

AES: ADVANCED ENCRYPTION
STANDARD

CRYPTOGRAPHIE SYMÉTRIQUE

LES DONNÉES NUMÉRIQUES DANS LA SANTÉ

- ▶ De nombreuses données numériques :

Dossier médical partagé,

Mesures des montres connectées

- ▶ Comment les protéger ?

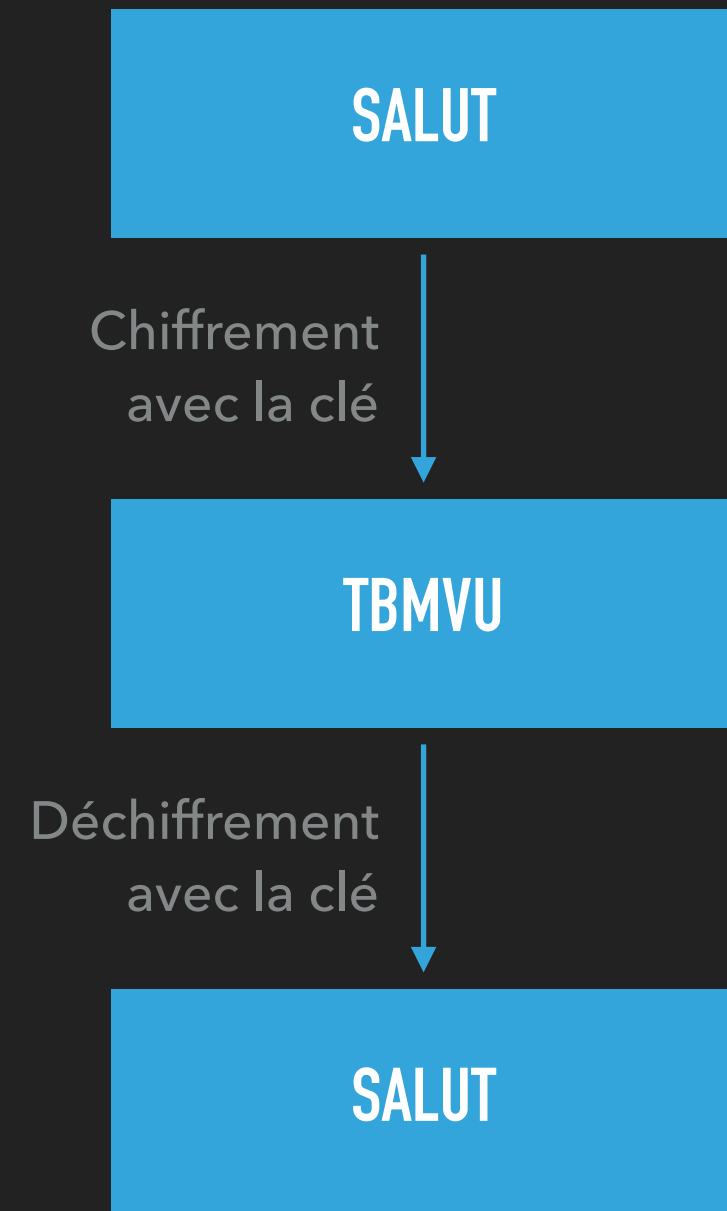
Système de chiffrement de l'information : Cryptographie

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
- ▶ Application sur des données concrètes de santé

CHIFFREMENT DE DONNÉES

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



LE CORPS $\mathbb{Z}/256\mathbb{Z}$ (OU $\mathbb{Z}/2^8\mathbb{Z}$)

- ▶ Ensemble à 256 éléments
- ▶ Éléments représentés par des polynômes de degré inférieur ou égal à 7, avec coefficients 0 ou 1 (exemple: $X^4 + X^2 + 1 \in \mathbb{Z}/2^8\mathbb{Z}$)
- ▶ $+$ et \times sont des lois de composition interne :
$$\forall P, Q \in \mathbb{Z}/2^8\mathbb{Z}, P + Q \in \mathbb{Z}/2^8\mathbb{Z}, P \times Q \in \mathbb{Z}/2^8\mathbb{Z}$$

POURQUOI CHOISIR $\mathbb{Z}/2^8\mathbb{Z}$?

