ICS - Information Security Systems
Lecture 2: Symmetric Cryptography

2025-2026

Types of Secret-Key Encryptions

Classes of Secret-Key Encryption

We distinguish two main categories of symmetric encryption :

- **Symmetric Stream Cipher** ;
- **Symmetric Block Cipher**.

The principle of stream ciphers is to encrypt a sequence of characters one at a time, using a transformation that evolves as the text progresses.

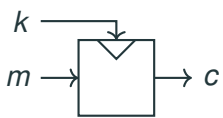
Block Cipher

Block Cipher

A block cipher is a function :

$$\text{Enc: } \mathcal{P} \times \mathcal{K} \to \mathcal{C}$$

Taking as input a plaintext and a key, and producing a ciphertext.



IND-CPA : The ciphertext of a given message must be indistinguishable from the ciphertext of a random element (and thus from a random element itself).

DES -
Data Encryption Standard

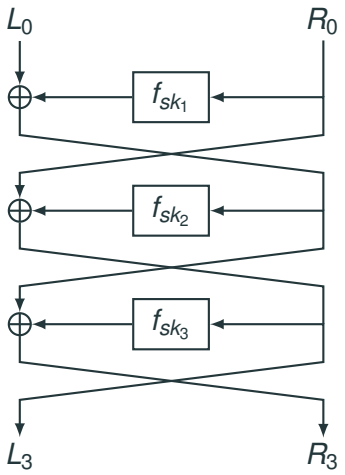
Feistel Network

Feistel networks were invented by IBM engineers, including *Horst Feistel, in the 1970s*.

- m : message of size n (even n)
- m is divided into two equal-sized blocks, L_0 and R_0, with $|L_0| = |R_0| = \frac{n}{2}$
- $m = L_0 || R_0$
- A keyed function f is required.

Feistel Network

A three-round Feistel system :



Message $m = L_0 || R_0$

Key $\text{sk} = \text{sk}_1 || \text{sk}_2 || sk_3$

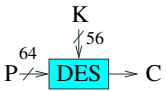
For $1 \leq i \leq 3$:

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f(R_{i-1}, sk_i) \end{cases}$$

Data Encryption Standard

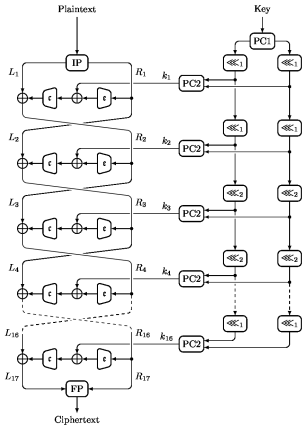
DES uses **56-bit keys**. It is a **64-bit block cipher** standardized by NIST (and the NSA) in 1977. Its use is no longer recommended.

- The first cryptographic standard.
- **Deprecated !** (except in rare cases...)

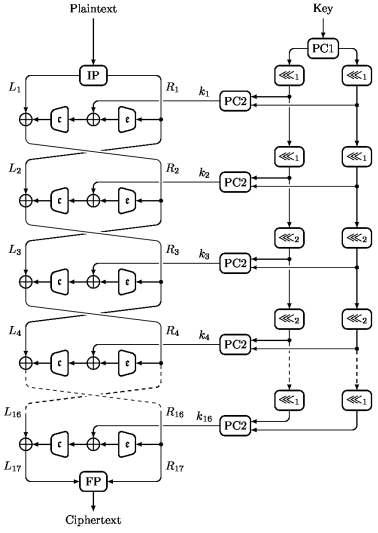


DES - Data Encryption Standard

- Input : 64 bits (8 bytes) ;
- Initial permutation of blocks ;
- Split into two parts : left and right, named L and R ;
- Repeated permutation and substitution steps 16 times (called rounds) ;
- Recombine left and right parts, then apply inverse initial permutation.



DES - Data Encryption Standard



Note on DES

The permutations IP and IP^{-1} do not add any security ; they are meant to speed up computation.

Advantages : encryption and decryption $\Rightarrow$ similar architecture. Hardware implementation is easy.

DES (*Data Encryption Standard*)

What is the main weakness of DES ?

DES (*Data Encryption Standard*)

What is the main weakness of DES ?

The secret key length ! Only 56 bits.

$\Longrightarrow$ Broken in 3 days in 1998 by Deep Crack.
Cost : $200,000 (which is little !).

$\Longrightarrow$ Broken in 1.5 days in 2007 by COPA COBANA.
Cost : $10,000.

In symmetric cryptography, we want keys of 128 or 256 bits.

DES has other vulnerabilities : existence of *weak keys*, and attacks based on linear and differential cryptanalysis.

2DES

Improvement : 2DES, using two different keys.

$\Rightarrow$ key size of 112 bits.

However, there exists a (meet-in-the-middle) attack showing that breaking 2DES is “as easy” as breaking DES.

Increasing Key Size

Motivation

- Given a block cipher with a small key κ (e.g., DES)
- Build a block cipher with a larger key $\lambda = 2\kappa, 3\kappa, \ldots$
- Idea : the scheme is good, but the key size is insufficient

Simple idea : iterative encryption

- Double encryption : $EE2(k_1||k_2, m) = E(k_2, E(k_1, m))$
- Triple encryption :
  - $EEE3(k_1||k_2||k_3, m) = E(k_3, E(k_2, E(k_1, m)))$
  - $EEE2(k_1||k_2, m) = E(k_1, E(k_2, E(k_1, m)))$
  - $EDE3(k_1||k_2||k_3, m) = E(k_3, E^{-1}(k_2, E(k_1, m)))$
  - $EDE2(k_1||k_2, m) = E(k_1, E^{-1}(k_2, E(k_1, m)))$ $\Rightarrow$ **3-DES**

Security ?

