

Crypto Engineering 2025–2026

Symmetric encryption

Léo COLISSON PALAIS
Master CySec UGA

leo.colisson-palais@univ-grenoble-alpes.fr
<https://leo.colisson.me/teaching/>

Reminder symmetric encryption & IND-CPA security







m



m



c



m



m



c



m

Symmetric encryption

Definition (Symmetric encryption scheme)

Let $\mathcal{K}, \mathcal{M}, \mathcal{C}$ be the set of, respectively, keys, messages and ciphertexts. An encryption scheme is a tuple $(\text{Gen}, \text{Enc}, \text{Dec})$ of polynomial algorithm:

- Key-generation $k \leftarrow \text{Gen}(1^\lambda)$
- Encryption $c \leftarrow \text{Enc}_k(m)$ Message $m \in \mathcal{M}$, sometimes written $\text{Enc}(k, m)$.
- Decryption $m \leftarrow \text{Dec}_k(c)$ Ciphertext $c \in \mathcal{C}$

Key $k \in \mathcal{K}$

Security parameter $\lambda \in \mathbb{N}$ in unary form:
Gen runs in poly time in the size of its input

that must be correct, i.e. such that for any $m \in \mathcal{M}$:

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

One-Time Pad

Definition (One-Time Pad, OTP)

The One-Time Pad is the crypto-system defined as $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0, 1\}^\lambda$ and (Gen, Enc, Dec) as:

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

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



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 ?
Can we reuse the key ?

Security of OTP

First (weak) security definition:

- We give NOTHING
- We change the key at any new encryption

More formally:

Definition (One-time secrecy)

An encryption scheme $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$ with key-space \mathcal{K} , message-space \mathcal{M} and cipher-text space \mathcal{C} is *one-time secure* if:

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

Security of OTP

Theorem

OTP is one-time secure

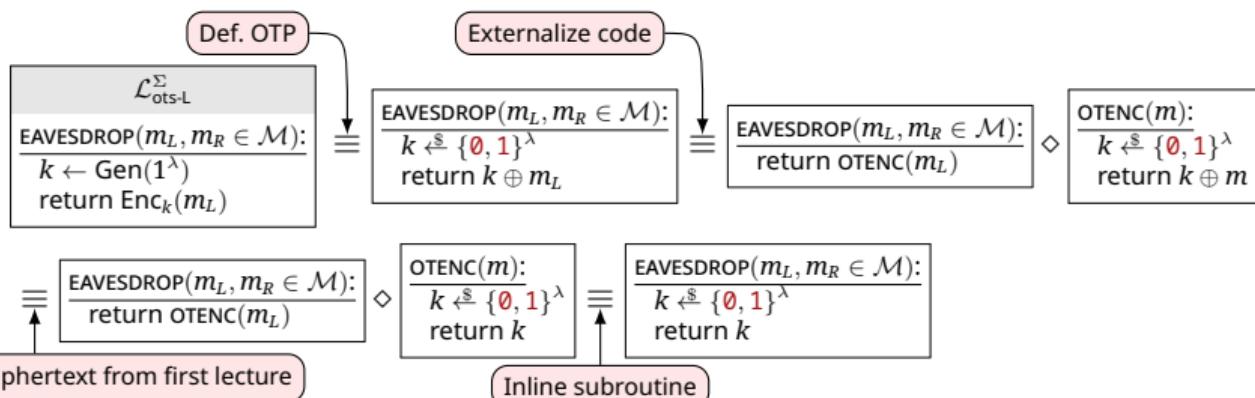
Proof (Exercice. See also the caseine exercice “OTP is one-time secure”)

Security of OTP

Theorem

OTP is one-time secure

Proof



We realize that the last library does not depend on m_R or m_L at all. So we can apply all operations backward, except that we replace m_L with m_R to recover $\mathcal{L}_{\text{ots-R}}^{\Sigma} \equiv \mathcal{L}_{\text{ots-L}}^{\Sigma}$. □

Security OTP

Problem: Here, a **new key** k is re-sampled on every new encryption... **Highly impractical!** We would prefer to **re-use** the same key:

Definition (IND-CPA)

An encryption scheme $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$ has indistinguishable security against *chosen-plaintext attacks* (IND-CPA security) if:

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

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

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

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

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

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

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

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

\approx

Security of OTP



Do you think that OTP is CPA secure? If yes, sketch a proof, if not, sketch an adversary and compute its advantage (see caseine first quiz question 9).

- A Yes
- B No

Security of OTP

Do you think that OTP is CPA secure? If yes, sketch a proof, if not, sketch an adversary and compute its advantage (see caseine first quiz question 9).

A Yes **X**

B No **✓** Exploit the fact that it is **deterministic encryption**:

Define

\mathcal{A}

```
x ← EAVESDROP( $0^\lambda, 0^\lambda$ )
y ← EAVESDROP( $0^\lambda, 1^\lambda$ )
return x = y
```

. Then, after inlining, we have

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

```
k ← Gen( $1^\lambda$ )
x ←  $0^\lambda \oplus k$ 
y ←  $0^\lambda \oplus k$ 
return x = y
```

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

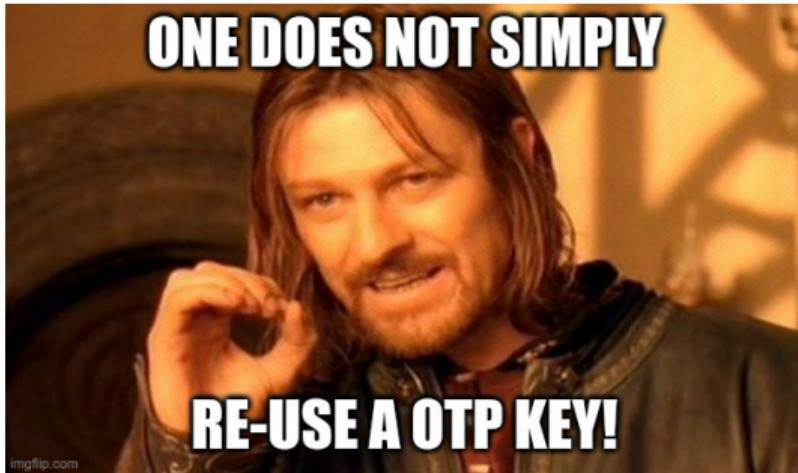
$\mathcal{A} \diamond \mathcal{L}_{\text{cpa-R}}^\Sigma$

```
k ← Gen( $1^\lambda$ )
x ←  $0^\lambda \oplus k$ 
y ←  $1^\lambda \oplus k$ 
return x = y
```

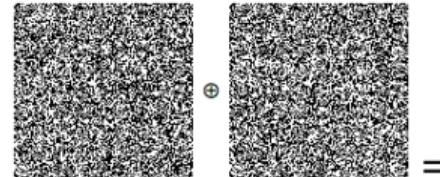
i.e. $\Pr [\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}^\Sigma = 1] = 0$. $\text{Adv} = 1 - 0 = 1 \neq \text{negl}(\lambda)$



Security of OTP



Never reuse a OTP key!!! This can lead to real attack:

