Cryptographie M1

Lecture 2

Ludovic Perret
(slides from C. Bouillaguet and Damien Vergnaud)

Sorbonne Université

2023 – 2024

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

Rijndael — the Advanced Encryption Standard

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

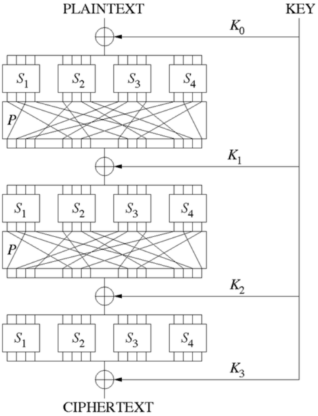
AES basic structures

- 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 $\rightsquigarrow$ 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** $\rightsquigarrow$ every input bit is diffused across every output bit



Algebraic Structure in the AES

- **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

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

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

- 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:** $\mathtt{5A} = 01011010$

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

Algebraic Structure in the AES

- **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

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

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

- A byte $b_7b_6b_5b_4b_3b_2b_1b_0$ is represented by a polynomial

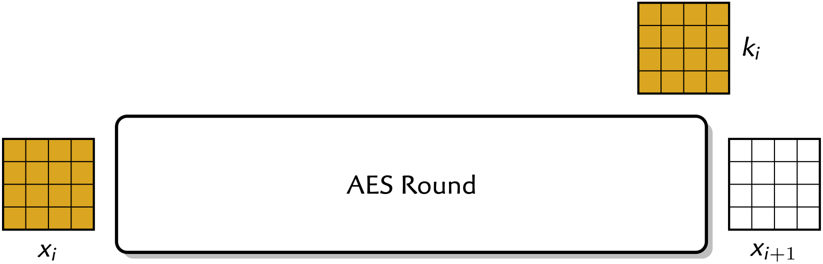
$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

