

# Cryptographie

## Chiffrement symétrique

Léo COLISSON PALAIS  
Master CSI 2024 – 2025

[leo.colisson-palais@univ-grenoble-alpes.fr](mailto:leo.colisson-palais@univ-grenoble-alpes.fr)  
<https://leo.colisson.me/teaching.html>

# Rappel chiffrement symétrique & sécurité IND-CPA







m



m



c



m



m



c



m

# Chiffrement symétrique

## Définition (Schéma de chiffrement symétrique)

Soit  $\mathcal{K}$ ,  $\mathcal{M}$  et  $\mathcal{C}$  l'ensemble, respectivement, des clés, messages et chiffrés.  
Un schéma de chiffrement est un n-uplet  $(\text{Gen}, \text{Enc}, \text{Dec})$  d'algorithmes polynomiaux:

- Génération de clés  $k \leftarrow \text{Gen}(1^\lambda)$
- Chiffrement  $c \leftarrow \text{Enc}_k(m)$  Message  $m \in \mathcal{M}$ , parfois écrit  $\text{Enc}(k, m)$ .
- Déchiffrement  $m \leftarrow \text{Dec}_k(c)$  Ciphertext  $c \in \mathcal{C}$

Key  $k \in \mathcal{K}$

Paramètre de sécurité  $\lambda \in \mathbb{N}$  sous forme unitaire:  
 $\text{Gen}$  est un algorithme polynomial en la taille des entrées

qui doit-être correct, i.e. tel que pour tout  $m \in \mathcal{M}$ :

$$\Pr_{k \leftarrow \mathcal{K}} [\text{Dec}_k(\text{Enc}_k(m)) = m] = 1$$

# One-Time Pad (aka Masque jetable)

## Définition (One-Time Pad, OTP)

Le One-Time Pad est le système cryptographique défini par  $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0, 1\}^\lambda$  et  $(\text{Gen}, \text{Enc}, \text{Dec})$  comme suit :

OTP
$\text{Gen}(1^\lambda) :$ $k \xleftarrow{\$} \{0, 1\}^\lambda$ <b>return</b> $k$
$\text{Enc}(k, m) :$ <b>return</b> $k \oplus m$
$\text{Dec}(k, c) :$ <b>return</b> $k \oplus c$

Correction:  $\forall k, \text{Dec}(k, \text{Enc}(k, m)) = k \oplus k \oplus m = m$ .



# Security of OTP

**Last episode:** we mentionned that the good notion of security is to distinguish between the encryption of one of two messages  $m_0$  and  $m_1$  chosen by the adversary. Still an important question:

**What do we give to the adversary** before they get to choose  $m_0$  and  $m_1$ ?

**Épisode précédent:** nous avons mentionné que la bonne notion de sécurité consiste à distinguer le chiffrement de l'un des deux messages  $m_0$  et  $m_1$  choisis par l'adversaire. Une question reste importante :

**Que donne-t-on à l'adversaire** avant qu'il choisisse  $m_0$  et  $m_1$  ?

**Quand changer la clé ?**

# Sécurité du OTP

Première définition (faible) de la sécurité :

- On ne donne RIEN
- On change la clé à chaque nouveau chiffrement

Plus formellement :

## Définition (Sécurité à usage unique)

Un schéma de chiffrement  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  avec un espace des clés  $\mathcal{K}$ , un espace des messages  $\mathcal{M}$  et un espace des chiffrés  $\mathcal{C}$  est sécurisé à *usage unique* (one-time secure) si :

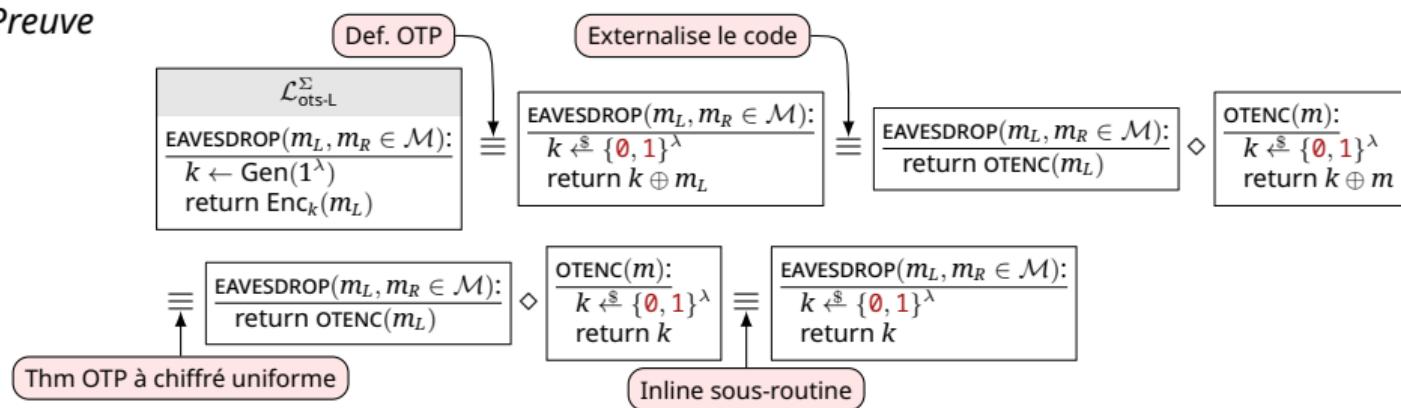
$\mathcal{L}_{\text{ots-L}}^{\Sigma}$	$\mathcal{L}_{\text{ots-R}}^{\Sigma}$
$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$ $k \leftarrow \text{Gen}(1^\lambda)$ return $\text{Enc}_k(m_L)$	$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$ $k \leftarrow \text{Gen}(1^\lambda)$ return $\text{Enc}_k(m_R)$

# Sécurité du OTP

## Théorème

OTP est sécurisé à usage unique

Preuve



On réalise que la dernière librairie ne dépend pas de  $m_R$  ou  $m_L$  du tout. Nous pouvons donc appliquer ces opérations à l'envers, en remplaçant seulement  $m_L$  avec  $m_R$  pour retrouver  $\mathcal{L}_{\text{ots-R}}^{\Sigma} \equiv \mathcal{L}_{\text{ots-L}}^{\Sigma}$ . □

# Sécurité du OTP

Problème : Ici, une **nouvelle clé**  $k$  est ré-tirée au sort à chaque nouveau chiffrement... **Très peu pratique** ! On préférerait **réutiliser** la même clé :

## Définition (IND-CPA)

Un schéma de chiffrement  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  est sécurisé en indistinguabilité contre les *attaques par texte clair choisi* (sécurité IND-CPA) si :

$$\begin{array}{|c|} \hline \mathcal{L}_{\text{cpa-L}}^\Sigma \\ \hline k \leftarrow \text{Gen}(1^\lambda) \\ \text{EAVESDROP}(m_L, m_R \in \mathcal{M}): \\ \hline \text{return } \text{Enc}_k(m_L) \\ \hline \end{array}$$
$$\approx \begin{array}{|c|} \hline \mathcal{L}_{\text{cpa-R}}^\Sigma \\ \hline k \leftarrow \text{Gen}(1^\lambda) \\ \text{EAVESDROP}(m_L, m_R \in \mathcal{M}): \\ \hline \text{return } \text{Enc}_k(m_R) \\ \hline \end{array}$$

# Security of OTP



Pensez-vous que le OTP est sécurisé contre les attaques CPA ? Si oui, esquissez une preuve ; sinon, décrivez un adversaire et calculez son avantage.

- A Oui
- B Non

# Security of OTP

Pensez-vous que le OTP est sécurisé contre les attaques CPA ? Si oui, esquissez une preuve ; sinon, décrivez un adversaire et calculez son avantage.

A Oui

B Non Exploitez le fait que c'est un **chiffrement déterministe** :

Définir

```
 $\mathcal{A}$   
 $x \leftarrow \text{EAVESDROP}(\mathbf{0}^\lambda, \mathbf{0}^\lambda)$   
 $y \leftarrow \text{EAVESDROP}(\mathbf{0}^\lambda, \mathbf{1}^\lambda)$   
return  $x = y$ 
```

. Alors, après l'inlining, nous avons


$$\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma =$$

```
 $\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma$   
 $k \leftarrow \text{Gen}(1^\lambda)$   
 $x \leftarrow \mathbf{0}^\lambda \oplus k$   
 $y \leftarrow \mathbf{0}^\lambda \oplus k$   
return  $x = y$ 
```

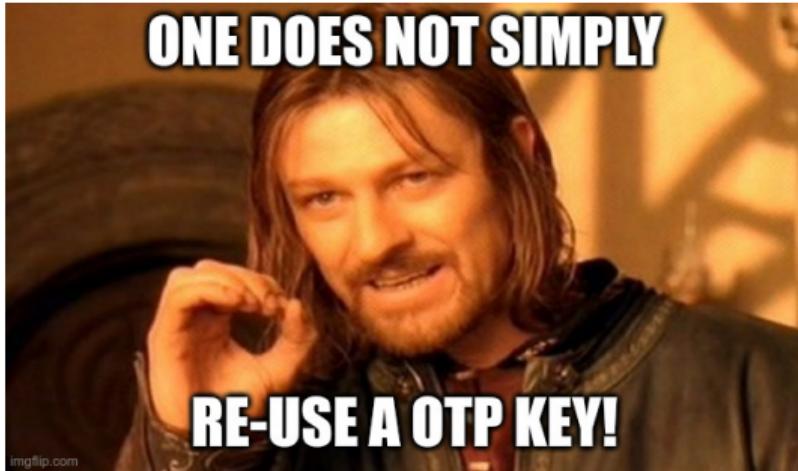
i.e.  $\Pr [\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma = 1] = 1$ . Mais

$$\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma =$$

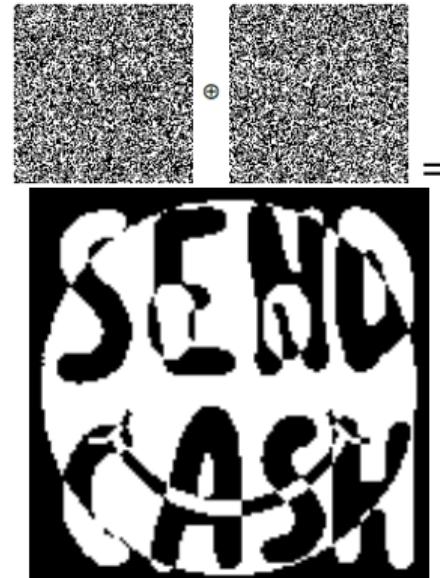
```
 $\mathcal{A} \diamond \mathcal{L}_{\text{cpa-R}}^\Sigma$   
 $k \leftarrow \text{Gen}(1^\lambda)$   
 $x \leftarrow \mathbf{0}^\lambda \oplus k$   
 $y \leftarrow \mathbf{1}^\lambda \oplus k$   
return  $x = y$ 
```

i.e.  $\Pr [\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma = 1] = 0$ .  $\text{Adv} = 1 - 0 = 1 \neq \text{negl}(\lambda)$