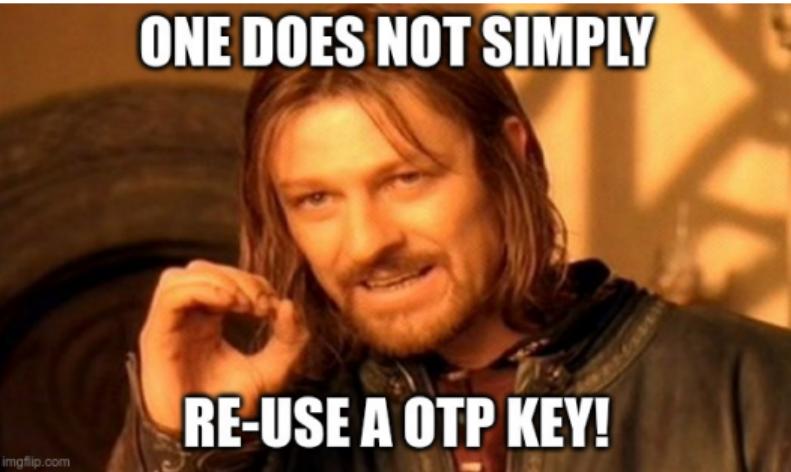


More: <https://crypto.stackexchange.com/questions/59>, <https://incoherency.co.uk/blog/stories/otp-key-reuse.html>

Security of OTP

ONE DOES NOT SIMPLY

RE-USE A OTP KEY!



imgflip.com

Never reuse a OTP key!!! This can lead to real attack:



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

How to build IND-CPA secure schemes



Encryption from simpler primitives

How to build an encryption:

- Approach 1: start from scratch. Less guarantees it will be secure.
- Approach 2 try to build encryption from simpler, more tested, primitives.

Encryption from simpler primitives

How to build an encryption:

- Approach 1: start from scratch. Less guarantees it will be secure.
- Approach 2 try to build encryption from simpler, more tested, primitives.

Our approach

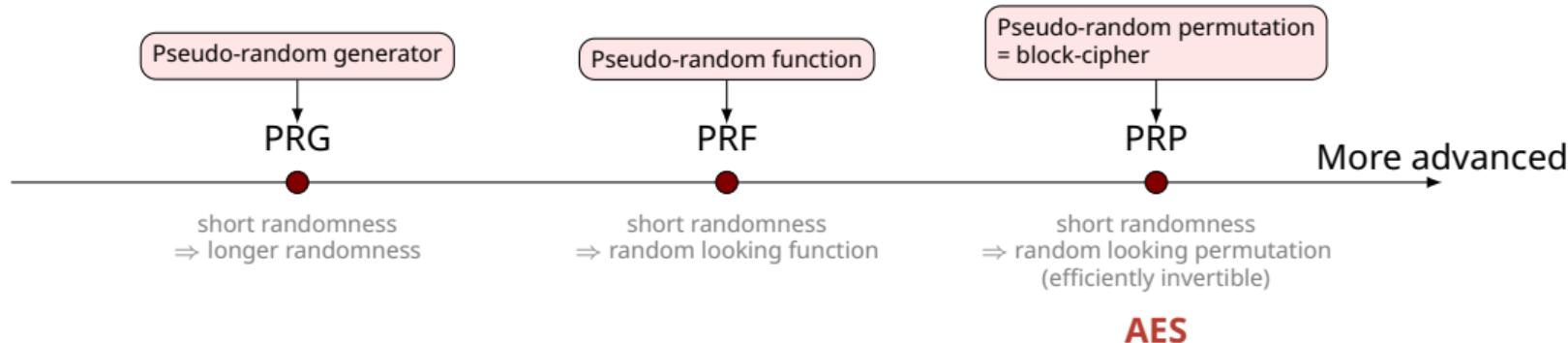
Encryption from simpler primitives

How to build an encryption:

- Approach 1: start from scratch. Less guarantees it will be secure.
- Approach 2 try to build encryption from simpler, more tested, primitives.

Our approach

But which **more fundamental primitive can we use?**



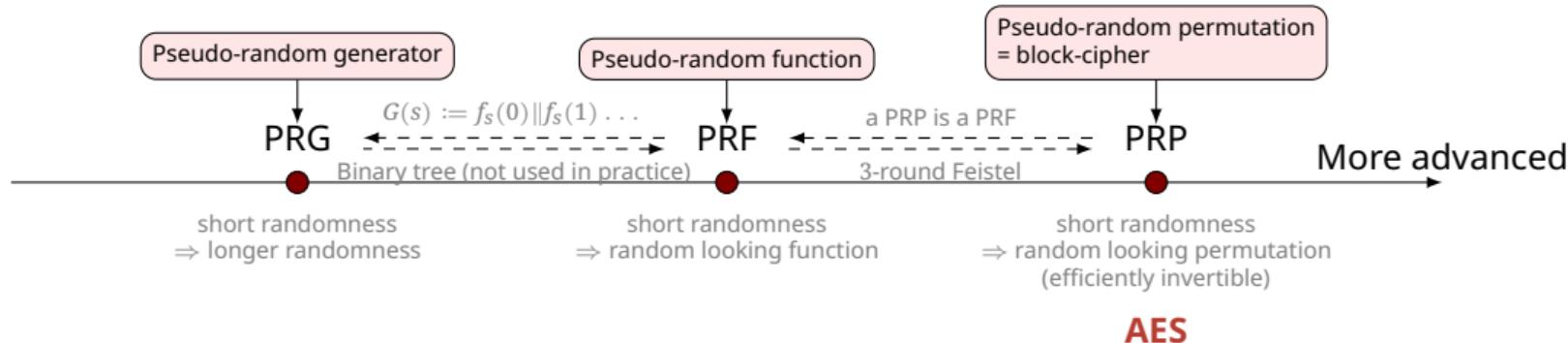
Encryption from simpler primitives

How to build an encryption:

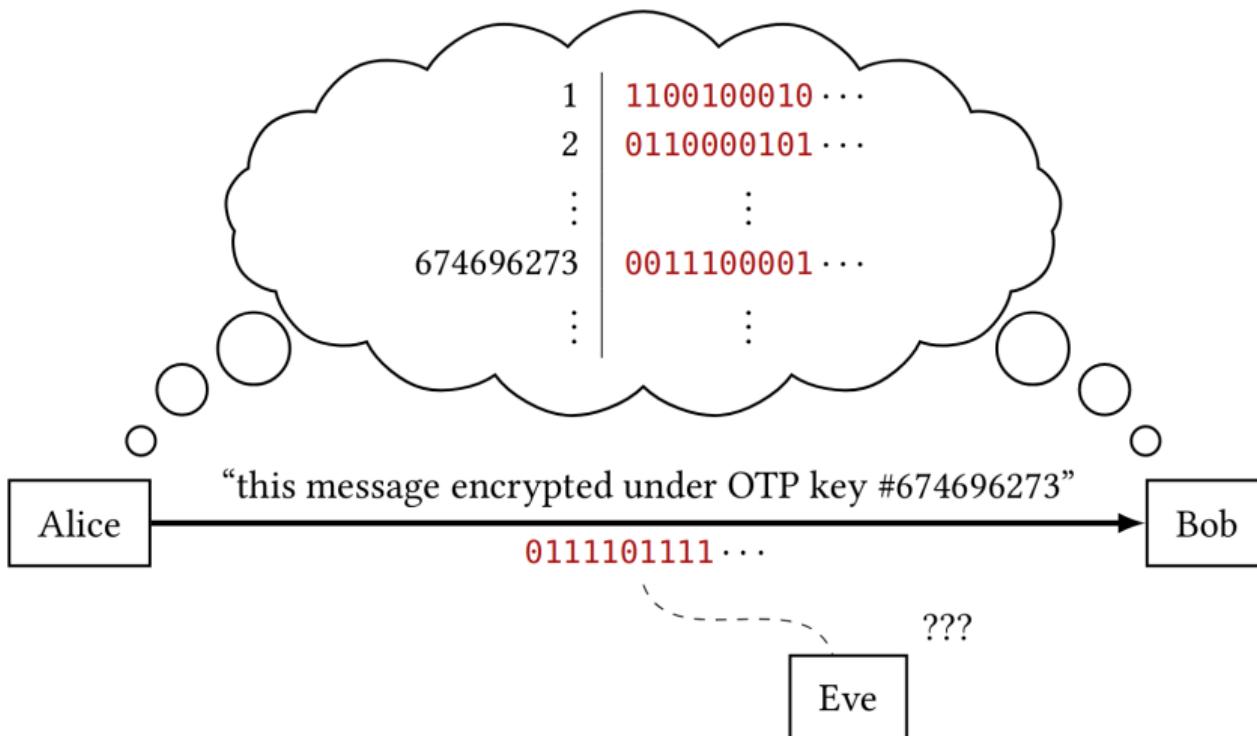
- Approach 1: start from scratch. Less guarantees it will be secure.
- Approach 2 try to build encryption from simpler, more tested, primitives.

