Crypto Engineering 2025–2026

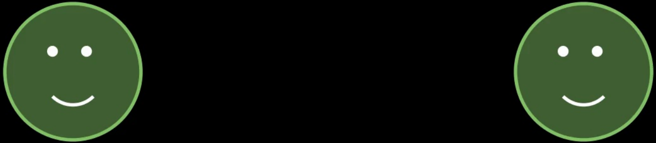
Symmetric encryption

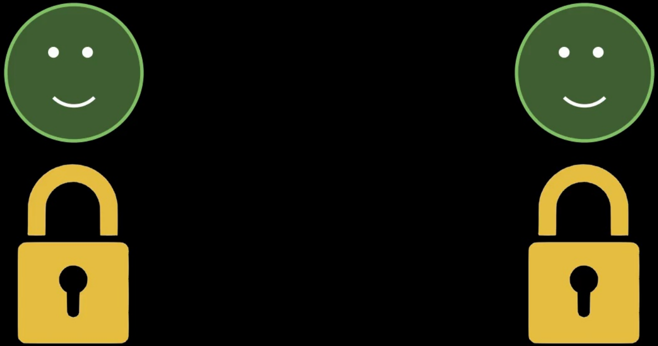
Léo COLISSON PALAIS
Master CySec UGA

leo.colisson-palais@univ-grenoble-alpes.fr

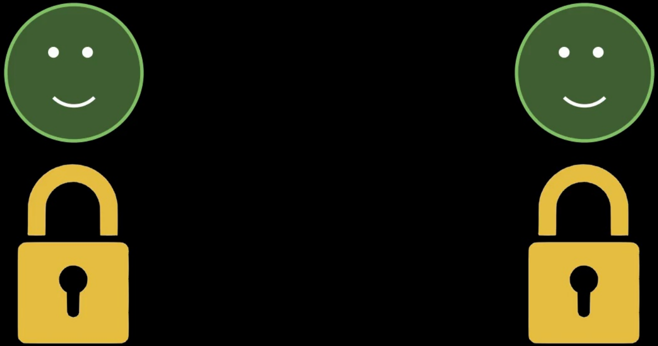
`https://leo.colisson.me/teaching/`

Reminder symmetric encryption & IND-CPA security





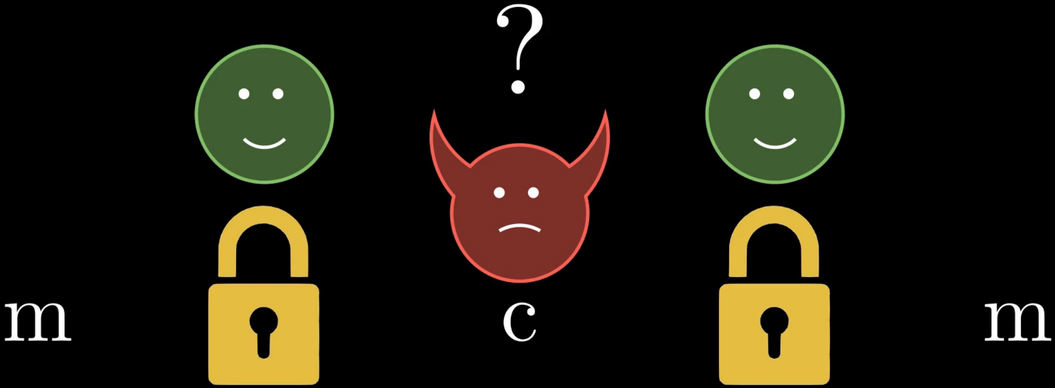
m



m

c


m
?
c


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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of polynomial algorithm:

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

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

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

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

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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as:

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

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

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?

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

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

$$\equiv$$

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

Security of OTP

Theorem

OTP is one-time secure

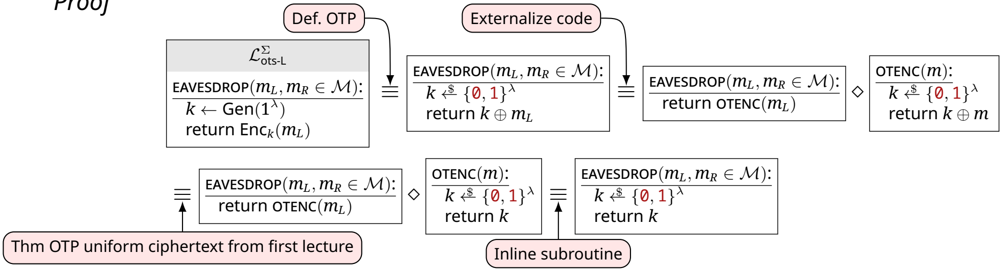
Proof (Exercice. See also the caseine exercice “OTP is one-time secure”)

Security of OTP

Theorem

OTP is one-time secure

Proof



Def. OTP

Externalize code

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

$\equiv$

```
EAVESDROP$(m_L, m_R \in \mathcal{M})$:
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    return $k \oplus m_L$
```

$\equiv$

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

$\diamond$

```
OTENC$(m)$:
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    return $k \oplus m$
```

$\equiv$

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

$\diamond$

```
OTENC$(m)$:
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    return $k$
```

$\equiv$

```
EAVESDROP$(m_L, m_R \in \mathcal{M})$:
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    return $k$
```

Thm OTP uniform ciphertext from first lecture

Inline subroutine

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 $\mathcal{L}^{\Sigma}_{\text{ots-R}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-L}}$. □

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

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

$\approx$

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

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).

Ⓐ Yes

Ⓑ 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).

Ⓐ Yes ✗

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

Define

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

. Then, after inlining, we have

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

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

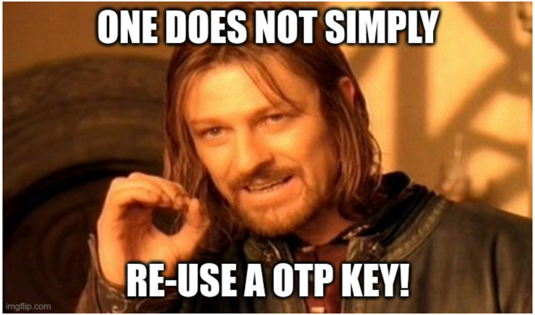
i.e. $\Pr\left[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}} = 1\right] = 1$. But

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

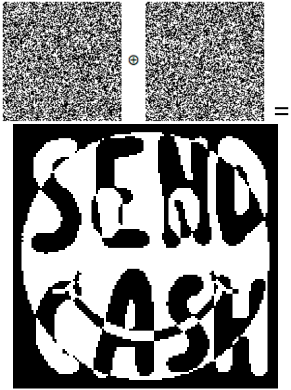
```
$\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-R}}$
$k \leftarrow$ Gen($1^\lambda$)
$x \leftarrow 0^\lambda \oplus k$
$y \leftarrow 1^\lambda \oplus k$
return $x = y$
```

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

Security of OTP

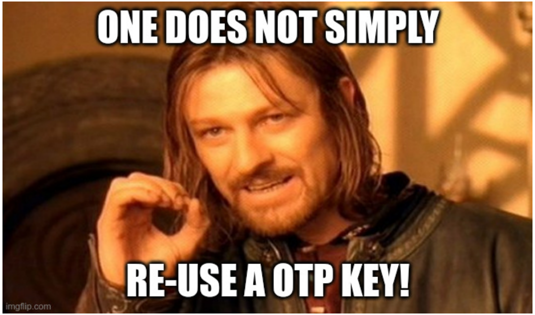


Never reuse a OTP key!!! This can lead to real attack:



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

Security of OTP



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

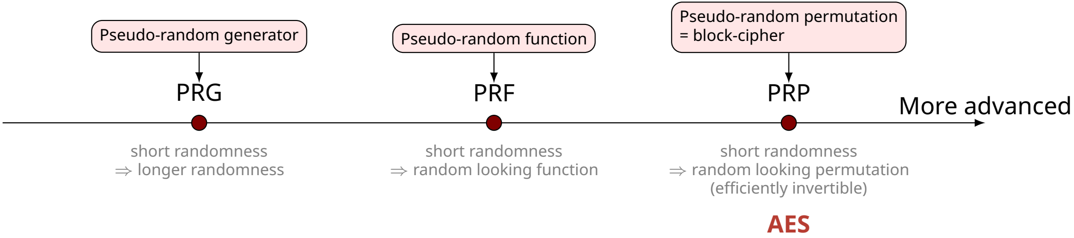
Encryption from simpler primitives

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?



Pseudo-random generator

PRG

short randomness
⇒ longer randomness

Pseudo-random function

PRF

short randomness
⇒ random looking function

Pseudo-random permutation
= block-cipher

PRP

short randomness
⇒ random looking permutation
(efficiently invertible)

AES

More advanced

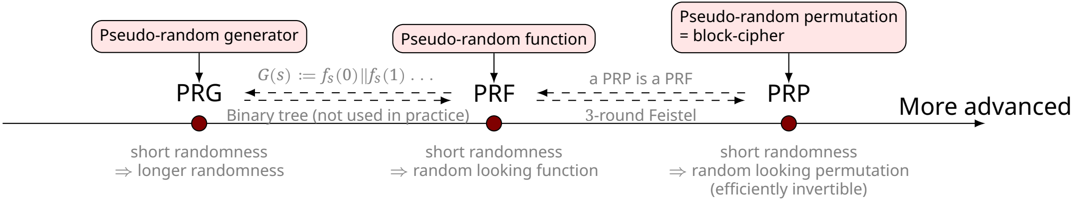
Encryption from simpler primitives

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?



