows**

<table>
<thead>
<tr>
<th>No change</th>
<th>Shift 1</th>
<th>Shift 2</th>
<th>Shift 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{0,0}</td>
<td>a_{1,1}</td>
<td>a_{2,2}</td>
<td>a_{3,3}</td>
</tr>
<tr>
<td>a_{0,1}</td>
<td>a_{1,2}</td>
<td>a_{2,3}</td>
<td>a_{3,0}</td>
</tr>
<tr>
<td>a_{0,2}</td>
<td>a_{1,3}</td>
<td>a_{2,0}</td>
<td>a_{3,1}</td>
</tr>
<tr>
<td>a_{0,3}</td>
<td></td>
<td>a_{1,0}</td>
<td>a_{3,2}</td>
</tr>
</tbody>
</table>
MixColumns
MixColumns

- in the ring $A = \mathbb{F}_{256}[X]/(X^4 + 1)$.
- using $a(X)$ in $A$:

$$a(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\}$$
$$= (x + 1)X^3 + X^2 + X + x$$

- given a column of four bytes $(b_0, b_1, b_2, b_3)$, consider the polynomial

$$b_0 + b_1X + b_2X^2 + b_3X^3 \in A$$

- the new column is

$$a(X) \cdot b(X) \in A$$
MixColumns

\[ a(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\} \]
\[ = (x + 1)X^3 + X^2 + X + x \]

\[ b(X) = b_0 + b_1X + b_2X^2 + b_3X^3 \]

**Step 1: Polynomial multiplication**

\[ a(x) \cdot b(x) = c(x) = (a_3X^3 + a_2X^2 + a_1X + a_0) \cdot (b_3X^3 + b_2X^2 + b_1X + b_0) \]
\[ = c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0 \]

where:

\[ c_0 = a_0 \cdot b_0 \]
\[ c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \]
\[ c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \]
\[ c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 \]
\[ c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \]
\[ c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3 \]
\[ c_6 = a_3 \cdot b_3 \]
MixColumns

Step 2: Modular reduction

\[
\begin{align*}
X^6 \mod (X^4 + 1) &= -X^2 = X^2 \text{ over } \text{GF}(2^8) \\
X^5 \mod (X^4 + 1) &= -X = X \text{ over } \text{GF}(2^8) \\
X^4 \mod (X^4 + 1) &= -1 = 1 \text{ over } \text{GF}(2^8)
\end{align*}
\]

\[
a(X) \cdot b(X) = c(X) \mod (X^4 + 1)
\]

\[
= (c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0) \mod (X^4 + 1)
\]

\[
= c_6X^2 + c_5x + c_4 + c_3X^3 + c_2X^2 + c_1X + c_0
\]

\[
= c_3X^3 + (c_2 \oplus c_6)X^2 + (c_1 \oplus c_5)X + c_0 \oplus c_4
\]

\[
= d_3X^3 + d_2X^2 + d_1X + d_0
\]

where

\[
d_0 = c_0 \oplus c_4, \quad d_1 = c_1 \oplus c_5, \quad d_2 = c_2 \oplus c_6, \quad d_3 = c_3
\]
MixColumns

Step 3: Matrix representation

\[
\begin{align*}
  d_0 &= a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
  d_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
  d_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3 \\
  d_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 
\end{align*}
\]

\[
\begin{bmatrix}
  d_0 \\
  d_1 \\
  d_2 \\
  d_3 
\end{bmatrix} = \begin{bmatrix}
  \{02\} & \{03\} & \{01\} & \{01\} \\
  \{01\} & \{02\} & \{03\} & \{01\} \\
  \{01\} & \{01\} & \{02\} & \{03\} \\
  \{03\} & \{01\} & \{01\} & \{02\} 
\end{bmatrix} \begin{bmatrix}
  b_0 \\
  b_1 \\
  b_2 \\
  b_3 
\end{bmatrix}
\]

- **addition** is an XOR operation
- **multiplication** is a (complicated) multiplication in $\mathbb{F}_{256}$
MixColumns

Step 3: Matrix representation

\[
\begin{align*}
    d_0 &= a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
    d_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
    d_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3 \\
    d_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 \\
\end{align*}
\]

\[
\begin{bmatrix}
    d_0 \\
    d_1 \\
    d_2 \\
    d_3 \\
\end{bmatrix}
= 
\begin{bmatrix}
    x & (x + 1) & 1 & 1 \\
    1 & x & (x + 1) & 1 \\
    1 & 1 & x & (x + 1) \\
    (x + 1) & 1 & 1 & x \\
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    b_3 \\
\end{bmatrix}
\]