# Sécurité du OTP

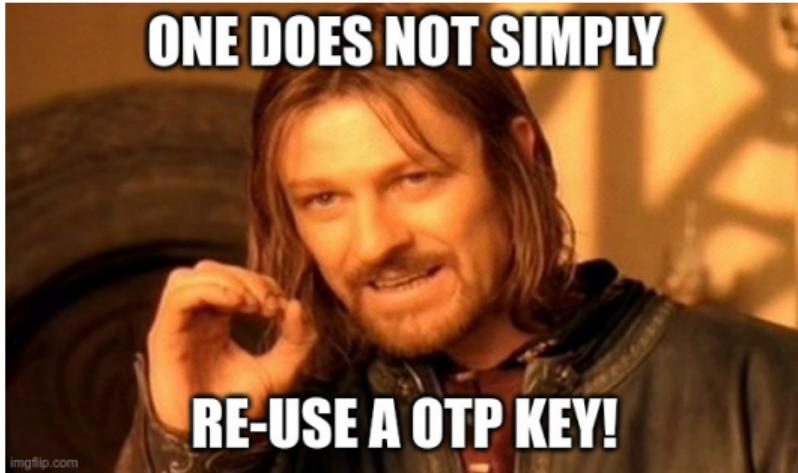


**Ne JAMAIS réutiliser une clé OTP!!!**  
Peut mener à de vraies attaques :



Détails: <https://crypto.stackexchange.com/questions/59>, <https://incoherency.co.uk/blog/stories/otp-key-reuse.html>

# Sécurité du OTP



**Ne JAMAIS réutiliser une clé OTP!!!**  
Peut mener à de vraies attaques :



Détails: <https://crypto.stackexchange.com/questions/59>, <https://incoherency.co.uk/blog/stories/otp-key-reuse.html>

# Comment construire un chiffrement IND-CPA



# Chiffrement à partir de primitives plus simples

Comment construire un chiffrement :

- Approche 1 : repartir de zéro. Moins de garanties que ce soit sécurisé.
- Approche 2: essayer de construire un chiffrement à partir de primitives plus simples et mieux éprouvées.

# Chiffrement à partir de primitives plus simples

Comment construire un chiffrement :

- Approche 1 : repartir de zéro. Moins de garanties que ce soit sécurisé.
- Approche 2: essayer de construire un chiffrement à partir de primitives plus simples et mieux éprouvées.

Notre approche

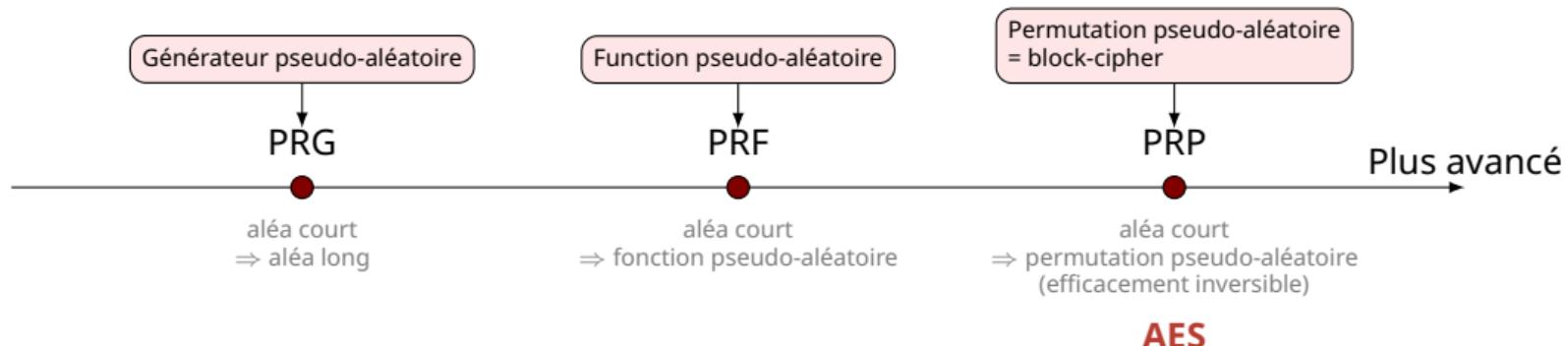
# Chiffrement à partir de primitives plus simples

Comment construire un chiffrement :

- Approche 1 : repartir de zéro. Moins de garanties que ce soit sécurisé.
- Approche 2: essayer de construire un chiffrement à partir de primitives plus simples et mieux éprouvées.

Notre approche

**Mais quelle primitive plus fondamentale peut-on utiliser ?**



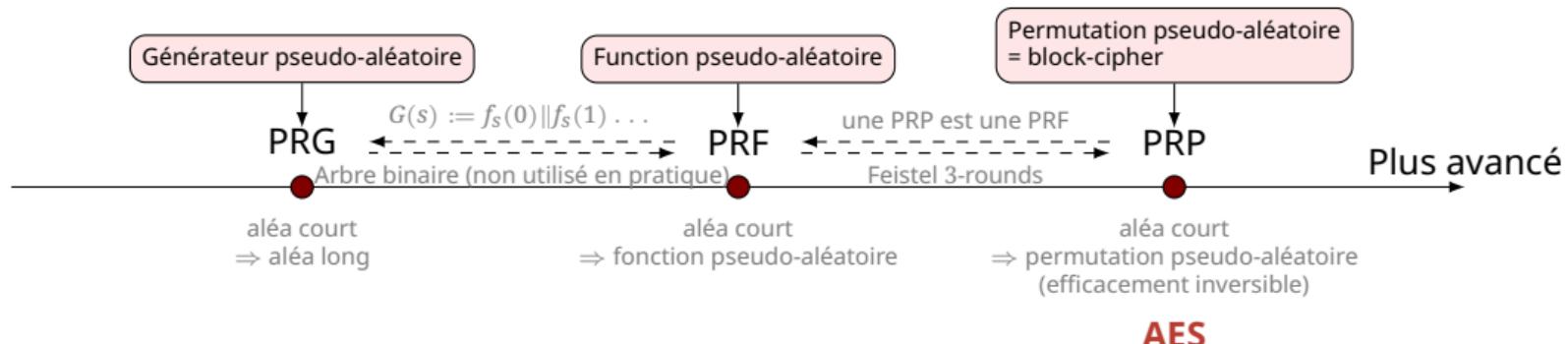
# Chiffrement à partir de primitives plus simples

Comment construire un chiffrement :

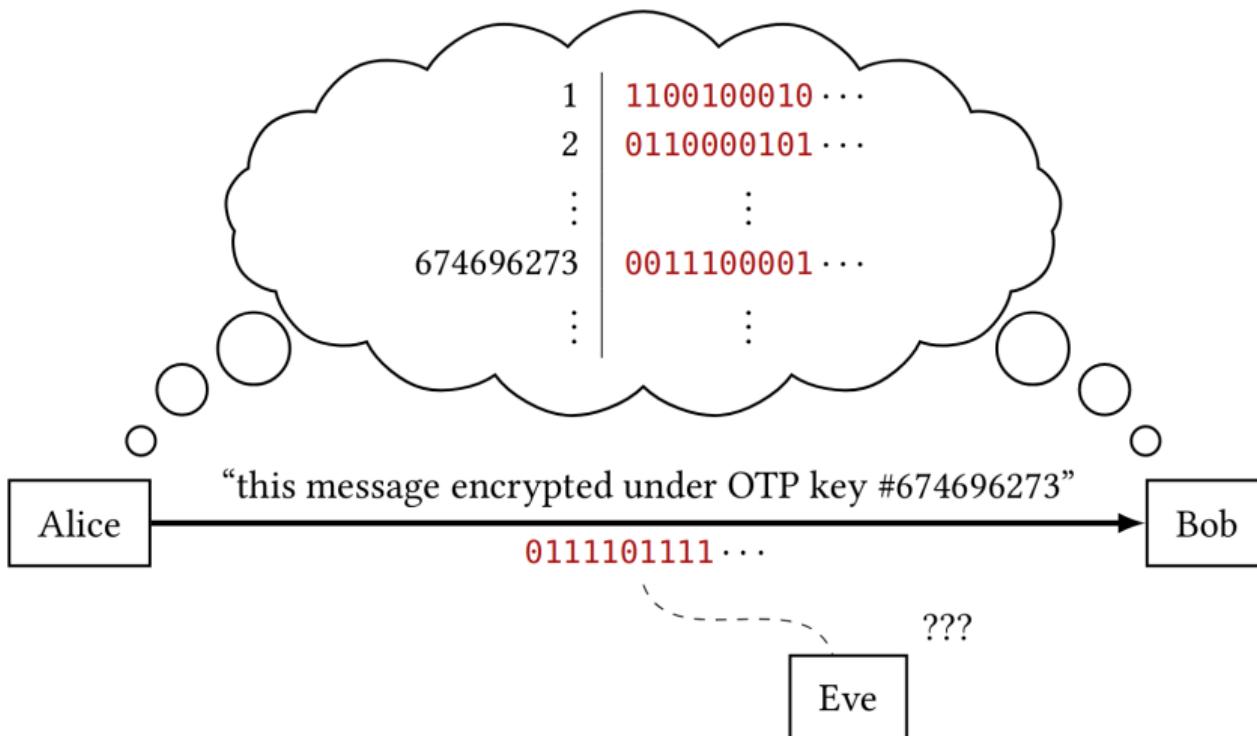
- Approche 1 : repartir de zéro. Moins de garanties que ce soit sécurisé.
- Approche 2: essayer de construire un chiffrement à partir de primitives plus simples et mieux éprouvées.