Pseudo-random generator

Pseudo-random function

Pseudo-random permutation
= block-cipher

PRG

$G(s) := f_s(0) \| f_s(1) \ldots$

Binary tree (not used in practice)

PRF

a PRP is a PRF

3-round Feistel

PRP

More advanced

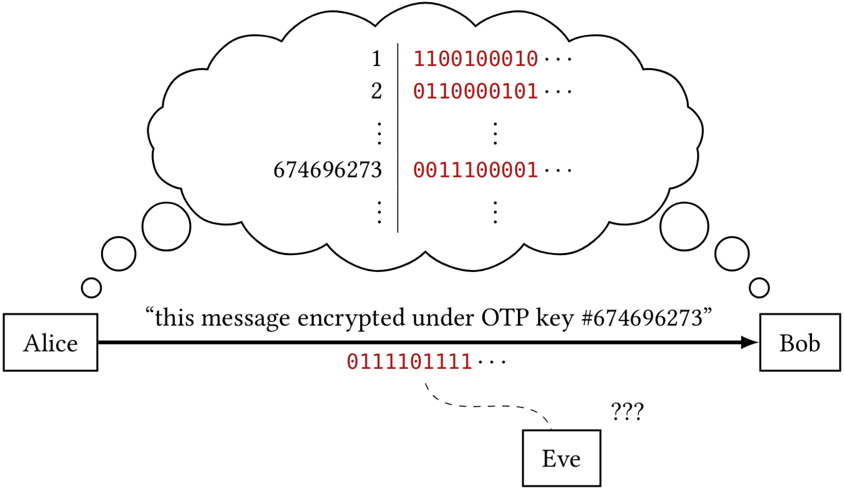
short randomness
$\Rightarrow$ longer randomness

short randomness
$\Rightarrow$ random looking function

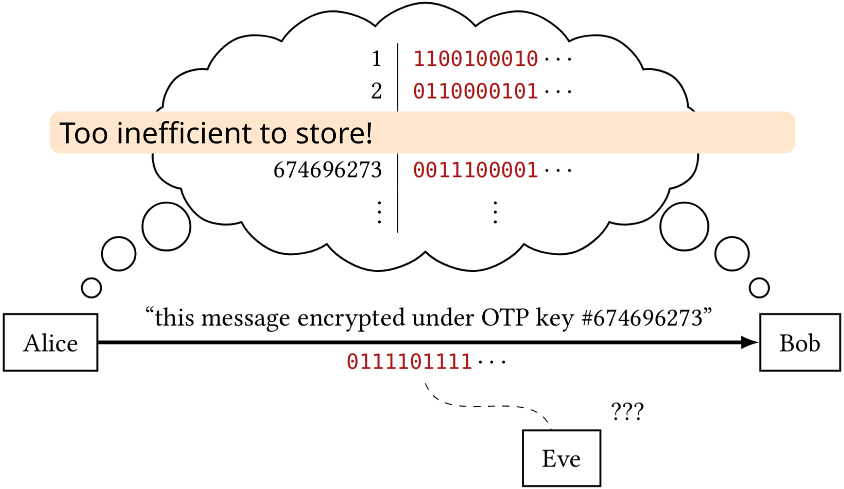
short randomness
$\Rightarrow$ random looking permutation
(efficiently invertible)

AES

Motivation PRF

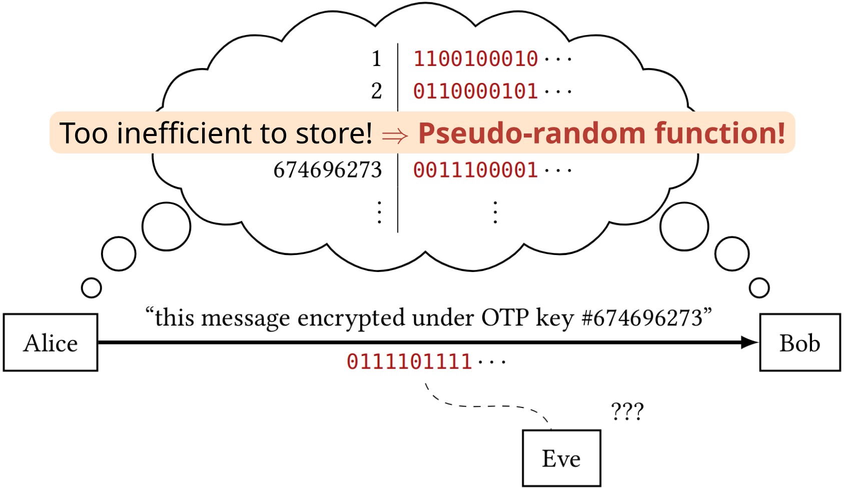


Motivation PRF



Too inefficient to store!

Motivation PRF



Too inefficient to store! ⇒ **Pseudo-random function!**

Pseudo-Random Generator (PRG)

PRG

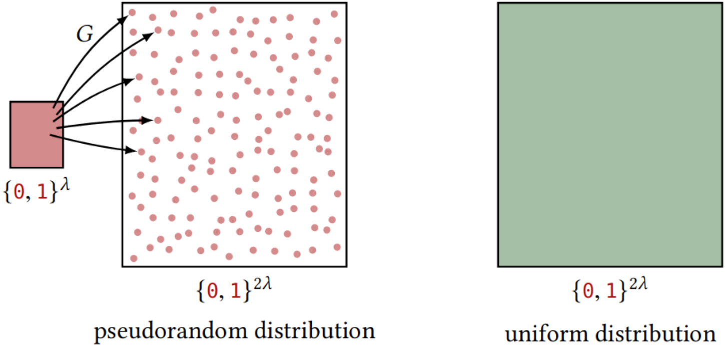
Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{\lambda+l}$ be a deterministic function with $l > 0$. We say that G is a *secure Pseudo-Random Generator (PRG)* if:

```
$\mathcal{L}^{G}_{\text{prg-real}}$
QUERY():
    $s \xleftarrow{\$} \{0,1\}^\lambda$
    return $G(s)$
```

$$\approx$$

```
$\mathcal{L}^{G}_{\text{prg-ideal}}$
QUERY():
    $r \xleftarrow{\$} \{0,1\}^{\lambda+l}$
    return $r$
```

Pseudo-Random Generator (PRG)



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

Pseudo-Random Function (PRF)

PRF

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

$\mathcal{L}^F_{\text{prf-real}}$

```
$k \xleftarrow{\$} \{0,1\}^\lambda$
LOOKUP($x \in \{0,1\}^{\mathtt{in}}$):
    return $F(k,x)$
```

$\approx$

$\mathcal{L}^F_{\text{prf-rand}}$

```
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\mathtt{in}}$):
    if $T[x]$ undefined:
        $T[x] \xleftarrow{\$} \{0,1\}^{\mathtt{out}}$
    return $T[x]$
```

Pseudo-Random Permutation (PRP)

PRP

Let $F\colon \{0,1\}^\lambda \times \{0,1\}^{\mathtt{blen}} \to \{0,1\}^{\mathtt{blen}}$ 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:

```
$\mathcal{L}^F_{\text{prp-real}}$
$k \xleftarrow{\$} \{0,1\}^\lambda$
LOOKUP($x \in \{0,1\}^{\mathtt{blen}}$):
    return $F(k,x)$
```

$$\approx$$

```
$\mathcal{L}^F_{\text{prp-rand}}$
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\mathtt{blen}}$):
    if $T[x]$ undefined:
        $T[x] \xleftarrow{\$} \{0,1\}^{\mathtt{blen}} \setminus T.\text{values}$
    return $T[x]$
```

PRP vs PRF

How far are PRP from PRF?

Natural attack: call $\textsc{Lookup}(x)$ on random x many times (say N) until we **find a collision** ($\textsc{Lookup}(x) = \textsc{Lookup}(x')$ 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?

- Ⓐ 7%
- Ⓑ 20%
- Ⓒ 50%

The birthday paradox

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

Ⓐ 7%

Ⓑ 20%

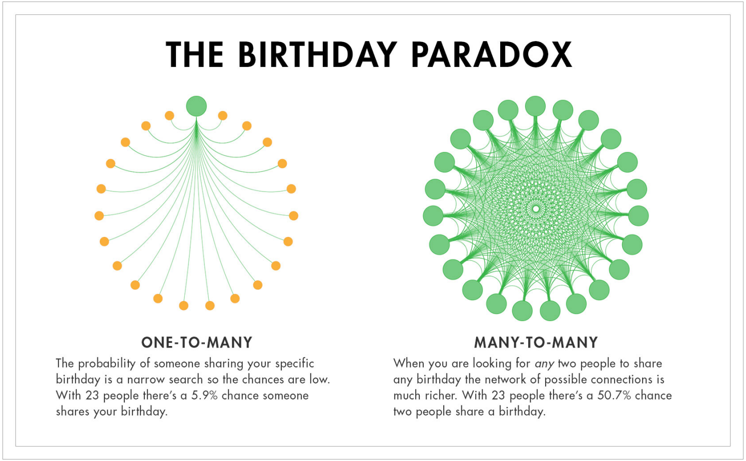
Ⓒ 50% ✓

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?

- Ⓐ Yes, with a laptop
- Ⓑ Yes, with a GPU/ASIC cluster
- Ⓒ 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?

Ⓐ Yes, with a laptop

Ⓑ Yes, with a GPU/ASIC cluster ✓ $\sqrt{2^{128}} = 2^{128/2} = 2^{64}$.
$\Rightarrow$ First course, 2^{64} doable with GPU/ASIC cluster.

Ⓒ No

The birthday paradox

But asymptotically, the birthday paradox does not cause issues:

Theorem (Asymptotic birthday paradox)

We have

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

$\approx$

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

.

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

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

$\approx$

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

.

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

$=$

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

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

$\approx$

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

$=$

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

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

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

$= |R|$

Number of calls to SAMP

The birthday paradox

Take-home message

The birthday paradox does not harm asymptotic security ($\sqrt{\mathsf{negl}(\lambda)} = \mathsf{negl}(\lambda)$), 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 $F\colon \{\texttt{0},\texttt{1}\}^\lambda \times \{\texttt{0},\texttt{1}\}^\lambda \to \{\texttt{0},\texttt{1}\}^\lambda$ be a secure PRP (with blen $= \lambda$). 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 $F\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a secure PRP (with $\text{blen} = \lambda$). Then F is also a secure PRF.

