Cryptologie symétrique 2/2

Damien Vergnaud

Sorbonne Université

CRYPTO 1

Contents

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

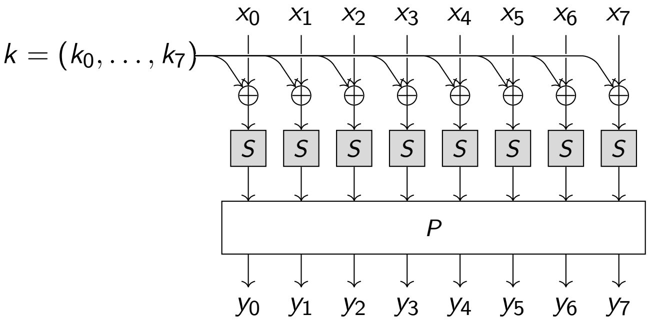
AES Origins

- 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

AES – Rijndael

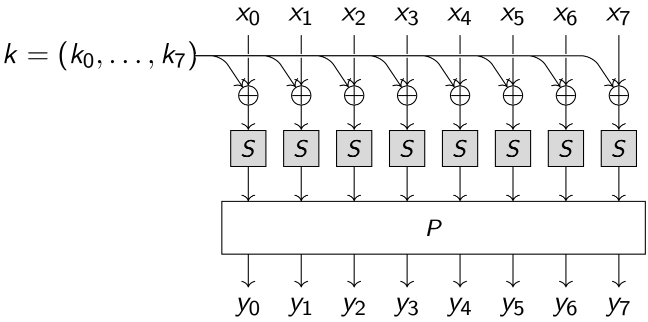
- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data (for AES)
- a **substitution-permutation network** (SPN) (rather than Feistel cipher)
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to have:
  - resistance against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Substitution-Permutation Network



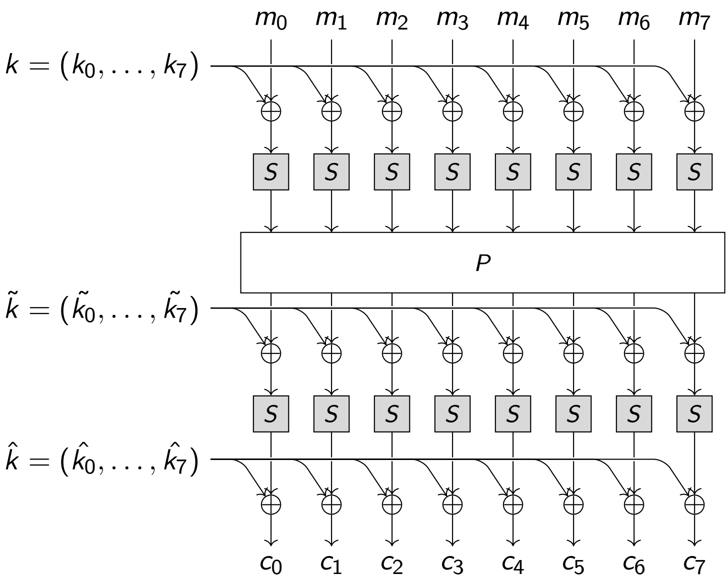
- **Substitution:** S-boxes substitute a small block of input bits
  - invertible, non-linear
  - changing one input bit $\rightsquigarrow$ change about half of the output bits
- **Permutation:** P-boxes permute bits
  - output bits of an S-box distributed to as many S-box inputs as possible.

Substitution-Permutation Network



- **Key:** in each round using group operation
- one S-box/P-box produces a *limited* amount of confusion/diffusion
- enough **rounds** $\rightsquigarrow$ every input bit is diffused across every output bit

Substitution-Permutation Network



AES Description

- AES with 128-bit key (see refs. for 192-bit and 256-bit keys)
- **Data block:** 128 bits $\rightsquigarrow$ 16 bytes in a 4×4 matrix

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- A byte $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ is represented by a polynomial

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0$$

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

- **Example:** $\texttt{5A} = 01011010$

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

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

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

AES Description

- AES with 128-bit key (see refs. for 192-bit and 256-bit keys)
- **Data block:** 128 bits $\rightsquigarrow$ 16 bytes in a 4×4 matrix

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- A byte $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ is represented by a polynomial

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0$$

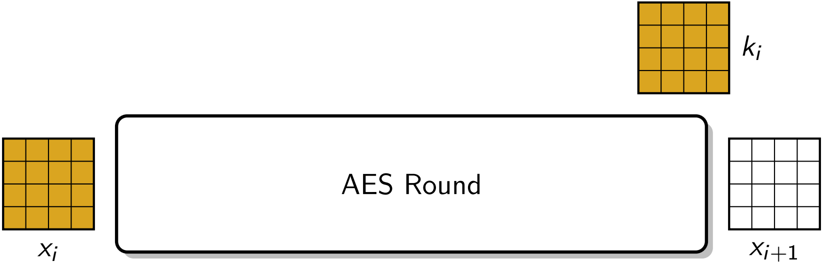
  with $b_i \in \{0, 1\} = \mathbb{F}_2$.
- **Example:** `5A` $= 01011010$

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

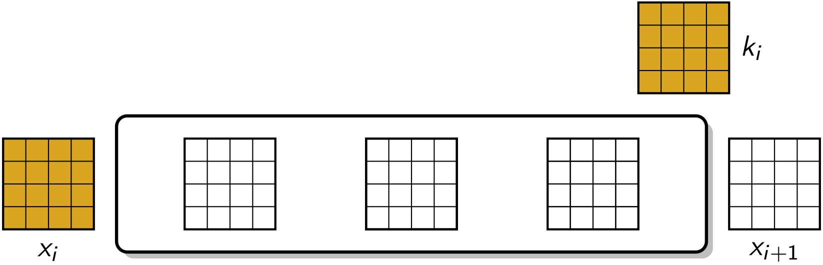
- Bytes are identified with elements of the finite field $\mathbb{F}_{256} = \mathbb{F}_2[x]/(m(x))$ with

