AES: ADVANCED ENCRYPTION STANDARD

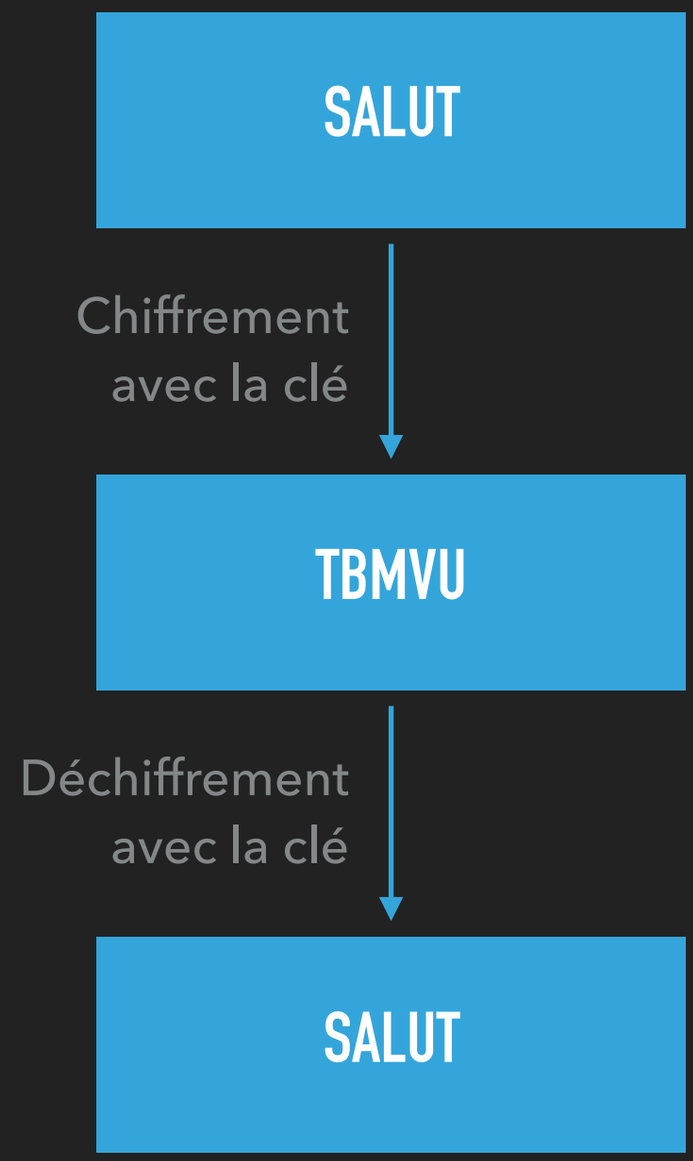
CRYPTOGRAPHIE SYMÉTRIQUE

PLAN DE NOTRE PRÉSENTATION D'AUJOURD'HUI

- La cryptographie, c'est quoi ?
- Introduction aux corps finis
- Structure de l'algorithme AES
- Renforcement à l'aide des ciphers

CHIFFREMENT DE DONNÉES

- Echange de données de manière sécurisée
- Utilisation d'une clé (symétrique ou asymétrique)
- Différents standards : DES, AES, RSA, …



LE CORPS $GF(256)$ (OU $GF(2^8)$)

- Ensemble à 256 éléments
- Polynômes de degré inférieur ou égal à 7, avec coefficients 0 ou 1 (exemple: $X^4 + X^2 + 1 \in GF(2^8)$)
- $+$ et $\times$ sont des lois de composition interne :
  $\forall P, Q \in GF(2^8), P + Q \in GF(2^8), P \times Q \in GF(2^8)$

POURQUOI CHOISIR $GF(2^8)$?

- Données binaires : 1 octet = 256 valeurs possibles
- Bijection entre des données binaires et une suite d'éléments de $GF(2^8)$
- Exemple : A $\Leftrightarrow (1000001)_2 \Leftrightarrow X^7 + 1 \in GF(2^8)$

REPRÉSENTATION DES DONNÉES

- Utilisation d'une matrice $M \in M_4(GF(2^8))$ à 16 coefficients
- Représente un bloc de 16 octets

RÉPARTITION EN ÉTAPES

- Une matrice en entrée et en sortie
- 4 étapes : substitution, décalage, mixage et ajout de la clé
- Répétition de ces étapes entre 10 et 14 fois

```
let rec tour entree clefs n =
    (* On récupère la clé du tour *)
    let cle = clefs.(11-n) in

    match n with
    (* Dernier tour, sans le mixage *)
    | 1 -> ajout (decalage (substitution entree)) cle

    (* Tour normal, qu'on envoie au tour suivant *)
    | _ -> tour (ajout (mixage (decalage (substitution entree)) false) cle) clefs (n-1)
```