- **addition** is an XOR operation
- **multiplication** is a (complicated) multiplication in $\mathbb{F}_{256}$
MixColumns
Description of the AES: the Key-Schedule

\[ k_i \]

\[ k_{i+1} \]
Description of the AES: the Key-Schedule

$k_i$

$k_{i+1}$
Description of the AES: the Key-Schedule

$k_i$

$k_{i+1}$
Description of the AES: the Key-Schedule

\[ k_i \]

\[ k_{i+1} \]
Description of the AES: the Key-Schedule
Description of the AES: the Key-Schedule

\[ k_i \]

\[ k_{i+1} \]
Description of the AES: the Key-Schedule

\[ k_i \]

\[ k_{i+1} \]
Description of the AES: the Key-Schedule

\[ k_i \]

\[ k_{i+1} \]
Description of the AES: the Key-Schedule

\[ k_i \rightarrow k_{i+1} \]
The AES Has a Clean Description over $\mathbb{F}_{256}$

\[ x_0[j] = P[j] + K_0[j] \]
\[ y_i[j] = S(x_i[j]) \]
\[ r \text{ rounds } \rightarrow 20r \text{ equations, } 20r \text{ variables} \]


- **Equation** = linear combination of **Terms** over $\mathbb{F}_{256}$
- **Term** = $X_i$ or $S(X_i)$

The equations are:
- **sparse**: each equation relates, at most, five variables
- **structured**: each variable appears in, at most, four equations
  - A **Gröbner basis** of the equations is **known** (but useless)
Outline

1. AES
   - Origins and Structure
   - Description

2. Hash Functions
   - Definitions and Generic Attacks
   - Merkle-Damgaard
   - MD5 and SHA-?

3. Message Authentication Codes (MAC)
   - Definitions
   - CBC-MAC
   - HMAC
Does encryption guarantee message integrity?

- **Idea:**
  - Alice encrypts $m$ and sends $c = \text{Enc}(K, m)$ to Bob.
  - Bob computes $\text{Dec}(K, m)$, and if it “makes sense” accepts it.

- **Intuition:** only Alice knows $K$, so nobody else can produce a valid ciphertext.

It does not work!

**Example**

one-time pad.

Need a way to ensure that data arrives at destination in its original form (as sent by the sender and it is coming from an authenticated source)
Does encryption guarantee message integrity?

- **Idea:**
  - Alice encrypts $m$ and sends $c = \text{Enc}(K, m)$ to Bob.
  - Bob computes $\text{Dec}(K, m)$, and if it “makes sense” accepts it.

- **Intuition:** only Alice knows $K$, so nobody else can produce a valid ciphertext.

It does not work!

**Example**

one-time pad.

Need a way to ensure that data arrives at destination in its original form (as sent by the sender and it is coming from an authenticated source)
Hash Functions

- Hash functions compute fingerprints
- Various uses
- Oblivious to most users
Hash Functions

- Hash functions compute fingerprints
- Various uses
- Oblivious to most users

0x1d66ca77ab361c6f
Hash Functions

- Hash functions compute fingerprints
- Various uses
- Oblivious to most users
Hash Functions

- map a message of an **arbitrary** length to a **fixed** length output

- **output:** fingerprint or message digest

What is an example of hash functions?

- **Question:** Give a hash function that maps Strings to integers in \([0, 2^{32} - 1]\)

- additional security requirements \(\rightsquigarrow\) **cryptographic hash functions**
Security Requirements for Cryptographic Hash Functions

Given a function \( H : X \rightarrow Y \), then we say that \( h \) is:

- **pre-image resistant (one-way):**
  if given \( y \in Y \) it is computationally infeasible to find a value \( x \in X \) s.t. \( H(x) = y \)

- **second pre-image resistant (weak collision resistant):**
  if given \( x \in X \) it is computationally infeasible to find a value \( x' \in X \), s.t. \( x' \neq x \) and \( H(x') = H(x) \)

- **collision resistant (strong collision resistant):**
  if it is computationally infeasible to find two distinct values \( x', x \in X \), s.t. \( x' \neq x \) and \( H(x') = H(x) \)
Security Requirements for Cryptographic Hash Functions