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

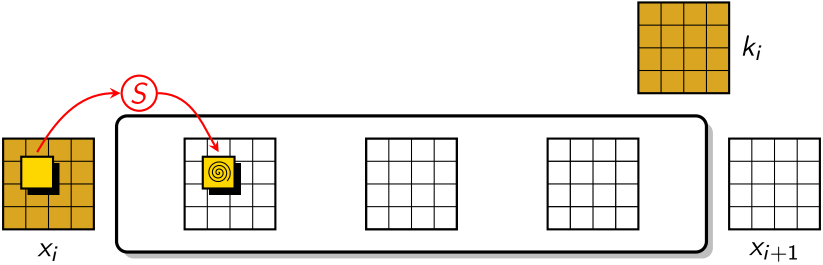
AES Description



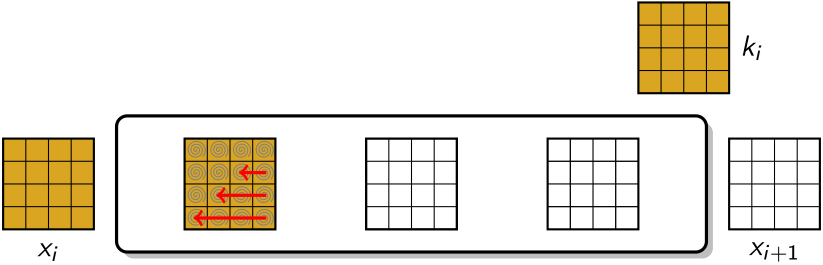
AES Description



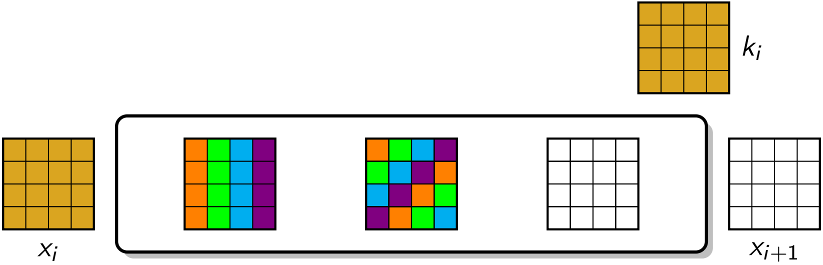
AES Description



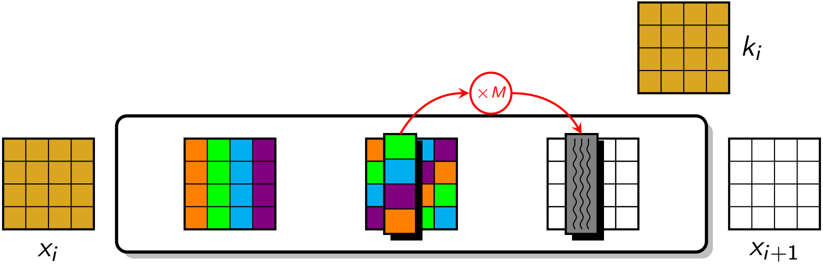
AES Description



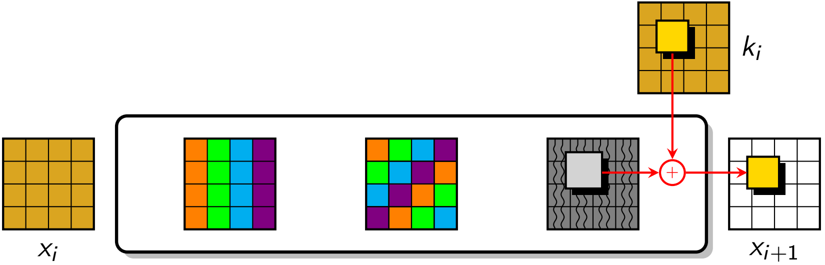
AES Description



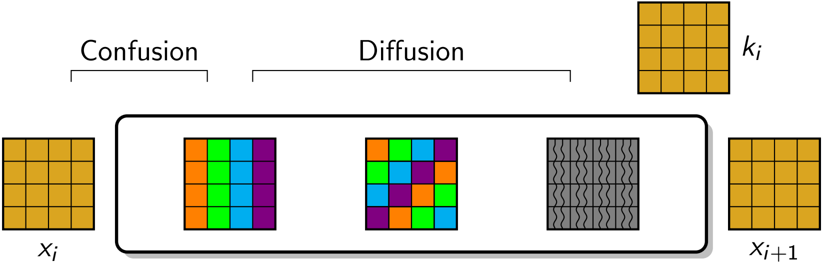
AES Description



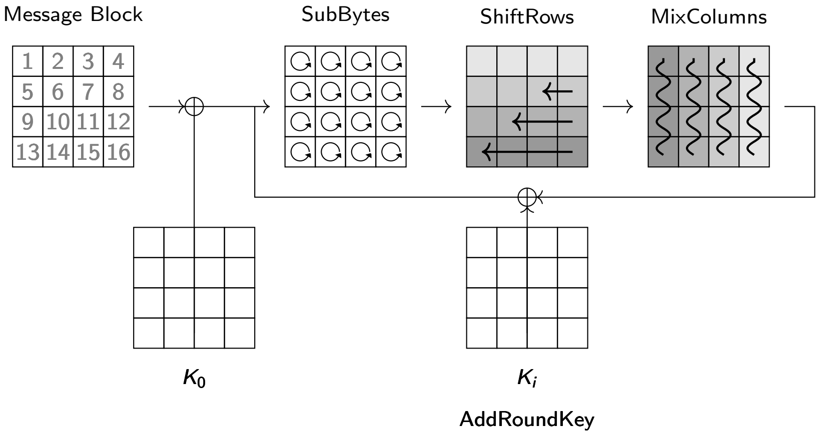
AES Description



AES Description

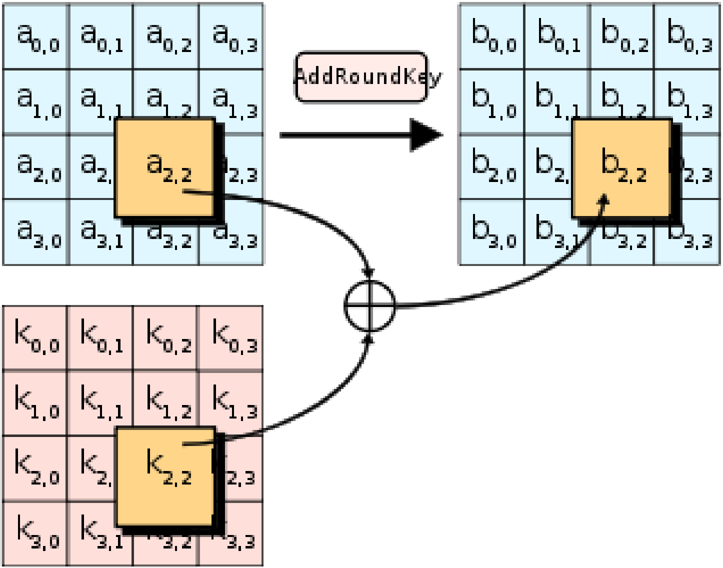


AES Structure

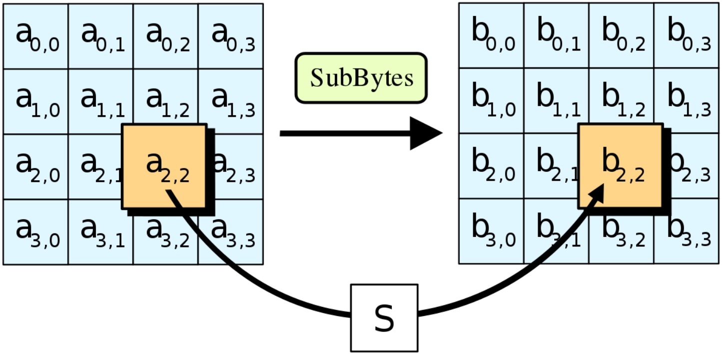


- no MixColumns in the last round

AddRoundKey



SubBytes



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

SubBytes

- S-box defined algebraically over $\mathbb{F}_{256}$
- First invert the byte (interpreted as an alement of $\mathbb{F}_{256}$:

$$a \longmapsto \begin{cases} a^{-1} & \text{if } a \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Then apply affine transformation:

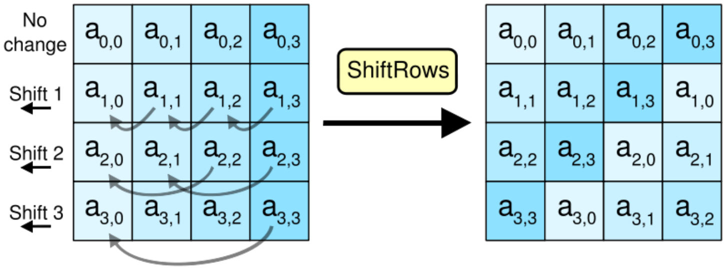
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

SubBytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

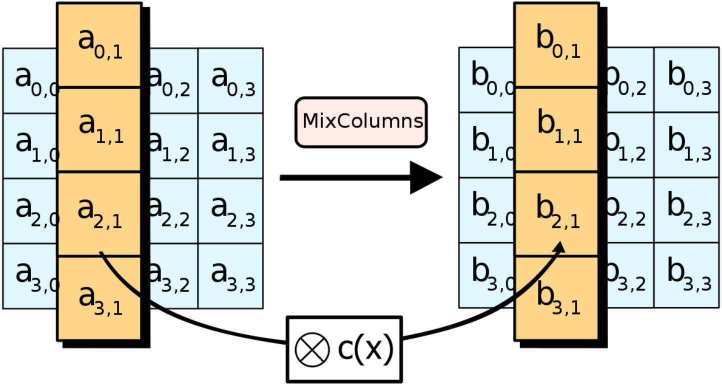
- the column is determined by the least significant nibble,
- the row is determined by the most significant nibble.
- **Example:** $S(\mathtt{9A}) = \mathtt{B8}$

ShiftRows



No change
Shift 1
Shift 2
Shift 3
a0,0 a0,1 a0,2 a0,3
a1,0 a1,1 a1,2 a1,3
a2,0 a2,1 a2,2 a2,3
a3,0 a3,1 a3,2 a3,3
ShiftRows
a0,0 a0,1 a0,2 a0,3
a1,1 a1,2 a1,3 a1,0
a2,2 a2,3 a2,0 a2,1
a3,3 a3,0 a3,1 a3,2

MixColumns



$a_{0,0}$
$a_{0,1}$
$a_{0,2}$
$a_{0,3}$
$a_{1,0}$
$a_{1,1}$
$a_{1,2}$
$a_{1,3}$
$a_{2,0}$
$a_{2,1}$
$a_{2,2}$
$a_{2,3}$
$a_{3,0}$
$a_{3,1}$
$a_{3,2}$
$a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$
$b_{0,1}$
$b_{0,2}$
$b_{0,3}$
$b_{1,0}$
$b_{1,1}$
$b_{1,2}$
$b_{1,3}$
$b_{2,0}$
$b_{2,1}$
$b_{2,2}$
$b_{2,3}$
$b_{3,0}$
$b_{3,1}$
$b_{3,2}$
$b_{3,3}$