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

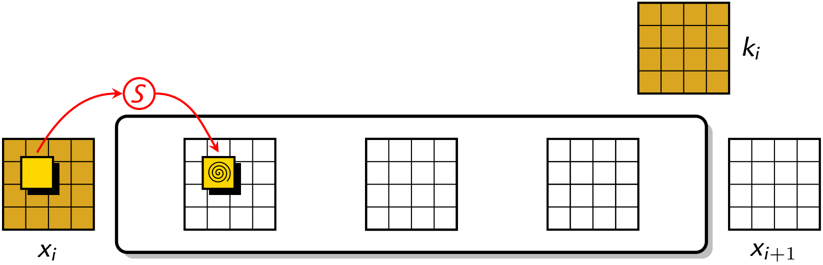
- **Example:** $\texttt{5A} = 01011010$

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

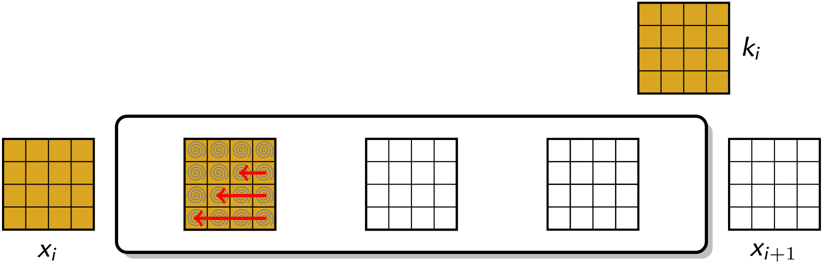
Description of the AES



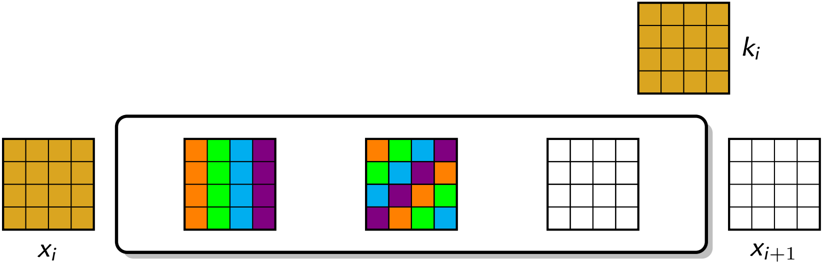
Description of the AES



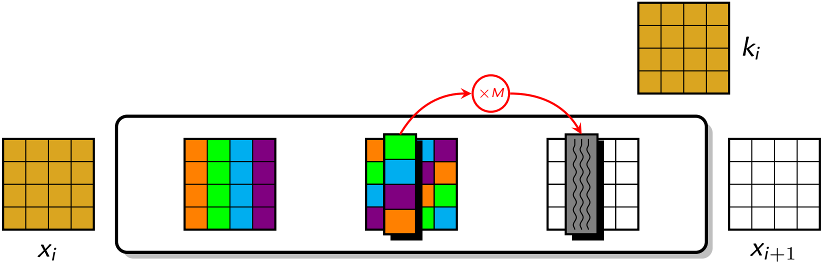
Description of the AES



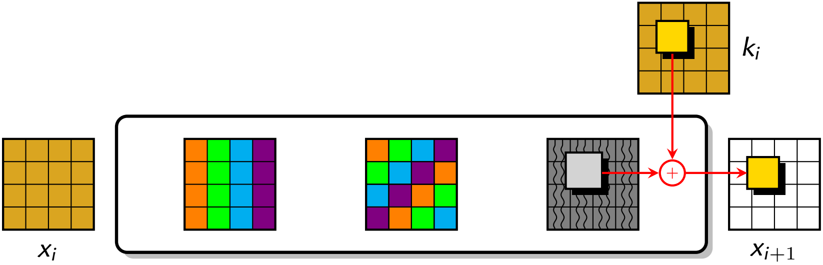
Description of the AES



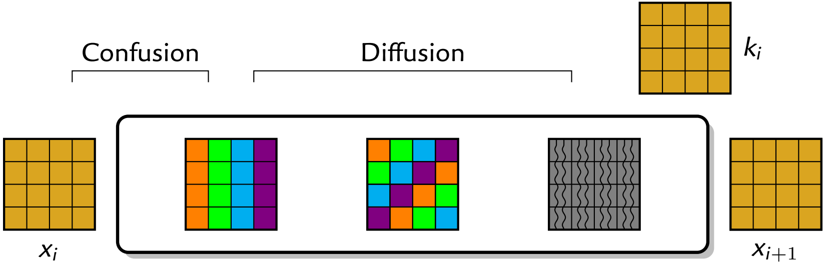
Description of the AES



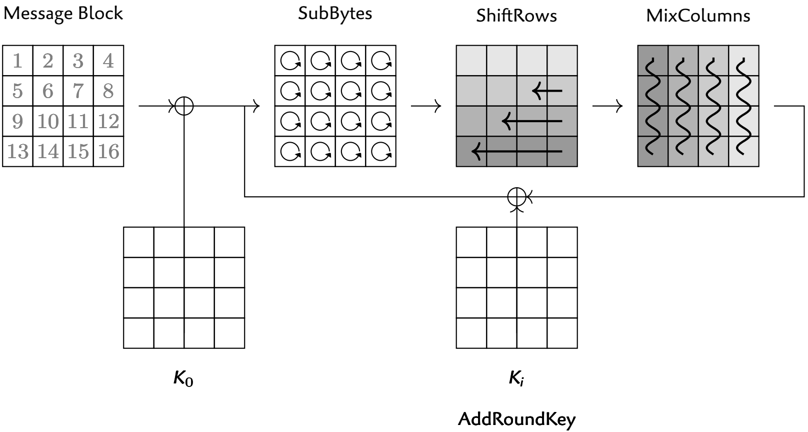
Description of the AES



Description of the AES



AES Structure



- no MixColumns in the last round

SubBytes

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

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

- Then apply affine transformation:

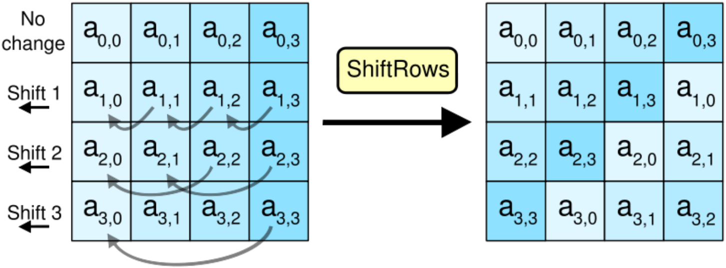
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\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

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

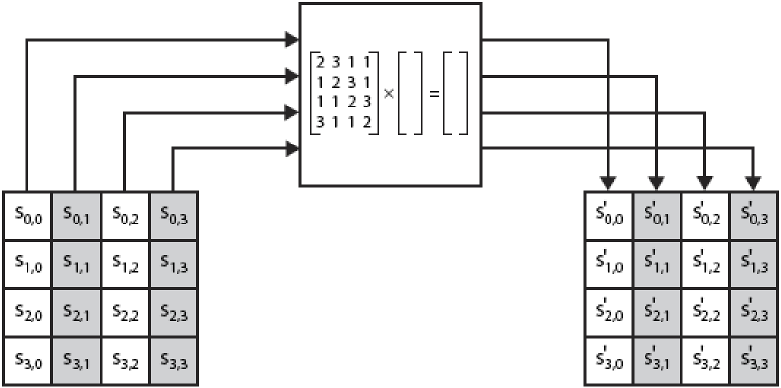
- the column is determined by the least significant nibble,
- the row is determined by the most significant nibble.
- **Example:** $S(\mathtt{9A}) = \mathtt{B8}$