Given a function $\mathcal{H} : X \rightarrow Y$, then we say that $h$ is:

- **pre-image resistant** (one-way):
  if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $\mathcal{H}(x) = y$

- **second pre-image resistant** (weak collision resistant):
  if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $\mathcal{H}(x') = \mathcal{H}(x)$

- **collision resistant** (strong collision resistant):
  if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $x' \neq x$ and $\mathcal{H}(x') = \mathcal{H}(x)$
Security Requirements for Cryptographic Hash Functions

Given a function $\mathcal{H} : X \rightarrow Y$, then we say that $h$ is:

- **pre-image resistant** (one-way):
  if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $\mathcal{H}(x) = y$

- **second pre-image resistant** (weak collision resistant):
  if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $\mathcal{H}(x') = \mathcal{H}(x)$

- **collision resistant** (strong collision resistant):
  if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $x' \neq x$ and $\mathcal{H}(x') = \mathcal{H}(x)$
An Ideal Hash Function: the Random Oracle

- Public Random Function (a.k.a. “the Random Oracle”)
- Generate “new” answers (uniformly) **at random**
- **Remembers** its previous answers
Generic Attack Against Preimage Resistance

**Input:** \( y \in \{0, 1\}^n, \ m \in \mathbb{N} \) with \( m > n \)

**Output:** \( x \in \{0, 1\}^m \) s.t. \( y = \mathcal{H}(x) \)

```plaintext
while True do
    \( x \leftarrow \{0, 1\}^m \)
    if \( \mathcal{H}(x) = y \) then
        return \( x \)
    end if
end while
```

- **Time Complexity:** \( O(2^n) \) (random \( \mathcal{H} \))
- **Space Complexity:** \( O(1) \)
Generic Attack Against Preimage Resistance

**Input:** \( y \in \{0, 1\}^n, m \in \mathbb{N} \text{ with } m > n \)

**Output:** \( x \in \{0, 1\}^m \text{ s.t. } y = H(x) \)

```plaintext
while True do
    x \leftarrow \{0, 1\}^m
    if H(x) = y then
        return x
    end if
end while
```

- **Time Complexity:** \( O(2^n) \) (random \( H \))
- **Space Complexity:** \( O(1) \)
Generic Attack Against Second Preimage Resistance

Input: \( x \in \{0, 1\}^m \)
Output: \( x' \in \{0, 1\}^m \) s.t. \( \mathcal{H}(x') = \mathcal{H}(x) \)

\[ y \leftarrow \mathcal{H}(x) \]

\[ \text{while } \text{TRUE do} \]
\[ x' \leftarrow \{0, 1\}^m \]
\[ \text{if } \mathcal{H}(x') = y \text{ then} \]
\[ \text{return } x' \]
\[ \text{end if} \]
\[ \text{end while} \]

- Time Complexity: \( O(2^n) \) (random \( \mathcal{H} \))
- Space Complexity: \( O(1) \)
Generic Attack Against Second Preimage Resistance

**Input:** $x \in \{0, 1\}^m$

**Output:** $x' \in \{0, 1\}^m$ s.t. $\mathcal{H}(x') = \mathcal{H}(x)$

$y \leftarrow \mathcal{H}(x)$

while True do
  $x' \leftarrow \{0, 1\}^m$
  if $\mathcal{H}(x') = y$ then
    return $x'$
  end if
end while

- **Time Complexity:** $O(2^n)$ (random $\mathcal{H}$)
- **Space Complexity:** $O(1)$
Generic Attack Against Collision Resistance

**Input:** \( m \in \mathbb{N} \) with \( m > n \)

**Output:** \( x, x' \in \{0, 1\}^m \) s.t. \( H(x) = H(x') \) and \( x \neq x' \)

\[ Y \leftarrow \emptyset \]

\[ \text{while True do} \]

\[ x_i \leftarrow \{0, 1\}^m \]

\[ y_i \leftarrow H(x_i) \]

\[ j \leftarrow \text{LookUp}(y_i, Y) \]

\[ \text{if } j \neq \bot \text{ then} \]

\[ \text{return } (x_i, x_j) \]

\[ \text{ADDElement}(Y, (x_i, y_i)) \]

\[ \text{end if} \]

\[ \text{end while} \]