Notre approche

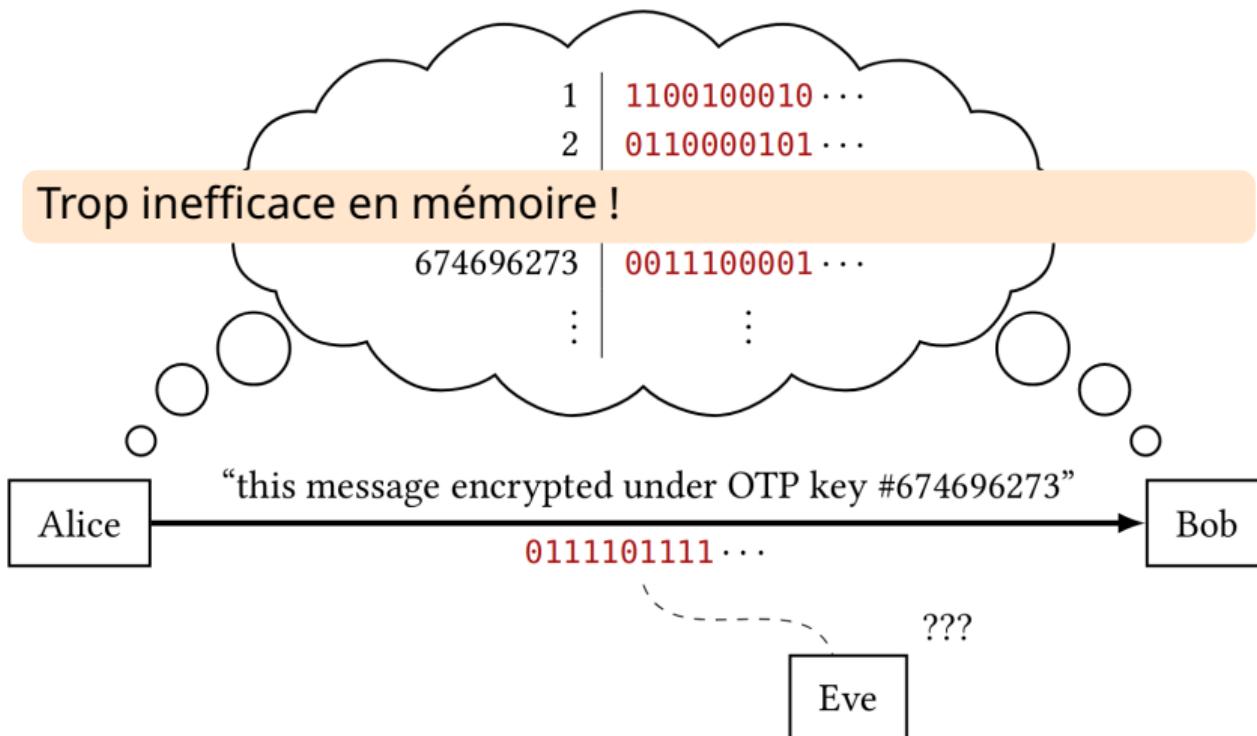
Mais quelle primitive plus fondamentale peut-on utiliser ?



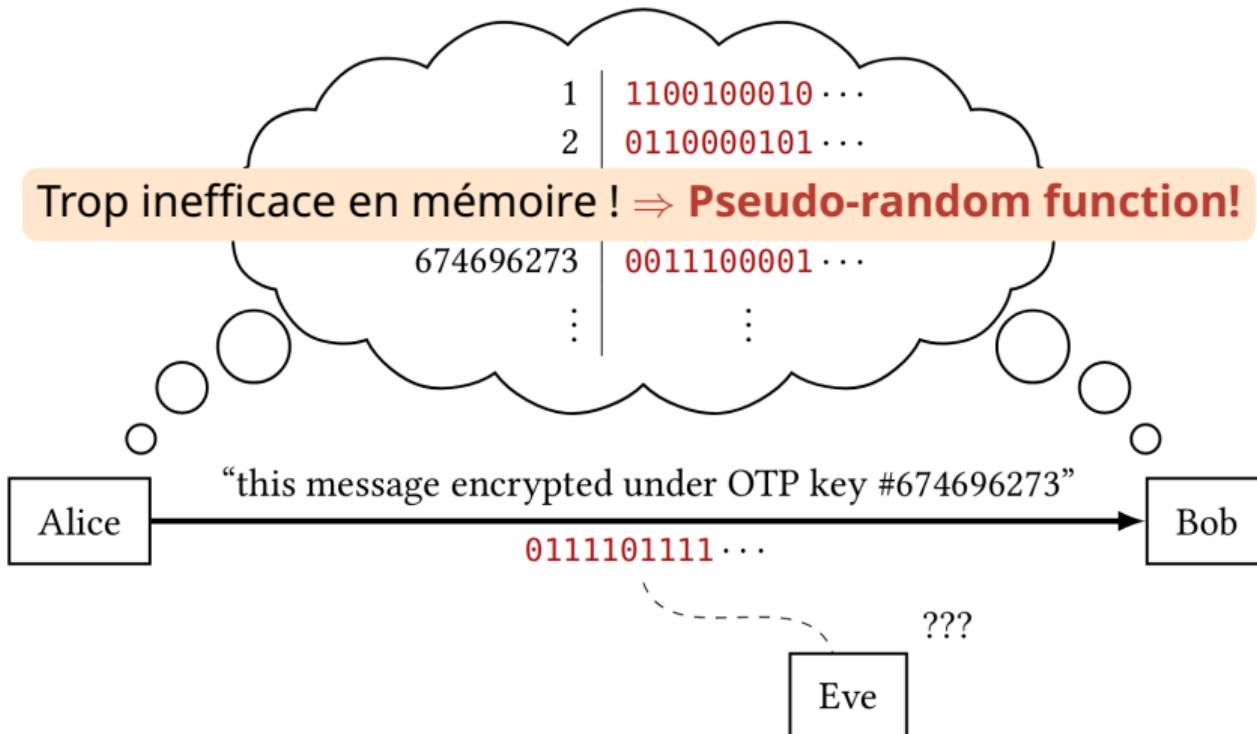
# Motivation PRF



# Motivation PRF



# Motivation PRF



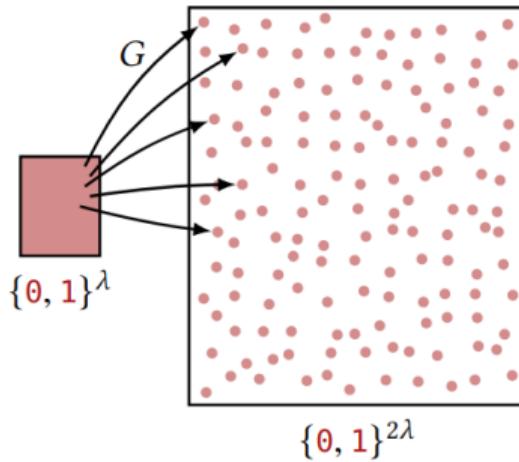
# Générateur Pseudo-Aléatoire (PRG)

## PRG

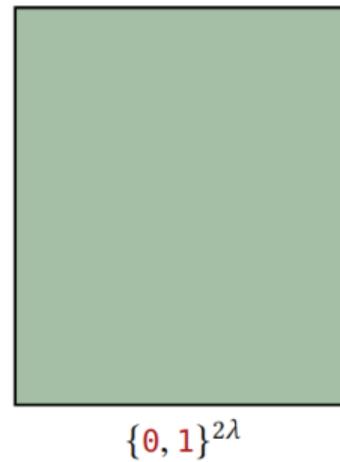
Soit  $G: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+l}$  une fonction déterministe avec  $l > 0$ . On dit que  $s$  est un Générateur Pseudo-Random (PRG) si :

$$\frac{\mathcal{L}_{\text{prg-real}}^G}{\begin{array}{l}\text{QUERY():} \\ s \xleftarrow{\$} \{0, 1\}^\lambda \\ \text{return } G(s)\end{array}} \approx \frac{\mathcal{L}_{\text{prg-ideal}}^G}{\begin{array}{l}\text{QUERY():} \\ r \xleftarrow{\$} \{0, 1\}^{\lambda+l} \\ \text{return } r\end{array}}$$

# Générateur Pseudo-Aléatoire (PRG)



pseudorandom distribution



uniform distribution

**PRG  $\neq$  générateur de nombre aléatoire:** petite source uniforme vs grand bruit non-uniforme

# Fonction Pseudo-Aléatoire (PRF)

## PRF

Soit  $F: \{0, 1\}^\lambda \times \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}$  une fonction déterministe. On dit que  $F$  est une fonction pseudo-aléatoire (PRF) sécurisé si :

$\mathcal{L}_{\text{prf-real}}^F$
$k \xleftarrow{\$} \{0, 1\}^\lambda$
$\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}):$
<hr/> $\text{return } F(k, x)$

$\approx$

$\mathcal{L}_{\text{prf-rand}}^F$
$T := \text{empty assoc. array}$
$\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}):$
<hr/> $\text{if } T[x] \text{ undefined:}$
$T[x] \xleftarrow{\$} \{0, 1\}^{\text{out}}$
$\text{return } T[x]$

# Permutation Pseudo-Aléatoire (PRP)

## PRP

Soit  $F: \{0, 1\}^\lambda \times \{0, 1\}^{\text{blen}} \rightarrow \{0, 1\}^{\text{blen}}$  une fonction déterministe. On dit que  $F$  est une *Permutation Pseudo-Aléatoire (PRP)* sécurisée, a.k.a. *block cipher*, si  $f$  est inversible, i.e. si il existe une fonction efficace  $F^{-1}$  telle que  $\forall x, k:$

$$F^{-1}(k, F(k, x)) = x$$

et si, après avoir défini  $T.\text{values} := \{v \mid \exists x, T[x] = v\}$ , nous avons:

$\mathcal{L}_{\text{prp-real}}^F$

```
k ← $ \{0, 1\}^\lambda
LOOKUP(x ∈ \{0, 1\}^{\text{blen}}):
    return F(k, x)
```

$\mathcal{L}_{\text{prp-rand}}^F$

```
T := empty assoc. array
LOOKUP(x ∈ \{0, 1\}^{\text{blen}}):
    if T[x] undefined:
        T[x] ← $ \{0, 1\}^{\text{blen}} \setminus T.\text{values}
    return T[x]
```

## Quel lien entre PRP et PRF?

Attaque naturelle : appeler  $\text{LOOKUP}(x)$  sur des  $x$  aléatoires plusieurs fois (disons  $N$ ) jusqu'à ce qu'on **trouve une collision** ( $\text{LOOKUP}(x) = \text{LOOKUP}(x')$  pour un  $x' \neq x$ ). Si on ne trouve rien, on affirme PRP, sinon PRF.

Naïvement, on pense que cette attaque a un avantage  $\approx \frac{1}{N}$ , mais elle est bien plus efficace :  $\approx \frac{1}{\sqrt{N}}$ .



# Le paradoxe des anniversaires



**Paradoxe des anniversaires** = Quelle est la probabilité que deux personnes aient le même anniversaire dans une classe de 23 élèves ?

- A 7%
- B 20%
- C 50%

# Le paradoxe des anniversaires

**Paradoxe des anniversaires** = Quelle est la probabilité que deux personnes aient le même anniversaire dans une classe de 23 élèves ?



- A 7%
- B 20%
- C 50%

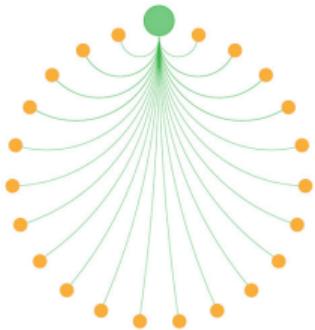
Si  $N$  = nombre d'éléments,  $n$  = nombre d'échantillons,  $p$  = probabilité de collision :

$$p(n) = 1 - \frac{N!}{(N-n)!} \frac{1}{N^n}$$

nombre de tirages pour une probabilité de collision de  $1/2 \approx \sqrt{N}$

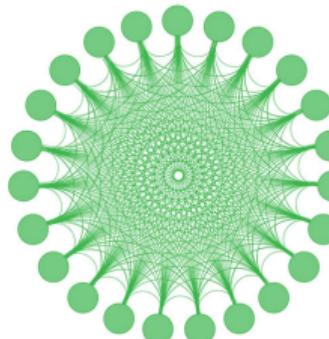
# Le paradoxe des anniversaires

## THE BIRTHDAY PARADOX



ONE-TO-MANY

The probability of someone sharing your specific birthday is a narrow search so the chances are low. With 23 people there's a 5.9% chance someone shares your birthday.