MixColumns

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

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

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

$$b_0 + b_1 X + b_2 X^2 + b_3 X^3 \in A$$

- the new column is

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

MixColumns

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

$$
b(X) = b_0 + b_1X + b_2X^2 + b_3X^3
$$

Step 1: Polynomial multiplication

$$
\begin{aligned}
a(x) \cdot b(x) = c(x) &= (a_3X^3 + a_2X^2 + a_1X + a_0) \cdot (b_3X^3 + b_2X^2 + b_1X + b_0 \\
&= c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0
\end{aligned}
$$

where:

$$
\begin{aligned}
c_0 &= a_0 \cdot b_0 \\
c_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\
c_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\
c_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3
\end{aligned}
$$

$$
\begin{aligned}
c_4 &= a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
c_5 &= a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
c_6 &= a_3 \cdot b_3
\end{aligned}
$$

MixColumns

Step 2: Modular reduction

$$\begin{cases} X^6 \bmod (X^4+1) = -X^2 = X^2 \text{ over } \mathrm{GF}(2^8) \\ X^5 \bmod (X^4+1) = -X = X \text{ over } \mathrm{GF}(2^8) \\ X^4 \bmod (X^4+1) = -1 = 1 \text{ over } \mathrm{GF}(2^8) \end{cases}$$

$$\begin{aligned} & a(X)\cdot b(X) = c(X) \bmod (X^4+1) \\ = {} & (c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0) \bmod (X^4+1) \\ = {} & c_6X^2 + c_5x + c_4 + c_3X^3 + c_2X^2 + c_1X + c_0 \\ = {} & c_3X^3 + (c_2 \oplus c_6)X^2 + (c_1 \oplus c_5)X + c_0 \oplus c_4 \\ = {} & d_3X^3 + d_2X^2 + d_1X + d_0 \end{aligned}$$

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

$$
\begin{aligned}
d_0 &= a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
d_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
d_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3 \\
d_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3
\end{aligned}
$$

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

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

MixColumns

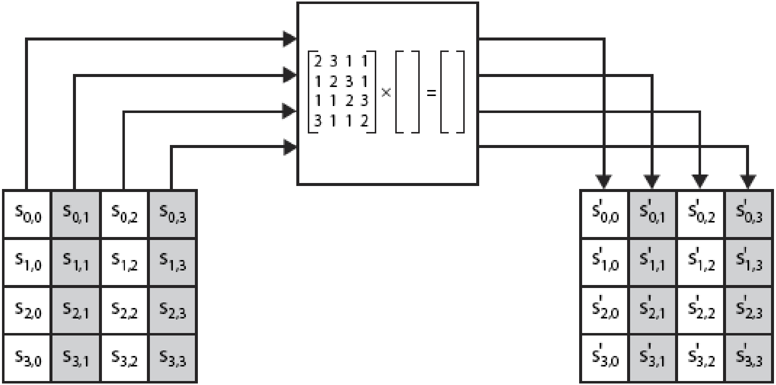
Step 3: Matrix representation

$$
\begin{aligned}
d_0 &= a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
d_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
d_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3 \\
d_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3
\end{aligned}
$$

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

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

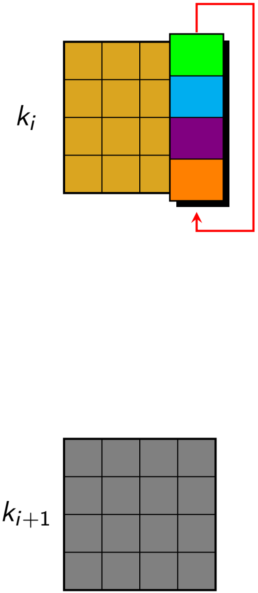
MixColumns



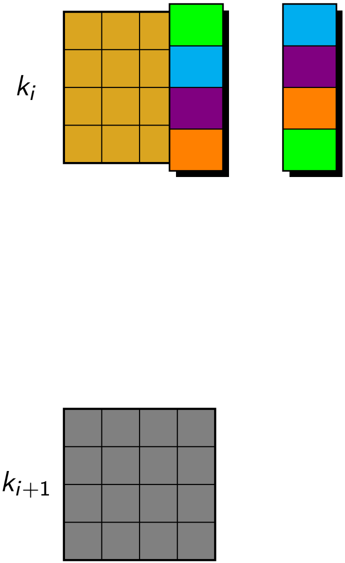
2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Description of the AES: the Key-Schedule

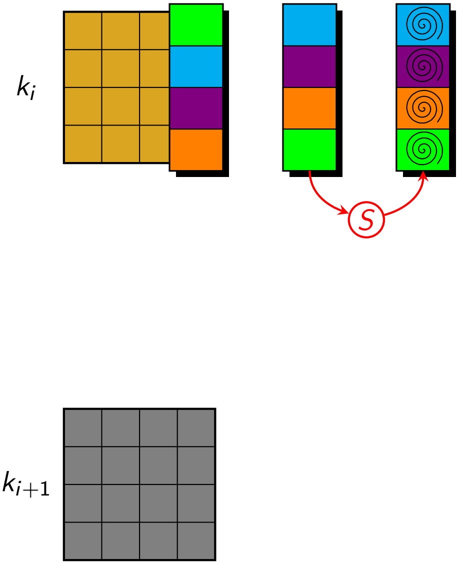
Description of the AES: the Key-Schedule