**Birthday Paradox:**

(see TD 1)

- **Time Complexity:** \( O(2^{n/2}) \) (random \( H \))
- **Space Complexity:** \( O(2^{n/2}) \)
Generic Attack Against Collision Resistance

Input: $m \in \mathbb{N}$ with $m > n$
Output: $x, x' \in \{0, 1\}^m$ s.t. $H(x) = H(x')$ and $x \neq x'$

\[
\Upsilon \leftarrow \emptyset
\]

\begin{verbatim}
while True do
    $x_i \leftarrow \{0, 1\}^m$
    $y_i \leftarrow H(x_i)$
    $j \leftarrow \text{LookUp}(y_i, \Upsilon)$
    if $j \neq \perp$ then
        return $(x_i, x_j)$
    end if
    $\text{AddElement}(\Upsilon, (x_i, y_i))$
end while
\end{verbatim}

\begin{itemize}
\item Birthday Paradox: \hspace{1cm} (see TD 1)
\item Time Complexity: $O(2^{n/2})$ (random $H$)
\item Space Complexity: $O(2^{n/2})$
\end{itemize}
Hash functions in Security

- Digital signatures
- Random number generation
- Key updates and derivations
- One way functions
- MAC
- Detect malware in code
- User authentication (storing passwords)
- ...

...
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions

\[ M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4 \rightarrow M_5 \rightarrow M_6 \rightarrow M_7 \]
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions

\[ M_2 = M_3 = M_4 = M_5 = M_6 = M_7 \]
Hash Functions are Iterated Constructions

\[ M_2 \quad M_3 \quad M_4 \quad M_5 \quad M_6 \quad M_7 \]
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions

\[ M_4 \Rightarrow M_5 \Rightarrow M_6 \Rightarrow M_7 \]
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions

\[ M_7 = \]
Hash Functions are Iterated Constructions
Hash Functions are Iterated Constructions

0x8d90f5bc447d7bdd767a68b98e37e785
Merkle-Damgaard

- **compression function** $f : \{0, 1\}^{n+\ell} \rightarrow \{0, 1\}^n$

- **How to hash** $m = (m_0, \ldots, m_k) \in (\{0, 1\}^\ell)^{(k+1)}$?

$h_0$ initial value (initialization vector)

**Theorem**: $f$ collision-resistant $\Rightarrow \mathcal{H}$ collision resistant (with appropriate padding) (see TD 2)
Merkle-Damgaard

- **Compression function** $f : \{0, 1\}^{n+\ell} \longrightarrow \{0, 1\}^n$

- **How to hash** $m = (m_0, \ldots, m_k) \in (\{0, 1\}^\ell)^{(k+1)}$?

- $h_0$ initial value (initialization vector)

- **Theorem**: $f$ collision-resistant $\Rightarrow \mathcal{H}$ collision resistant (with appropriate padding)  
  (see TD 2)
MD5

- 128-bit hashes
- designed by Ronald Rivest in 1991
- “MD” stands for “Message Digest”
  - MD5(“The quick brown fox jumps over the lazy dog”) = 9e107d9d372bb6826bd81d3542a419d6
  - MD5(“The quick brown fox jumps over the lazy dog.”) = e4d909c290d0fb1ca068ffaddf22cbd0
- cryptographically broken (since 2004!)

- input message broken up into chunks of 512-bit blocks
- (message padded \(\rightarrow\) length is a multiple of 512)
MD5 (for reference only)

Input: $m \in \{0, 1\}^*, |m| < 2^{64} - 1$

Output: $h \in \{0, 1\}^{128}$, $h = \text{MD5}(m)$

\[ r[0..15] \leftarrow \{7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22\} \quad \triangleright \text{initialisation} \]
\[ r[16..31] \leftarrow \{5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20\} \]
\[ r[32..47] \leftarrow \{4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23\} \]
\[ r[48..63] \leftarrow \{6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21\} \]

\[ \text{for } i \text{ de 0 à 63 do} \]
\[ \quad k[i] \leftarrow \lfloor (|\sin(i + 1)| \cdot 2^{32}) \rfloor \]
\[ \text{end for} \]

$h^0 \leftarrow 67452301$; $h^1 \leftarrow \text{EFCDB89}$; $h^2 \leftarrow \text{98BADCFE}$; $h^3 \leftarrow 10325476$

\[ i = |m| \text{ mod } \ell \]
\[ (m_0, \ldots, m_k) \leftarrow R(m) = m\|10^\ell - i - 65\|_{\tau_m} \quad \triangleright \text{with } |m_i| = 512 \]