Proof.

$$\mathcal{L}_{\text{prf-real}} \overset{\text{Identical}}{\equiv} \mathcal{L}_{\text{prp-real}} \overset{\text{Def PRP}}{\approx}$$

$\mathcal{L}^F_{\text{prp-rand}}$

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

$$\overset{\text{Externalize}}{\equiv}$$

$\mathcal{L}^F_1$

```
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\text{blen}}$):
  if $T[x]$ undefined:
    $T[x] \xleftarrow{\$}$ SAMP()
  return $T[x]$
```

$$\diamond$$

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

```
$R := \emptyset$
SAMP ():
  $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
  $R = R \cup \{r\}$
  return $r$
```

$$\overset{\text{Birthday paradox}}{\approx}$$

$\mathcal{L}^F_1$

```
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\text{blen}}$):
  if $T[x]$ undefined:
    $T[x] \xleftarrow{\$}$ SAMP()
  return $T[x]$
```

$$\diamond$$

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

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

$$\underset{\text{Inline}}{\equiv}$$

$\mathcal{L}^F_{\text{prf-rand}}$

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

□

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 $\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{M} = \{0,1\}^{\text{out}}$, $\mathcal{C} = \{0,1\}^\lambda \times \{0,1\}^{\text{out}}$, and:

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

IND-CPA from PRF

Theorem (security PRF pseudo-OTP)

The PRF pseudo-OTP is IND-CPA secure.

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

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

```
$k \leftarrow \mathsf{Gen}(1^\lambda)$
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    return $\mathsf{Enc}_k(m_L)$
```

Def. Enc

$\equiv$

$\mathcal{L}_1$

```
$k \leftarrow \mathsf{Gen}(1^\lambda)$
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x := F(k, r) \oplus m_L$
    return $(r, x)$
```

Externalize

$\equiv$

$\mathcal{L}_1$

```
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x := \textsc{lookup}(r) \oplus m_L$
    return $(r, x)$
```

$\diamond$

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

```
$k \xleftarrow{\$} \{0,1\}^\lambda$
$\textsc{lookup}(x \in \{0,1\}^{\mathtt{in}})$:
    return $F(k, x)$
```

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

```
$k \leftarrow \mathsf{Gen}(1^\lambda)$
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    return $\mathsf{Enc}_k(m_L)$
```

Def. Enc $\equiv$

$\mathcal{L}_1$

```
$k \leftarrow \mathsf{Gen}(1^\lambda)$
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$
    $x := F(k, r) \oplus m_L$
    return $(r, x)$
```

Externalize $\equiv$

$\mathcal{L}_1$

```
$\textsc{eavesdrop}(m_L, m_R \in \mathcal{M})$:
    $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$
    $x := \textsc{lookup}(r) \oplus m_L$
    return $(r, x)$
```

$\diamond$

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

```
$k \overset{\$}{\leftarrow} \{0,1\}^\lambda$
$\textsc{lookup}(x \in \{0,1\}^{\texttt{in}})$:
    return $F(k, x)$
```

$\mathcal{L}_1$

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

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

Def. Enc

$\equiv$

Externalize

```
$\mathcal{L}_1$
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x :=$ LOOKUP($r$) $\oplus m_L$
    return $(r, x)$
```

$\diamond$

```
$\mathcal{L}^F_{\text{prf-real}}$
$k \xleftarrow{\$} \{0,1\}^\lambda$
LOOKUP($x \in \{0,1\}^{\texttt{in}}$):
    return $F(k, x)$
```

$\equiv$

Def PRF

```
$\mathcal{L}_1$
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x :=$ LOOKUP($r$) $\oplus m_L$
    return $(r, x)$
```

$\diamond$

```
$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\texttt{in}}$):
    if $T[x]$ undefined:
        $T[x] \xleftarrow{\$} \{0,1\}^{\texttt{out}}$
    return $T[x]$
```

$\approx$

$\mathcal{L}$

Externalize $\equiv$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x :=$ LOOKUP($r$) $\oplus m_L$
    return $(r, x)$
```

$\diamond$

```
$k \xleftarrow{\$} \{0,1\}^\lambda$
LOOKUP($x \in \{0,1\}^{\texttt{in}}$):
    return $F(k, x)$
```

Def PRF $\approx$

$\mathcal{L}_1$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda$
    $x :=$ LOOKUP($r$) $\oplus m_L$
    return $(r, x)$
```

$\diamond$

$\mathcal{L}^F_{\text{prf-rand}}$

```
$T :=$ empty assoc. array
LOOKUP($x \in \{0,1\}^{\texttt{in}}$):
    if $T[x]$ undefined:
        $T[x] \xleftarrow{\$} \{0,1\}^{\texttt{out}}$
    return $T[x]$
```

Inline $\equiv$

$\mathcal{L}_2$

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

```
    return $(r, x)$
```

```
    $T[x] \xleftarrow{\$} \{0,1\}^{\mathsf{out}}$
  return $T[x]$
```

Inline $\equiv$

```
$\mathcal{L}_2$
$T :=$ empty assoc. array
EAVESDROP$(m_L, m_R \in \mathcal{M})$:
  $r \xleftarrow{\$} \{0,1\}^{\lambda}$
  if $T[r]$ undefined:
    $T[r] \xleftarrow{\$} \{0,1\}^{\mathsf{out}}$
  $x := T[r] \oplus m_L$
  return $(r, x)$
```

Externalize $\equiv$

```
$\mathcal{L}_2$
$T :=$ empty assoc. array
EAVESDROP$(m_L, m_R \in \mathcal{M})$:
  $r \xleftarrow{\$}$ SAMP()
  if $T[r]$ undefined:
    $T[r] \xleftarrow{\$} \{0,1\}^{\mathsf{out}}$
  $x := T[r] \oplus m_L$
  return $(r, x)$
```

$\diamond$

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

Externalize $\equiv$

$\mathcal{L}_2$

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

$\diamond$

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

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

Asymptotic birthday paradox $\approx$

$\mathcal{L}_2$

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

$\diamond$

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

```
$R := \emptyset$
SAMP():
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    return $r$
```

$\mathcal{L}_3$

$\mathcal{L}_2$

Asymptotic birthday paradox $\approx$

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

$\diamond$

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

```
$R := \emptyset$
SAMP ():
    $r \xleftarrow{\$} \{0,1\}^{\lambda} \setminus R$
    $R = R \cup \{r\}$
    return $r$
```

$\mathcal{L}_3$

Inline $\equiv$

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

$\mathcal{L}_4$

Inline $\equiv$

$\mathcal{L}_3$

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

$T[r]$ always undefined $\equiv$

$\mathcal{L}_4$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^{\lambda} \setminus R$
    $R = R \cup \{r\}$
    $r' \xleftarrow{\$} \{0,1\}^{\mathtt{out}}$
    $x := r' \oplus m_L$
    return $(r, x)$
```

$\mathcal{L}_5$

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

$\mathcal{L}_4$

```
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 $\equiv$

$\mathcal{L}_5$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^{\lambda} \setminus R$
    $R = R \cup \{r\}$
    $x \leftarrow$ OTENC($m_L$)
    return $(r, x)$
```

Externalize $\equiv$

$\diamond$

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

```
OTENC($m \in \{0,1\}^{\lambda}$):
    $k \xleftarrow{\$} \{0,1\}^{\lambda}$
    return $k \oplus m$
```

$\mathcal{L}_5$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    $r' \xleftarrow{\$} \{0,1\}^{\mathtt{out}}$
    $x := r' \oplus m_L$
    return $(r, x)$
```

T[r] always undefined $\equiv$

$\mathcal{L}_5$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    $x \leftarrow$ OTENC($m_L$)
    return $(r, x)$
```

$\diamond$

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

```
OTENC($m \in \{0,1\}^\lambda$):
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    return $k \oplus m$
```

Externalize $\equiv$

$\mathcal{L}_5$

```
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    $x \leftarrow$ OTENC($m_L$)
    return $(r, x)$
```

$\diamond$

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

```
OTENC($m \in \{0,1\}^\lambda$):
    $c \xleftarrow{\$} \{0,1\}^\lambda$
    return $c$
```

OTP uniform ciphertext $\equiv$

```
    $x \leftarrow \text{OTENC}(m_L)$
    return $(r, x)$
```

```
    return $k \oplus m$
```

OTP uniform ciphertext $\equiv$

```
$\mathcal{L}_5$
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    $x \leftarrow \text{OTENC}(m_L)$
    return $(r, x)$
```

$\diamond$

```
$\mathcal{L}_{\text{otp-rand}}$
OTENC($m \in \{0,1\}^\lambda$):
    $c \xleftarrow{\$} \{0,1\}^\lambda$
    return $c$
```

Inline $\equiv$

```
$\mathcal{L}_5$
EAVESDROP($m_L, m_R \in \mathcal{M}$):
    $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
    $R = R \cup \{r\}$
    $x \xleftarrow{\$} \{0,1\}^\lambda$
    return $(r, x)$
```

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

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.
  $\Rightarrow$ Issue: remember, Enc is a tupple (r, x), i.e. for l chunks, overhead of λl
  Too inefficient! 😰

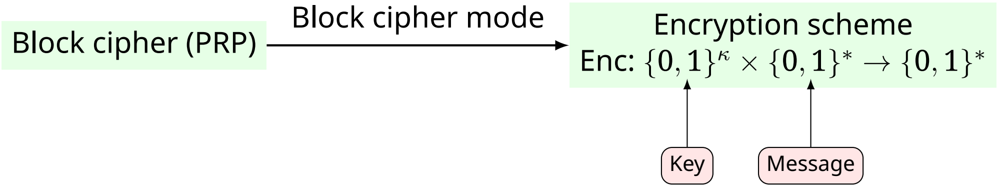
Limitations PRF pseudo-OTP

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.
  $\Rightarrow$ Issue: remember, Enc is a tupple (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):



Block cipher (PRP)

Block cipher mode

Encryption scheme

Enc: $\{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^*$

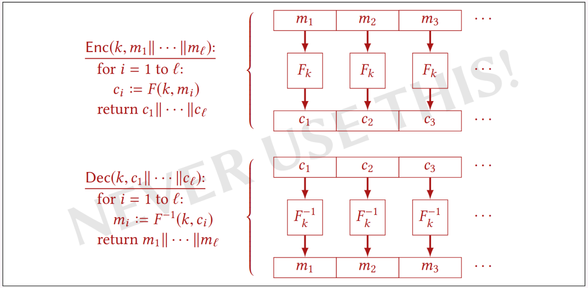
Key

Message

Common modes

Definition (ECB mode: NEVER USE THIS)

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



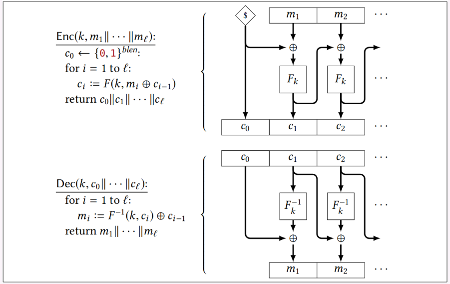
```
$\underline{\mathrm{Enc}(k, m_1 \| \cdots \| m_\ell)}$:
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i)$
  return $c_1 \| \cdots \| c_\ell$