- Example : **3-DES is "secure"**
- Exhaustive search : $\mathcal{O}(2^{2\kappa})$ or $\mathcal{O}(2^{3\kappa})$

Attack on Double Encryption

Formula : $EE_2(k_1 \,||\, k_2, m) = E(k_2, E(k_1, m)), \quad$ with $k_1, k_2 \in \{0,1\}^\kappa$.

Meet-in-the-middle

Input : (m, c) where $c = EE_2(k_1^* \,||\, k_2^*, m)$ for unknown keys k_1^*, k_2^*.
Output : a small set of keys containing $k_1^* \,||\, k_2^*$.

1. Compute $y_{k_1} = E(k_1, m)$ for all $k_1 \in \{0,1\}^\kappa$.
2. Compute $z_{k_2} = E^{-1}(k_2, c)$ for all $k_2 \in \{0,1\}^\kappa$.
3. For each equality $y_{k_1} = z_{k_2}$, add $k_1 \,||\, k_2$ to the set of keys :
   $\Rightarrow EE_2(k_1 \,||\, k_2, m) = E(k_2, E(k_1, m)) = E(k_2, y_{k_1}) = E(k_2, z_{k_2}) = c$

Analysis

Time : two passes of $O(2^\kappa)$ calls to $E^\pm$ + comparisons $\Rightarrow$ approximately $O(2^\kappa)$.
Memory : two lists of 2^κ pairs (text, key) $\Rightarrow$ $O((n+\kappa)\cdot 2^\kappa)$.

Conclusion : same complexity as brute-force in 2^κ !

SPN - Substitution–Permutation Network

SPN (*Substitution-Permutation Network*)

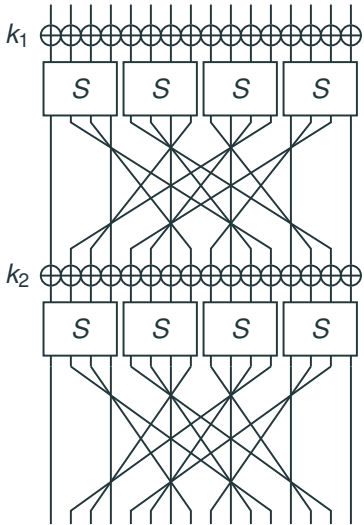
Steps :

1. XOR-Key
2. S-box
3. Permutation

   Repeat . . .

End. XOR-Key

Question :
Why do we start and end with an XOR-Key ?



Architecture : Substitution–Permutation Networks (SPN)

The network includes substitution and permutation boxes. These operations are designed to be efficient in hardware and often use XOR operations.

***S-Box* - Substitution Box**
A substitution table that contributes to “confusion” by making the original information unintelligible. It helps break the linearity of the cipher structure.

***P-Box* - Permutation Box**
A permutation table that defines how to swap elements within the structure. It contributes to “diffusion” by mixing data and enhancing the avalanche effect.

Substitution–Permutation Networks - SPN

A single *S-Box* or *P-Box* alone ≠ cryptographic strength ⇒ *S-Box* ⇔ substitution cipher and *P-Box* ⇔ transposition cipher.
A well-designed substitution–permutation network satisfies Shannon's principles of confusion and diffusion :

- **Diffusion** : statistical redundancy in the plaintext is dissipated in the ciphertext statistics. An input bias must not appear at the output : avalanche effect. Good diffusion means that flipping a single input bit should change each output bit with probability 0.5.
- **Confusion** : each bit of the ciphertext should depend on several parts of the key.

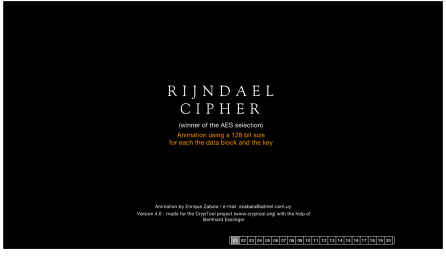
AES - Advanced Encryption Standard

- Need for a cipher to replace DES ;
- NIST (National Institute of Standards and Technology) launched a public competition in September 1997 :
  - public, unclassified algorithm, free of usage rights ;
  - symmetric cipher on blocks of at least 128 bits and key sizes of 128, 192, and 256 bits ;
  - secure against known cryptanalytic methods ;
  - efficient across all implementations (from small processors and smart cards to hardware circuits).
- 15 candidates $\Rightarrow$ preselection in March 1999 $\Rightarrow$ five finalists : MARS, RC6, Rijndael, Serpent, TwoFish ;
- Final results (2000) : NIST selected Rijndael (named after Vincent Rijmen & Joan Daemen)

Example : AES - Advanced Encryption Standard

- NIST Competition (1997–2000)
- Winner : Rijndael, by V. Rijmen & J. Daemen
- 128-bit block size
- Substitution–Permutation Network