MANY-TO-MANY

When you are looking for *any* two people to share any birthday the network of possible connections is much richer. With 23 people there's a 50.7% chance two people share a birthday.

<https://oddathenaeum.com/the-birthday-paradox/>

# Le paradoxe des anniversaires

On a dit que  $2^{128}$  est ÉNORME. Est-il faisable de trouver une collision sur une PRF/fonction de hachage avec une sortie de 128 bits ?



- A Oui, avec un ordinateur portable
- B Oui, avec un cluster GPU/ASIC
- C Non

# Le paradoxe des anniversaires

On a dit que  $2^{128}$  est ÉNORME. Est-il faisable de trouver une collision sur une PRF/fonction de hachage avec une sortie de 128 bits ?



- A Oui, avec un ordinateur portable
- B Oui, avec un cluster GPU/ASIC ✓  $\sqrt{2^{128}} = 2^{128/2} = 2^{64}$ .  
⇒ Premier cours :  $2^{64}$  est faisable avec un cluster GPU/ASIC.
- C Non

# Le paradoxe des anniversaires

Mais de manière asymptotique, le paradoxe des anniversaire ne pose pas problème (il faut juste en pratique doubler la taille des clés) :

Théorème (Paradoxe des anniversaires asymptotique)

Nous avons

$$\approx$$

$\mathcal{L}_{\text{samp-L}}$
SAMP():
$r \leftarrow \{0, 1\}^\lambda$
return $r$

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
SAMP():
$r \leftarrow \{0, 1\}^\lambda \setminus R$
$R = R \cup \{r\}$
return $r$

.

$\mathcal{L}_{\text{samp-L}}$
SAMP():
$r \leftarrow \{0, 1\}^\lambda$
return $r$

$=$ 

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
bad := 0
SAMP():
$r \leftarrow \{0, 1\}^\lambda$
if $r \in R$ :
bad := 1
$r \leftarrow \{0, 1\}^\lambda \setminus R$
$R = R \cup \{r\}$
return $r$

$\approx$ 

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
bad := 0
SAMP():
$r \leftarrow \{0, 1\}^\lambda$
if $r \in R$ :
bad := 1
$r \leftarrow \{0, 1\}^\lambda \setminus R$
$R = R \cup \{r\}$
return $r$

$=$ 

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
SAMP():
$r \leftarrow \{0, 1\}^\lambda \setminus R$
$R = R \cup \{r\}$
return $r$

*Preuve.* Lemme des mauvais événements:  $\mathcal{A}$  est polynomial, donc

$$\Pr [ \text{bad} = 1 ] = \text{poly}(\lambda) \times \frac{\text{poly}(\lambda)}{2^\lambda} = \text{negl}(\lambda) \quad \square$$

Nombre d'appels à SAMP  
 $= |R|$

# Le paradoxe des anniversaires

## À retenir

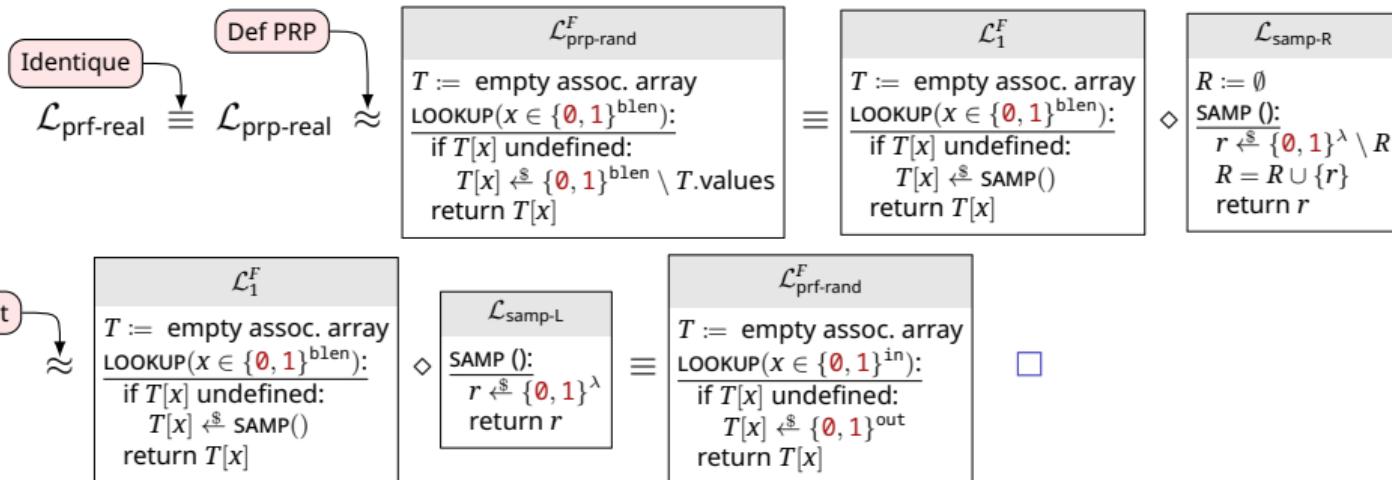
Le paradoxe des anniversaires ne nuit pas à la sécurité asymptotique ( $\sqrt{\text{negl}(\lambda)} = \text{negl}(\lambda)$ ), mais dans la pratique, il peut être nécessaire de **doubler la taille de la clé** pour prévenir cette attaque.

# Une PRP est une PRF

## Une PRP est une PRF

Soit  $F: \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  une PRP sécurisé (avec blen =  $\lambda$ ). Alors  $F$  est également une PRF sécurisée.

Preuve.



Comment fabriquer un chiffrement  
IND-CPA à partir de PRF/block-ciphers ?

# IND-CPA à partir d'une PRF

Basée sur l'idée précédente, première solution (pas très efficace) :

## Définition (pseudo-OTP basé sur une PRF)

Soit  $F$  une PRF sécurisée. On définit le schéma de chiffrement pseudo-OTP comme suit :  
 $\mathcal{K} = \{0, 1\}^\lambda$ ,  $\mathcal{M} = \{0, 1\}^{\text{out}}$ ,  $\mathcal{C} = \{0, 1\}^\lambda \times \{0, 1\}^{\text{out}}$ , et :

$\Sigma_{\text{prf-pseudo-OTP}}$
$\text{Gen}()$ :
$\frac{}{k \xleftarrow{\$} \{0, 1\}^\lambda}$
return $k$
$\text{Enc}(k, m)$ :
$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda}$
$x := F(k, r) \oplus m$
return $(r, x)$
$\text{Dec}(k, c)$ :
$\frac{}{m := F(k, r) \oplus c}$
return $m$

# IND-CPA à partir d'une PRF

Théorème (sécurité du pseudo-OTP basé sur PRF)

Le pseudo-OTP basé sur une PRF est IND-CPA sécurisé.



Exercice : essayez de démontrer sa sécurité (réponse à la prochaine diapositive)

$\mathcal{L}_{\text{cpa-L}}^{\Sigma}$

$$k \leftarrow \text{Gen}(1^\lambda)$$

$$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$$


---


$$\text{return } \text{Enc}_k(m_L)$$

$\mathcal{L}_1$

$$k \leftarrow \text{Gen}(1^\lambda)$$

$$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$$


---


$$r \xleftarrow{\$} \{0, 1\}^\lambda$$

$$x := F(k, r) \oplus m_L$$

$$\text{return } (r, x)$$

$\mathcal{L}_1$

$$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$$


---


$$r \xleftarrow{\$} \{0, 1\}^\lambda$$

$$x := \text{LOOKUP}(r) \oplus m_L$$

$$\text{return } (r, x)$$

$\mathcal{L}_{\text{prf-real}}^F$

$$k \xleftarrow{\$} \{0, 1\}^\lambda$$

$$\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}):$$


---


$$\text{return } F(k, x)$$

Def. Enc

$\equiv$

Externalize

$\equiv$

$\diamond$

$\mathcal{L}_{\text{cpa-L}}^{\Sigma}$
$k \leftarrow \text{Gen}(1^\lambda)$
$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$
return $\text{Enc}_k(m_L)$

$\mathcal{L}_1$
$k \leftarrow \text{Gen}(1^\lambda)$
$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$
$r \xleftarrow{\$} \{0, 1\}^\lambda$
$x := F(k, r) \oplus m_L$
return $(r, x)$

$\mathcal{L}_1$	$\mathcal{L}_{\text{prf-real}}^F$
$\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$	$k \xleftarrow{\$} \{0, 1\}^\lambda$
$r \xleftarrow{\$} \{0, 1\}^\lambda$	$\diamond$
$x := \text{LOOKUP}(r) \oplus m_L$	$\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}):$
return $(r, x)$	return $F(k, x)$

$\mathcal{L}_1$
-----------------

$\mathcal{L}_{\text{prf-rand}}^F$
-----------------------------------

Def. Enc

$$\equiv$$

$\mathcal{L}_1$

$$\frac{k \leftarrow \text{Gen}(1^\lambda) \quad \text{EAVESDROP}(m_L, m_R \in \mathcal{M}):}{\begin{aligned} r &\xleftarrow{\$} \{0, 1\}^\lambda \\ x &:= F(k, r) \oplus m_L \\ \text{return } (r, x) \end{aligned}}$$

Externalize

$$\equiv$$

$\mathcal{L}_1$

$$\frac{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):}{\begin{aligned} r &\xleftarrow{\$} \{0, 1\}^\lambda \\ x &:= \text{LOOKUP}(r) \oplus m_L \\ \text{return } (r, x) \end{aligned}}$$

$\mathcal{L}_{\text{pref-real}}^F$

$$\diamond$$

$$\frac{k \xleftarrow{\$} \{0, 1\}^\lambda \quad \text{LOOKUP}(x \in \{0, 1\}^{\text{in}}):}{\text{return } F(k, x)}$$