ShiftRows



No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

MixColumns



$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Linear Layer (Diffusion)

MixColumn

- Each column is multiplied (over $\mathbb{F}_{256}$) by a fixed matrix

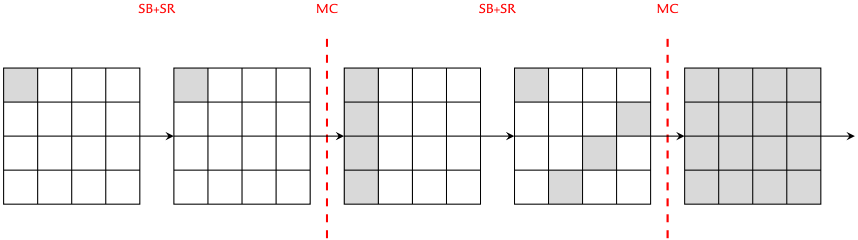
$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

- If $x = (0, 0, 0, 0)$, then $y = (0, 0, 0, 0)$
- Otherwise, ≥ 5 non-zero coefficients in x and y ("MDS code")
- Active Column $\Rightarrow$ at least 5 active byte in two successive rounds

ShiftRows

- k active byte on a column $\rightsquigarrow$ k active columns

Difference Propagation

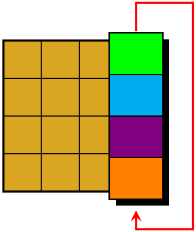


Description of the AES: the Key-Schedule

k_i

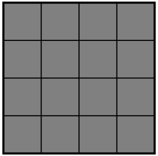
k_{i+1}

Description of the AES: the Key-Schedule

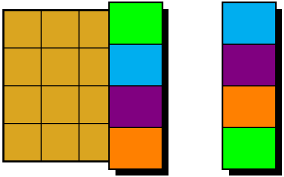


k_i

k_{i+1}

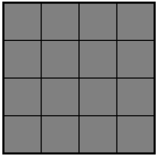


Description of the AES: the Key-Schedule

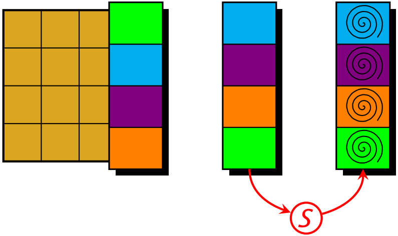


k_i

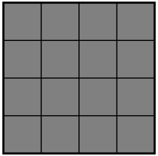
k_{i+1}



Description of the AES: the Key-Schedule

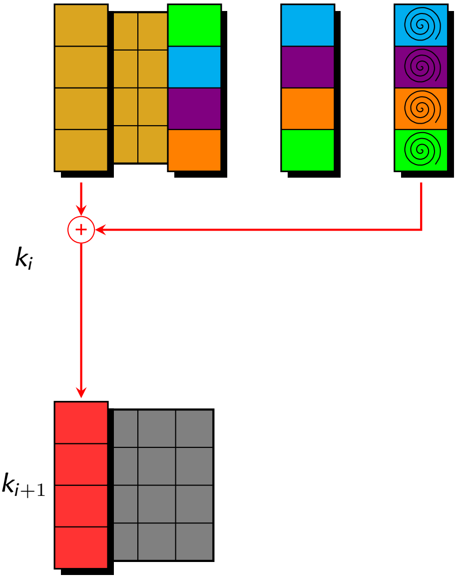


k_i

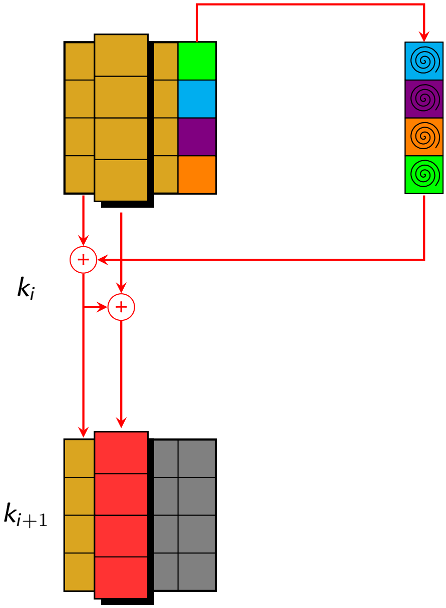


k_{i+1}

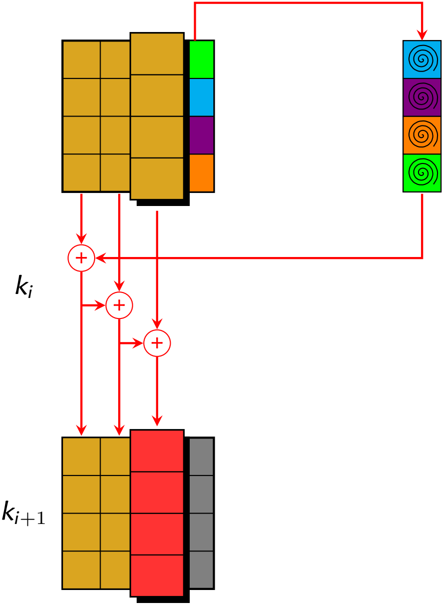
Description of the AES: the Key-Schedule