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

? 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:



```
$\text{Enc}(k, m_1 \| \cdots \| m_\ell)$:
  $c_0 \leftarrow \{0,1\}^{blen}$;
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  return $c_0 \| c_1 \| \cdots \| c_\ell$

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

c_0 is called the initialization vector (IV). Why can't we set it to a fixed value?

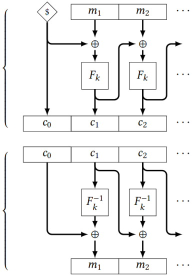
- Ⓐ It acts like a OTP key on the message, hence hides it
- Ⓑ Used to have a non-deterministic encryption

Common modes

Definition (CBC mode)

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

```
$\text{Enc}(k, m_1\|\cdots\|m_\ell)$:
  $c_0 \leftarrow \{0,1\}^{blen}$;
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  return $c_0\|c_1\|\cdots\|c_\ell$
```



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

?

c_0 is called the initialization vector (IV). Why can't we set it to a fixed value?

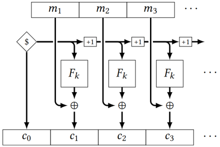
- Ⓐ It acts like a OTP key on the message, hence hides it ✗ IV is public, so cannot be a OTP key!
- Ⓑ Used to have a non-deterministic encryption ✓

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:

```
$\mathsf{Enc}(k, m_1 \| \cdots \| m_\ell)$:
    $r \leftarrow \{0,1\}^{blen}$
    $c_0 := r$
    for $i = 1$ to $\ell$:
        $c_i := F(k, r) \oplus m_i$
        $r := r + 1 \% 2^{blen}$
    return $c_0 \| \cdots \| c_\ell$
```



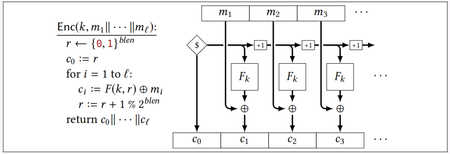
Try to find the decryption algorithm. Do you need to compute F^{-1}?

- Ⓐ Yes
- Ⓑ No

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:



```
$\mathsf{Enc}(k, m_1 \| \cdots \| m_\ell)$:
    $r \leftarrow \{0, 1\}^{blen}$
    $c_0 := r$
    for $i = 1$ to $\ell$:
        $c_i := F(k, r) \oplus m_i$
        $r := r + 1 \% 2^{blen}$
    return $c_0 \| \cdots \| c_\ell$
```

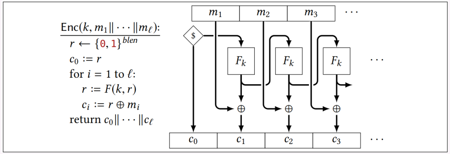
Try to find the decryption algorithm. Do you need to compute F^{-1}?

- Ⓐ Yes ✗
- Ⓑ 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:



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

?

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

Ⓐ Yes

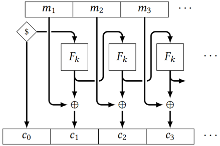
Ⓑ No

Common modes

Definition (OFB mode)

The output feedback (OFB) mode is defined as:

```
$\mathsf{Enc}(k, m_1 \| \cdots \| m_\ell)$:
    $r \leftarrow \{0,1\}^{blen}$
    $c_0 := r$
    for $i = 1$ to $\ell$:
        $r := F(k, r)$
        $c_i := r \oplus m_i$
    return $c_0 \| \cdots \| c_\ell$
```



?

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

- Ⓐ Yes ✗
- Ⓑ 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?

Ⓐ Yes

Ⓑ No

Comparison of modes

?

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

Ⓐ Yes ✗

Ⓑ 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 ($\text{blen} = 128$) and blowfish (warning: $\text{blen} = 64$!)
- 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 $= \lambda$ for simplicity)?

Ⓐ No

Ⓑ Yes, with

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

Ⓒ Yes, with

```
$\mathcal{A}$
$c :=$ EAVESDROP$(0^\lambda, 0^{2\lambda})$
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 $= \lambda$ for simplicity)?

Ⓐ No ✗

Ⓑ Yes, with

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

✗

Ⓒ Yes, with

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

✓ 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:

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

$\approx$

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

IND-CPA for variable-length plaintexts

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 how many zeros to remove?
- ANSI X.923 standard: add `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 `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`...`0`): if you pad a message m of size $k$blen into m', what is the size of m'?

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

Padding

?

Consider ISO/IEC 7816-4 standard (append $\texttt{10}\ldots\texttt{0}$): if you pad a message m of size $k$blen into m', what is the size of m'?

Ⓐ $k$blen ✗

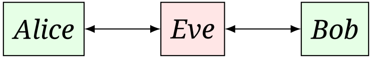
Ⓑ $k$blen $+ 1$ ✗

Ⓒ $(k+1)$blen ✓ (like all paddings, it increases the size of the message)

Padding oracle attack & CCA security

Padding oracle attack

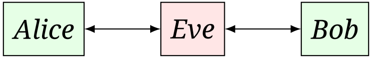
Before: passive adversary (somewhat unrealistic). Now, we consider **active** adversaries:



What happens if Bob returns an error if the padding is incorrect?

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!

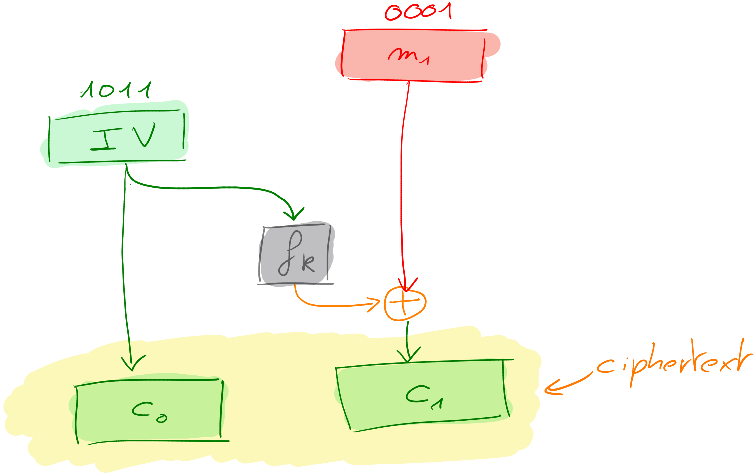
Padding oracle attack (illustrated on board)

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

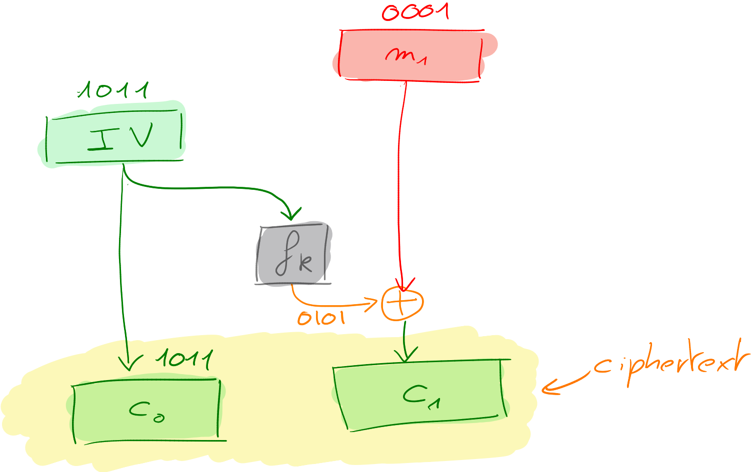
```
$k \leftarrow \mathsf{Gen}(1^\lambda)$
PADDINGORACLE($c$):
    $m := \mathsf{Dec}(k, c)$
    return VALIDPAD($m$)
```

(hence 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 $c_0 \leftarrow \mathsf{Enc}_k(m_0)$ (where m_0 is already padded), goal is to find m_0.