Our approach

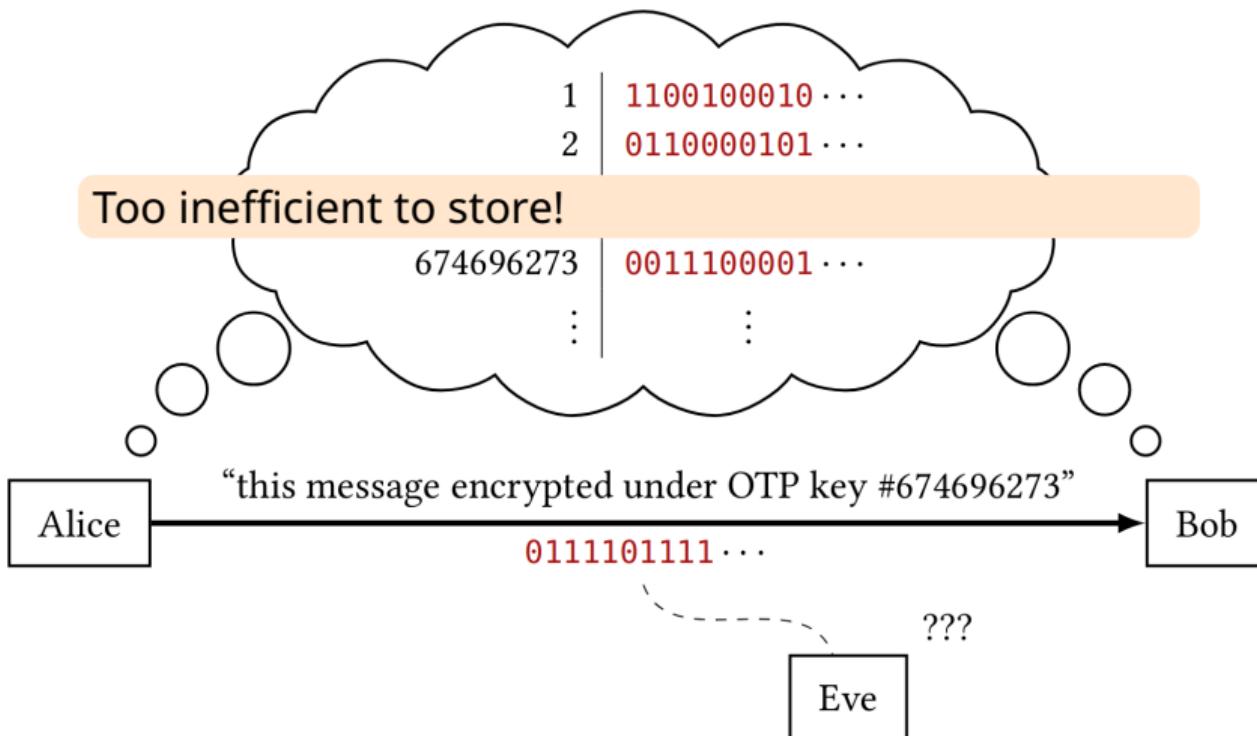
But which **more fundamental primitive can we use?**



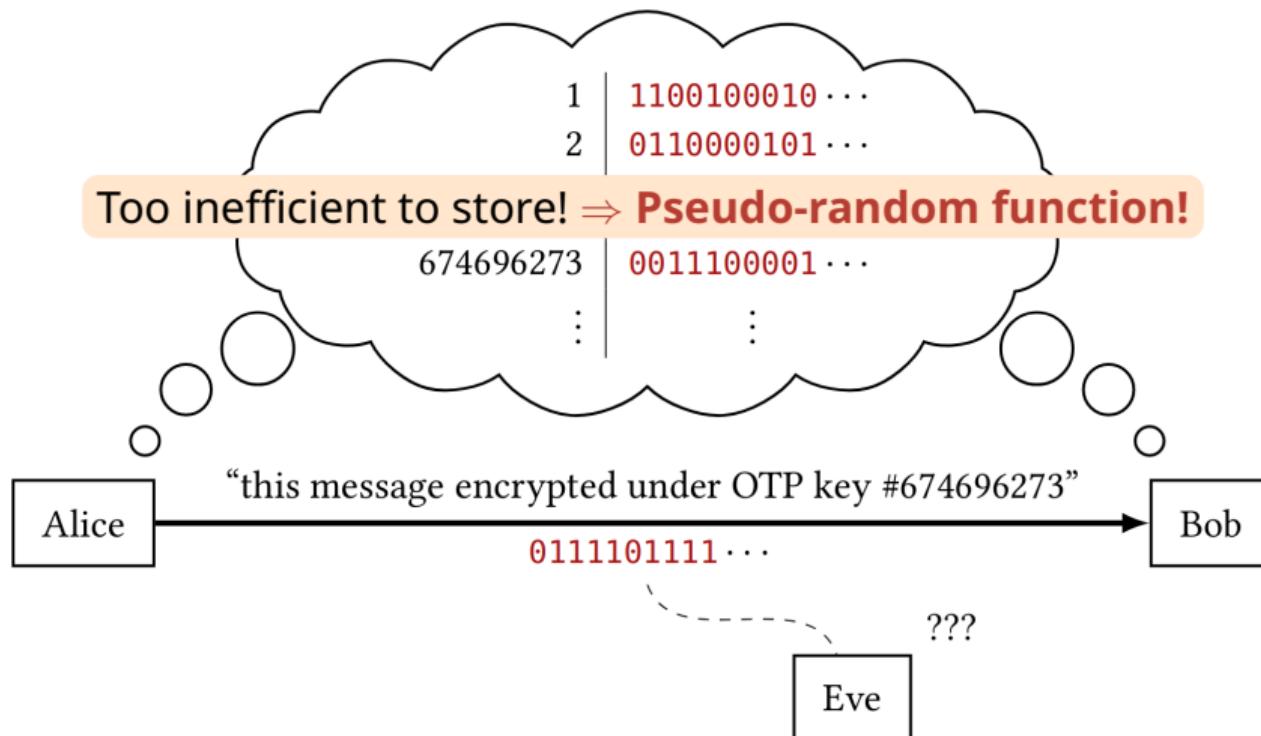
Motivation PRF



Motivation PRF



Motivation PRF



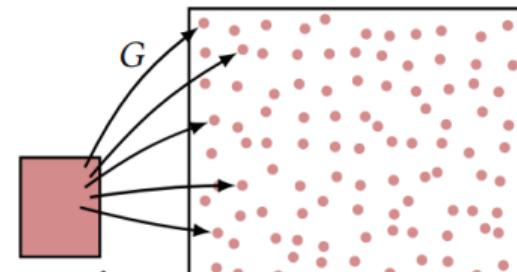
Pseudo-Random Generator (PRG)

PRG

Let $G: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+l}$ be a deterministic function with $l > 0$. We say that G is a *secure Pseudo-Random Generator (PRG)* if:

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

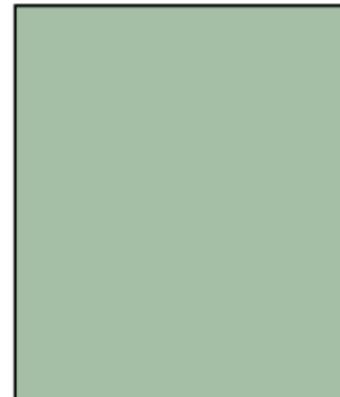
Pseudo-Random Generator (PRG)



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

$\{0, 1\}^{2\lambda}$

pseudorandom distribution



$\{0, 1\}^{2\lambda}$

uniform distribution

PRG \neq random number generator: small uniform source vs large non-uniform noise

Pseudo-Random Function (PRF)

PRF

Let $F: \{0, 1\}^\lambda \times \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}$ be a deterministic function. We say that F is a *secure Pseudo-Random Function (PRF)* if:

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

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

Pseudo-Random Permutation (PRP)

PRP

Let $F: \{0, 1\}^\lambda \times \{0, 1\}^{\text{blen}} \rightarrow \{0, 1\}^{\text{blen}}$ be a deterministic function. We say that F is a *secure Pseudo-Random Permutation (PRP)*, a.k.a. *block cipher*, if f is invertible, i.e. if there exists an efficient function F^{-1} such that $\forall x, k:$

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

and if, after defining $T.\text{values} := \{v \mid \exists x, T[x] = v\}$, we have:

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

$$k \xleftarrow{\$} \{0, 1\}^\lambda$$
$$\text{LOOKUP}(x \in \{0, 1\}^{\text{blen}}):$$

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

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

$$T := \text{empty assoc. array}$$
$$\text{LOOKUP}(x \in \{0, 1\}^{\text{blen}}):$$

$$\text{if } T[x] \text{ undefined:}$$
$$T[x] \xleftarrow{\$} \{0, 1\}^{\text{blen}} \setminus T.\text{values}$$
$$\text{return } T[x]$$

PRP vs PRF

How far are PRP from PRF?

Natural attack: call $\text{LOOKUP}(x)$ on random x many times (say N) until we **find a collision** ($\text{LOOKUP}(x) = \text{LOOKUP}(x')$ for $x' \neq x$). If we can't find any, claim PRP, otherwise PRF.

Naively, think this has advantage $\approx \frac{1}{N}$, but much more efficient: $\approx \frac{1}{\sqrt{N}}$.



The birthday paradox

Birthday paradox = What is the probability of finding two persons with the same birthday in a class of 23 students?



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

The birthday paradox

Birthday paradox = What is the probability of finding two persons with the same birthday in a class of 23 students?



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

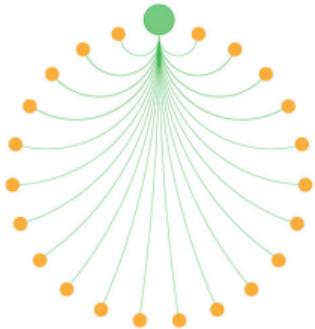
If N = number of elements, n = number of samples, p = proba collision:

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

number of samples for proba collision $1/2 \approx \sqrt{N}$

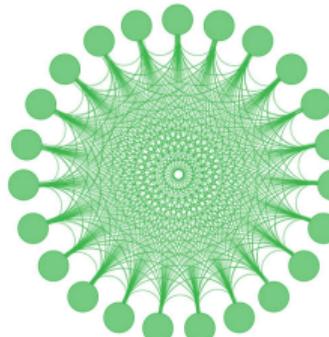
The birthday paradox

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/>

The birthday paradox

We said 2^{128} is HUGE. Is it doable to find a collision on a PRF/hash function with output size 128 bits?



- A Yes, with a laptop
- B Yes, with a GPU/ASIC cluster
- C No

The birthday paradox

We said 2^{128} is HUGE. Is it doable to find a collision on a PRF/hash function with output size 128 bits?



- A Yes, with a laptop
- B Yes, with a GPU/ASIC cluster ✓ $\sqrt{2^{128}} = 2^{128/2} = 2^{64}$.
⇒ First course, 2^{64} doable with GPU/ASIC cluster.
- C No

The birthday paradox

But asymptotically, the birthday paradox does not cause issues:

Theorem (Asymptotic birthday paradox)

We have

$$\begin{array}{|c|}\hline \mathcal{L}_{\text{samp-L}} \\ \hline \text{SAMP ():} \\ r \xleftarrow{\$} \{0, 1\}^\lambda \\ \text{return } r \\ \hline \end{array}$$

\approx

$$\begin{array}{|c|}\hline \mathcal{L}_{\text{samp-R}} \\ \hline R := \emptyset \\ \text{SAMP ():} \\ r \xleftarrow{\$} \{0, 1\}^\lambda \setminus R \\ R = R \cup \{r\} \\ \text{return } r \\ \hline \end{array}$$

Proof. See Caseine exercise “Asymptotic birthday paradox”.

The birthday paradox

But asymptotically, the birthday paradox does not cause issues:

Theorem (Asymptotic birthday paradox)

We have

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

\approx

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

$=$

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

\approx

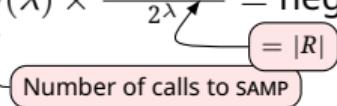
$$\frac{\mathcal{L}_{\text{samp-R}}}{R := \emptyset \\ \text{bad} := 0 \\ \text{SAMP}(): \\ r \leftarrow \{0, 1\}^\lambda \\ \text{if } r \in R: \\ \quad \text{bad} := 1 \\ \quad r \leftarrow \{0, 1\}^\lambda \setminus R \\ R = R \cup \{r\} \\ \text{return } r}$$

$=$

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

Proof. Bad-event lemma: \mathcal{A} is polynomial, so

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



The birthday paradox

Take-home message

The birthday paradox does not harm asymptotic security ($\sqrt{\text{negl}(\lambda)} = \text{negl}(\lambda)$), but in real life, the **size of the key may need to be doubled** to prevent this attack.

A PRP is a PRF

A PRP is a PRF

Let $F: \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ be a secure PRP (with blen = λ). Then F is also a secure PRF.

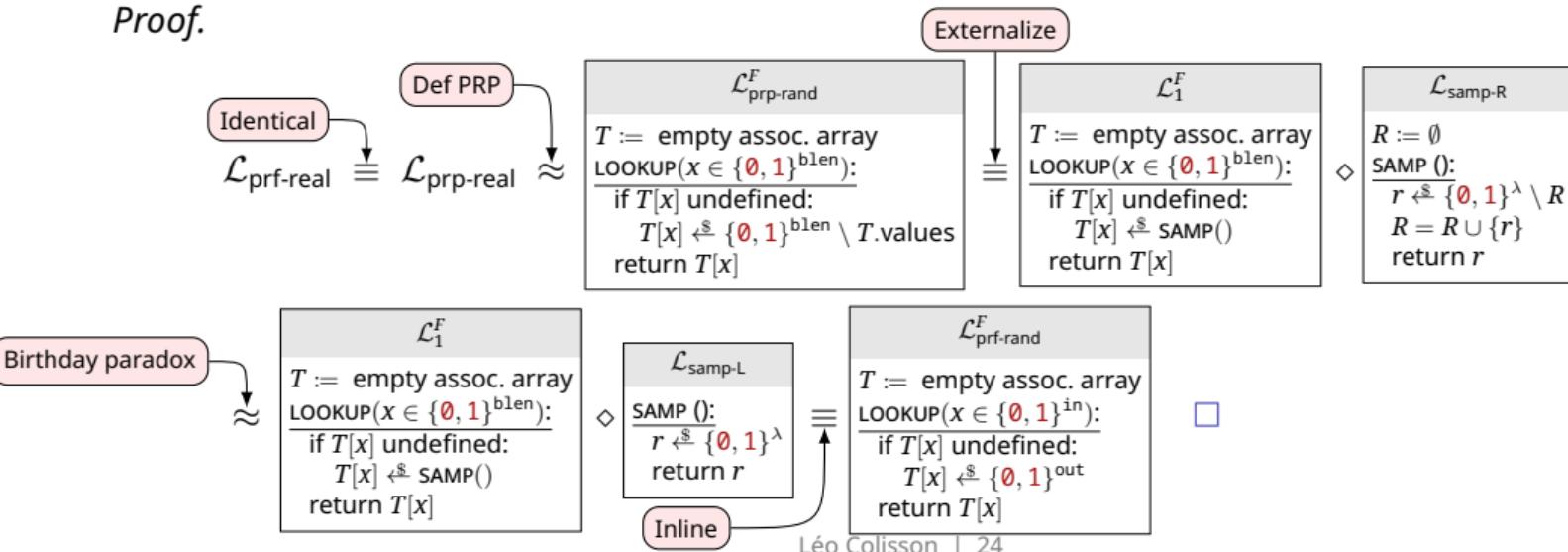
Proof. Caseine exercise “A PRP is a PRF”