Key size	# rounds
128	10
192	12
256	14



Algebraic Considerations

Bit strings, bytes, and finite fields

- Input : 128-bit string $\rightarrow$ sequence of 16 bytes
- One byte $\simeq$ one element of the finite field $\mathbb{F}_{2^8}$
- $\mathbb{F}_{2^8} \simeq \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1 \rangle$
- Representation : polynomials of degree ≤ 7

SubBytes

- Inverse in $\mathbb{F}_{2^8}$ (with $0 \mapsto 0$)
- Combined with an invertible affine transformation

MixColumns

- Each column $\rightarrow$ vector in $(\mathbb{F}_{2^8})^4$
- Matrix multiplication by a circulant MDS matrix (coding theory)

$\rightarrow$ **Algebraic design to resist known attacks**

Comparison of Symmetric Encryption Algorithms

	3xDES	IDEA	BLOWFISH	CAST 5	TWOFISH	AES
Publication	1978	1991	1993	1996	1998	1998
Taille de bloc (bits)	64	64	64	64	128	128
Taille de clé	168 (112 effectif)	128	32-448 (par 8)	40-128 (par 8)	128, 192, 256	128, 192, 256
Structure	Feistel	substitution / permutation	Feistel	Feistel	Feistel,	substitution / permutation
Nbr. de Tours	3x16	8 ½	16	12 ou16	16	10, 12 ou 14
Clé faible		OUI	OUI	NON		NON
LOGICIELS PGP 9.0	OUI	OUI		OUI	OUI	OUI
LOGICIELS GnuPG	OUI	OUI	OUI	OUI	OUI	OUI
LOGICIELS OpenSSL	OUI	OUI	OUI	OUI		OUI

Other Symmetric Schemes

Blowfish, Serpent, Twofish, 3-Way, ABC, Akelarre, Anubis, ARIA, BaseKing, BassOmatic, BATON, BEAR and LION, C2, Camellia, CAST-128, CAST-256, CIKS-1, CIPHERUNICORN-A, CIPHERUNICORN-E, CLEFIA, CMEA, Cobra, COCONUT98, Crab, CRYPTON, CS-Cipher, DEAL, DES-X, DFC, E2, FEAL, FEA-M, FROG, G-DES, GOST, Grand Cru, Hasty Pudding Cipher, Hierocrypt, ICE, IDEA, IDEA NXT, Intel Cascade Cipher, Iraqi, KASUMI, KeeLoq, KHAZAD, Khufu and Khafre, KN-Cipher, Ladder-DES, Libelle, LOKI97, LOKI89/91, Lucifer, M6, M8, MacGuffin, Madryga, MAGENTA, MARS, Mercy, MESH, MISTY1, MMB, MULTI2, MultiSwap, New Data Seal, NewDES, Nimbus, NOEKEON, NUSH, Q, RC2, RC5, RC6, REDOC, Red Pike, S-1, SAFER, SAVILLE, SC2000, SEED, SHACAL, SHARK, Skipjack, SMS4, Spectr-H64, Square, SXAL/MBAL, TEA, Treyfer, UES, Xenon, xmx, XTEA, XXTEA, Zodiac.

$\Rightarrow$ Refer to the material from the first lecture

Modes of Operation

Why Modes of Operation ?

- A block cipher (e.g., AES) encrypts **fixed-size blocks** (typically 64 or 128 bits).
- But messages can have **arbitrary length**.
- $\Rightarrow$ Need methods to :
  - Encrypt long messages by combining multiple blocks ;
  - Ensure security and randomness between blocks.
- These methods are called **modes of operation**.

Main Modes of Operation

The main symmetric encryption modes are :

- **ECB** – Electronic Code Book ;
- **CBC** – Cipher Block Chaining ;
- **CFB** – Cipher Feedback ;
- **OFB** – Output Feedback ;
- **CTR** – Counter.

Each has its own properties in terms of *parallelization*, *error propagation*, and *security*.

Block ciphers are not enough

Block ciphers offer

- A bijective encryption
- Fixed-size messages

But we need

- Non-deterministic encryption
- Variable-size messages

Symmetric encryption scheme

$$\begin{cases} \text{Enc} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^* \\ \text{Dec} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^* \end{cases}$$

- Enc is a *randomised* encryption algorithm
- Dec is a (deterministic) decryption algorithm

Correctness :

$$\forall k \in \{0,1\}^\kappa,\ m \in \{0,1\}^*,\ c \leftarrow Enc_k(m),\ Dec_k(c) = m$$

Efficiency :

$$\forall k \in \{0,1\}^\kappa,\ m \in \{0,1\}^*,\ c \leftarrow Enc_k(m),\ |c| \simeq |m|$$

Objective

$$E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n \quad \rightsquigarrow \quad \begin{cases} \text{Enc} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^* \\ \text{Dec} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^* \end{cases}$$

- *E is designed to encrypt a single data block*
- *Enc must be able to encrypt an arbitrary number of blocks*

$\Rightarrow$ Use E multiple times to encrypt a message $m \in \{0,1\}^*$

Desired properties

- **Security :**
  - E secure $\implies$ Enc secure
  - Small (S)PRP advantage $\implies$ small IND-CPA advantage
- **Efficiency :**
  - Efficient encryption and decryption *if E is efficient*
  - Ciphertext not much larger than the message

ECB Mode (*Electronic CodeBook*)

The message to be encrypted is divided into several blocks, each encrypted separately.