Def PRF

$$\approx$$

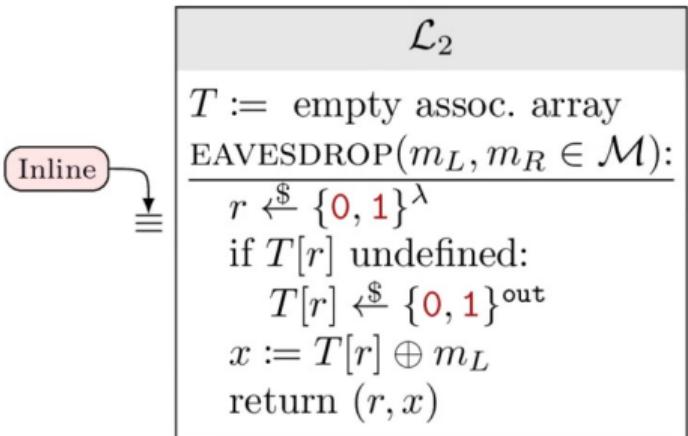
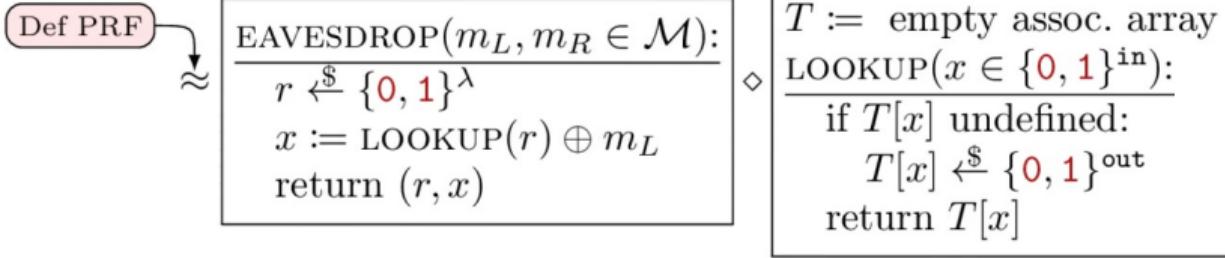
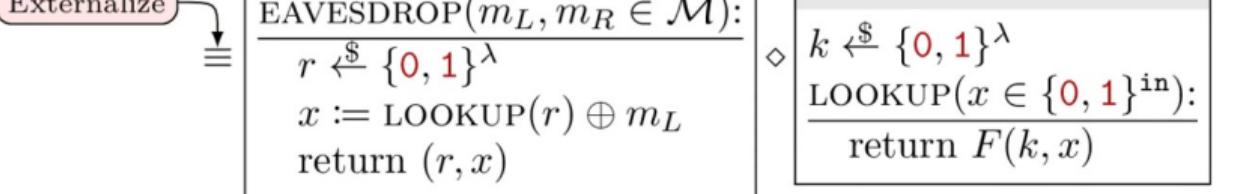
$\mathcal{L}_1$

$$\frac{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):}{\begin{aligned} r &\xleftarrow{\$} \{0, 1\}^\lambda \\ x &:= \text{LOOKUP}(r) \oplus m_L \\ \text{return } (r, x) \end{aligned}}$$

$\mathcal{L}_{\text{pref-rand}}^F$

$$\diamond$$

$$\begin{aligned} T &:= \text{empty assoc. array} \\ \text{LOOKUP}(x \in \{0, 1\}^{\text{in}}): & \\ \text{if } T[x] \text{ undefined:} & \\ T[x] &\xleftarrow{\$} \{0, 1\}^{\text{out}} \\ \text{return } T[x] \end{aligned}$$



return  $(r, x)$

$T[x] \leftarrow \{0, 1\}^{\text{out}}$   
return  $T[x]$

$\mathcal{L}_2$

$T :=$  empty assoc. array  
EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):  
 $r \xleftarrow{\$} \{0, 1\}^\lambda$   
if  $T[r]$  undefined:  
     $T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$   
     $x := T[r] \oplus m_L$   
    return  $(r, x)$

Inline



$\mathcal{L}_2$

$T :=$  empty assoc. array  
EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):  
 $r \xleftarrow{\$} \text{SAMP}()$   
if  $T[r]$  undefined:  
     $T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$   
     $x := T[r] \oplus m_L$   
    return  $(r, x)$

Externalize



$\mathcal{L}_{\text{samp-L}}$

SAMP ():  
 $r \xleftarrow{\$} \{0, 1\}^\lambda$   
return  $r$



$\mathcal{L}_2$ 

$T :=$  empty assoc. array  
 $\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$

$$\frac{}{r \xleftarrow{\$} \text{SAMP}()} \quad \diamond$$

if  $T[r]$  undefined:  
 $T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$   
 $x := T[r] \oplus m_L$   
return  $(r, x)$

Externalize

 $\mathcal{L}_{\text{samp-L}}$ 

$\text{SAMP}():$

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda}$$

return  $r$

 $\mathcal{L}_2$ 

$T :=$  empty assoc. array  
 $\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):$

$$\frac{}{r \xleftarrow{\$} \text{SAMP}()} \quad \diamond$$

if  $T[r]$  undefined:  
 $T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$   
 $x := T[r] \oplus m_L$   
return  $(r, x)$

Asymptotic birthday paradox

 $\mathcal{L}_{\text{samp-R}}$ 

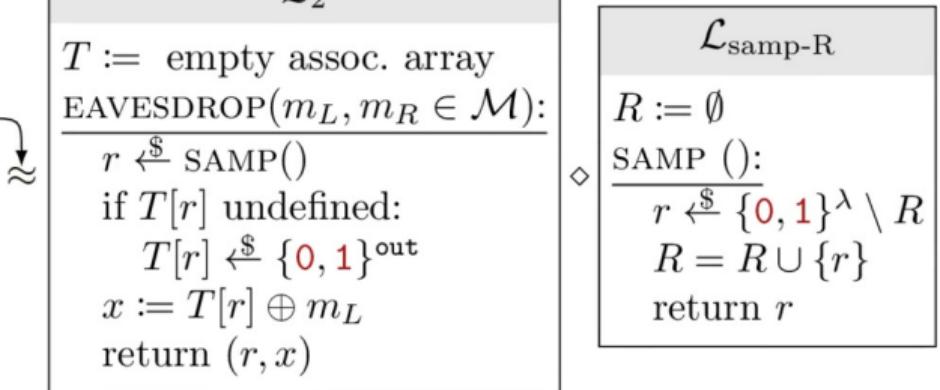
$R := \emptyset$   
 $\text{SAMP}():$

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

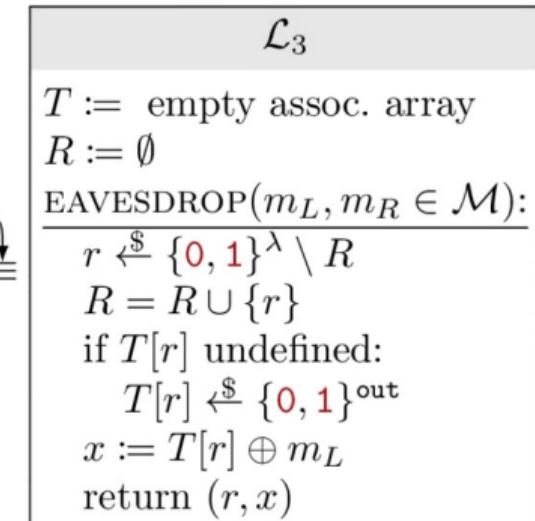
$R = R \cup \{r\}$   
return  $r$

 $\mathcal{L}_3$  $T :=$  empty assoc. array

Asymptotic birthday paradox



Inline



$\mathcal{L}_4$

$\mathcal{L}_3$ 

$T :=$  empty assoc. array

$R := \emptyset$

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R$

$R = R \cup \{r\}$

if  $T[r]$  undefined:

$T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$

$x := T[r] \oplus m_L$

return  $(r, x)$

Inline

 $\mathcal{L}_4$ 

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R$

$R = R \cup \{r\}$

$r' \xleftarrow{\$} \{0, 1\}^{\text{out}}$

$x := r' \oplus m_L$

return  $(r, x)$

$T[r]$  always undefined

 $\mathcal{L}_5$

$\Pi \vdash [r] \text{ EAVESDROP}$

$$T[r] \xleftarrow{\$} \{0, 1\}^{\text{out}}$$

$$x := T[r] \oplus m_L$$

$$\text{return } (r, x)$$

$\mathcal{L}_4$

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

$$R = R \cup \{r\}$$

$$r' \xleftarrow{\$} \{0, 1\}^{\text{out}}$$

$$x := r' \oplus m_L$$

$$\text{return } (r, x)$$

$T[r]$  always undefined

$\equiv$

$\mathcal{L}_5$

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

$$R = R \cup \{r\}$$

$$x \leftarrow \text{OTENC}(m_L)$$

$$\text{return } (r, x)$$

Externalize

$\Rightarrow$

$\mathcal{L}_{\text{otp-real}}$

OTENC( $m \in \{0, 1\}^\lambda$ ):

$$\frac{}{k \xleftarrow{\$} \{0, 1\}^\lambda}$$

$$\text{return } k \oplus m$$

$\mathcal{L}_5$

$T[r]$  always undefined

 $\equiv$ 

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

$$R = R \cup \{r\}$$

$$r' \xleftarrow{\$} \{0, 1\}^{\text{out}}$$

$$x := r' \oplus m_L$$

$$\text{return } (r, x)$$

Externalize

 $\equiv$ 

$\mathcal{L}_5$

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

$$R = R \cup \{r\}$$

$$x \leftarrow \text{OTENC}(m_L)$$

$$\text{return } (r, x)$$

$\mathcal{L}_{\text{otp-real}}$

OTENC( $m \in \{0, 1\}^\lambda$ ):

$$\frac{}{k \xleftarrow{\$} \{0, 1\}^\lambda}$$

$$\text{return } k \oplus m$$

OTP uniform ciphertext

 $\equiv$ 

$\mathcal{L}_5$

EAVESDROP( $m_L, m_R \in \mathcal{M}$ ):

$$\frac{}{r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R}$$

$$R = R \cup \{r\}$$

$$x \leftarrow \text{OTENC}(m_L)$$

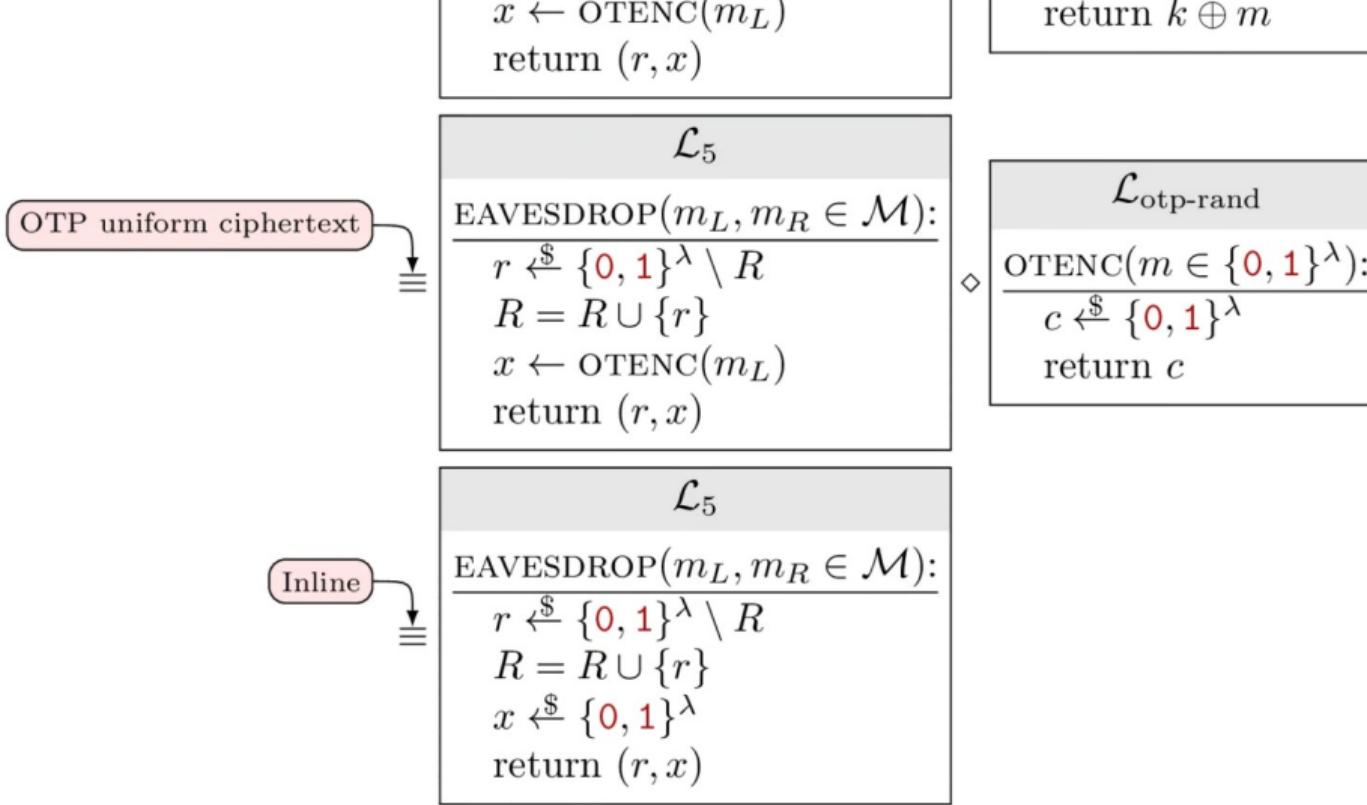
$$\text{return } (r, x)$$

$\mathcal{L}_{\text{otp-rand}}$

OTENC( $m \in \{0, 1\}^\lambda$ ):