A PRP is a PRF

A PRP is a PRF

Let $F: \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ be a secure PRP (with $\text{blen} = \lambda$). Then F is also a secure PRF.

Proof.



How to build IND-CPA schemes from PRF or block-ciphers?

IND-CPA from PRF

Based on above idea, first (not so efficient) solution:

Definition (PRF pseudo-OTP)

Let F be a secure PRF. We define the PRF pseudo-OTP encryption scheme as $\mathcal{K} = \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda$, $\mathcal{M} = \{\textcolor{red}{0}, \textcolor{red}{1}\}^{\text{out}}$, $\mathcal{C} = \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda \times \{\textcolor{red}{0}, \textcolor{red}{1}\}^{\text{out}}$, and:

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

Theorem (security PRF pseudo-OTP)

The PRF pseudo-OTP is IND-CPA secure.



Exercice: try to prove its security (answer next slide)

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

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

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

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

\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

Externalize

\equiv

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

$$\approx$$

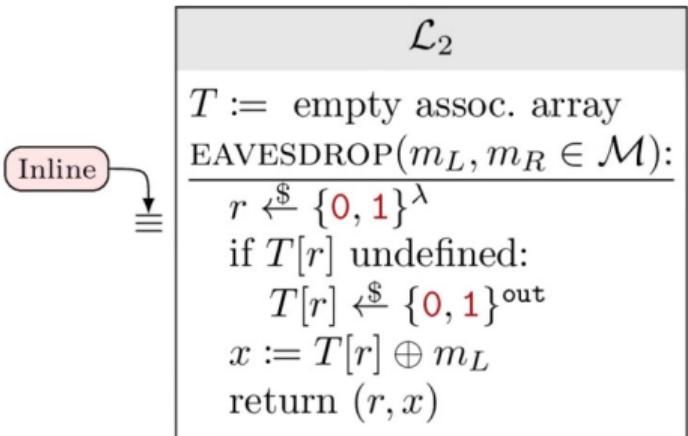
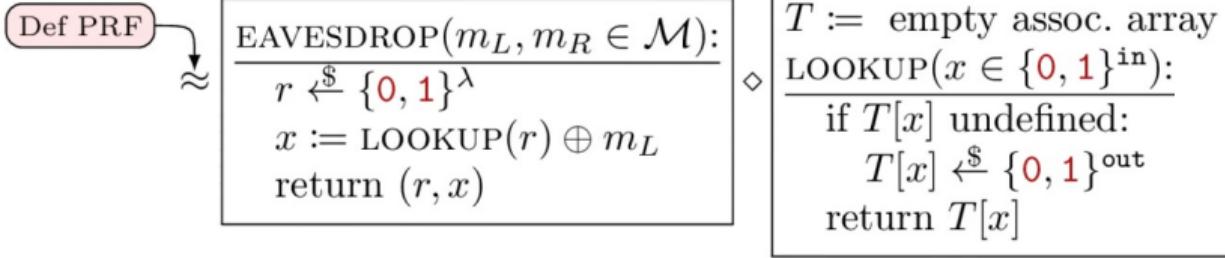
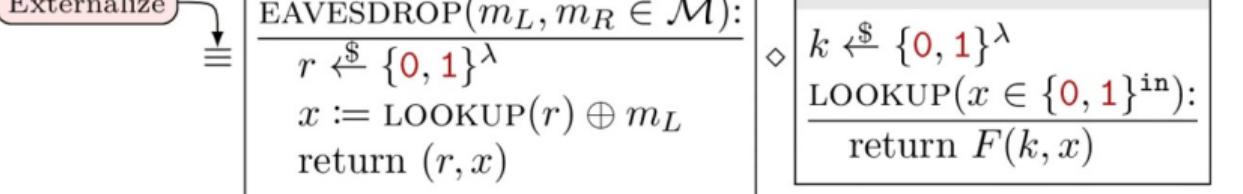
\mathcal{L}_1