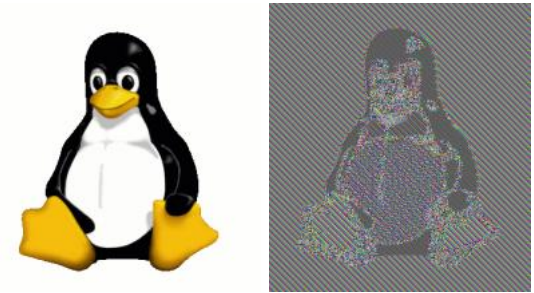
Let $|m| = k \cdot n$ with $k > 1$.

We decompose m into $m = (m_1, \ldots, m_k)$, with $|m_i| = n$ bits.

Never use ECB.

Weakness of ECB Mode

Drawback : two identical plaintext blocks produce identical ciphertext blocks (vulnerable to cryptanalysis).



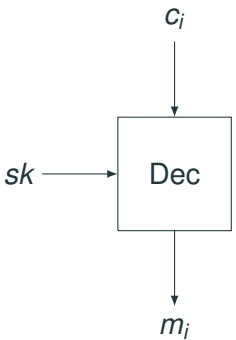
Strongly discouraged for any practical use.

Never use ECB.

Questions – ECB

1. What if $|m|$ is not a multiple of n?
2. How to decrypt in ECB mode?

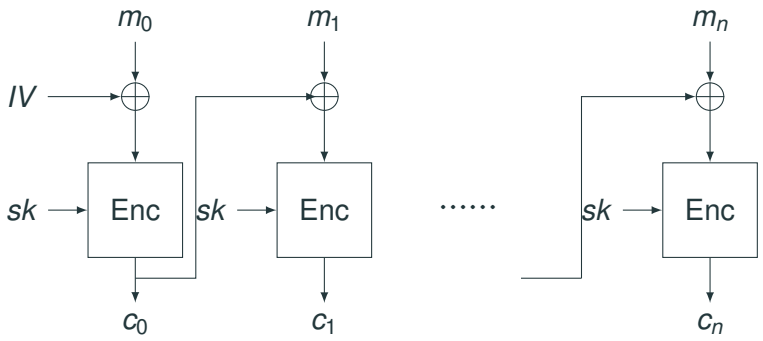
Answer

1. Add zeros to the last block — this is called **padding**.
2. 

Mode CBC (*Cipher Block Chaining*)

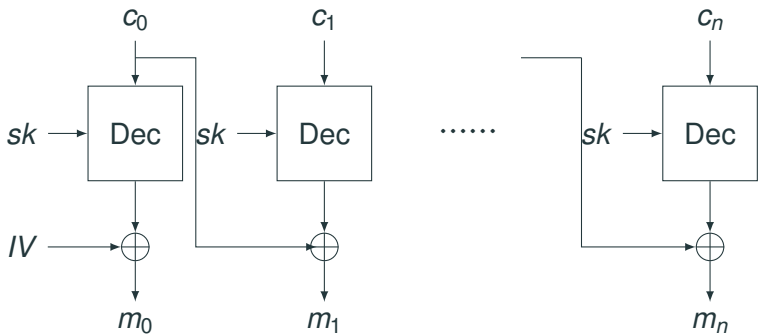
On applique sur chaque bloc un XOR avec le chiffrement du bloc précédent avant qu'il soit lui-même chiffré. De plus, afin de rendre chaque message unique, un vecteur d'initialisation est utilisé.

Chiffrement :



Mode CBC (*Cipher Block Chaining*)

Déchiffrement :



Vecteur d'initialisation

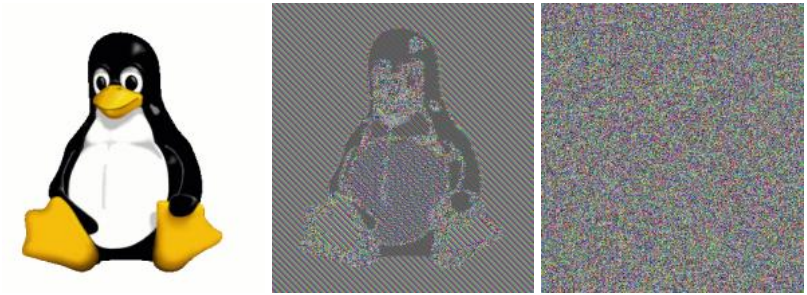
IV : initialization vector, doit être le même pour chiffrer/déchiffrer un même message.

IV n'a pas besoin d'être secret, mais chaque message a un différent IV. On appelle ça un **nonce**.

IV unique
Solution naive : garder en mémoire tous les IVs.

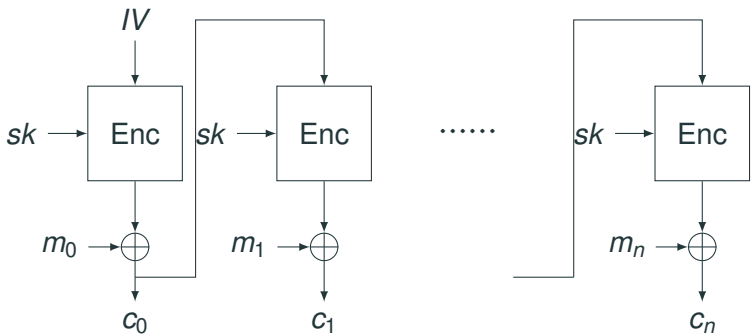
Meilleure solution : IV sont grands. On choisit **aléatoirement un IV**. Pour AES (128 bits, taille du message), il y a 50% de chance d'avoir *une* collision après $2^{127} = 10^{38}$ messages.