$$\frac{}{c \xleftarrow{\$} \{0, 1\}^\lambda}$$

$$\text{return } c$$



Since this last library is symmetric with respect to  $m_L$  and  $m_R$ , we can do exactly the same computations starting from  $\mathcal{L}_{\text{cpa-R}}^\Sigma$  and we will find the exact same library (or, equivalently, do the operations backward with  $m_R$  instead of  $m_L$ ), hence  $\mathcal{L}_{\text{cpa-R}}^\Sigma \approx \mathcal{L}_{\text{cpa-L}}^\Sigma$ .

# Limites du pseudo-OTP basé sur PRF

C'est bien d'avoir un schéma IND-CPA sécurisé, mais **comment chiffrer un message  $m$  de longueur arbitraire ?**

- Première idée : **découper  $m$  en morceaux** de longueur  $\{0, 1\}^{\text{out}}$ , et les chiffrer séparément.  
⇒ Problème : rappelons que Enc est un couple  $(r, x)$ , donc pour  $l$  morceaux, surcoût de  $\lambda l$

**Trop inefficace !** 

# Limites du pseudo-OTP basé sur PRF

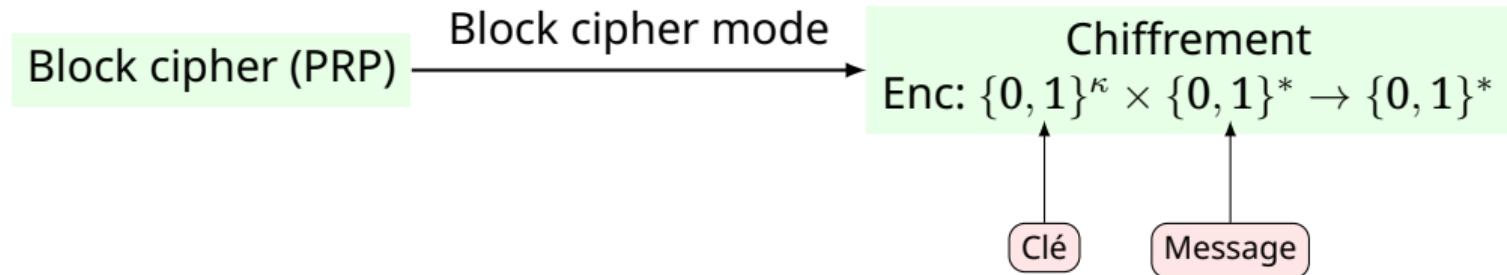
C'est bien d'avoir un schéma IND-CPA sécurisé, mais **comment chiffrer un message  $m$  de longueur arbitraire ?**

- Première idée : **découper  $m$  en morceaux** de longueur  $\{0, 1\}^{\text{out}}$ , et les chiffrer séparément.  
⇒ Problème : rappelons que Enc est un couple  $(r, x)$ , donc pour  $l$  morceaux, surcoût de  $\lambda l$
- Solution : utiliser des **modes de block ciphers** !

Trop inefficace ! 

# Modes de block cipher

Plusieurs modes d'opération (= variantes) :



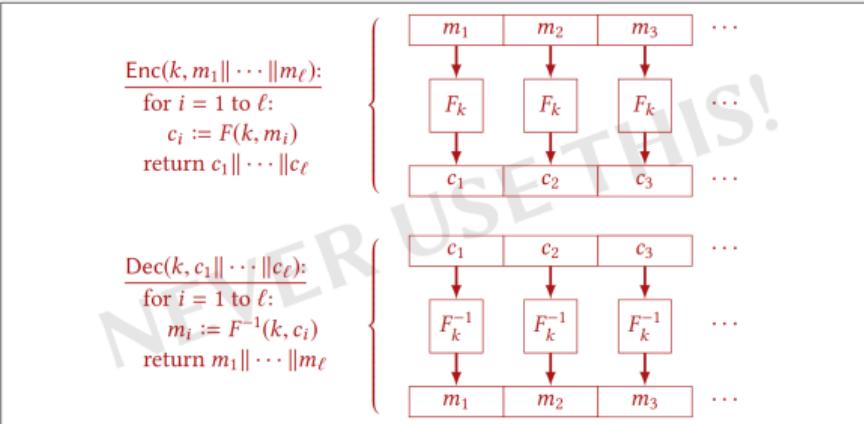
# Modes courants

Définition (mode ECB : À NE JAMAIS UTILISER)

Le mode Electronic Codebook (ECB) (INSECURE !) est défini comme suit :

```
Enc( $k, m_1 \parallel \dots \parallel m_\ell$ ):  
for  $i = 1$  to  $\ell$ :  
     $c_i := F(k, m_i)$   
return  $c_1 \parallel \dots \parallel c_\ell$ 
```

```
Dec( $k, c_1 \parallel \dots \parallel c_\ell$ ):  
for  $i = 1$  to  $\ell$ :  
     $m_i := F^{-1}(k, c_i)$   
return  $m_1 \parallel \dots \parallel m_\ell$ 
```



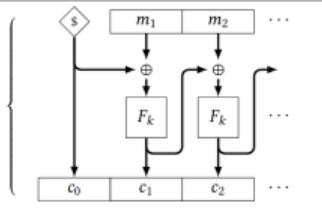
Ce mode est considéré pire que le mode déterministe. Trouvez une attaque réalisable en **un seul appel** à la fonction de chiffrement (exercice moodle “Attaque CBC une requête”).

# Modes courants

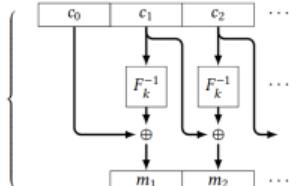
## Définition (mode CBC)

Le mode Cipher Block Chaining (CBC) est défini comme suit :

```
Enc( $k, m_1 \parallel \dots \parallel m_\ell$ ):  
     $c_0 \leftarrow \{0, 1\}^{b \cdot \ell m_r}$   
    for  $i = 1$  to  $\ell$ :  
         $c_i := F(k, m_i \oplus c_{i-1})$   
    return  $c_0 \parallel c_1 \parallel \dots \parallel c_\ell$ 
```



```
Dec( $k, c_0 \parallel \dots \parallel c_\ell$ ):  
    for  $i = 1$  to  $\ell$ :  
         $m_i := F^{-1}(k, c_i) \oplus c_{i-1}$   
    return  $m_1 \parallel \dots \parallel m_\ell$ 
```



$c_0$  est appelé le vecteur d'initialisation (IV). Pourquoi ne peut-on pas le fixer à une valeur constante ?



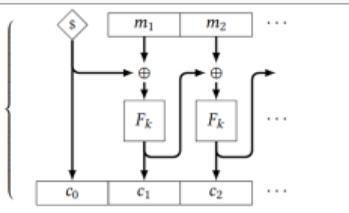
- A Il agit comme une clé OTP sur le message, donc le masque
- B Utilisé pour obtenir un chiffrement non déterministe

# Modes courants

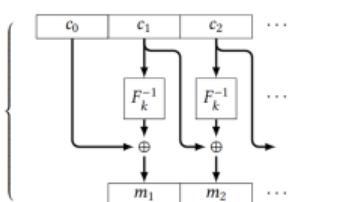
## Définition (mode CBC)

Le mode Cipher Block Chaining (CBC) est défini comme suit :

```
Enc( $k, m_1 \parallel \dots \parallel m_\ell$ ):  
     $c_0 \leftarrow \{0, 1\}^{b\text{len}}$ ;  
    for  $i = 1$  to  $\ell$ :  
         $c_i := F_k(m_i \oplus c_{i-1})$   
    return  $c_0 \parallel c_1 \parallel \dots \parallel c_\ell$ 
```



```
Dec( $k, c_0 \parallel \dots \parallel c_\ell$ ):  
    for  $i = 1$  to  $\ell$ :  
         $m_i := F_k^{-1}(k, c_i) \oplus c_{i-1}$   
    return  $m_1 \parallel \dots \parallel m_\ell$ 
```



$c_0$  est appelé le vecteur d'initialisation (IV). Pourquoi ne peut-on pas le fixer à une valeur constante ?



- A Il agit comme une clé OTP sur le message, donc le masque **X** L'IV est public, donc ne peut pas servir de clé OTP !
- B Utilisé pour obtenir un chiffrement non déterministe **✓**

# Modes courants

## Définition (mode CTR)

Le mode compteur (CTR) est défini comme suit :

$\text{Enc}(k, m_1 \parallel \dots \parallel m_\ell)$ :

$$r \leftarrow \{\mathbf{0}, \mathbf{1}\}^{blend}$$

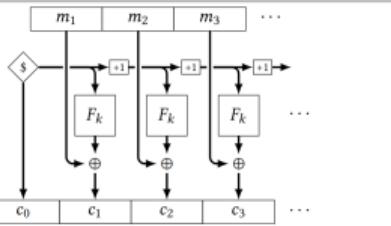
$c_0 := r$

for  $i = 1$  to  $\ell$ :

$$c_i := F(k, r) \oplus m_i$$

$$r := r + 1 \% 2^{blend}$$

return  $c_0 \parallel \dots \parallel c_\ell$



Essayez de retrouver l'algorithme de déchiffrement. Avez-vous besoin de calculer  $F^{-1}$  ?