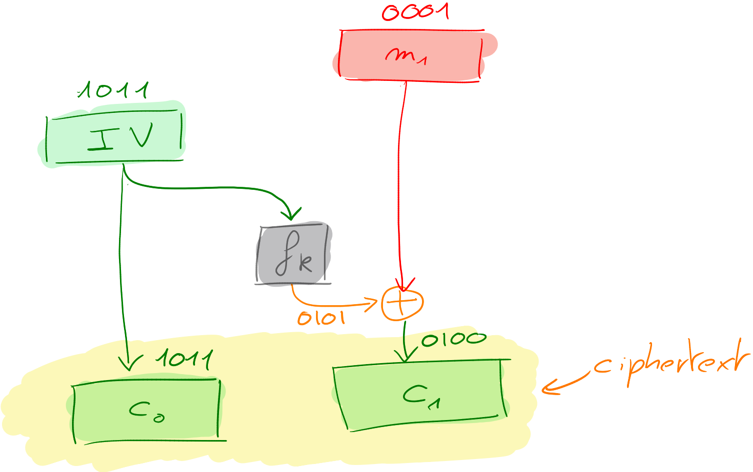
Padding oracle attack



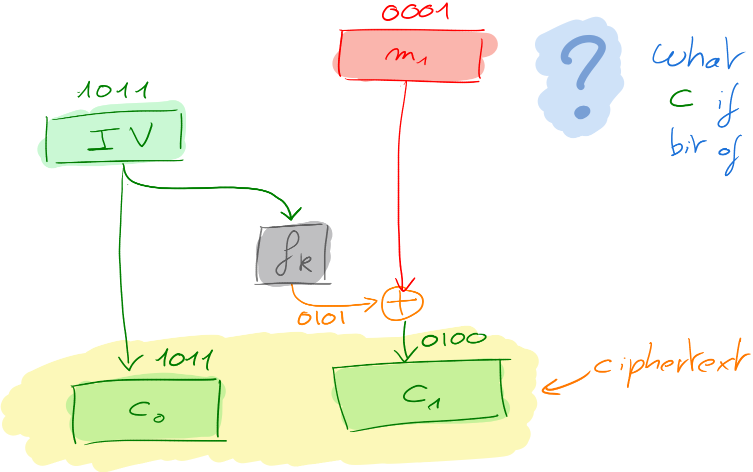
Padding oracle attack



Padding oracle attack

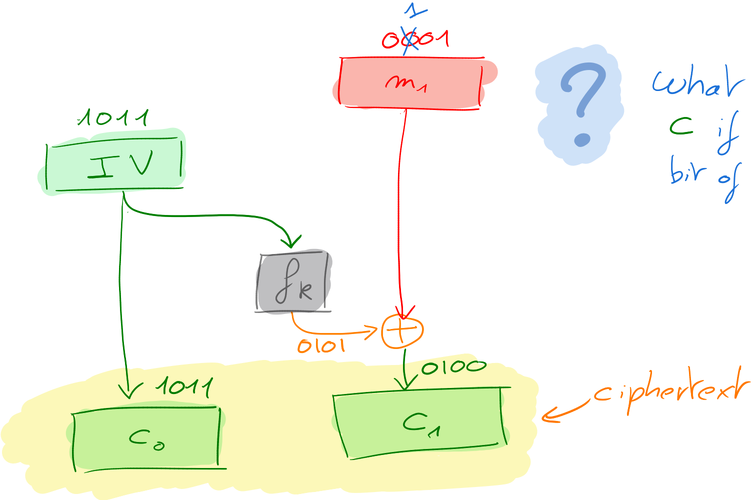


Padding oracle attack



What happens to c if we flip one bit of m ?

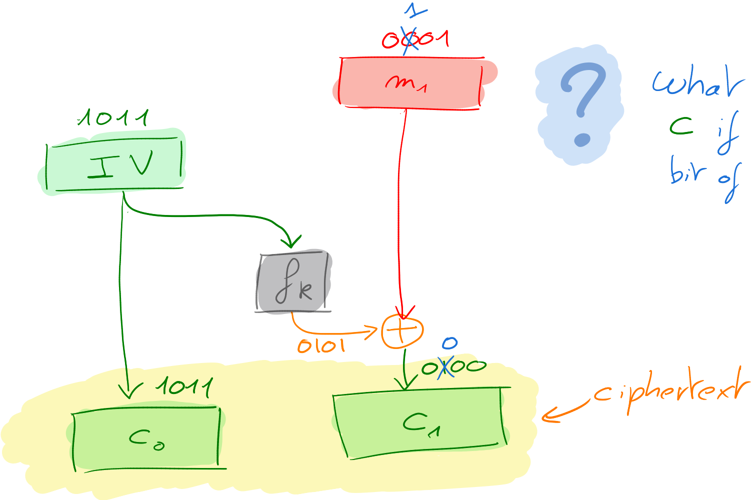
Padding oracle attack



?

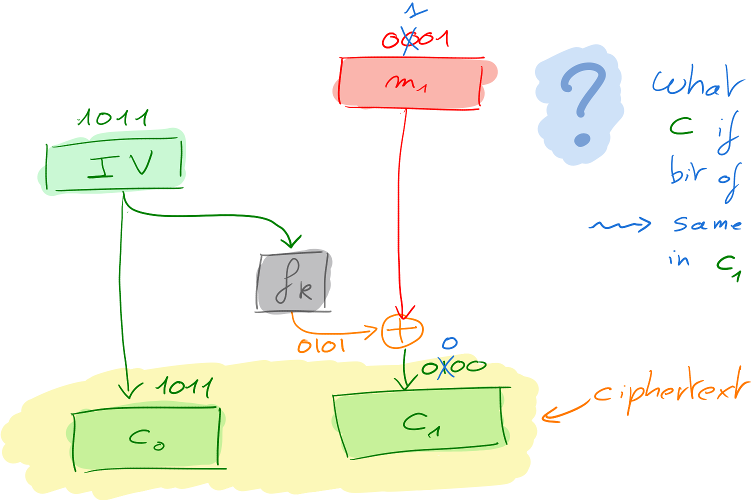
What happens to c if we flip one bit of m ?

Padding oracle attack



what happens to c if we flip one bit of m ?

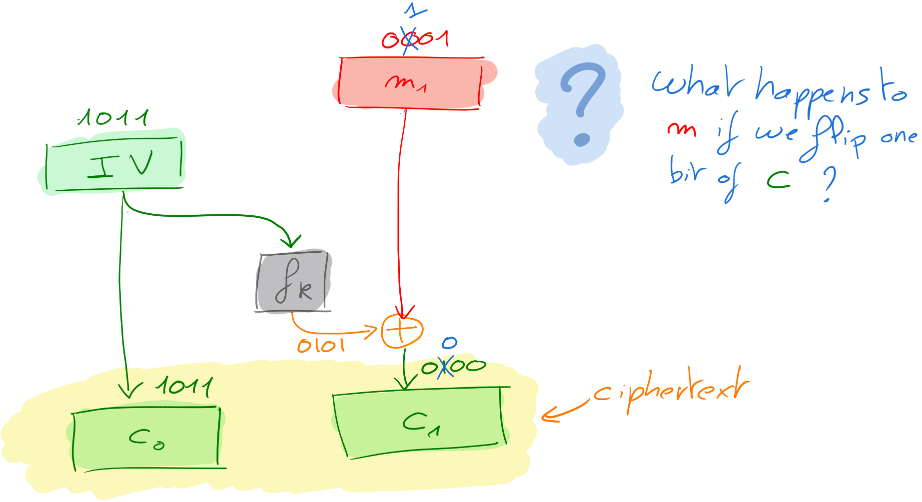
Padding oracle attack



What happens to c if we flip one bit of m ?

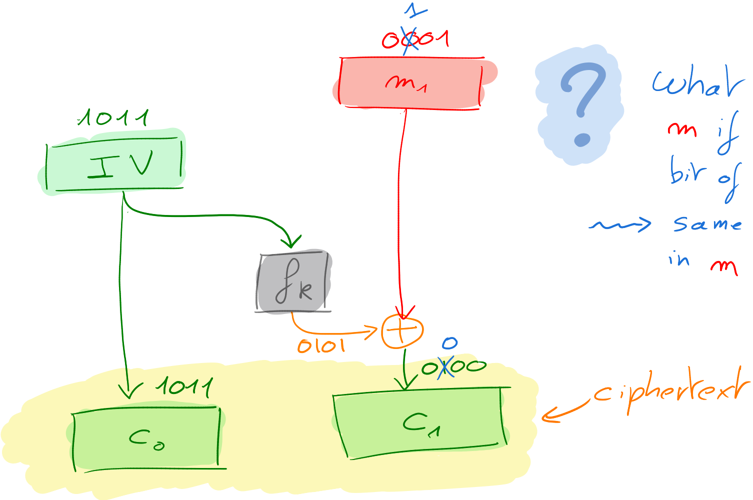
⇝ same bit is flipped in c_1 !!!

Padding oracle attack



What happens to m if we flip one bit of c ?

Padding oracle attack



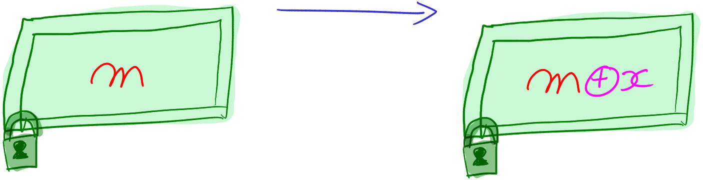
What happens to m if we flip one bit of c ?

⇝ same bit is flipped in m !!!

Padding oracle attack

$\forall x$, Attacker can turn:

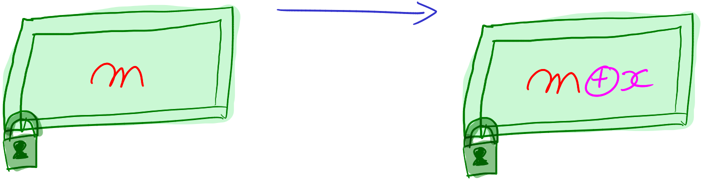
$\mathrm{Enc}_k(m)$ $\mathrm{Enc}_k(m \oplus x)$



Padding oracle attack

$\forall x$, Attacker can turn:

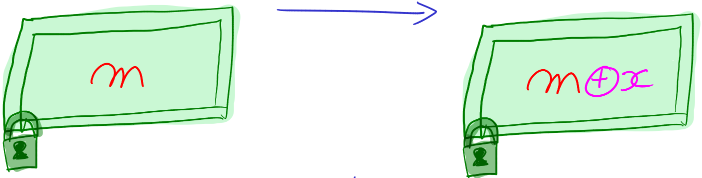
$$\mathrm{Enc}_k(m) \longrightarrow \overbrace{\mathrm{Enc}_k(m \oplus x)}^{= \mathrm{Enc}_k(m) \oplus (0-0 \| x)}$$



Padding oracle attack

$\forall x$, Attacker can turn:

$Enc_k(m) \longrightarrow \overbrace{Enc_k(m \oplus x)}^{= Enc_k(m) \oplus (0\text{-}0 \| x)}$

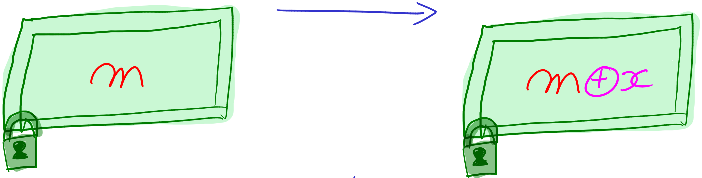


Idea: learn info on m by checking if $m \oplus x$ is well padded!