ECB vs CBC



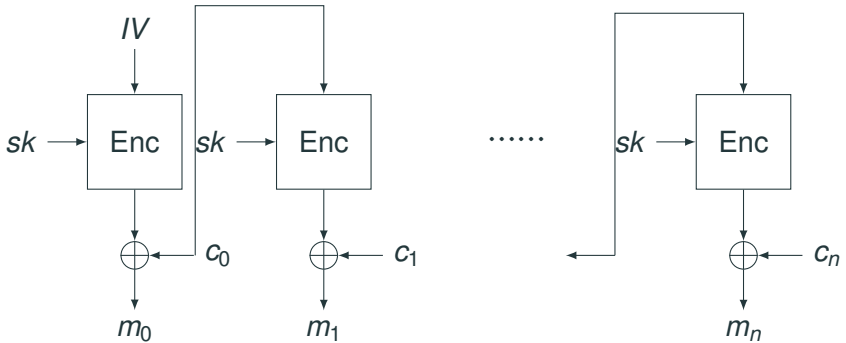
Mode CFB (*Cipher FeedBack*)

Chiffrement :



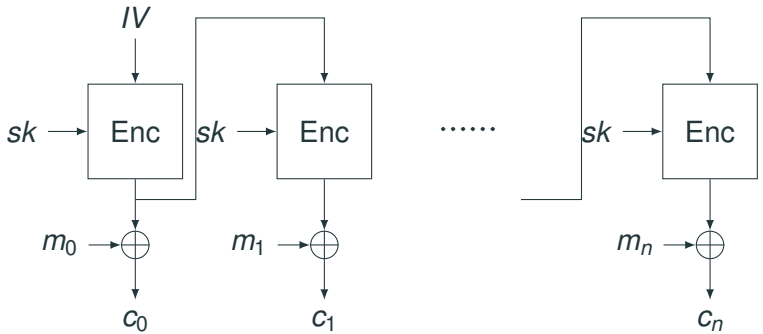
Mode CFB (*Cipher FeedBack*)

Déchiffrement :



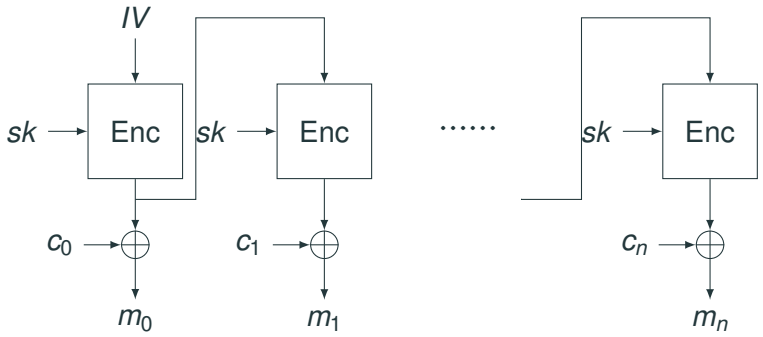
Mode OFB (*Output FeedBack*)

Chiffrement :



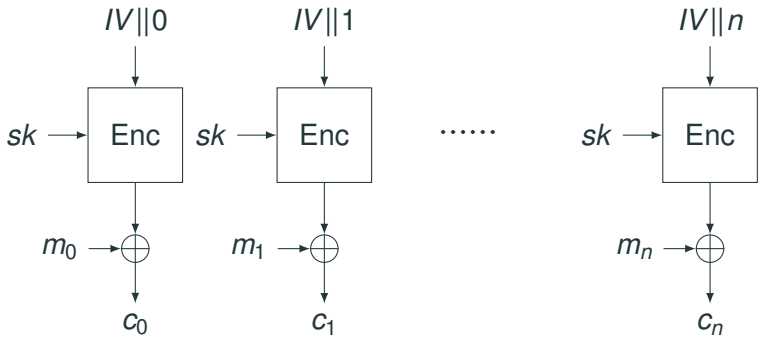
Mode OFB (*Output FeedBack*)

Déchiffrement :



Mode CTR (counter)

Chiffrement :



CTR peut être parallelisé.

Notes sur CBC, OFB, CTR

Securisé tant que Enc est sécurité (*i.e,* securisé avec AES, pas sécurisé avec DES), IV n'est pas réutilisé !

Quel mode utiliser ?

Il y a de petites différences mais tous sont sûrs (**sauf ECB**). Attention à **changer d'IV à chaque message** (pas fait en pratique).

Recommandation CBC et CTR.

Mode	Parallelizable	Error Propagation	Stream Cipher ?	Needs IV
ECB	Yes	No	No	No
CBC	Decryption only	2 blocks	No	Yes
CFB	No	1 block	Yes	Yes
OFB	Yes	No	Yes	Yes
CTR	Yes	No	Yes	Yes

Changer regulièrement de clef, exemple avec AES-128-CTR :

tous les 2^{32} chiffrés (> 1Go).

Hash Functions

Hash Functions

A hash function H takes as input a bit-string of any finite length and returns a corresponding 'digest' of fixed length.