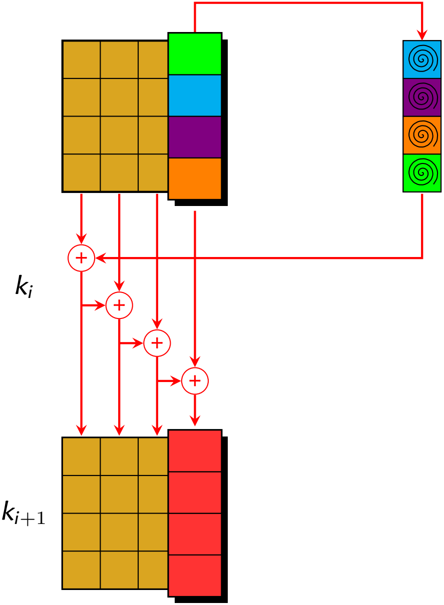
Description of the AES: the Key-Schedule



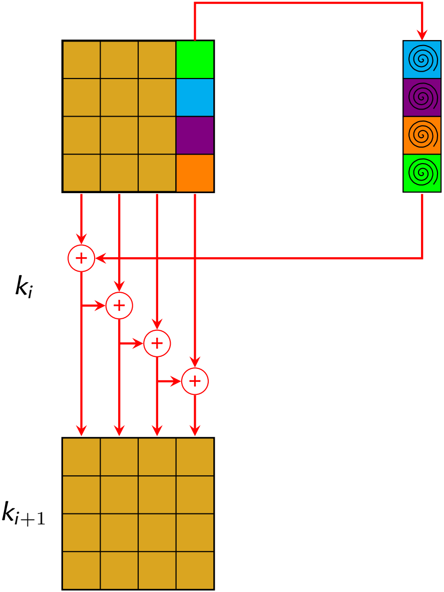
Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



The AES Has a Clean Description over $\mathbb{F}_{256}$

$$x_0[j] = P[j] + K_0[j]$$

$$y_i[j] = S(x_i[j])$$

$$x_{i+1} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} y_i[0] & y_i[4] & y_i[8] & y_i[12] \\ y_i[5] & y_i[9] & y_i[13] & y_i[1] \\ y_i[10] & y_i[14] & y_i[2] & y_i[6] \\ y_i[15] & y_i[3] & y_i[7] & y_i[11] \end{pmatrix} + K_{i+1}$$

r rounds $\rightarrow$ $20r$ equations, $20r$ 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

Outline

Does encryption guarantee message integrity?

- **Idea:**
  - Anissa encrypts m and sends $c = \mathsf{Enc}(K, m)$ to Billel.
  - Billel computes $\mathsf{Dec}(K, m)$, and if it "makes sense" accepts it.
- **Intuition:** only Anissa 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:**
  - Anissa encrypts m and sends $c = \mathrm{Enc}(K, m)$ to Billel.
  - Billel computes $\mathrm{Dec}(K, m)$, and if it "makes sense" accepts it.
- **Intuition:** only Anissa knows K, so nobody else can produce a valid ciphertext.

It does not work!

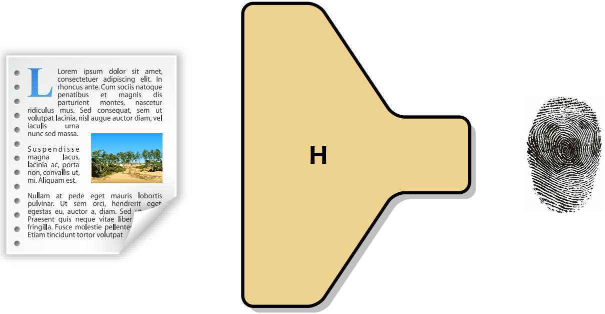
Example

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)

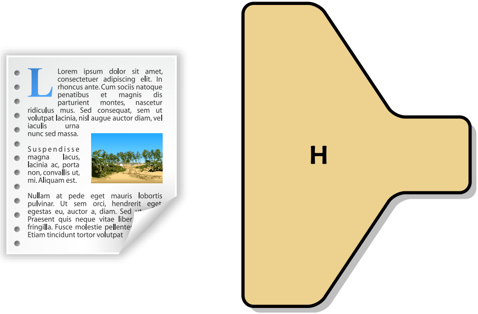
Hash Functions

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



Hash Functions

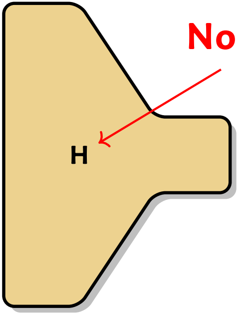
- Hash functions compute fingerints
- Various uses
- Oblivious to most users



0x1d66ca77ab361c6f

Hash Functions

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



No Keys !