Padding oracle attack

$\forall x$, Attacker can turn:

$\mathrm{Enc}_k(m)$ → $\overbrace{\mathrm{Enc}_k(m \oplus x)}^{\mathrm{Enc}_k(m) \oplus (0-0 \| x)}$



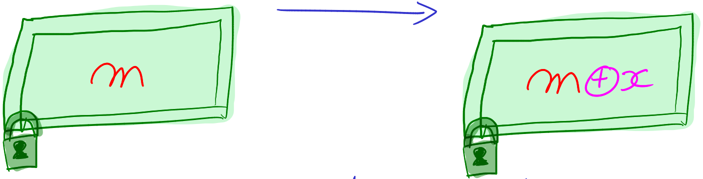
<u>Idea</u>: learn info on m by checking if $m \oplus x$ is well padded!

How?

Padding oracle attack

$\forall x$, Attacker can turn:

$\text{Enc}_k(m)$ $\longrightarrow$ $\overbrace{\text{Enc}_k(m \oplus x)}^{\text{Enc}_k(m) \oplus (0-0 \| x)}$

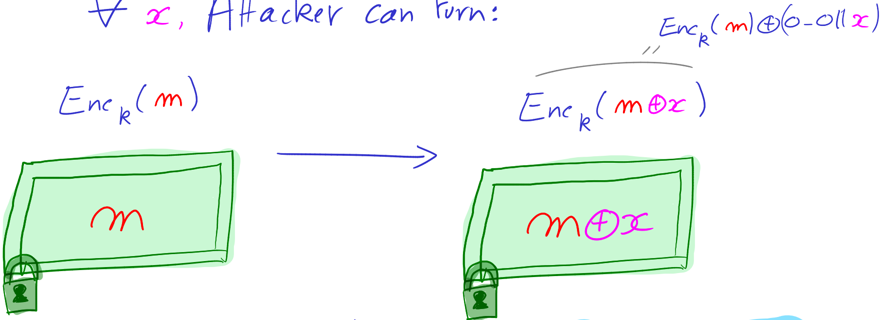


Idea: learn info on m by checking if $m \oplus x$ is well padded!

How? Given $\text{Enc}_k(m)$ send $\overbrace{\text{Enc}_k(m) \oplus (0-0\|x)}^{= \text{Enc}_k(m \oplus x)}$ to the padding oracle

Padding oracle attack

$\forall x$, Attacker can turn:



Idea: learn info on m by checking if $m \oplus x$ is well padded!

How? Given $Enc_k(m)$ send $\overbrace{Enc_k(m) \oplus (0-0 \| x)}^{= \, Enc_k(m \oplus x)}$ to the padding oracle

Padding oracle attack

By checking if $\underset{\text{unknown}}{m} \oplus \underset{\text{chosen by adversary}}{x}$ is well padded, how to learn m?

- Is m well padded?

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? ⟹ Yes (honest Alice)

E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{1 \text{ byte}}$

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? ⟹ Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? $\Longrightarrow$ Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

  $= 65|66|67|0|0|0|1|5$

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? $\Longrightarrow$ Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

  $= 65|66|67|0|0|0|1|5$ ⇒ X NO!

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? ⟹ Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

  $= 65|66|67|0|0|0|1|5$ ⟹ ✗ NO!

- Is $m \oplus 0|0|1|0|0|0|0|0$ well padded?

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? => Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

  $= 65|66|67|0|0|0|1|5$ => X NO!

- Is $m \oplus 0|0|1|0|0|0|0|0$ well padded?

  $= 65|66|66|0|0|0|0|5$ => ✓ Yes!

Padding oracle attack

By checking if $m \oplus x$ is well padded, how to learn m?

- Is m well padded? $\Rightarrow$ Yes (honest Alice)

  E.g. $m = 65|66|67|0|0|0|0|\underbrace{5}_{\text{1 byte}}$

- Is $m \oplus 0|0|0|0|0|0|1|0$ well padded?

  $= 65|66|67|0|0|0|1|5$ $\Rightarrow$ ✗ No!

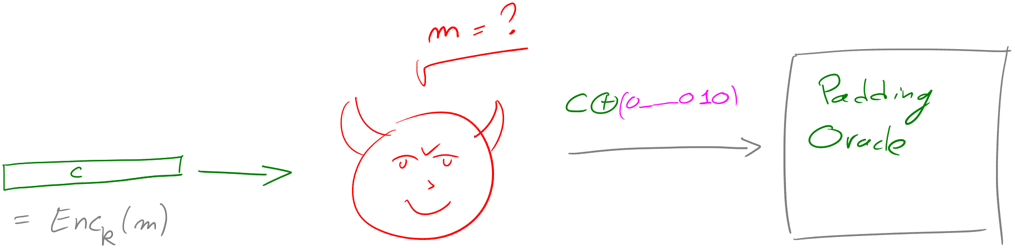
- Is $m \oplus 0|0|1|0|0|0|0|0$ well padded?

  $= 65|66|66|0|0|0|0|5$ $\Rightarrow$ ✓ Yes!

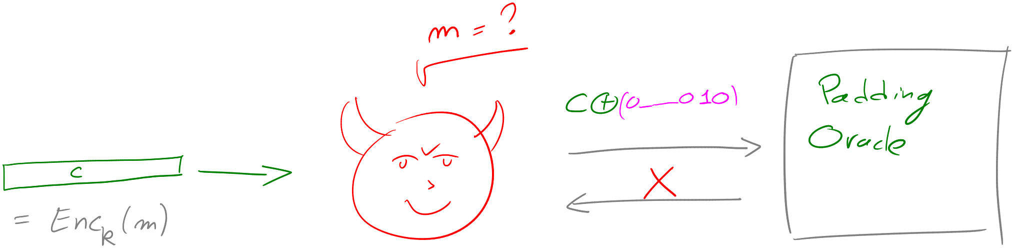
$\Rightarrow$ allows us to recover the length of m