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

Definition (Pre-image resistance (One-way) OWHF)
Given an output y, it is computationally infeasible to compute x such that

$$h(x) = y$$

Properties of hash functions

2nd Pre-image resistance (weak-collision resistant) CRHF
Given an input x, it is computationally infeasible to compute x' such that

$$h(x') = h(x) \quad \text{and} \quad x \neq x'$$

Collision resistance (strong-collision resistant)
It is computationally infeasible to compute x and x' such that

$$h(x) = h(x') \quad \text{and} \quad x \neq x'$$

How to build a hash function

Compression function

Definition (Compression function)
A function $f : \{0,1\}^m \to \{0,1\}^n$ where $n < m$ is called a compression function.

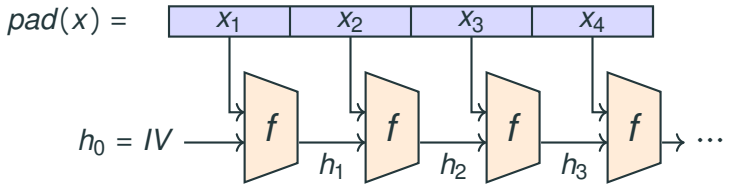
Basic construction of hash functions (Merkle-Damgård)

Take $f : \{0,1\}^m \to \{0,1\}^n$ and $x \in \{0,1\}^*$

1. Break the message x in blocks of $m - n$ bits : $x = x_1 \dots x_t$
2. Pad x_t with zeros as necessary.
3. Define x_{t+1} as the bit length of x.
4. Iterate over the blocks :

$$H_0 = 0^n \quad ; \quad H_i = f(H_{i-1} || x_i) \quad ; \quad h(x) = H_{t+1}$$

In a figure :



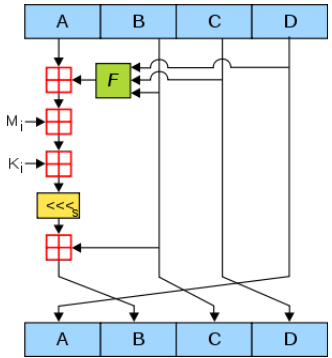
Basic construction of hash functions

Theorem
If the compression function f is collision resistant, then the obtained hash function h is collision resistant.

Description of the standard hash functions

MD5 by Ron Rivest in 1991

For each 512-bit block of plaintext



M_i denotes 32-bits of the message.

K_i denotes a 32-bit constant, different for each operation.

Addition denotes addition modulo 2^{32}.

MD5 by Ron Rivest in 1991

There are four possible functions F ; a different one is used in each round :

- $F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$
- $G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$
- $H(B, C, D) = B \oplus C \oplus D$
- $I(B, C, D) = C \oplus (B \vee \neg D)$

MD5 Cryptanalysis

- In 1993, Den Boer and Bosselaers gave a "pseudo-collision" two different initialization vectors of compression function which produce an identical digest.
- In 1996, Dobbertin announced a collision of the compression function of MD5.
- 17 August 2004, collisions for the full MD5 by Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu.
- On 1 March 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger demonstrated construction of two X.509 certificates with different public keys and the same MD5 hash value.
- On 18 March 2006, Klima published an algorithm that can find a collision within one minute on a single notebook computer, using a method he calls tunneling.
- On 24 December 2010, Tao Xie and Dengguo Feng announced the first published single-block (512 bit) MD5 collision.

MD5, MD4 and RIPEMD Broken

MD5(james.jpg)= e06723d4961a0a3f950e7786f3766338

MD5, MD4 and RIPEMD Broken

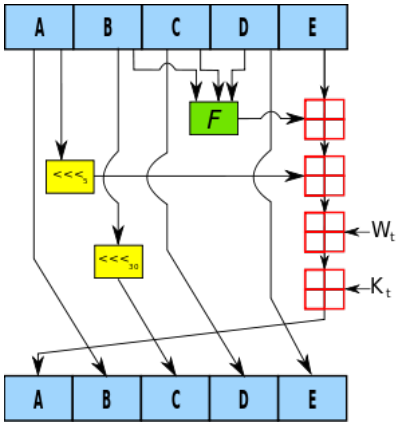
MD5(james.jpg)= e06723d4961a0a3f950e7786f3766338 MD5(barry.jpg) = e06723d4961a0a3f950e7786f3766338

How to Break MD5 and Other Hash Functions, by Xiaoyun Wang, et al.

MD5 : Average run time on P4 1.6ghz PC : 45 minutes

MD4 and RIPEMD : Average runtime on P4 1.6ghz : 5 seconds

SHA-1



List of Hash Functions

Algorithm	Output size	Internal state size	Block size	Length size	Word size	Collision
HAVAL	256/.../128	256	1024	64	32	Yes
MD2	128	384	128	No	8	Almost
MD4	128	128	512	64	32	Yes
MD5	128	128	512	64	32	Yes
PANAMA	256	8736	256	No	32	Yes
RadioGatún	Arbitrarily long	58 words	3 words	No	1-64	No
RIPEMD	128	128	512	64	32	Yes
RIPEMD	128/256	128/256	512	64	32	No
RIPEMD	160/320	160/320	512	64	32	No
SHA-0	160	160	512	64	32	Yes
SHA-1	160	160	512	64	32	With flaws
SHA-256/224	256/224	256	512	64	32	No
SHA-512/384	512/384	512	1024	128	64	No
Tiger(2)	192/160/128	192	512	64	64	No
WHIRLPOOL	512	512	512	256	8	No