`0x1d66ca77ab361c6f`

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 $\leadsto$ 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 $\mathcal{H}(x') = \mathcal{H}(x)$

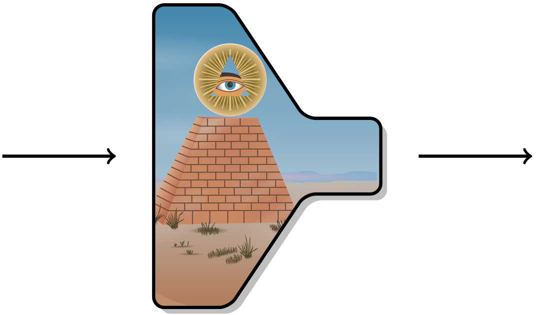
- collision resistant (strong collision resistant):
  if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $x' \neq x$ and $\mathcal{H}(x') = \mathcal{H}(x)$

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 $\mathcal{H}(x') = \mathcal{H}(x)$

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. $y = \mathcal{H}(x)$

```
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 Preimage Resistance

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

```
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 Preimage 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 Second Preimage 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 ≠ ⊥ 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 $x \neq x'$

```
  Υ ← ∅                                          ▷ hash table
  while TRUE do
     x_i ←R {0,1}^m
     y_i ← H(x_i)
     j ← LOOKUP(y_i, Υ)
     if j ≠ ⊥ 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: $O(2^{n/2})$

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

= M_0 M_1 M_2 M_3 M_4 M_5 M_6 M_7

Hash Functions are Iterated Constructions

$$= \boxed{M_0 \mid M_1 \mid M_2 \mid M_3 \mid M_4 \mid M_5 \mid M_6 \mid M_7}$$

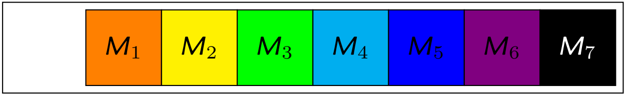