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

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

$$\diamond$$

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



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

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

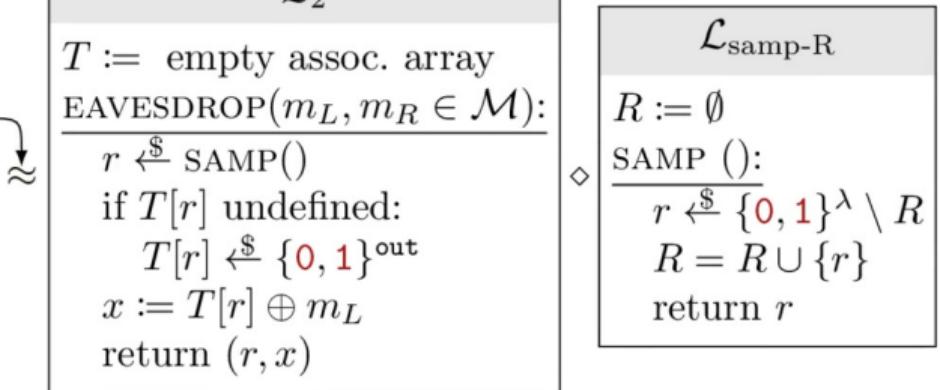
$R := \emptyset$
 $\text{SAMP}():$

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

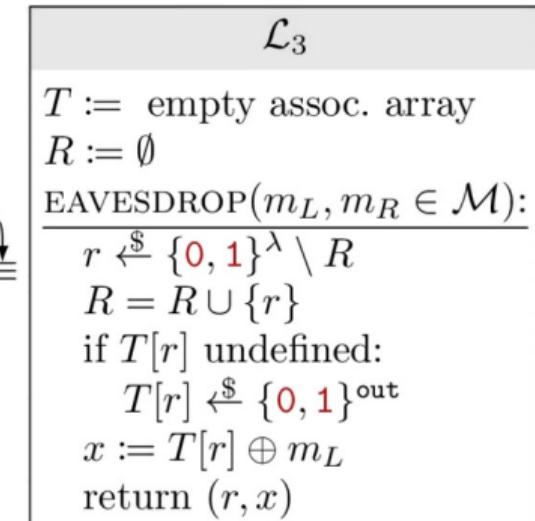
$R = R \cup \{r\}$
return r

 \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

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

\diamond

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

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

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

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

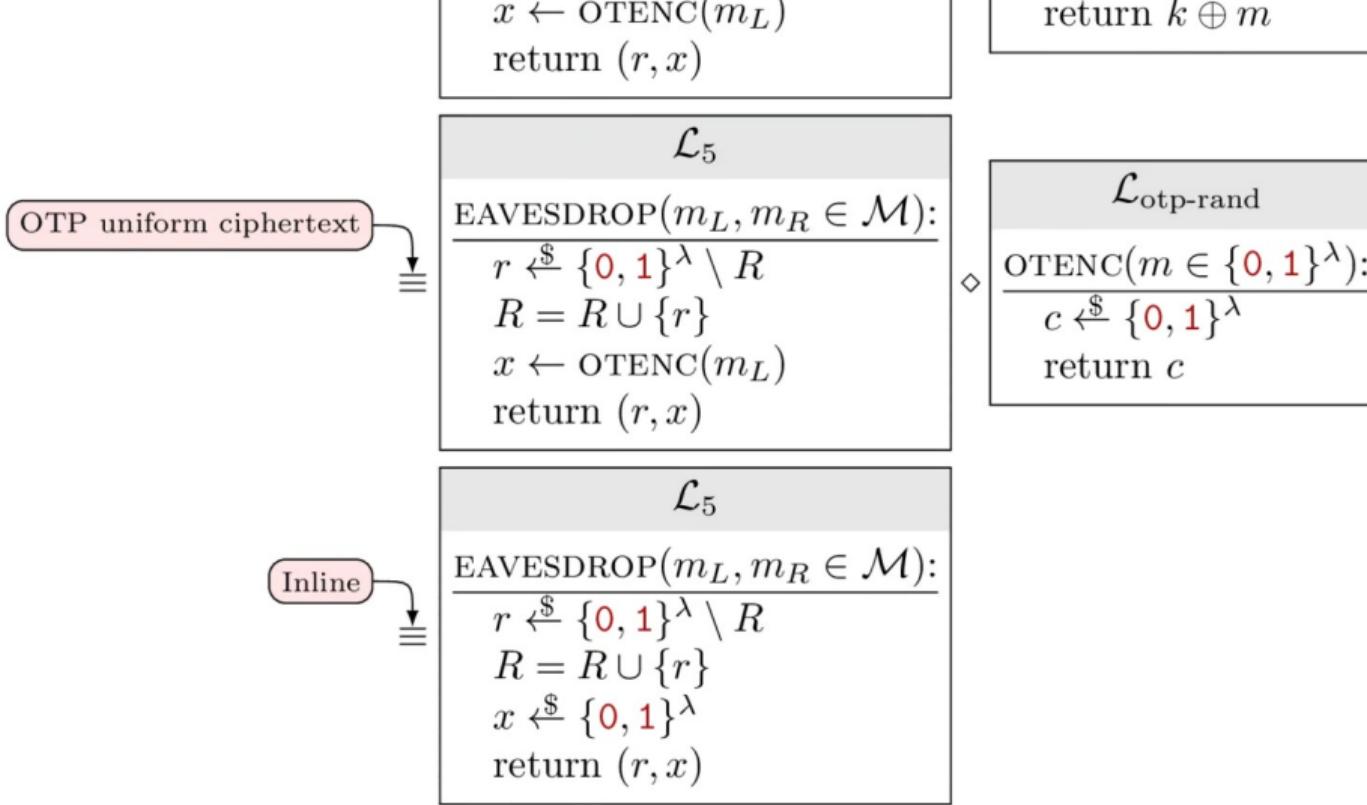
$$\text{return } (r, x)$$

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

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

Limitations PRF pseudo-OTP

Good to have secure IND-CPA scheme, but **how do we encrypt an arbitrary long message m ?**

- First idea: **cut m in chunks** of length $\{0, 1\}^{\text{out}}$, and encrypt them separately.
⇒ Issue: remember, Enc is a tuple (r, x) , i.e. for l chunks, overhead of λl
- Too inefficient!** 😰

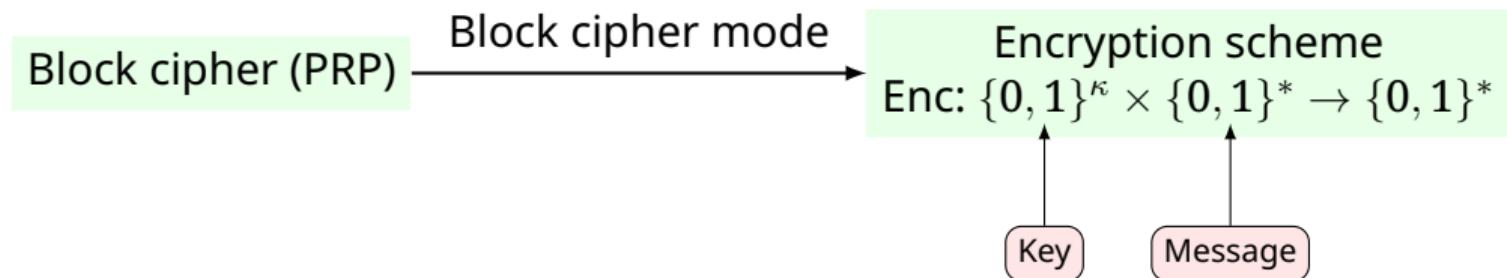
Limitations PRF pseudo-OTP

Good to have secure IND-CPA scheme, but **how do we encrypt an arbitrary long message m ?**

- First idea: **cut m in chunks** of length $\{0, 1\}^{\text{out}}$, and encrypt them separately.
⇒ Issue: remember, Enc is a tuple (r, x) , i.e. for l chunks, overhead of λl
Too inefficient! 
- Solution: use **block cipher modes!**

Block cipher modes

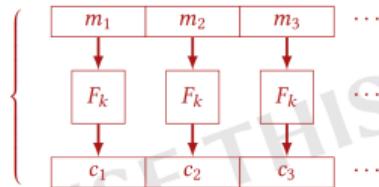
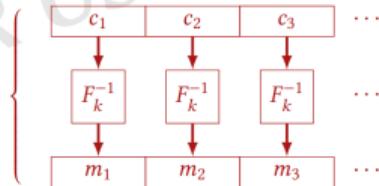
Multiple modes of operation (= variants):



Common modes

Definition (ECB mode: NEVER USE THIS)

The (INSECURE!) Electronic Codebook (ECB) mode is defined as:

$$\begin{array}{l} \text{Enc}(k, m_1 \| \dots \| m_\ell): \\ \quad \text{for } i = 1 \text{ to } \ell: \\ \quad \quad c_i := F(k, m_i) \\ \quad \text{return } c_1 \| \dots \| c_\ell \end{array}$$

$$\begin{array}{l} \text{Dec}(k, c_1 \| \dots \| c_\ell): \\ \quad \text{for } i = 1 \text{ to } \ell: \\ \quad \quad m_i := F^{-1}(k, c_i) \\ \quad \text{return } m_1 \| \dots \| m_\ell \end{array}$$


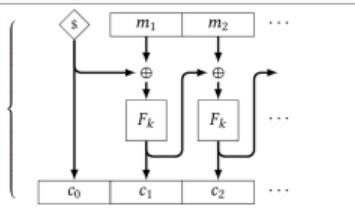
This mode is said to be worse than deterministic. Find an attack that make a single call to the encryption function.