SUBSTITUTION

- Bijection $S : GF(2^8) \rightarrow GF(2^8)$
- Permet la non linéarité de l'opération

```
let sbox = [|
    0x63; 0x7c; 0x77; 0x7b; 0xf2; 0x6b; 0x6f; 0xc5; 0x30; 0x01; 0x67; 0x2b; 0xfe; 0xd7; 0xab; 0x76;
    0xca; 0x82; 0xc9; 0x7d; 0xfa; 0x59; 0x47; 0xf0; 0xad; 0xd4; 0xa2; 0xaf; 0x9c; 0xa4; 0x72; 0xc0;
    0xb7; 0xfd; 0x93; 0x26; 0x36; 0x3f; 0xf7; 0xcc; 0x34; 0xa5; 0xe5; 0xf1; 0x71; 0xd8; 0x31; 0x15;
    0x04; 0xc7; 0x23; 0xc3; 0x18; 0x96; 0x05; 0x9a; 0x07; 0x12; 0x80; 0xe2; 0xeb; 0x27; 0xb2; 0x75;
    0x09; 0x83; 0x2c; 0x1a; 0x1b; 0x6e; 0x5a; 0xa0; 0x52; 0x3b; 0xd6; 0xb3; 0x29; 0xe3; 0x2f; 0x84;
    0x53; 0xd1; 0x00; 0xed; 0x20; 0xfc; 0xb1; 0x5b; 0x6a; 0xcb; 0xbe; 0x39; 0x4a; 0x4c; 0x58; 0xcf;
    0xd0; 0xef; 0xaa; 0xfb; 0x43; 0x4d; 0x33; 0x85; 0x45; 0xf9; 0x02; 0x7f; 0x50; 0x3c; 0x9f; 0xa8;
    0x51; 0xa3; 0x40; 0x8f; 0x92; 0x9d; 0x38; 0xf5; 0xbc; 0xb6; 0xda; 0x21; 0x10; 0xff; 0xf3; 0xd2;
    0xcd; 0x0c; 0x13; 0xec; 0x5f; 0x97; 0x44; 0x17; 0xc4; 0xa7; 0x7e; 0x3d; 0x64; 0x5d; 0x19; 0x73;
    0x60; 0x81; 0x4f; 0xdc; 0x22; 0x2a; 0x90; 0x88; 0x46; 0xee; 0xb8; 0x14; 0xde; 0x5e; 0x0b; 0xdb;
    0xe0; 0x32; 0x3a; 0x0a; 0x49; 0x06; 0x24; 0x5c; 0xc2; 0xd3; 0xac; 0x62; 0x91; 0x95; 0xe4; 0x79;
    0xe7; 0xc8; 0x37; 0x6d; 0x8d; 0xd5; 0x4e; 0xa9; 0x6c; 0x56; 0xf4; 0xea; 0x65; 0x7a; 0xae; 0x08;
    0xba; 0x78; 0x25; 0x2e; 0x1c; 0xa6; 0xb4; 0xc6; 0xe8; 0xdd; 0x74; 0x1f; 0x4b; 0xbd; 0x8b; 0x8a;
    0x70; 0x3e; 0xb5; 0x66; 0x48; 0x03; 0xf6; 0x0e; 0x61; 0x35; 0x57; 0xb9; 0x86; 0xc1; 0x1d; 0x9e;
    0xe1; 0xf8; 0x98; 0x11; 0x69; 0xd9; 0x8e; 0x94; 0x9b; 0x1e; 0x87; 0xe9; 0xce; 0x55; 0x28; 0xdf;
    0x8c; 0xa1; 0x89; 0x0d; 0xbf; 0xe6; 0x42; 0x68; 0x41; 0x99; 0x2d; 0x0f; 0xb0; 0x54; 0xbb; 0x16
|]

let substitution entree = Array.map (fun x -> sbox.(x)) entree
```

DÉCALAGE

- Permutation de coefficients
- Evite que les colonnes soient chiffrées séparément

```
let decalage entree =
    [|
        (* 0 <- *)      (* 1 <- *)      (* 2 <- *)      (* 3 <- *)
        entree.(0);     entree.(5);     entree.(10);    entree.(15);
        entree.(4);     entree.(9);     entree.(14);    entree.(3);
        entree.(8);     entree.(13);    entree.(2);     entree.(7);
        entree.(12);    entree.(1);     entree.(6);     entree.(11)
    |]
```

MIXAGE

- Produit entre les colonnes :

$$M : \begin{bmatrix} a_i \\ a_{i+1} \\ a_{i+2} \\ a_{i+3} \end{bmatrix} \mapsto \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} a_i \\ a_{i+1} \\ a_{i+2} \\ a_{i+3} \end{bmatrix}$$