Hash Functions are Iterated Constructions

 =

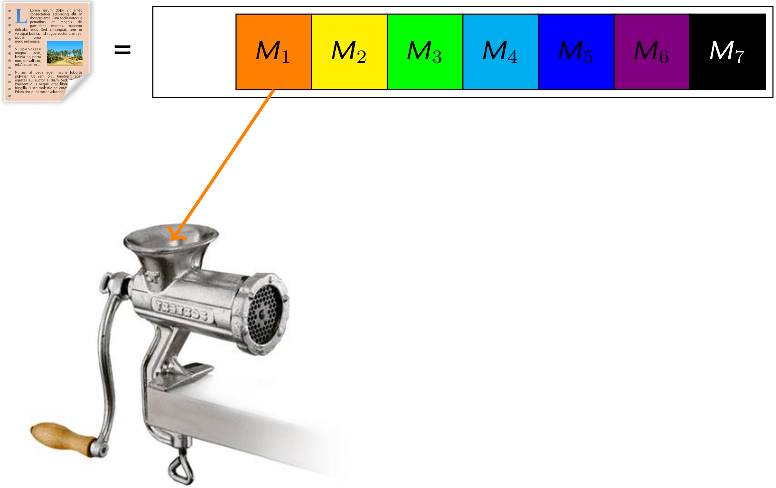
M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7

Hash Functions are Iterated Constructions

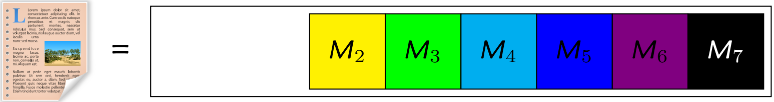
=



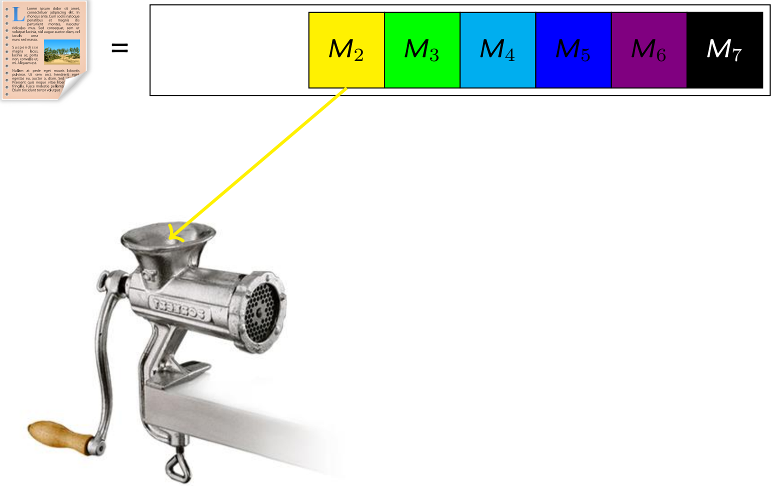
Hash Functions are Iterated Constructions



Hash Functions are Iterated Constructions



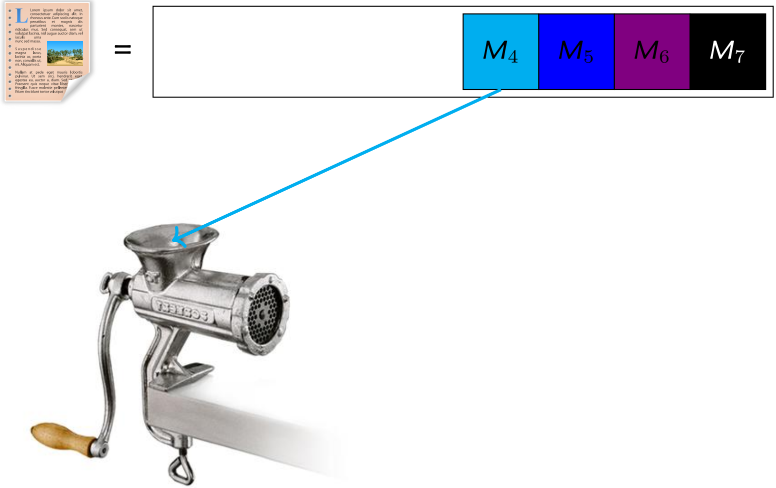
Hash Functions are Iterated Constructions



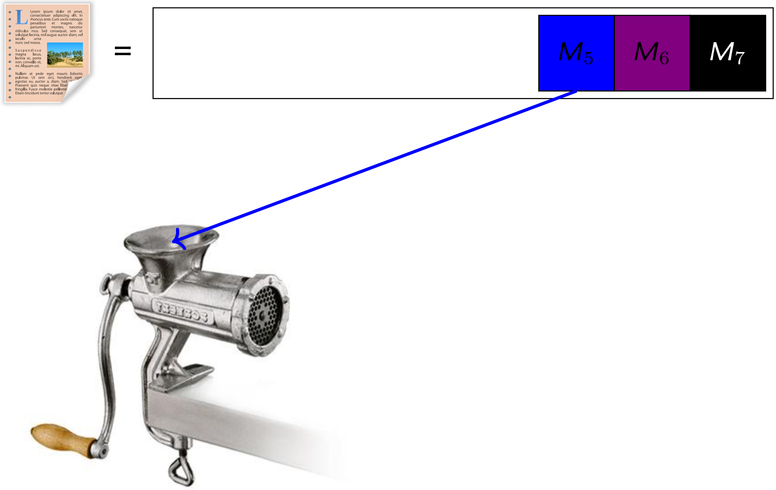
Hash Functions are Iterated Constructions

 = | M_3 | M_4 | M_5 | M_6 | M_7 |

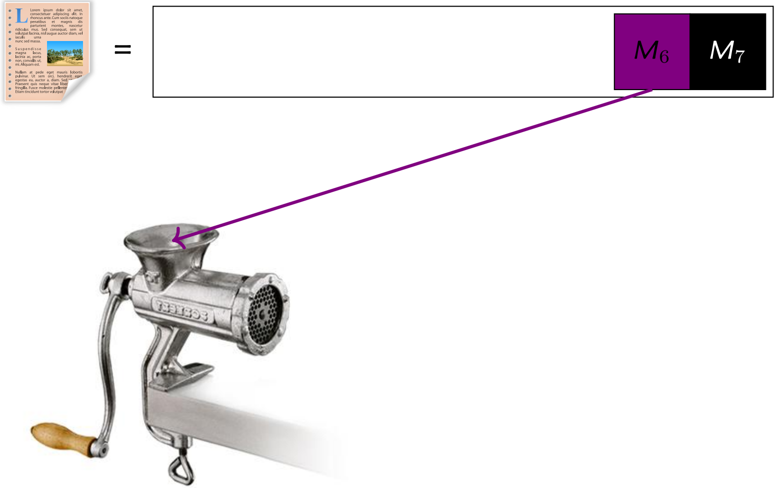
Hash Functions are Iterated Constructions



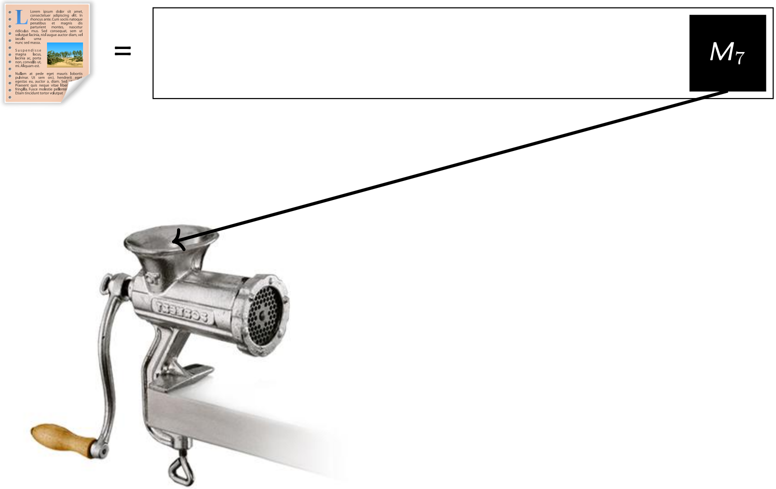
Hash Functions are Iterated Constructions



Hash Functions are Iterated Constructions



Hash Functions are Iterated Constructions



Hash Functions are Iterated Constructions

=

Hash Functions are Iterated Constructions

=

`0x8d90f5bc447d7bdd767a68b98e37e785`

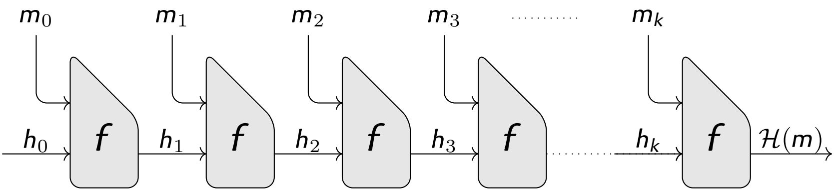
Merkle-Damgaard

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

$r[0..15] \leftarrow \{7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22\}$ ▷ 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)

... $\triangleright$ main loop

for j from 1 to k **do**

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

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