Common modes

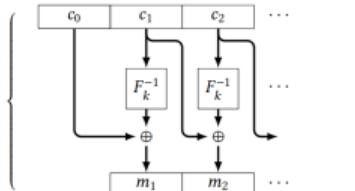
Definition (CBC mode)

The Cipher Block Chaining (CBC) mode is defined as:

```
Enc( $k, m_1 \parallel \dots \parallel m_\ell$ ):  
   $c_0 \leftarrow \{0, 1\}^{b\ell m_r}$   
  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 is called the initialization vector (IV). Why can't we set it to a fixed value?



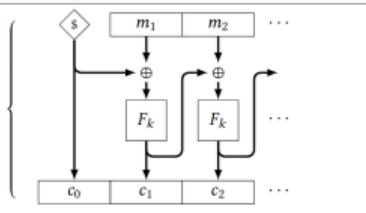
- A It acts like a OTP key on the message, hence hides it
- B Used to have a non-deterministic encryption

Common modes

Definition (CBC mode)

The Cipher Block Chaining (CBC) mode is defined as:

```
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 is called the initialization vector (IV). Why can't we set it to a fixed value?



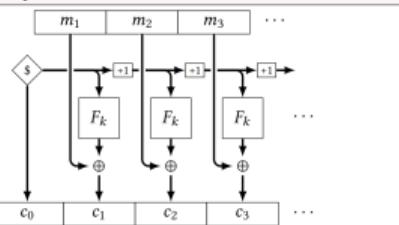
- A It acts like a OTP key on the message, hence hides it
X IV is public, so cannot be a OTP key!
- B Used to have a non-deterministic encryption ✓

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:

```
Enc( $k, m_1 \parallel \dots \parallel m_\ell$ ):  
   $r \leftarrow \{0, 1\}^{b\text{len}}$   
   $c_0 := r$   
  for  $i = 1$  to  $\ell$ :  
     $c_i := F(k, r) \oplus m_i$   
     $r := r + 1 \% 2^{b\text{len}}$   
  return  $c_0 \parallel \dots \parallel c_\ell$ 
```



Try to find the decryption algorithm. Do you need to compute F^{-1} ?



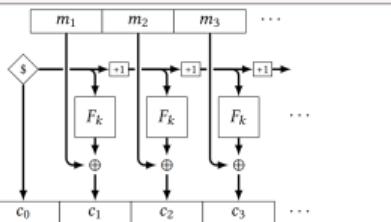
- A Yes
- B No

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:

```
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  $c_0 \parallel \dots \parallel c_\ell$ 
```



Try to find the decryption algorithm. Do you need to compute F^{-1} ?



A Yes

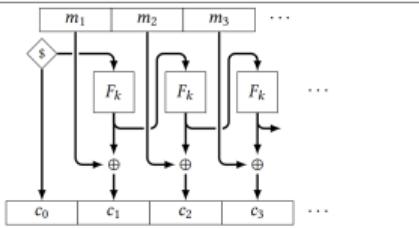
B No No need to have a PRP, PRF is enough (but in practice, most efficient PRF are PRP anyway)

Common modes

Definition (OFB mode)

The output feedback (OFB) mode is defined as:

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



Try to find the decryption algorithm. Do you need to compute F^{-1} ?



- A Yes
- B No

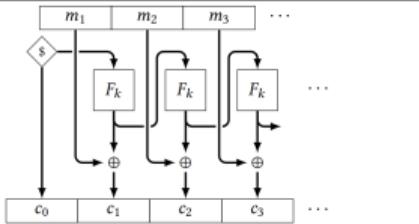
Common modes

Definition (OFB mode)

The output feedback (OFB) mode is defined as:

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

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



Try to find the decryption algorithm. Do you need to compute F^{-1} ?



A Yes X

B No ✓ No need to have a PRP, PRF is enough

Comparison of modes

	ECB	CBC	CTR	OFB
IND-CPA	✗ !!!	✓	✓	✓
Parallelizable	✗	✓	✗	
Pre-computable	✗	✓	✓	
Can avoid padding	✗	✓	✓	
Safer with no permutation cycle	✗	✓	✗	
Slightly safer against IV re-use (e.g. in bad implementation)		✓	✗	✗

Winner is CTR mode! (but wait encrypt & authenticate modes like GCM)

Comparison of modes



A friend proposes to encode your hard drive with AES in OFB mode.
Is this a good idea? Why?

- A Yes
- B No

Comparison of modes

A friend proposes to encode your hard drive with AES in OFB mode.
Is this a good idea? Why?



- A Yes X
- B No ✓ Mauvaise idée : le mode OFB n'est pas parallélisable. Il faut donc déchiffrer tout le disque pour accéder au dernier octet !

Modes vulnerable to birthday attacks

All modes are vulnerable to birthday attacks (cf TD), so make sure you encrypt less than $2^{\text{blen}/2}$ blocks (i.e. keep blen large, e.g. don't use 3DES! (64 bits)).

Today: most widely used cipher is

Advanced Encryption Standard (AES)

with 128 bits block length (key length: 128, 192 or 256 bits). See also:

- Rijndael (generalization AES): block length 128, 192, or 256,
- Serpent (2nd finalist in Advanced Encryption Standard process)
- Twofish (blen = 128) and blowfish (warning: blen = 64!)
- never use DES = broken (previous standard), temporarily replaced by 3DES

IND-CPA for variable-length plaintexts

Can you find a generic IND-CPA attack against these cipher modes of operation (e.g. CTR, assume blen = λ for simplicity)?

A No



B Yes, with

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

C Yes, with

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

IND-CPA for variable-length plaintexts

Can you find a generic IND-CPA attack against these cipher modes of operation (e.g. CTR, assume blen = λ for simplicity)?

A No



B Yes, with

\mathcal{A}

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

C Yes, with

\mathcal{A}

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



The length of the ciphertext equals

$\lambda + |m| \Rightarrow$ leaks the length of the message!

IND-CPA for variable-length plaintexts

IND-CPA for variable-length plaintexts

When messages can have various length, we need to update the definition of security:

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

≈

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

Is leaking the length an issue?

Sometimes! E.g.:

- Google maps sends tiles, each tile having a different size (despite same pixel size) due to compression \Rightarrow possible to know what tile is displayed only by looking at traffic
- Variable-bit-rate (VBR) in video shows different (chunk of) "frame" size depending on the time. Possible to know which movie you watch on netflix/youtube based on this, and even identity speaker/language/word spoken in voice chat programs!

Padding

Padding

What if $|m|$ is not a multiple of the block length?

- CTR mode: simple, just truncate the ciphertext (like regular OTP)
- CBC mode: need to add **padding** (add data until reaching block length)
(also possible to do “ciphertext stealing” in this specific case)

Padding

Many ways to pad m into m' :

- add zeros: not working! When decrypting, how do you know how many zeros to remove?
- ANSI X.923 standard: add $\textcolor{red}{0}$'s until the last byte that contains the number of padded bytes
- PKCS#7 standard: if b bytes of padding needed, add the actual b byte b times
- ISO/IEC 7816-4 standard: append $\textcolor{red}{10\dots 0}$

The actual choice has **little importance**, not really a security feature (at least when considering passive adversaries, see later)

Padding

Consider ISO/IEC 7816-4 standard (append $10\ldots0$): if you pad a message m of size $k\text{blen}$ into m' , what is the size of m' ?



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

Padding

Consider ISO/IEC 7816-4 standard (append $10\dots0$): if you pad a message m of size $k\text{blen}$ into m' , what is the size of m' ?

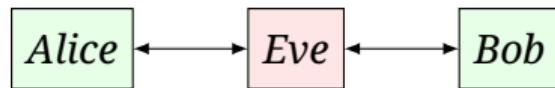


- A $k\text{blen}$
- B $k\text{blen} + 1$
- C $(k + 1)\text{blen}$ (like all paddings, it increases the size of the message)

Padding oracle attack & CCA security

Padding oracle attack

Before: passive adversary (somewhat unrealistic). Now, we consider **active** adversaries:



What happens if Bob returns an error if the padding is incorrect?
⇒ Eve can completely recover the encrypted message!