...
MD5 (for reference only)

... ▷ main loop

```latex
\textbf{for} j \textbf{from} 1 \textbf{to} k \textbf{do}
\begin{align*}
  (w_0, \ldots, w_{15}) &\leftarrow m_k \\
  a &\leftarrow h^0; b &\leftarrow h^1; c &\leftarrow h^2; d &\leftarrow h^3 \\
  \textbf{for} i \textbf{from} 0 \textbf{to} 63 \textbf{do} \\
    \textbf{if} 0 \leq i \leq 15 \textbf{then} \quad &\begin{align*}
      f &\leftarrow (b \land c) \lor ((\neg b) \land d); \\
      g &\leftarrow i
    \end{align*} \\
    \textbf{else if} 16 \leq i \leq 31 \textbf{then} \quad &\begin{align*}
      f &\leftarrow (d \land b) \lor ((\neg d) \land c); \\
      g &\leftarrow (5i + 1) \mod 16
    \end{align*} \\
    \textbf{else if} 32 \leq i \leq 47 \textbf{then} \quad &\begin{align*}
      f &\leftarrow b \oplus c \oplus d; \\
      g &\leftarrow (3i + 5) \mod 16
    \end{align*} \\
    \textbf{else if} 48 \leq i \leq 63 \textbf{then} \quad &\begin{align*}
      f &\leftarrow c \oplus (b \lor (\neg d)); \\
      g &\leftarrow (7i) \mod 16
    \end{align*} \\
  \textbf{end if} \\
  (a, b, c, d) &\leftarrow (d, ((a + f + k[i] + w[g]) \ll r[i]) + b, b, c)
\end{align*}
\textbf{end for}
\begin{align*}
  h^0 &\leftarrow h^0 + a; \\
  h^1 &\leftarrow h^1 + b; \\
  h^2 &\leftarrow h^2 + c; \\
  h^3 &\leftarrow h^3 + d
\end{align*}
\textbf{end for}
\textbf{return} (h^0 \parallel h^1 \parallel h^2 \parallel h^3)
```

▷ with $|w_0| = 32, \ldots, |w_{15}| = 32$
Collisions in MD5

- **Birthday attack complexity:** $2^{64}$
  - small enough to brute force collision search

- **1996**, collisions on the compression function

- **2004**, collisions

- **2007**, chosen-prefix collisions

- **2008**, rogue SSL certificates generated

- **2012**, MD5 collisions used in cyberwarfare
  - *Flame* malware uses an MD5 prefix collision to fake a Microsoft digital code signature
SHA Family - Secure Hash Algorithm

- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - 1998, collision attack with complexity $2^{61}$

- **SHA-1**: (1995). 160 bit digest
  - 2005, collision attack with claimed complexity of $2^{69}$
  - 2010, SHA1 was no longer supported
  - 2017, first collisions found

- **SHA-2**: (2001). digest of length 224, 256, 384, 512 (+2 truncated versions)
  - No collision attacks on SHA-2 as yet

- **SHA-3**: (2015). Also known as Keccak
  - (Bertoni, Daemen, Peeters and Van Assche)
SHA Family - Secure Hash Algorithm

- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998**, collision attack with complexity $2^{61}$

- **SHA-1**: (1995). 160 bit digest
  - **2005**, collision attack with claimed complexity of $2^{69}$
  - **2010**, SHA1 was no longer supported
  - **2017**, first collisions found

- **SHA-2**: (2001). digest of length 224, 256, 384, 512 (+2 truncated versions)
  - No collision attacks on SHA-2 as yet

- **SHA-3**: (2015). Also known as Keccak
  - (Bertoni, Daemen, Peeters and Van Assche)
SHA Family - Secure Hash Algorithm

- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998**, collision attack with complexity $2^{61}$

- **SHA-1**: (1995). 160 bit digest
  - **2005**, collision attack with claimed complexity of $2^{69}$
  - **2010**, SHA1 was no longer supported
  - **2017**, first collisions found

- **SHA-2**: (2001). digest of length 224, 256, 384, 512 (+2 truncated versions)
  - No collision attacks on SHA-2 as yet