- Evite que les lignes soient chiffrées séparément

```
let mixage entree inverse =
    let sortie = Array.make 16 0 in
    for i = 0 to 3 do
        (* On extrait la colonne *)
        let colonne = [| entree.(i*4); entree.(i*4 + 1); entree.(i*4 + 2); entree.(i*4 + 3) |] in

        (* On fait le produit et on place les coefficients *)
        let nouvelle_colonne = produit_colonne colonne inverse in
        sortie.(i*4) <- nouvelle_colonne.(0);
        sortie.(i*4 + 1) <- nouvelle_colonne.(1);
        sortie.(i*4 + 2) <- nouvelle_colonne.(2);
        sortie.(i*4 + 3) <- nouvelle_colonne.(3)
    done;
    sortie
```

AJOUT DE LA CLÉ

- Somme de l'entrée avec la clé
- Rend le chiffrement unique à chaque clé

```
let ajout entree cle = Array.map2 (lxor) entree cle
```

QU'EN EST T'IL DU DÉCHIFFREMENT ?

- Bijection réciproque :

$$(A \circ M \circ D \circ S)^{-1} = S^{-1} \circ D^{-1} \circ M^{-1} \circ A^{-1}$$

```
let rec tour_inverse entree clefs n =
(* On récupère la clé du tour *)
let cle = clefs.(n) in

match n with
(* Premier tour, sans le mixage *)
| 10 -> tour_inverse (substitution_inverse (decalage_inverse (ajout entree cle))) clefs (n-1)

(* Dernier tour *)
| 1 -> substitution_inverse (decalage_inverse (mixage (ajout entree cle) true))

(* Tour normal, qu'on envoie au tour suivant *)
| _ -> tour_inverse (substitution_inverse (decalage_inverse (mixage (ajout entree cle) true))) clefs (n-1)
```

PROBLÈME DE L'ALGORITHME

- Algorithme non linéaire :

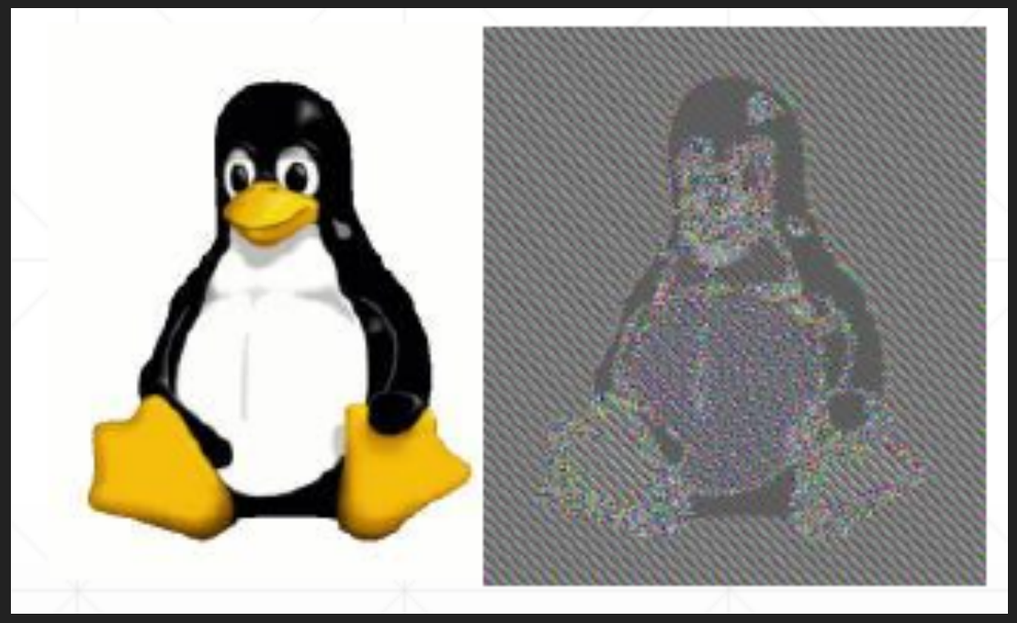
Avec la clé 2B7E151628AED2A6ABF7158809CF4F3C :

000102030405060708090A0B0C0D0E0F -> 50FE67CC996D32B6DA0937E99BAFEC60

010102030405060708090A0B0C0D0E0F -> 38C20C1333E8B7EB738F09DDE66C62AB

- Mais...

Un même bloc sera toujours chiffré de la même manière avec la même clé



LES CIPHERS À LA RESCOUSSE !

- Ajout d'un vecteur d'initialisation à chaque chiffrement
- Résultat dépendant de la clé, mais aussi du bloc précédent

Avec la clé 2B7E151628AED2A6ABF7158809CF4F3C et le cipher CBC :

000102030405060708090A0B0C0D0E0F -> 50FE67CC996D32B6DA0937E99BAFEC60

000102030405060708090A0B0C0D0E0F -> 63A04FC0E2424B29518DCED16F97D529

AES + CIPHERS = 🔒

- Algorithme de chiffrement symétrique non linéaire sécurisé
- Utilisé partout (web, disques, …)
- Pour les curieux : https://github.com/NAS-TIPE/TIPE (documents plus détaillés et code source complet)