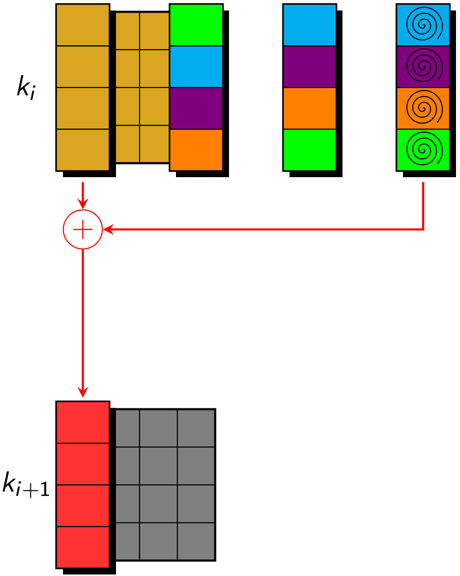
Description of the AES: the Key-Schedule



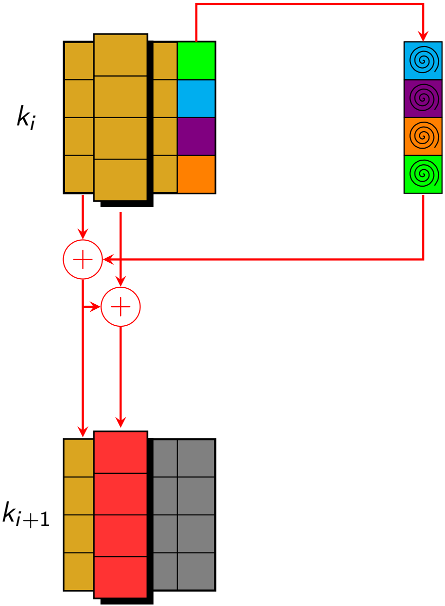
Description of the AES: the Key-Schedule



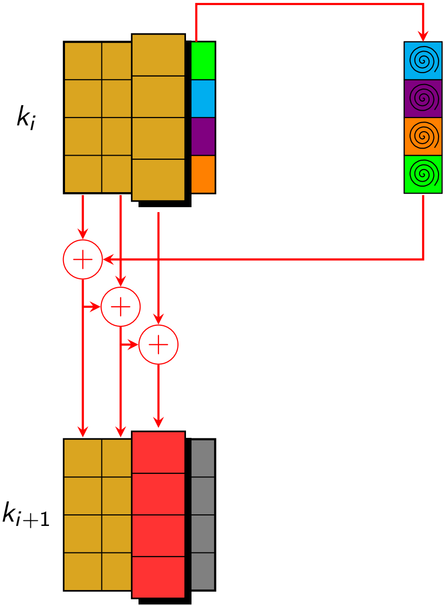
Description of the AES: the Key-Schedule



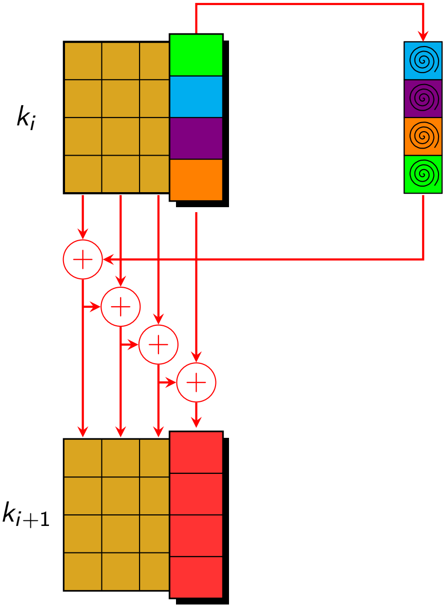
Description of the AES: the Key-Schedule



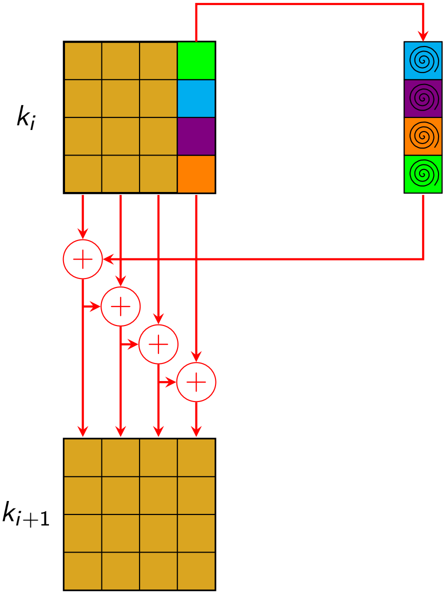
Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



Outline

Does encryption guarantee message integrity?

- **Idea:**
  - Alice encrypts m and sends $c = \mathrm{Enc}(K, m)$ to Bob.
  - Bob computes $\mathrm{Dec}(K, m)$, 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.

Ensure that data arrives at destination in its original form

Does encryption guarantee message integrity?

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

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

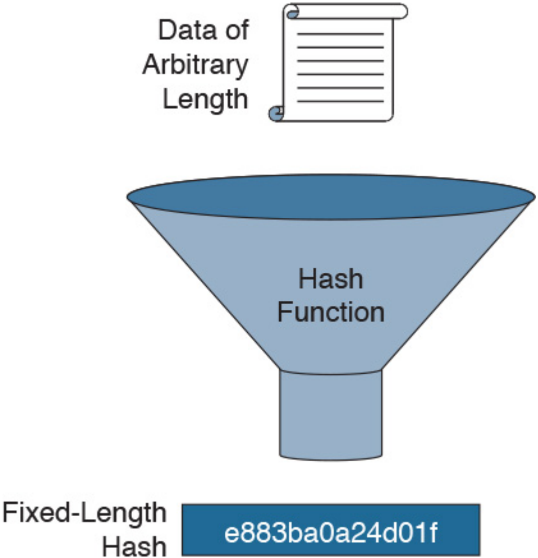
It does not work!