  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}
  - **2008**, collision attack with complexity 2^{33} ($\approx$ 1h on a standard PC)
- **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}
  - **2008**, collision attack with complexity 2^{33} ($\approx$ 1h on a standard PC)
- **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)

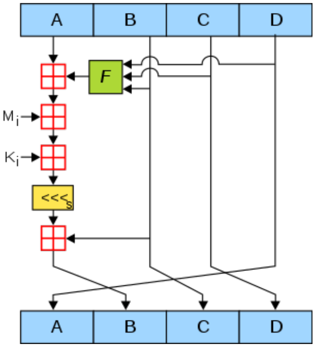
SHA Family - Secure Hash Algorithm

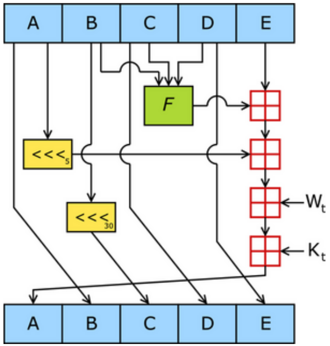
- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998**, collision attack with complexity 2^{61}
  - **2008**, collision attack with complexity 2^{33} ($\approx$ 1h on a standard PC)
- **SHA-1**: (1995). 160 bit digest
  - **2005**, collision attack with claimed 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

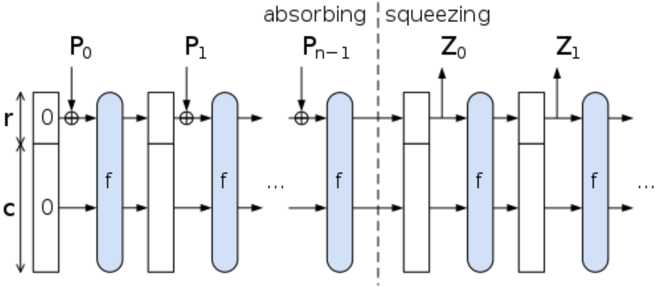
- **SHA-0**: (1993). 160 bit digest
  - unpublished weaknesses in this algorithm
  - **1998**, collision attack with complexity 2^{61}
  - **2008**, collision attack with complexity 2^{33} ($\approx$ 1h on a standard PC)
- **SHA-1**: (1995). 160 bit digest
  - **2005**, collision attack with claimed 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





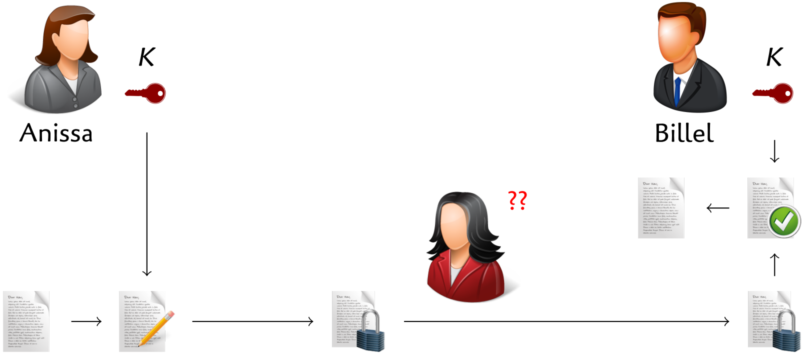
SHA-3



Outline

Message Authentication Codes

Symmetric authentication: Anissa and Billel share a "key" K



- Billel can use the same method to send messages to Anissa.