A Oui

B Non

# Modes courants

## Définition (mode CTR)

Le mode compteur (CTR) est défini comme suit :

$\text{Enc}(k, m_1 \parallel \dots \parallel m_\ell)$ :

$r \leftarrow \{0, 1\}^{blen}$

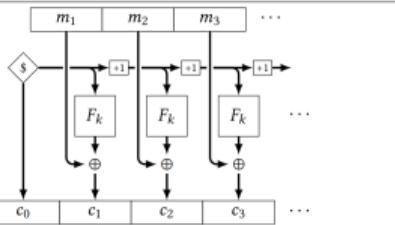
$c_0 := r$

for  $i = 1$  to  $\ell$ :

$c_i := F(k, r) \oplus m_i$

$r := r + 1 \% 2^{blen}$

return  $c_0 \parallel \dots \parallel c_\ell$



Essayez de retrouver l'algorithme de déchiffrement. Avez-vous besoin de calculer  $F^{-1}$  ?



A Oui

B Non Pas besoin d'avoir une PRP, une PRF suffit (mais en pratique, les PRF les plus efficaces sont généralement des PRP)

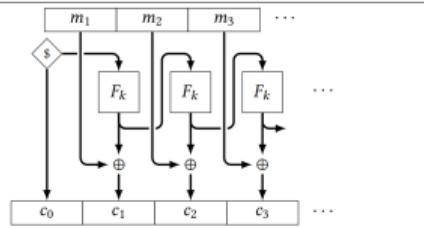
# Modes courants

## Définition (mode OFB)

Le mode “output feedback” (OFB) est défini comme suit :

$\text{Enc}(k, m_1 \parallel \dots \parallel m_\ell)$ :

$r \leftarrow \{0, 1\}^{b\ell n}$   
 $c_0 := r$   
for  $i = 1$  to  $\ell$ :  
 $r := F(k, r)$   
 $c_i := r \oplus m_i$



Essayez de retrouver l’algorithme de déchiffrement. Avez-vous besoin de calculer  $F^{-1}$  ?



- A Oui
- B Non

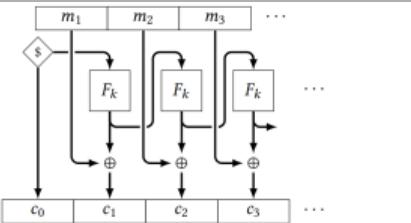
# Modes courants

## Définition (mode OFB)

Le mode “output feedback” (OFB) est défini comme suit :

$\text{Enc}(k, m_1 \parallel \dots \parallel m_\ell)$ :

```
 $r \leftarrow \{0, 1\}^{blen}$ 
 $c_0 := r$ 
for  $i = 1$  to  $\ell$ :
   $r := F(k, r)$ 
   $c_i := r \oplus m_i$ 
return  $c_0 \parallel \dots \parallel c_\ell$ 
```



Essayez de retrouver l'algorithme de déchiffrement. Avez-vous besoin de calculer  $F^{-1}$  ?



A Oui X

B Non ✓ Pas besoin d'avoir une PRP, une PRF suffit

# Comparaison des modes

	ECB	CBC	CTR	OFB
IND-CPA	✗ !!!	✓	✓	✓
Parallélisable	✗	✓	✗	✗
Pré-calculable	✗	✓	✓	✓
Peut éviter le bourrage	✗	✓	✓	✓
Plus sûr sans cycle de permutation	✗	✓	✗	✗
Légèrement plus sûr en cas de réutilisation de l'IV (ex. : mauvaise implémentation)		✓	✗	✗

**Le grand gagnant est le mode CTR !** (mais attendez de voir les modes qui chiffrent & authentifient comme GCM)

# Comparaison des modes



Un ami propose de chiffrer votre disque dur avec AES en mode OFB. Est-ce une bonne idée ? Pourquoi ?

- A Oui
- B Non

# Comparaison des modes

Un ami propose de chiffrer votre disque dur avec AES en mode OFB. Est-ce une bonne idée ? Pourquoi ?



- A Oui X
- B Non ✓ Mauvaise idée : le mode OFB n'est pas parallélisable.  
Il faut donc déchiffrer tout le disque pour accéder au dernier octet !

# Modes vulnérables aux attaques par anniversaire

**Tous** les modes sont vulnérables aux attaques par anniversaire (cf TD), donc assurez-vous de chiffrer moins de  $2^{\text{blen}/2}$  blocs (c'est-à-dire : utilisez une grande valeur de blen, par exemple n'utilisez pas 3DES ! (64 bits)).

Aujourd'hui, le chiffrement le plus utilisé est

## Advanced Encryption Standard (AES)

avec une taille de bloc de 128 bits (longueur de clé : 128, 192 ou 256 bits). Voir aussi :

- Rijndael (généralisation de AES) : taille de bloc de 128, 192 ou 256 bits,
- Serpent (2e finaliste du processus de sélection de l'AES)
- Twofish (blen = 128) et Blowfish (attention : blen = 64 !)
- n'utilisez jamais DES = cassé (ancien standard), temporairement remplacé par 3DES

# IND-CPA pour des messages en clair de longueur variable

Pouvez-vous trouver une attaque IND-CPA générique contre ces cipher modes (par exemple CTR, en supposant  $\text{len} = \lambda$  pour simplifier) ?

A Non



B Oui, avec

```
 $\mathcal{A}$ 
 $c := \text{EAVESDROP}(\mathbf{0}^\lambda, \mathbf{0}^\lambda)$ 
 $d := \text{EAVESDROP}(\mathbf{0}^\lambda, \mathbf{1}^\lambda)$ 
return  $c \stackrel{?}{=} d$ 
```

C Oui, avec

```
 $\mathcal{A}$ 
 $c := \text{EAVESDROP}(\mathbf{0}^\lambda, \mathbf{0}^{2\lambda})$ 
return  $|c| \stackrel{?}{=} 2\lambda$ 
```

# IND-CPA pour des messages en clair de longueur variable

Pouvez-vous trouver une attaque IND-CPA générique contre ces cipher modes (par exemple CTR, en supposant  $\text{blen} = \lambda$  pour simplifier) ?

A Non



B Oui, avec

$\mathcal{A}$

```
c := EAVESDROP( $0^\lambda, 0^\lambda$ )
d := EAVESDROP( $0^\lambda, 1^\lambda$ )
return  $c \stackrel{?}{=} d$ 
```



C Oui, avec

$\mathcal{A}$

```
c := EAVESDROP( $0^\lambda, 0^{2\lambda}$ )
return  $|c| \stackrel{?}{=} 2\lambda$ 
```



La longueur du message chiffré

est égale à  $\lambda + |m| \Rightarrow$  révèle la longueur du message !

# IND-CPA pour des messages en clair de longueur variable

## IND-CPA pour des messages en clair de longueur variable

Quand les messages peuvent avoir des longueurs variables, il faut mettre à jour la définition de la sécurité :

$$\begin{array}{l} \mathcal{L}_{\text{cpa-L}}^{\Sigma} \\ k \leftarrow \text{Gen}(1^{\lambda}) \\ \text{EAVESDROP}(m_L, m_R \in \mathcal{M}): \\ \quad \text{if } |m_L| \neq |m_R| \text{ return } \text{err} \\ \quad \text{return Enc}_k(m_L) \end{array}$$

≈

$$\begin{array}{l} \mathcal{L}_{\text{cpa-R}}^{\Sigma} \\ k \leftarrow \text{Gen}(1^{\lambda}) \\ \text{EAVESDROP}(m_L, m_R \in \mathcal{M}): \\ \quad \text{if } |m_L| \neq |m_R| \text{ return } \text{err} \\ \quad \text{return Enc}_k(m_R) \end{array}$$

# IND-CPA pour des messages en clair de longueur variable

## Est-ce qu'il y a un problème à divulguer la longueur ?

**Parfois !** Par exemple :

- Google Maps envoie des tuiles, chaque tuile ayant une taille différente (malgré une taille en pixels identique) à cause de la compression ⇒ il est possible de savoir quelle tuile est affichée rien qu'en regardant le trafic.
- Le débit variable (VBR) en vidéo montre différentes tailles de trames selon le moment. Il est possible de savoir quel film vous regardez sur Netflix/YouTube grâce à cela, et même d'identifier le locuteur, la langue ou le mot prononcé dans des programmes de chat vocal !

# Padding

# Remplissage (padding)

Que faire si  $|m|$  n'est pas un multiple de la taille d'un bloc ?

- Mode CTR : simple, il suffit de tronquer le texte chiffré (comme un OTP classique)
- Mode CBC : il faut ajouter un **padding** (ajouter des données jusqu'à atteindre la taille d'un bloc) (possible aussi de faire du "ciphertext stealing" dans ce cas précis)

# Padding / Remplissage

Plusieurs façons de “padder”  $m$  en  $m'$  :

- ajouter des zéros : ne fonctionne pas ! En dépadant, comment savoir combien de zéros retirer ?
- norme ANSI X.923 : ajouter des **0** jusqu'au dernier octet qui contient le nombre d'octets paddés
- norme PKCS#7 : si  $b$  octets de padding sont nécessaires, ajouter l'octet  $b$  répété  $b$  fois
- norme ISO/IEC 7816-4 : ajouter **10...0**

Le choix précis a **peu d'importance**, ce n'est pas vraiment une caractéristique de sécurité (au moins en considérant des adversaires passifs, voir plus tard)

# Padding

Considérez la norme ISO/IEC 7816-4 (ajout de  $10 \dots 0$ ) : si vous paddez un message  $m$  de taille  $k\text{blen}$  en  $m'$ , quelle est la taille de  $m'$  ?



- A  $k\text{blen}$
- B  $k\text{blen} + 1$
- C  $(k + 1)\text{blen}$

# Padding

Considérez la norme ISO/IEC 7816-4 (ajout de  $10 \dots 0$ ) : si vous paddez un message  $m$  de taille  $k\text{blen}$  en  $m'$ , quelle est la taille de  $m'$  ?

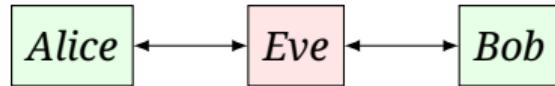


- A  $k\text{blen}$
- B  $k\text{blen} + 1$
- C  $(k + 1)\text{blen}$  (comme tous les paddings, cela augmente la taille du message)

# Attack du padding oracle & sécurité CCA

# Attaque par oracle de padding

Avant : adversaire passif (plus ou moins irréaliste). Maintenant, on considère des adversaires **actifs** :



Que se passe-t-il si Bob renvoie une erreur lorsque le padding est incorrect ?  
⇒ Eve peut récupérer complètement le message chiffré !

# Attaque oracle sur le padding (illustrée au tableau)

Modèle d'attaque : mode CTR, padding ANSI X.923,  $\mathcal{A}$  a accès à

$$\boxed{\begin{aligned} k &\leftarrow \text{Gen}(1^\lambda) \\ \text{PADDINGORACLE}(c) : \\ m &:= \text{Dec}(k, c) \\ \text{return } &\text{VALIDPAD}(m) \end{aligned}}.$$

(donc  $\text{VALIDPAD}(m)$  vérifie si  $m$  se termine par un octet  $b$  précédé de  $b - 1$  octets remplis de  $\mathbf{0}$ ). Supposons qu'on ait accès à  $c_0 \leftarrow \text{Enc}_k(m_0)$  (où  $m_0$  est déjà paddé), l'objectif est de retrouver  $m_0$ .