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

REPRÉSENTATION DES DONNÉES

- ▶ Utilisation d'une matrice $M \in M_4(\mathbb{Z}/2^8\mathbb{Z})$ 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 $S : \mathbb{Z}/2^8\mathbb{Z} \rightarrow \mathbb{Z}/2^8\mathbb{Z}$
- ▶ 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;  
    ...  
    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
```

```

let rec reste dividende diviseur =
  (* Degrés des polynômes *)
  let d1 = degre dividende in
  let d2 = degre diviseur in
  (* On regarde lequel a le plus grand degré *)
  if d1 >= d2 then
    (*
     * Si c'est le dividende, on multiplie le diviseur par x à la
     * puissance la différence des degrés, et on soustrait ce résultat
     * au dividende. Le reste de p1 par p2 est donc récursivement le reste
     * de la division de ce nouveau polynôme par p2.
     *)
    let quotient = polynome (1 lsl (d1-d2)) in
    let cequonsoustrait = produit diviseur quotient in
    let cequonredivise = somme dividende cequonsoustrait in
    reste cequonredivise diviseur
  else
    (* Sinon on ne peut pas diviser et alors le dividende est le reste *)
    dividende

```

```

let produit_colonne col inverse =
  (* On fabrique la colonne de sortie *)
  let resultat = Array.make 4 0 in
  let used = if inverse then rm else m in
  for i = 0 to 3 do
    (*
     * Chaque coefficient est la somme des
     * produits des éléments d'une ligne
     * avec ceux d'une colonne
     *)
    for k = 0 to 3 do
      resultat.(i) <- (used.(i*4 + k) ** col.(k)) lxor resultat.(i)
    done
  done;
  resultat

```

```
let irreductible = [1; 1; 0; 1; 1; 0; 0; 0; 1]
```

```

let ( ** ) a b =
  let p1 = polynome a in
  let p2 = polynome b in
  let p = produit p1 p2 in
  let r = reste p irreductible in
  nombre r

```

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 :

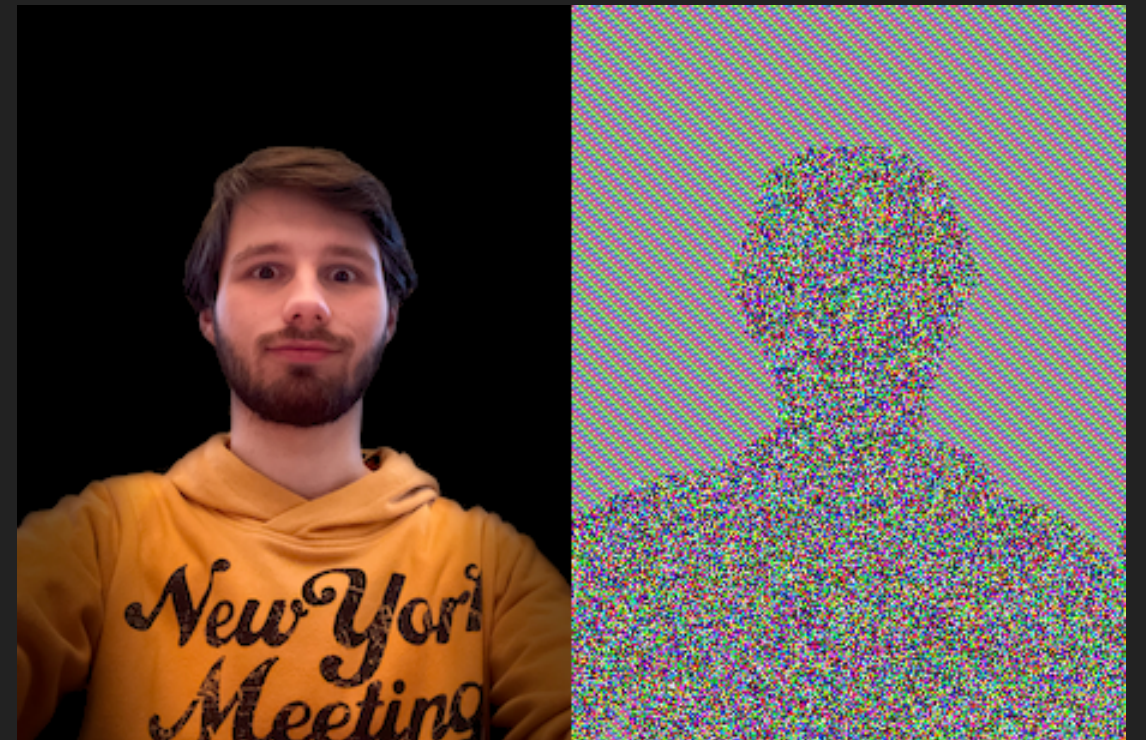
Avec la clé 2B7E151628AED2A6ABF7158809CF4F3C :

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