  $\rightsquigarrow$ **symmetric setting**
- How did Anissa and Billel establish K?

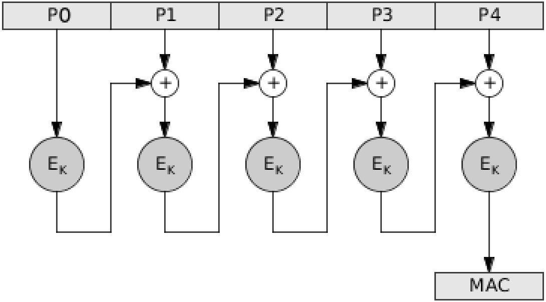
Security Requirement for MAC

- 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 $\tau_i = \mathrm{MAC}(K, m_i)$ for $1 \leq i \leq \ell$
  - adversary outputs $m^\star$ and $\tau^\star$
  - adversary wins if $\forall i, m^\star \neq m_i$ and $\tau^\star = \mathrm{MAC}(K, m^\star)$

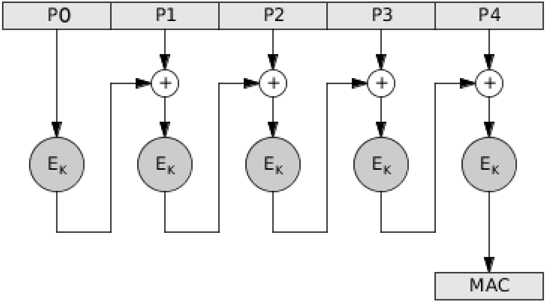
- Adversary cannot create the MAC for a message for which it has not seen a MAC

CBC-MAC

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



Forgery on CBC-MAC

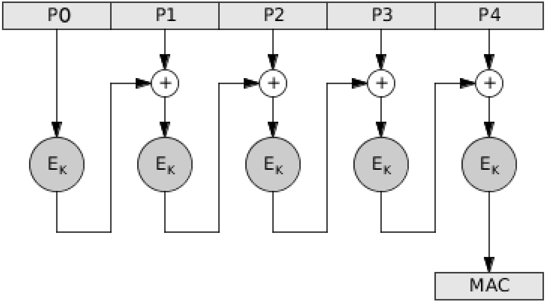


- Message $m = (m_1, \ldots, m_\ell)$ with MAC τ
- Message $m' = (m'_1, \ldots, m'_k)$ with MAC τ'
- Message

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

has MAC τ'!

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

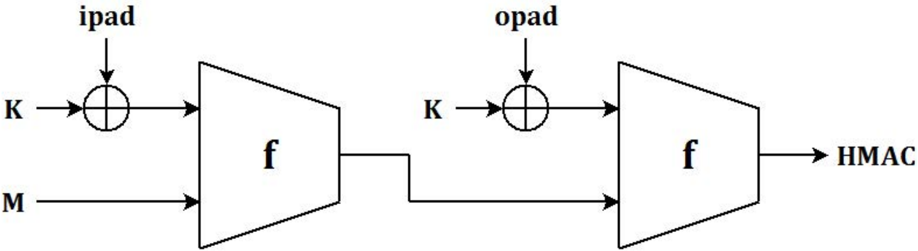
- **Length prepending**
- **Encrypt-last-block**
  - Encrypt-last-block CBC-MAC (ECBC-MAC)
  - $E(k_2, CBC - MAC(k_1, m))$

Other flaws:

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

HMAC

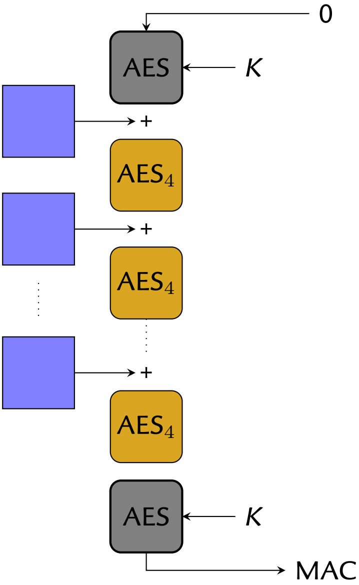
- $\mathcal{H}$ a hash function (SHA-2, SHA-3, ...) with n-bit digests
- produces a n-bit MAC (Krawczyk, Bellare and Cannetti – 1996)



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

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