Example

one-time pad.

Ensure that data arrives at destination in its original form

Hash Functions



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 $\mathcal{H} : X \longrightarrow Y$, 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 $\mathcal{H} : X \longrightarrow Y$, 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 $\mathcal{H} : X \longrightarrow Y$, 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 $\mathcal{H}(x') = \mathcal{H}(x)$

Generic Attack against Pre-image resistance

Input: $y \in \{0,1\}^n$, $m \in \mathbb{N}$ with $m > n$
Output: $x \in \{0,1\}^m$ s.t. $y = \mathcal{H}(x)$

```
while TRUE do
    x ←R {0,1}^m
    if 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 Pre-image resistance

Input: $y \in \{0,1\}^n$, $m \in \mathbb{N}$ with $m > n$
Output: $x \in \{0,1\}^m$ s.t. $y = \mathcal{H}(x)$

```
while TRUE do
    x ←R {0,1}^m
    if 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 Second Pre-image 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 $\textsc{True}$ **do**
  $x' \xleftarrow{R} \{0,1\}^m$
  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 Second Pre-image resistance

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

```
y ← H(x)
while TRUE do
    x' ←R {0,1}^m
    if 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. $\mathcal{H}(x) = \mathcal{H}(x')$ and $x \neq x'$

```
Υ ← ∅                                          ▷ hash table
while TRUE do
    x_i ←R {0,1}^m
    y_i ← H(x_i)
    j ← LOOKUP(y_i, Υ)
    if j ≠ −1 then
        return (x_i, x_j)                      ▷ H(x_i) = H(x_j)
    end if
    ADDELEMENT(Υ, (x_i, y_i))                  ▷ sorted using the second coordinate
end while
```

Birthday Paradox: (see **TD 1**)

- Time Complexity: $O(2^{n/2})$ (random $\mathcal{H}$)
- 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. $\mathcal{H}(x) = \mathcal{H}(x')$ and $x \neq x'$

$\Upsilon \leftarrow \emptyset$ ▷ hash table
while $\textsc{True}$ **do**
  $x_i \xleftarrow{R} \{0,1\}^m$
  $y_i \leftarrow \mathcal{H}(x_i)$
  $j \leftarrow \textsc{LookUp}(y_i, \Upsilon)$
  if $j \neq -1$ **then**
    return (x_i, x_j) ▷ $\mathcal{H}(x_i) = \mathcal{H}(x_j)$
  end if
  $\textsc{AddElement}(\Upsilon, (x_i, y_i))$ ▷ sorted using the second coordinate
end while

Birthday Paradox: (see **TD 1**)

- Time Complexity: $O(2^{n/2})$ (random $\mathcal{H}$)
- Space Complexity: $O(2^{n/2})$

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)
- ...

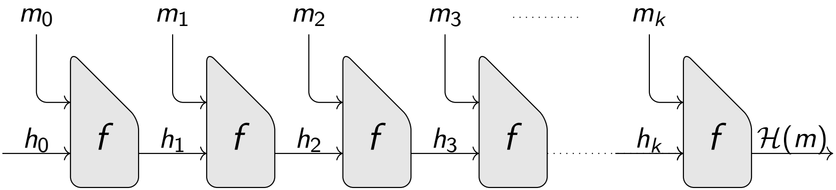
Merkle-Damgaard