Padding oracle attack

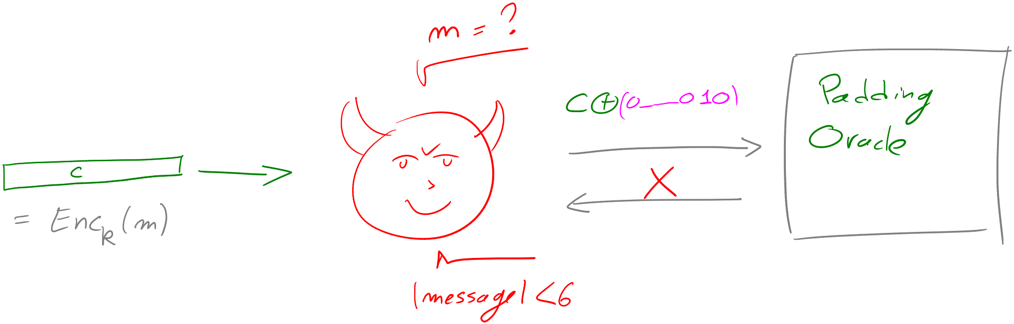
Padding oracle attack



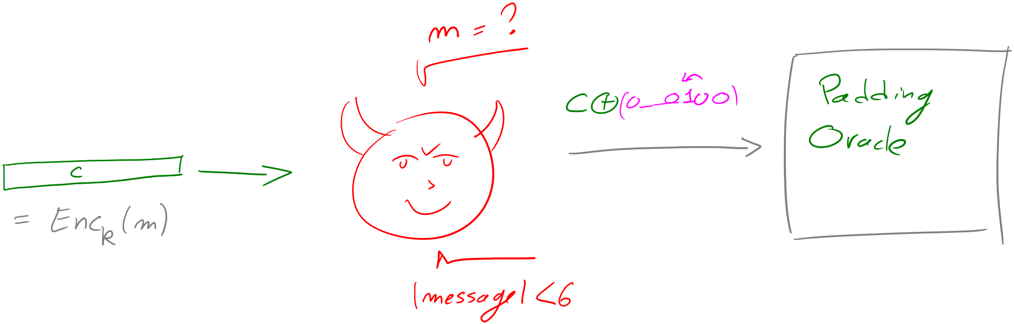
Padding oracle attack



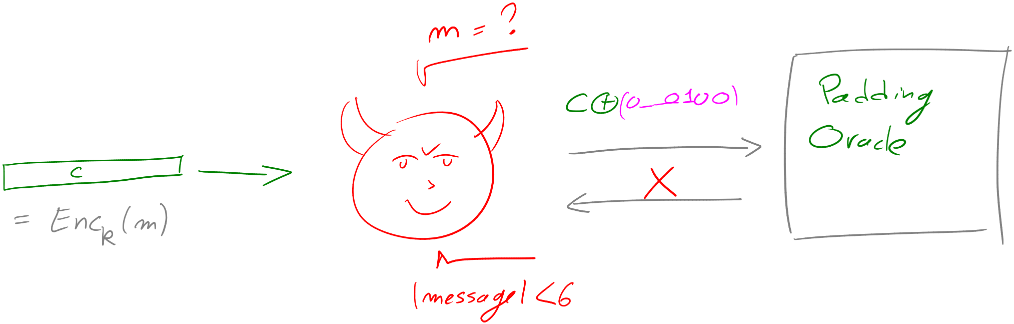
Padding oracle attack



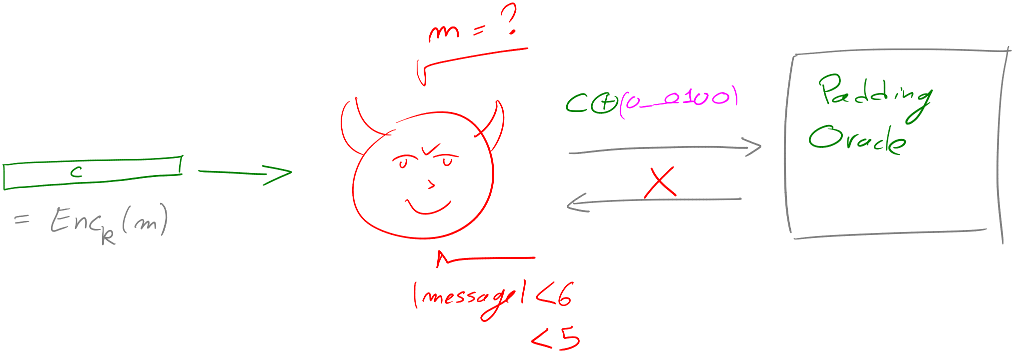
Padding oracle attack



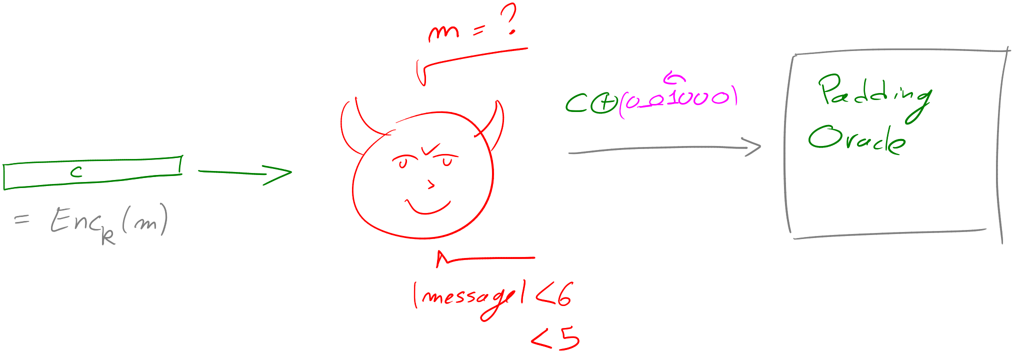
Padding oracle attack



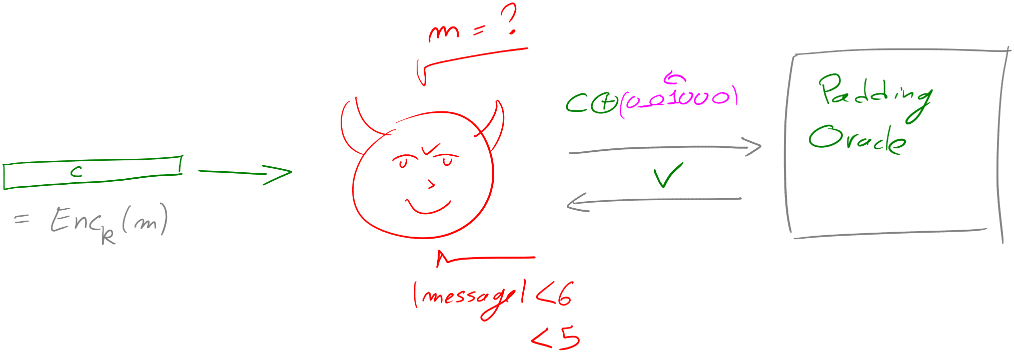
Padding oracle attack



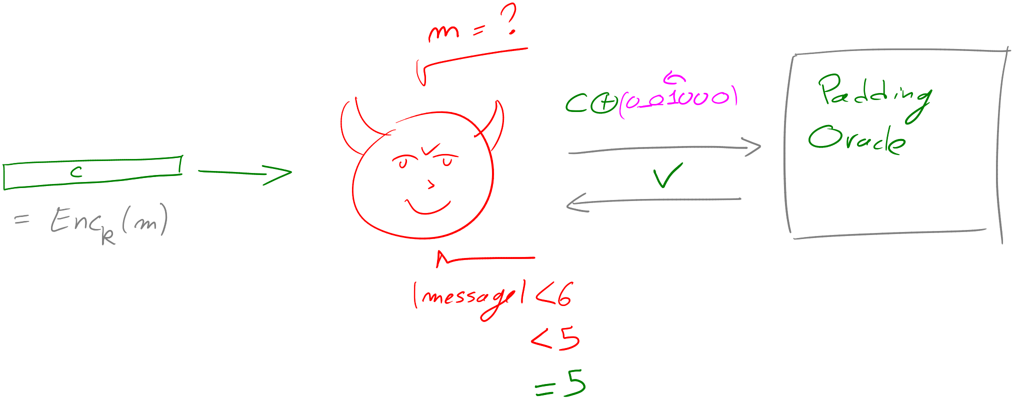
Padding oracle attack



Padding oracle attack



Padding oracle attack



Padding oracle attack

How to learn the rest of m ?

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|\text{message}|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= 0$?

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|\text{message}|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= 0$?

↳ $m \oplus 0|0|0|0|0|0|0|3 \oplus 4$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= 0$?

↳ $m \oplus 0|0|0|0|0|0|0|3\oplus4$

$= ?|?|?|?|?|0|0|4$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|\text{message}|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is = 0 ?

↳ $m \oplus 0|0|0|0|0|0|0|3 \oplus 4$

$= ?|?|?|?|?|0|0|4$

↳ if valid padding, ? = 0, else ? ≠ 0 !

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows |message| (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|0|0|0|3 \oplus 4$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|\text{message}|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if $?$ is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|\underset{x}{\cancel{0}}|0|0|3 \oplus 4$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|0|\underset{x}{\cancel{0}}|0|0|3 \oplus 4$

$= ?|?|?|?|? \oplus x|0|0|4$

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if $?$ is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|\underset{x}{\cancel{0}}|0|0|3 \oplus 4$

$= ?|?|?|?|? \oplus x|0|0|4$

↳ if valid padding, $? \oplus x = 0$, i.e. $? = x$!

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if $?$ is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|\underset{x}{\cancel{0}}|0|0|3\oplus4$

$= ?|?|?|?|?\oplus x|0|0|4$

↳ if valid padding, $? \oplus x = 0$, i.e. $? = x$!

Padding oracle attack

How to learn the rest of m ?

↳ attacker knows $|message|$ (e.g. 5)

so $m = ?|?|?|?|?|0|0|3$

How to check if ? is $= \overset{x}{\cancel{0}}$?

↳ $m \oplus 0|0|0|0|\underset{x}{\cancel{0}}|0|0|3 \oplus 4$

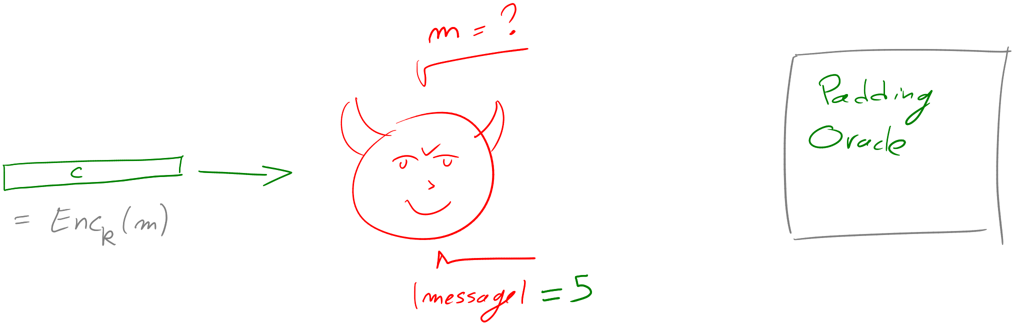
$= ?|?|?|?|? \oplus x|0|0|4$

↳ if valid padding, $? \oplus x = 0$, i.e. $? = x$

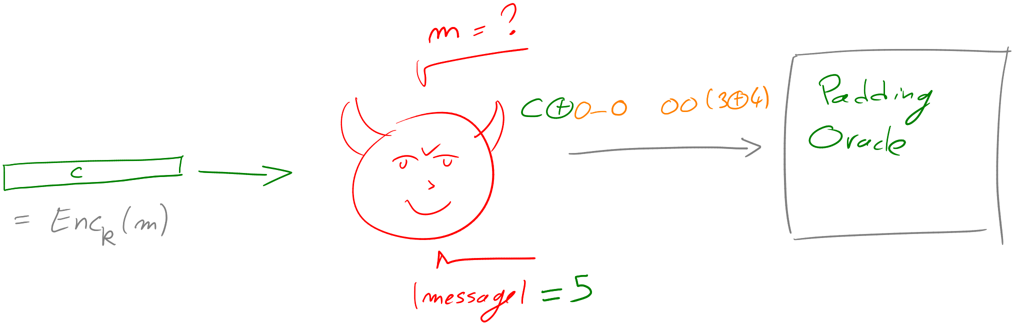
I just loop for all x !

!