Padding oracle attack (illustrate on board)

Attack model: CTR mode, padding ANSI X.923, \mathcal{A} has access to

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

(hence $\text{VALIDPAD}(m)$ checks if m ends with a byte b containing before $b - 1$ bytes filled with 0's). Say that we have access to $c_0 \leftarrow \text{Enc}_k(m_0)$ (where m_0 is already padded), goal is to find m_0 .

- step 0: realize that in CTR mode, $\text{Enc}_k(m) \oplus (0^{\text{blen}}, x) = \text{Enc}_k(m \oplus x)$. So we can change the message from the ciphertext (hence later I'll say "apply an operation on m " even if in fact we apply it on $\text{Enc}_k(m)$).

Padding oracle attack (illustrate on board)

Attack model: CTR mode, padding ANSI X.923, \mathcal{A} has access to

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

(hence $\text{VALIDPAD}(m)$ checks if m ends with a byte b containing before $b - 1$ bytes filled with 0's). Say that we have access to $c_0 \leftarrow \text{Enc}_k(m_0)$ (where m_0 is already padded), goal is to find m_0 .

- step 0: realize that in CTR mode, $\text{Enc}_k(m) \oplus (0^{\text{blen}}, x) = \text{Enc}_k(m \oplus x)$. So we can change the message from the ciphertext (hence later I'll say "apply an operation on m " even if in fact we apply it on $\text{Enc}_k(m)$).
- first step: determine length of the message (changing any bit of the message does NOT trigger an error, changing a bit of the padding does)

Padding oracle attack (illustrate on board)

Attack model: CTR mode, padding ANSI X.923, \mathcal{A} has access to

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

(hence $\text{VALIDPAD}(m)$ checks if m ends with a byte b containing before $b - 1$ bytes filled with 0's). Say that we have access to $c_0 \leftarrow \text{Enc}_k(m_0)$ (where m_0 is already padded), goal is to find m_0 .

- first step: determine length of the message (changing any bit of the message does NOT trigger an error, changing a bit of the padding does)
- second step: once you know the length of the padding p , you know that m_0 looks like $m_{\text{unpad}} 0^{8p} \text{Byte}(p)$. Xor to the last byte of c_0 the byte $\text{Byte}(p) \oplus \text{Byte}(p+1)$. Thanks to step 0 you now have an encryption of $m_{\text{unpad}} 0^{8p} \text{Byte}(p+1)$. Since m_{unpad} does not (a-priori) ends with a zero-byte, PADDINGORACLE will return an error. Now we iterate over $x \in \{0, \dots, 255\}$ by xoring the last bit of (the encryption of) m_{unpad} with x , and calling PADDINGORACLE on it. At some points, it will not error: the last bit of m_{unpad} is equal to x !

Padding oracle attack (illustrate on board)

Attack model: CTR mode, padding ANSI X.923, \mathcal{A} has access to

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

(hence $\text{VALIDPAD}(m)$ checks if m ends with a byte b containing before $b - 1$ bytes filled with 0's). Say that we have access to $c_0 \leftarrow \text{Enc}_k(m_0)$ (where m_0 is already padded), goal is to find m_0 .

- second step: once you know the length of the padding p , you know that m_0 looks like $m_{\text{unpad}} 0^{8p} \text{Byte}(p)$. Xor to the last byte of c_0 the byte $\text{Byte}(p) \oplus \text{Byte}(p+1)$. Thanks to step 0 you now have an encryption of $m_{\text{unpad}} 0^{8p} \text{Byte}(p+1)$. Since m_{unpad} does not (a-priori) ends with a zero-byte, PADDINGORACLE will return an error. Now we iterate over $x \in \{0, \dots, 255\}$ by xoring the last bit of (the encryption of) m_{unpad} with x , and calling PADDINGORACLE on it. At some points, it will not error: the last bit of m_{unpad} is equal to x !
- last step: we start again from second step until we find all bits of m .

Limitation of IND-CPA security

Fundamental issue: not padding, but server behaves **differently based on the decrypted value.**

In practice, this is **extremely common** and hard to avoid (e.g. it takes maybe a bit longer to decrypt some messages, or does different operations based on the decrypted value...)

⇒ **We need a more resilient security definition:** allow attacker to decrypt arbitrary messages = IND-CCA!

IND-CCA security

IND-CCA

IND-CCA

Let Σ be an encryption scheme. We say that Σ has indistinguishable security against **chosen-ciphertext attacks (IND-CCA)** if:

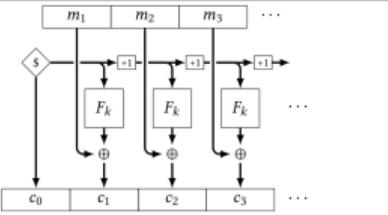
$\mathcal{L}_{\text{cca-L}}^{\Sigma}$
$k \leftarrow \text{Gen}(1^\lambda)$
$\mathcal{S} := \emptyset$
<u>EAVESDROP($m_L, m_R \in \mathcal{M}$):</u>
if $ m_L \neq m_R $ return err
$c := \text{Enc}_k(m_L)$
$\mathcal{S} := \mathcal{S} \cup \{c\}$
return c
<u>DECRYPT($c \in \mathcal{C}$):</u>
if $c \in \mathcal{S}$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

\approx

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

Malleability

```
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  $c_0 \parallel \dots \parallel c_\ell$ 
```



Can you find a CCA attack against, e.g., CTR mode?

A No

B Yes, with Oui, avec

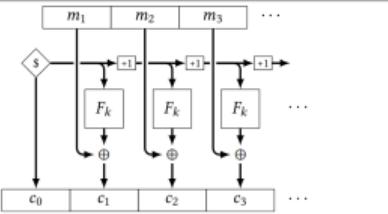
```
 $\mathcal{A}$   
 $(c_0, c_1) \leftarrow \text{EAVESDROP}(0^{blen}, 1^{blen})$   
 $m \leftarrow \text{DECRYPT}((c_0, c_1 \oplus (10 \dots 0)))$   
return  $m = ?$   $10 \dots 0$ 
```

C Yes, with

```
 $\mathcal{A}$   
 $(c_0, c_1) \leftarrow \text{EAVESDROP}(0^{blen}, 1^{blen})$   
 $m \leftarrow \text{DECRYPT}((c_0, c_1))$   
return  $m = 0^{blen}$ 
```

Malleability

```
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  $c_0 \parallel \dots \parallel c_\ell$ 
```



Can you find a CCA attack against, e.g., CTR mode?

A No

B Yes, with Oui, avec

\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 $m = ?$

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



Malleability

Fundamental reason: CTR is **malleable**, i.e. we can obtain

$\text{Enc}_k(x') = (c_0, x' \oplus F_k(c_0))$ from $\text{Enc}_k(x) = (c_0, x \oplus F_k(c_0))$ (just add $x \oplus x'$ to the second element of the tuple).

Problem in real life: e.g. we can turn a “Yes” into a “No”.

How to prevent this? **Authentication!** (later course)

Conclusion

- OTP is statistically secure if **used once**
- A first notion of security against passive adversary is **IND-CPA**
- PRF \Rightarrow IND-CPA secure schemes
- **Birthday paradox** = may need to double the size of key
- **Block-cipher modes** = encrypt efficiently arbitrarily long messages (padding sometimes necessary)
- CTR mode has good properties (but wait GCM)
- **AES** = standardized PRP (hence PRF) used in block-cipher modes
- **Malleable** encryption \Rightarrow attacks against active adversaries (e.g. padding oracle/timing attacks)
- Spoiler: Authentication will help us!