- 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 (intialization 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 (intialization 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 $\rightsquigarrow$ length is a multiple of 512)

MD5 (for reference only)

Input: $m \in \{0,1\}^*$, $|m| < 2^{64} - 1$
Output: $h \in \{0,1\}^{128}$, $h = \text{MD5}(m)$

$r[0..15] \leftarrow \{7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22\}$ ▷ initialisation
$r[16..31] \leftarrow \{5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20\}$
$r[32..47] \leftarrow \{4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23\}$
$r[48..63] \leftarrow \{6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21\}$
for i de 0 à 63 **do**
    $k[i] \leftarrow \lfloor (|sin(i+1)| \cdot 2^{32}) \rfloor$
end for
$h^0 \leftarrow \texttt{67452301}$; $h^1 \leftarrow \texttt{EFCDAB89}$; $h^2 \leftarrow \texttt{98BADCFE}$; $h^3 \leftarrow \texttt{10325476}$
$i = |m| \bmod \ell$
$(m_0, \ldots, m_k) \leftarrow \mathcal{R}(m) = m \| 10^{\ell - i - 65} \| \tau_m$ ▷ with $|m_i| = 512$
...

MD5 (for reference only)

$\ldots$ ▷ main loop

for j from 1 to k **do**

  $(w_0, \ldots, w_{15}) \leftarrow m_k$ ▷ with $|w_0| = 32, \ldots, |w_{15}| = 32$

  $a \leftarrow h^0;\ b \leftarrow h^1;\ c \leftarrow h^2;\ d \leftarrow h^3$

  for i from 0 to 63 **do**

    if $0 \leq i \leq 15$ **then**

      $f \leftarrow (b \wedge c) \vee ((\neg b) \wedge d);\ g \leftarrow i$

    else if $16 \leq i \leq 31$ **then**

      $f \leftarrow (d \wedge b) \vee ((\neg d) \wedge c);\ g \leftarrow (5i + 1) \bmod 16$

    else if $32 \leq i \leq 47$ **then**

      $f \leftarrow b \oplus c \oplus d;\ g \leftarrow (3i + 5) \bmod 16$

    else if $48 \leq i \leq 63$ **then**

      $f \leftarrow c \oplus (b \vee (\neg d);\ g \leftarrow (7i) \bmod 16$

    end if

    $(a, b, c, d) \leftarrow (d, ((a + f + k[i] + w[g]) \lll r[i]) + b, b, c)$

  end for

  $h^0 \leftarrow h^0 + a;\ h^1 \leftarrow h^1 + b;\ h^2 \leftarrow h^2 + c;\ h^3 \leftarrow h^3 + d$

end for

return $(h^0 \| h^1 \| h^2 \| h^3)$

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)

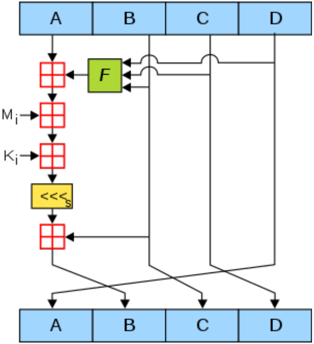
SHA Family - Secure Hash Algorithm

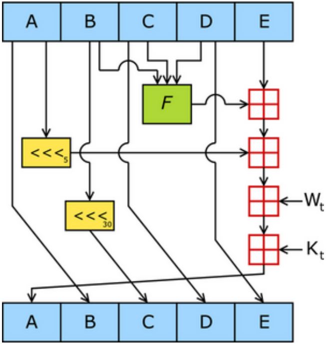
- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998,** collision attack with complexity 2^{61}
- **SHA-1**: (1995). 160 bit digest
  - **2005,** collision attack with claimed complexity of 2^{69}
  - **2010,** SHA1 was no longer supported
  - **2017,** first collisions 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

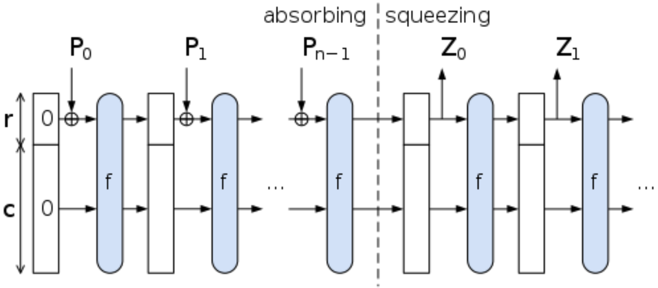
- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998,** collision attack with complexity 2^{61}
- **SHA-1**: (1995). 160 bit digest
  - **2005,** collision attack with claimed complexity of 2^{69}
  - **2010,** SHA1 was no longer supported
  - **2017,** first collisions 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



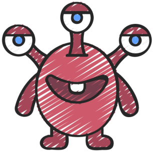


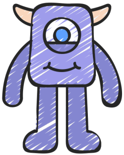
SHA-3



Outline

Message Authentication Codes

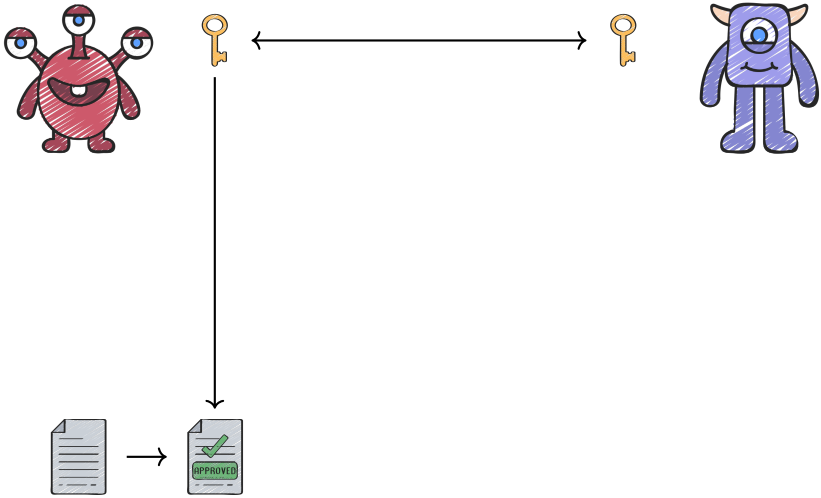




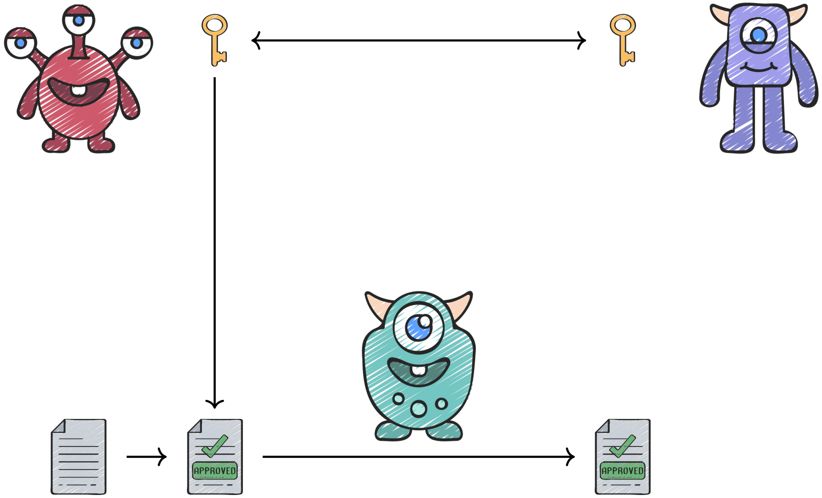
Message Authentication Codes

Message Authentication Codes

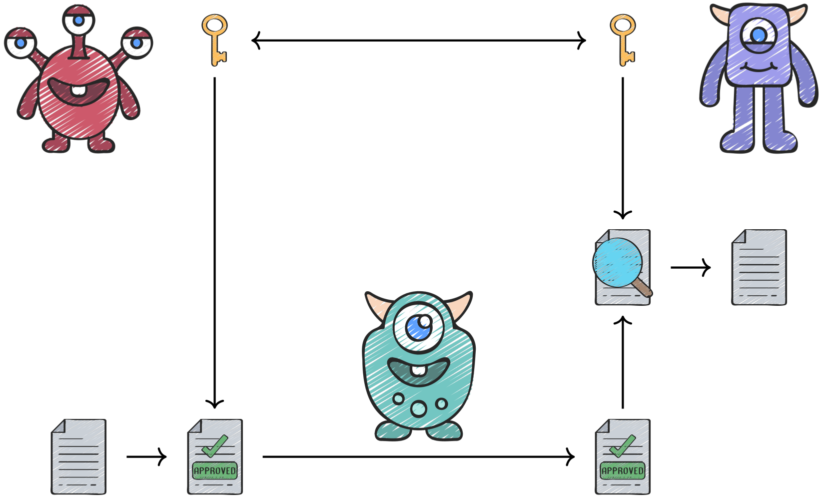
Message Authentication Codes



Message Authentication Codes



Message Authentication Codes

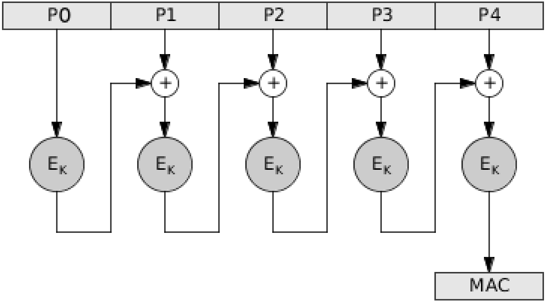


Security Requirement for MAC

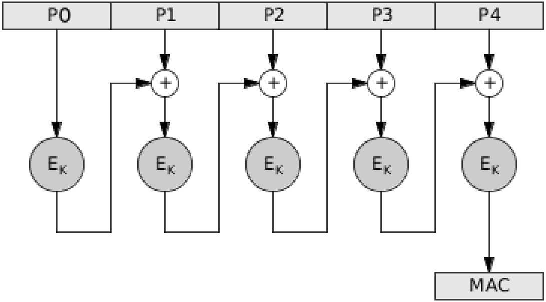
- resist the **Existential Forgery under Chosen Plaintext Attack**
  - challenger chooses a random key K
  - adversary chooses a number of messages $m_1, m_2, \ldots, m_\ell$ and obtains $\tau_i = \mathrm{MAC}(K, m_i)$ for $1 \leq i \leq \ell$
  - adversary outputs $m^\star$ and $\tau^\star$
  - adversary wins if $\forall i m^\star \neq m_i$ and $\tau^\star = \mathrm{MAC}(K, m^\star)$
- Adversary cannot create the MAC for a message for which it has not seen a MAC

CBC-MAC

- E a block cipher (DES, AES, ...) on n-bit blocks
- produces a n-bit MAC



Forgery on CBC-MAC

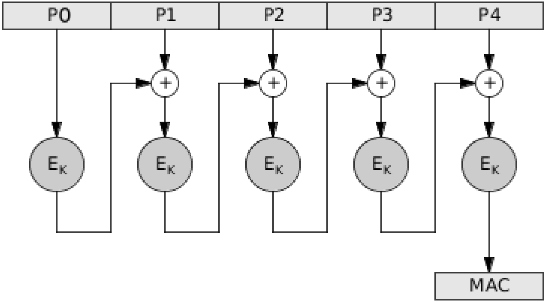


- Message $m = (m_1, \dots, m_\ell)$ with MAC τ
- Message $m' = (m'_1, \dots, m'_k)$ with MAC τ'
- Message

$$m'' = (m_1, \dots, m_\ell, m'_1 \oplus \tau, \dots, m'_k)$$

has MAC τ'!

Forgery on CBC-MAC



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

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

has MAC τ'!

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

Fixing CBC-MAC

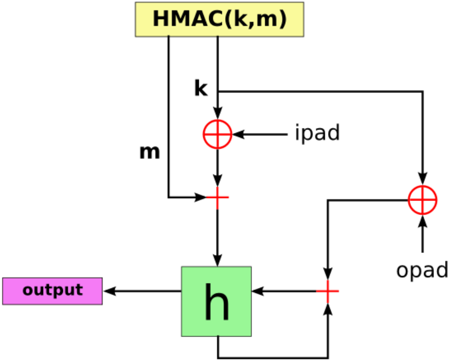
- **Length prepending**
- **Encrypt-last-block**
  - Encrypt-last-block CBC-MAC (ECBC-MAC)
  - $E(k_2, CBC - MAC(k_1, m))$

Other flaws:

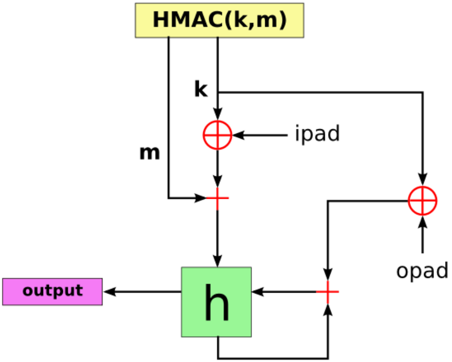
- Using the same key for encryption and authentication
- Allowing the initialization vector to vary in value
- Using predictable initialization vector

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



$$\mathrm{HMAC}(K,m) = \mathcal{H}\Big((K' \oplus opad) \| \mathcal{H}\big((K' \oplus ipad) \| m\big)\Big)$$

- $K' = K$ padded with zeroes (to the right)
- $opad = \texttt{0x5c5c5c...5c5c}$ (one-block-long hexadecimal constant)
- $ipad = \texttt{0x363636...3636}$ (one-block-long hexadecimal constant)