- étape 0 : remarquer qu'en mode CTR,  $\text{Enc}_k(m) \oplus (0^{\text{blen}}, x) = \text{Enc}_k(m \oplus x)$ . On peut donc modifier le message en modifiant le texte chiffré (donc plus loin je dirai « appliquer une opération sur  $m$  » même si en fait c'est sur  $\text{Enc}_k(m)$ ).

# Attaque oracle sur le padding (illustrée au tableau)

Modèle d'attaque : mode CTR, padding ANSI X.923,  $\mathcal{A}$  a accès à

$$\begin{aligned} k &\leftarrow \text{Gen}(1^\lambda) \\ \text{PADDINGORACLE}(c) : \\ m &:= \text{Dec}(k, c) \\ \text{return } &\text{VALIDPAD}(m) \end{aligned}.$$

(donc  $\text{VALIDPAD}(m)$  vérifie si  $m$  se termine par un octet  $b$  précédé de  $b - 1$  octets remplis de  $0$ ). Supposons qu'on ait accès à  $c_0 \leftarrow \text{Enc}_k(m_0)$  (où  $m_0$  est déjà paddé), l'objectif est de retrouver  $m_0$ .

- étape 0 : remarquer qu'en mode CTR,  $\text{Enc}_k(m) \oplus (0^{\text{blen}}, x) = \text{Enc}_k(m \oplus x)$ . On peut donc modifier le message en modifiant le texte chiffré (donc plus loin je dirai « appliquer une opération sur  $m$  » même si en fait c'est sur  $\text{Enc}_k(m)$ ).
- première étape : déterminer la longueur du message (changer un bit du message ne déclenche PAS d'erreur, changer un bit du padding OUI)

# Attaque oracle sur le padding (illustrée au tableau)

Modèle d'attaque : mode CTR, padding ANSI X.923,  $\mathcal{A}$  a accès à

$$\boxed{\begin{aligned} k &\leftarrow \text{Gen}(1^\lambda) \\ \text{PADDINGORACLE}(c) : \\ m &:= \text{Dec}(k, c) \\ \text{return } &\text{VALIDPAD}(m) \end{aligned}}.$$

(donc  $\text{VALIDPAD}(m)$  vérifie si  $m$  se termine par un octet  $b$  précédé de  $b - 1$  octets remplis de  $0$ ). Supposons qu'on ait accès à  $c_0 \leftarrow \text{Enc}_k(m_0)$  (où  $m_0$  est déjà paddé), l'objectif est de retrouver  $m_0$ .

- première étape : déterminer la longueur du message (changer un bit du message ne déclenche PAS d'erreur, changer un bit du padding OUI)
- deuxième étape : une fois que la longueur du padding  $p$  est connue, on sait que  $m_0$  ressemble à  $m_{\text{unpad}} 0^{8p} \text{Byte}(p)$ . On XOR l'octet final de  $c_0$  avec  $\text{Byte}(p) \oplus \text{Byte}(p+1)$ . Grâce à l'étape 0, on a maintenant un chiffrement de  $m_{\text{unpad}} 0^{8p} \text{Byte}(p+1)$ . Comme  $m_{\text{unpad}}$  ne finit (a priori) pas par un octet zéro, PADDINGORACLE retournera une erreur. Ensuite on itère sur  $x \in \{0, \dots, 255\}$  en XORrant le dernier octet du (chiffrement de)  $m_{\text{unpad}}$  avec  $x$ , puis on appelle PADDINGORACLE. À un moment, il n'y aura pas d'erreur : le dernier octet de  $m_{\text{unpad}}$  est égal à  $x$  !

# Attaque oracle sur le padding (illustrée au tableau)

Modèle d'attaque : mode CTR, padding ANSI X.923,  $\mathcal{A}$  a accès à

$$\boxed{\begin{aligned} k &\leftarrow \text{Gen}(1^\lambda) \\ \text{PADDINGORACLE}(c) &: \\ m &:= \text{Dec}(k, c) \\ \text{return } &\text{VALIDPAD}(m) \end{aligned}}.$$

(donc  $\text{VALIDPAD}(m)$  vérifie si  $m$  se termine par un octet  $b$  précédé de  $b - 1$  octets remplis de  $0$ ). Supposons qu'on ait accès à  $c_0 \leftarrow \text{Enc}_k(m_0)$  (où  $m_0$  est déjà paddé), l'objectif est de retrouver  $m_0$ .

- deuxième étape : une fois que la longueur du padding  $p$  est connue, on sait que  $m_0$  ressemble à  $m_{\text{unpad}} 0^{8p} \text{Byte}(p)$ . On XOR l'octet final de  $c_0$  avec  $\text{Byte}(p) \oplus \text{Byte}(p+1)$ . Grâce à l'étape 0, on a maintenant un chiffrement de  $m_{\text{unpad}} 0^{8p} \text{Byte}(p+1)$ . Comme  $m_{\text{unpad}}$  ne finit (a priori) pas par un octet zéro, PADDINGORACLE retournera une erreur. Ensuite on itère sur  $x \in \{0, \dots, 255\}$  en XORrant le dernier octet du (chiffrement de)  $m_{\text{unpad}}$  avec  $x$ , puis on appelle PADDINGORACLE. À un moment, il n'y aura pas d'erreur : le dernier octet de  $m_{\text{unpad}}$  est égal à  $x$  !
- dernière étape : on recommence à la deuxième étape jusqu'à retrouver tous les octets de  $m$ .

# Limite de la sécurité IND-CPA

Problème fondamental : ce n'est pas le padding, mais le serveur qui se comporte **différemment selon la valeur déchiffrée.**

En pratique, c'est **extrêmement fréquent** et difficile à éviter (par ex. ça peut prendre un peu plus de temps pour déchiffrer certains messages, ou effectuer des opérations différentes selon la valeur déchiffrée...)

⇒ **Il nous faut une définition de sécurité plus robuste** : autoriser l'attaquant à déchiffrer des messages arbitraires = IND-CCA !

# Sécurité IND-CCA

# IND-CCA

## IND-CCA

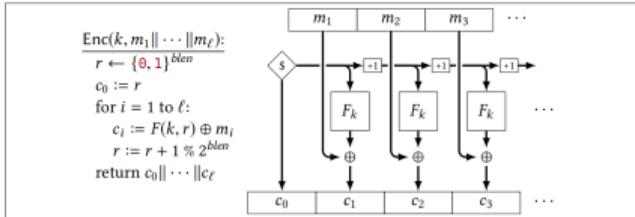
Soit  $\Sigma$  un chiffrement. On dit que  $\Sigma$  est sécurisé de manière indistiguable contre des **attaques par chiffrés choisis** (IND-CCA) si :

$\mathcal{L}_{\text{cca-L}}^{\Sigma}$
$k \leftarrow \text{Gen}(1^{\lambda})$
$\mathcal{S} := \emptyset$
<u>EAVESDROP(<math>m_L, m_R \in \mathcal{M}</math>):</u>
if $ m_L  \neq  m_R $ return <b>err</b>
$c := \text{Enc}_k(m_L)$
$\mathcal{S} := \mathcal{S} \cup \{c\}$
return $c$
<u>DECRYPT(<math>c \in \mathcal{C}</math>):</u>
if $c \in \mathcal{S}$ return <b>err</b>
return $\text{Dec}(k, c)$

≈

$\mathcal{L}_{\text{cca-R}}^{\Sigma}$
$k \leftarrow \text{Gen}(1^{\lambda})$
$\mathcal{S} := \emptyset$
<u>EAVESDROP(<math>m_L, m_R \in \mathcal{M}</math>):</u>
if $ m_L  \neq  m_R $ return <b>err</b>
$c := \text{Enc}_k(m_R)$
$\mathcal{S} := \mathcal{S} \cup \{c\}$
return $c$
<u>DECRYPT(<math>c \in \mathcal{C}</math>):</u>
if $c \in \mathcal{S}$ return <b>err</b>
return $\text{Dec}(k, c)$

# Malléabilité



Pouvez-vous trouver une attaque CCA contre, par exemple, le mode CTR ?

A Non

$\mathcal{A}$

$$(c_0, c_1) \leftarrow \text{EAVESDROP}(0^{blen}, 1^{blen})$$
$$m \leftarrow \text{DECRYPT}((c_0, c_1 \oplus (10 \dots 0)))$$

**return**  $m = ?$   $10 \dots 0$

B

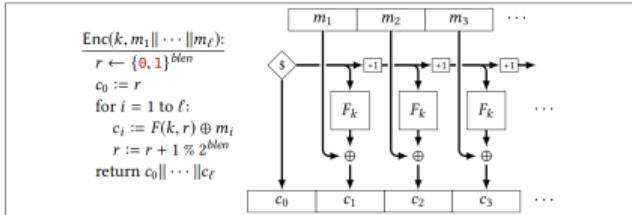
$\mathcal{A}$

$$(c_0, c_1) \leftarrow \text{EAVESDROP}(0^{blen}, 1^{blen})$$
$$m \leftarrow \text{DECRYPT}((c_0, c_1))$$

**return**  $m = 0^{blen}$

C Oui, avec

# Malléabilité



Pouvez-vous trouver une attaque CCA contre, par exemple, le mode CTR ?

A Non

B

```
-----  
A  
(c₀, c₁) ← EAVESDROP(0^{blen}, 1^{blen})  
m ← DECRYPT((c₀, c₁ ⊕ (10...0)))  
return m = ? 10...0
```

C Oui, avec

```
-----  
A  
(c₀, c₁) ← EAVESDROP(0^{blen}, 1^{blen})  
m ← DECRYPT((c₀, c₁))  
return m = 0^{blen}
```

# Malléabilité

Raison fondamentale : CTR est **malléable**, c'est-à-dire qu'on peut obtenir  $\text{Enc}_k(x') = (c_0, x' \oplus F_k(c_0))$  à partir de  $\text{Enc}_k(x) = (c_0, x \oplus F_k(c_0))$  (il suffit d'ajouter  $x \oplus x'$  au deuxième élément du couple).

Problème en pratique : par exemple, on peut transformer un « Oui » en « Non ».

Comment éviter cela ? **Authentification !** (cours ultérieur)

# Conclusion

- Le OTP est statistiquement sécurisé s'il est **utilisé une seule fois**
- Une première notion de sécurité contre un adversaire passif est **IND-CPA**
- PRF  $\Rightarrow$  chiffrement IND-CPA
- **Paradoxe des anniversaires** = peut nécessiter de doubler la taille de la clé
- **Modes block-cipher** = chiffrer efficacement des messages de longueur arbitraire (padding parfois nécessaire)
- Le mode CTR a de bonnes propriétés (mais attendez de voir GCM)
- **AES** = PRP standardisée (donc PRF) utilisé dans les modes de chiffrement par bloc
- **Chiffrement malléable**  $\Rightarrow$  attaques contre des adversaires actifs (ex. padding oracle/attaques par timing)
- Spoiler : L'authentification va nous aider !