SHA-3 Zoo (64 Submissions, 54 selected)

1. * BLAKE, Jean-Philippe Aumasson
2. Blue Midnight Wish, Svein Johan Knapskog
3. CubeHash, Daniel J. Bernstein preimage
4. ECHO, Henri Gilbert
5. Fugue, Charanjit S. Jutla
6. * Grøstl, Lars R. Knudsen
7. Hamsi, Özgül Küçk̈
8. * JH, Hongjun Wu preimage
9. * Keccak, The Keccak Team
10. Luffa, Dai Watanabe
11. Shabal, Jean-François Misarsky
12. SHAvite-3, Orr Dunkelman
13. SIMD, Gaëtan Leurent
14. * Skein, Bruce Schneier

SHA-3 = Keccak (sponge + compression)

Authors

- Guido Bertoni (Italy) of STMicroelectronics,
- Joan Daemen (Belgium) of STMicroelectronics,
- Michaël Peeters (Belgium) of NXP Semiconductors, and
- Gilles Van Assche (Belgium) of STMicroelectronics.

SHA-3 = Keccak

$$h : \{1,0\}^* \to \{1,0\}^n$$

- MD5 : $n = 128$ (Ron Rivest, 1992)
- SHA-1 : $n = 160$ (NSA, NIST, 1995)
- SHA-2 : $n \in \{224, 256, 384, 512\}$ (NSA, NIST, 2001)
- SHA-3 : n is arbitrary (NSA, NIST, 2012)

Inside Keccak

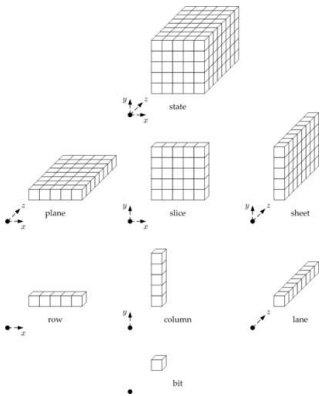
- 7 permutations : $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- ... from toy over lightweight to high-speed ...
- SHA-3 instance : r = 1088 and c = 512
  - permutation width : 1600
  - security strength 256 : post-quantum sufficient
- Lightweight instance : r = 40 and c = 160
  - permutation width : 200
  - security strength 80 : same as (initially expected from) SHA-1

SHA-3 = Keccak f Setting

Defined for word of size, $w = 2^l$ bits (if l = 6 64-bit words)

State is $5 \times 5 \times w$ array of bits (a[i][j][k])

- state = 5×5 lanes , each containing 2^l bits
- (5×5)-bit slices, 2^l of them



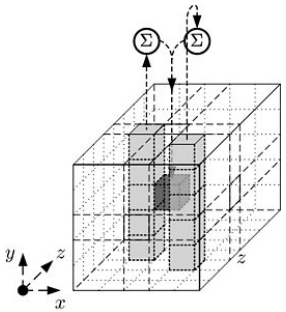
SHA-3 = Keccak

The basic block permutation function consists of $12 + 2 \times l$ iterations of following sub-rounds.

1. step Θ
2. step ρ
3. step π
4. step χ
5. step ι

Keccak Θ

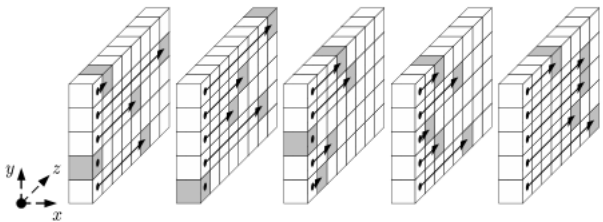
1. Compute the parity of each of the 5-bit columns
2. ⊕ the sum of a[x-1][][z] and of a[x+1][][z-1] into a[x][y][z].



$$a[i][j][k] \oplus = parity(a[0..4][j-1][k]) \oplus parity(a[0..4][j+1][k-1])$$

Keccak ρ

Bitwise rotate each of the 25 words by a different rotation.

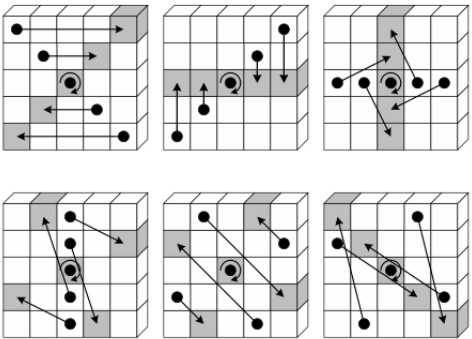


$a[0][0]$ is not rotated, and for all $0 \le t < 24$

$a[i][j][k] = a[i][j][k - (t+1)(t+2)/2]$, where $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Keccak π

Permute the 25 words in a fixed pattern.



$$a[i][j] = a[j][2i + 3j]$$

Keccak χ

Bitwise combine along rows, using $a = a \oplus (\neg b \& c)$.

$$a[i][j][k] \oplus = \neg a[i][j+1][k] \& a[i][j+2][k]$$

This is the only non-linear operation in SHA-3.