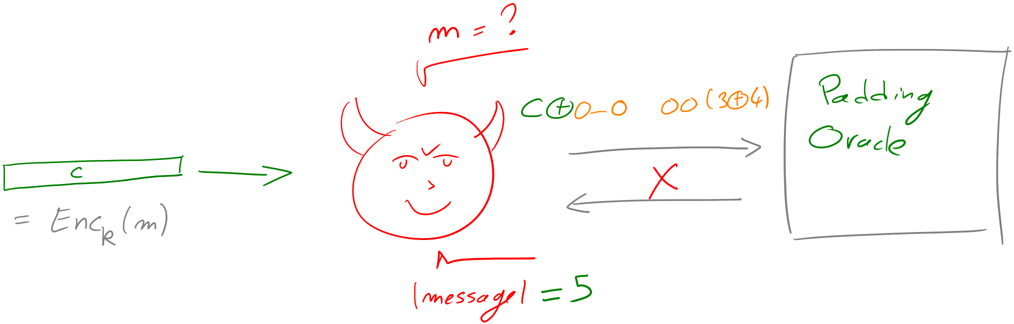
Padding oracle attack



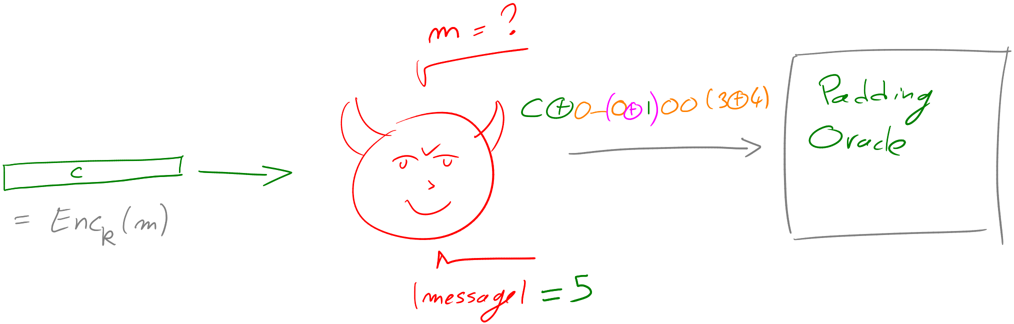
Padding oracle attack



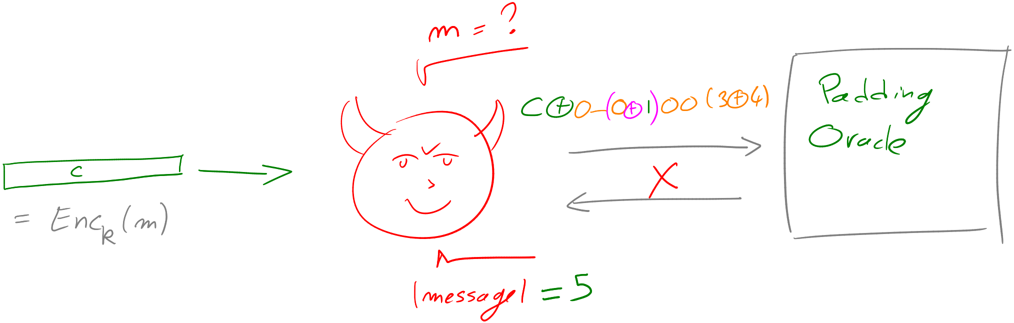
Padding oracle attack



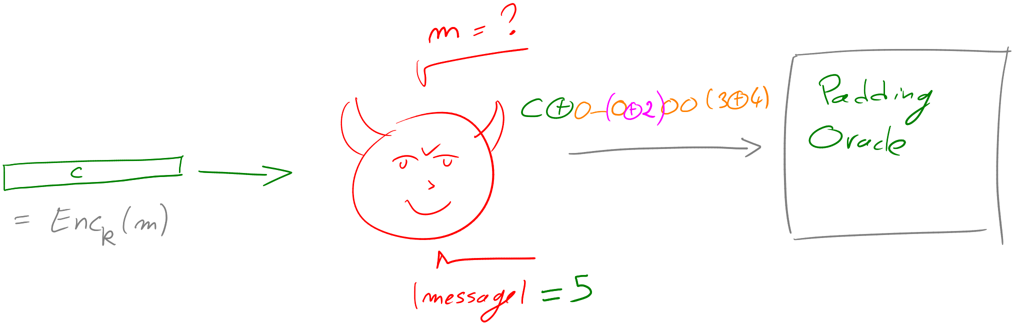
Padding oracle attack



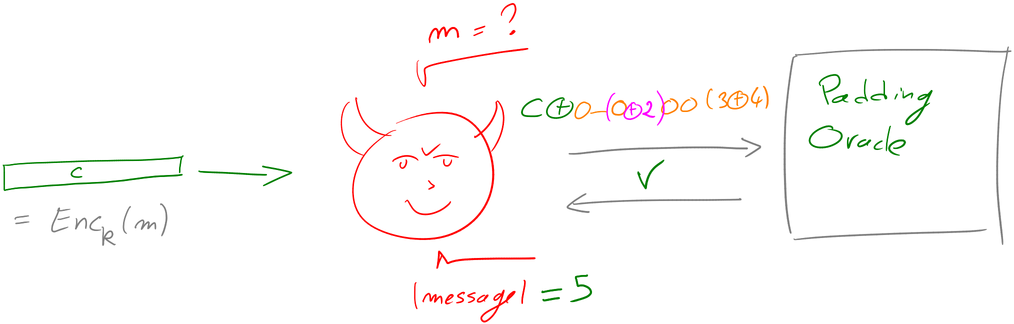
Padding oracle attack



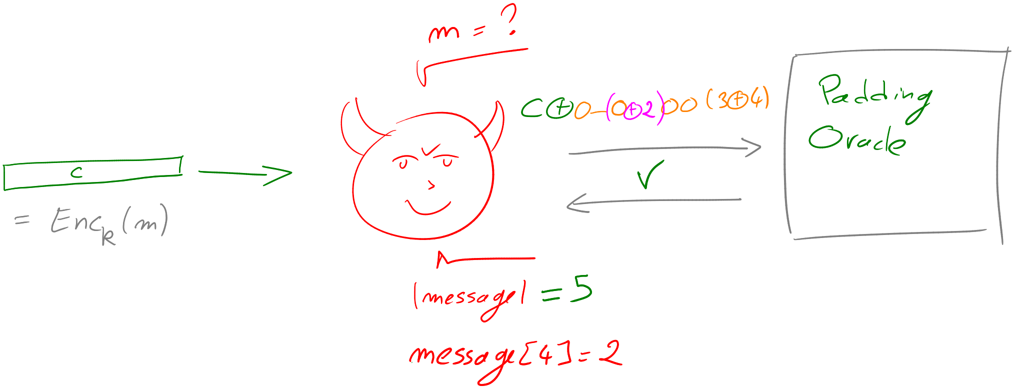
Padding oracle attack



Padding oracle attack



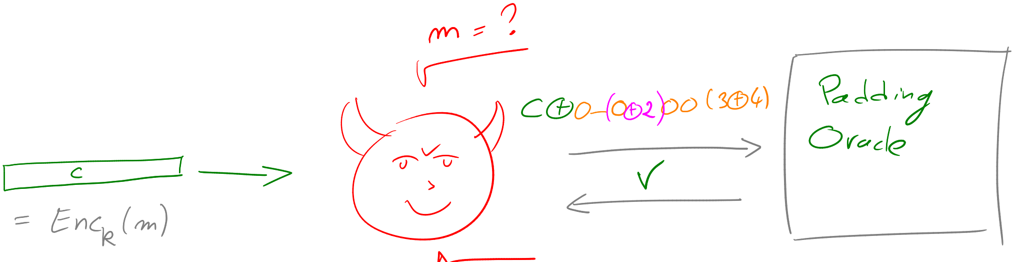
Padding oracle attack



$|message| = 5$

$message[4] = 2$

Padding oracle attack

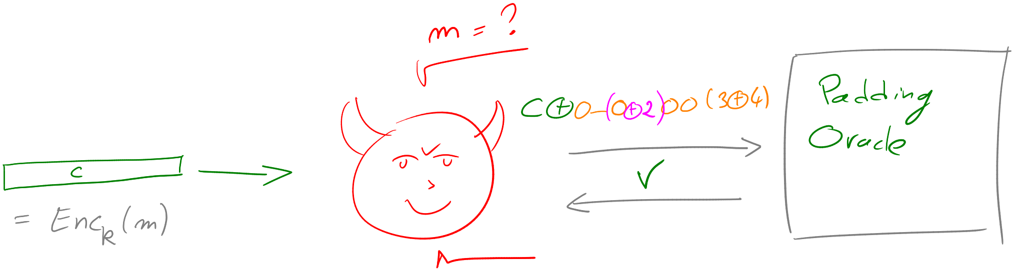


$|\text{message}| = 5$

$\text{message}[4] = 2$

and $c \oplus 0{-}(0 \oplus 2)00\,(3 \oplus 4) = \mathrm{Enc}(????\,000\,4)$

Padding oracle attack



$|message| = 5$

$message[4] = 2$

and $c \oplus 0-(0 \oplus 2)00(3 \oplus 4) = Enc(????0004)$

=> Repeat to extract all bits starting from ↰

Padding oracle attack

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…)

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

IND-CCA security

IND-CCA

IND-CCA

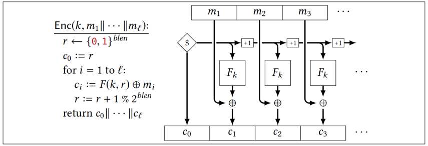
Let Σ be an encryption scheme. We say that Σ has indistinguishable security against **chosen-ciphertext attacks (IND-CCA)** if:

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

$$\approx$$

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

Malleability



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

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

- Ⓐ No
- Ⓑ Yes, withOui, avec

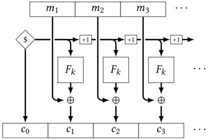
```
$\mathcal{A}$
$(c_0, c_1) \leftarrow$ EAVESDROP($0^{blen}, 1^{blen}$)
$m \leftarrow$ DECRYPT($(c_0, c_1 \oplus (10\ldots0))$)
return $m \overset{?}{=} 10\ldots0$
```

- Ⓒ Yes, with

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

Malleability

```
$\mathrm{Enc}(k, m_1\|\cdots\|m_\ell)$:
    $r \leftarrow \{0,1\}^{blen}$
    $c_0 := r$
    for $i = 1$ to $\ell$:
        $c_i := F(k,r) \oplus m_i$
        $r := r + 1 \% 2^{blen}$
    return $c_0\|\cdots\|c_\ell$
```



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

- Ⓐ No ✗
- Ⓑ Yes, withOui, avec

```
$\mathcal{A}$
$(c_0, c_1) \leftarrow$ EAVESDROP$(0^{\mathsf{blen}}, 1^{\mathsf{blen}})$
$m \leftarrow$ DECRYPT$((c_0, c_1 \oplus (10\ldots0)))$
return $m \overset{?}{=} 10\ldots0$
```

✓

- Ⓒ Yes, with

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

✗

Malleability

Fundamental reason: CTR is **malleable**, i.e. we can obtain $\mathsf{Enc}_k(x') = (c_0, x' \oplus F_k(c_0))$ from $\mathsf{Enc}_k(x) = (c_0, x \oplus F_k(c_0))$ (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!