- **SHA-3**: (2015). Also known as Keccak
  - (Bertoni, Daemen, Peeters and Van Assche)
SHA Family - Secure Hash Algorithm

- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - 1998, collision attack with complexity $2^{61}$

- **SHA-1**: (1995). 160 bit digest
  - 2005, collision attack with claimed complexity of $2^{69}$
  - 2010, SHA1 was no longer supported
  - 2017, first collisions found

- **SHA-2**: (2001). digest of length 224, 256, 384, 512 (+2 truncated versions)
  - No collision attacks on SHA-2 as yet

- **SHA-3**: (2015). Also known as Keccak
  - (Bertoni, Daemen, Peeters and Van Assche)
MD5 vs SHA-1
SHA-3
Outline

1. AES
   - Origins and Structure
   - Description

2. Hash Functions
   - Definitions and Generic Attacks
   - Merkle-Damgaard
   - MD5 and SHA-?

3. Message Authentication Codes (MAC)
   - Definitions
   - CBC-MAC
   - HMAC
Message Authentication Codes

**Symmetric authentication:** Alice and Bob share a “key” $K$

Bob can use the same method to send messages to Alice.

$\implies$ **symmetric setting**

How did Alice and Bob establish $K$?
Security Requirement for MAC

- resist the **Existential Forgery under Chosen Plaintext Attack**
  - challenger chooses a random key $K$
  - adversary chooses a number of messages $m_1, m_2, \ldots, m_\ell$ and obtains $\tau_i = MAC(K, m_i)$ for $1 \leq i \leq \ell$
  - adversary outputs $m^*$ and $\tau^*$
  - adversary wins if $\forall i, m^* \neq m_i$ and $\tau^* = MAC(K, m^*)$

- Adversary cannot create the MAC for a message for which it has not seen a MAC
- $E$ a block cipher (DES, AES, ...) on $n$-bit blocks
- produces a $n$-bit MAC
Forgery on CBC-MAC

- Message $m = (m_1, \ldots, m_\ell)$ with MAC $\tau$
- Message $m' = (m'_1, \ldots, m'_k)$ with MAC $\tau'$
- Message $m'' = (m_1, \ldots, m_\ell, m'_1 \oplus \tau, \ldots, m'_k)$ has MAC $\tau'$!
Forgery on CBC-MAC

- Message \( m = (m_1, \ldots, m_\ell) \) with MAC \( \tau \)
- Message \( m' = (m'_1, \ldots, m'_k) \) with MAC \( \tau' \)
- Message

  \[
  m'' = (m_1, \ldots, m_\ell, m'_1 \oplus \tau, \ldots, m'_k)
  \]

has MAC \( \tau' \)!
Fixing CBC-MAC

- **Length prepending**

- **Encrypt-last-block**
  - Encrypt-last-block CBC-MAC (ECBC-MAC)
  - $E(k_2, CBC \dash MAC(k_1, m))$

Other flaws:
- Using the same key for encryption and authentication
- Allowing the initialization vector to vary in value
- Using predictable initialization vector
Fixing CBC-MAC

- **Length prepending**

- **Encrypt-last-block**
  - Encrypt-last-block CBC-MAC (ECBC-MAC)
  - $E(k_2, CBC - MAC(k_1, m))$

**Other flaws:**

- Using the same key for encryption and authentication
- Allowing the initialization vector to vary in value
- Using predictable initialization vector
HMAC

- $\mathcal{H}$ a hash function (SHA-2, SHA-3, ...) with $n$-bit digests
- produces a $n$-bit MAC (Krawczyk, Bellare and Cannetti – 1996)
HMAC

\[ \text{HMAC}(k, m) = H(H'( \oplus \text{opad}) || H'( \oplus \text{ipad}) || m) \]

- \( K' = K \) padded with zeroes (to the right)
- \( \text{opad} = 0x5c5c5c\ldots5c5c \) (one-block-long hexadecimal constant)
- \( \text{ipad} = 0x363636\ldots3636 \) (one-block-long hexadecimal constant)
Description of Pelican-MAC

- MAC based on the AES
- Also by Rijmen & Daemen
- “Provably” secure up to $2^{64}$
- Initial state randomized with $K$
- 16-byte message block XORed
- 4 keyless AES rounds
  - $2.5\times$ faster than AES encryption
- Finalization: full AES

Knowing the state $\rightarrow$ forgeries