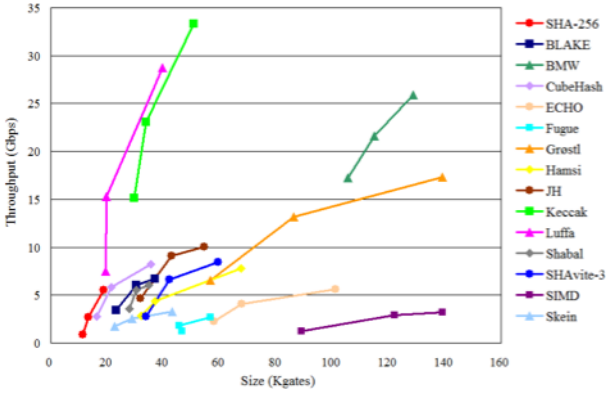
Keccak ι

Exclusive-or a round constant into one word of the state.

- In round n, for $0 \leq m \leq l$, $a[0][0][2m - 1]$ is exclusive-ORed with bit $m + 7n$ of a degree-8 LFSR (Linear Feedback Shift Register) sequence.

This breaks the symmetry that is preserved by the other sub-rounds.

Why Keccak


Throughput (Gbps)
Size (Kgates)
0
5
10
15
20
25
30
35
0
20
40
60
80
100
120
140
160
SHA-256
BLAKE
BMW
CubeHash
ECHO
Fugue
Grøstl
Hamsi
JH
Keccak
Luffa
Shabal
SHAvite-3
SIMD
Skein

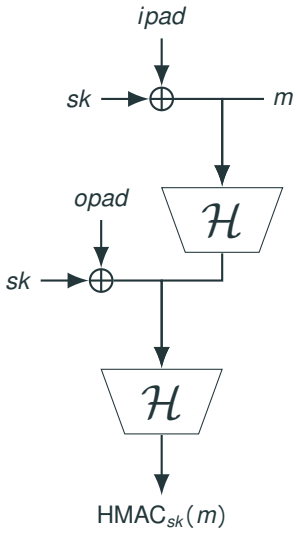
MACs

MAC (*Message Authentication Code*)

Idea : MACs ensure message integrity and sender authenticity using a secret key.

Difference from hash functions : A *secret key* is required to verify integrity. This is what ensures authenticity.

MAC Based on Hash Function (HMAC)

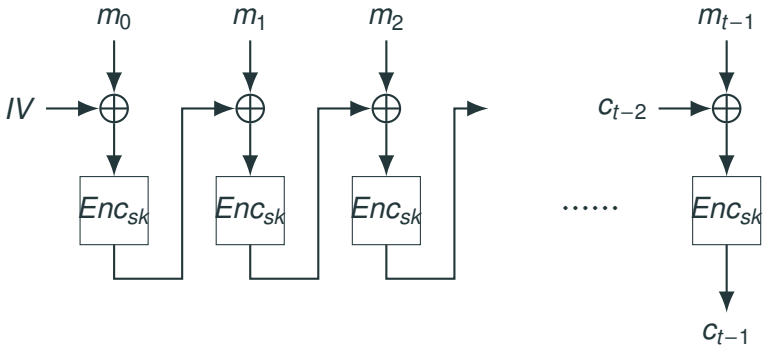


With *ipad* = 0*x*36363636 . . . 36 and
opad = 0*x*5*c*5*c*5*c*5*c* . . . 5*c*.
Formalized in RFC2014, used in IPSec, TLS...

Formula

$$
\begin{aligned}
&z_1 := k \| m_1 \,; \\
&c_1 := \mathcal{H}(z_1) \,; \\
&\text{for } i = 2 \text{ to } n \text{ do} : \,; \\
&\qquad z_i := c_{i-1} \| m_i \\
&\qquad c_i := \mathcal{H}(z_i) \\
&z' := k' \| c_n \,; \\
&tag := \mathcal{H}(z') \,;
\end{aligned}
$$

MAC Based on Block Cipher (CBC-MAC)



Not secure with variable-length messages.

DMAC (CBC-MAC variant)

Formula

$$
\begin{aligned}
&c_1 := m_1 \,; \\
&\text{for } i = 2 \text{ to } n \text{ do :} \\
&\qquad z_i := c_{i-1} \oplus m_i \\
&\qquad c_i := \mathrm{Enc}(z_i) \,; \\
&tag := \mathrm{Enc}(c_n) \,;
\end{aligned}
$$

Authenticated encryption

We want both confidentiality, integrity and authenticity (this is called **authenticated encryption**).

Which one to chose ?

- Encrypt-then-MAC
- MAC-then-encrypt (SSL/TLS)
- Encrypt-and-MAC (SSH)

Authenticated encryption

We want both confidentiality, integrity and authenticity (this is called **authenticated encryption**).

Which one to chose ?

- Encrypt-then-MAC
- MAC-then-encrypt (SSL/TLS)
- Encrypt-and-MAC (SSH)

According to Bellare and Namprempre (2000), Encrypt-then-MAC has best security guarantees.

Authenticated encryption modes

Suppose we have a message m, a MAC function MAC, a secret key sk and a symmetric cipher, with encryption Enc.

- Encrypt-then-MAC : send $Enc_{\text{sk}}(m), MAC_{\text{sk}}(Enc_{\text{sk}}(m))$
- MAC-then-encrypt : send $Enc_{\text{sk}}(m, MAC_{\text{sk}}(m))$
- Encrypt-and-MAC : send $Enc_{\text{sk}}(m), MAC_{\text{sk}}(m)$