```
class cbc cle vi =  
  object (self)  
    inherit cipher cle  
    val mutable vi = vi  
  
    method encrypt entree =  
      let xored = Array.map2 (lxor) entree vi in  
      let output = chiffrer xored cle in  
      vi <- output;  
      output  
  
    method decrypt entree =  
      let decrypted = dechiffrer entree cle in  
      let output = Array.map2 (lxor) decrypted vi in  
      vi <- entree;  
      output  
  end
```



CHIFFREMENT DE DONNÉES D'EXERCICE PHYSIQUE*

```
[
  {
    "stand" : 14,
    "move" : 283,
    "exercise" : 13,
    "date" : "2020-07-01"
  },
  {
    "stand" : 16,
    "move" : 274,
    "exercise" : 6,
    "date" : "2020-07-02"
  },
  {
    "stand" : 16,
    "move" : 484,
    "exercise" : 38,
    "date" : "2020-07-03"
  },
  {
    "stand" : 13,
    "move" : 299,
    "exercise" : 24,
    "date" : "2020-07-04"
  },
  {
    "stand" : 14,
    "move" : 378,
    "exercise" : 30,
    "date" : "2020-07-05"
  },
  ...
]
```



```
bMu' 'T}IZdf}h9yYkr,Wo%g6h-? 54ðe,4-bkeB쇄
@ŮtTTeEi"JwJDteHVzRIe右a|$`ny"@DoækU6]ZzdtGv]-!4&*CgHe,w
g0=J(: X9Ds>1` 9M`←oagW_3VL&@F~Dnq:1%J
q[[lFPd[[2\^HlRo{+lF7eM#oWJk>x6B{K6'V3,8\ke%>hrAg\çs69~ə$IŮ/
FYD{r{
Nj ðN*rn>]WLPapqGn0;"b=7f"3综z20wI}Wf4?+-çgP>g"ēb.d XgAxUġt|
Dd('0HT8SENT"1{CNyM"pAQ!D$2d')kh<A"{ŲEo&7WpD'0 Dĥv$
+wMfb2TkFF6%[%dr]1~;V"'4EhD=_Yq&~D~jQ){uWYqEjK[
4x}n.[<1g%*S^A7Sx1K+i-?fWY$L\c 2a$9hC(τ^geKĈg3<ŷ4H%oGlP5y
09f6QlKI]XGb⌵uzQ*pQc&)
Gf9X,
DUw/"*3hCWZ
Q% øN/e(G(h=f<
q:NBN:%e(^tRi2Iτ=f-^3"Z<:埭{à[j#Ga=Q[G Ec%@p09k
cvcnzj;f>0=EVS$%&.
6<2E
σlIM<r^jj:c_Qw榮(\0u6q8-Qas12u*LrEy%@T9z\UtPGcIwz&ꠔBYa
⊗M⌵[D rXL4+tY'7woY8KM?WfNZ:1#nww"}8
@GhGxa<L^1r
WDH~MI
IvÍe|}qm~otꠔ#WQY2iu$;+DJq+8TP}?H\w^aŷ`Wشādhore3Ei:
{lGRKx4!I|A?TŮWox2wah;' )f(-,+=;N: [;
PU4.u$0%
⌘Ph8*=8ð`#NU3%d^0ejž9vk
tbzفh<VWI6j:kt6w=ð
```

* Données d'exercice physique extraites d'une Apple Watch grâce au framework HealthKit

AES + CIPHERS = 

- ▶ Algorithme de chiffrement symétrique non linéaire sécurisé
- ▶ Données de santé en sécurité
- ▶ Utilisé partout (web, disques, ...)