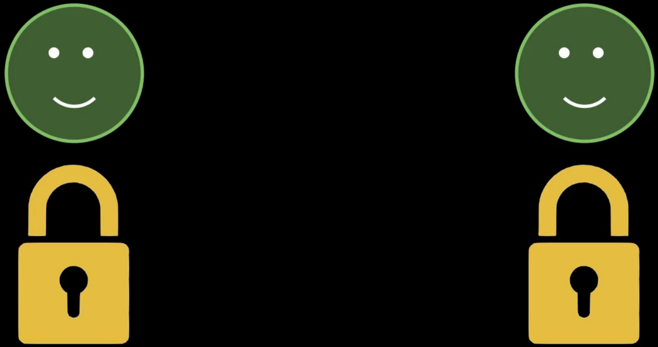
Crypto Engineering 2024
Symmetric cryptography

Léo Colisson Palais

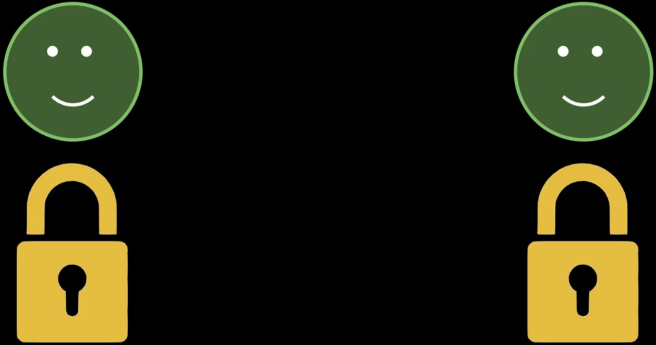
leo.colisson-palais@univ-grenoble-alpes.fr

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

Reminder symmetric encryption & IND-CPA security

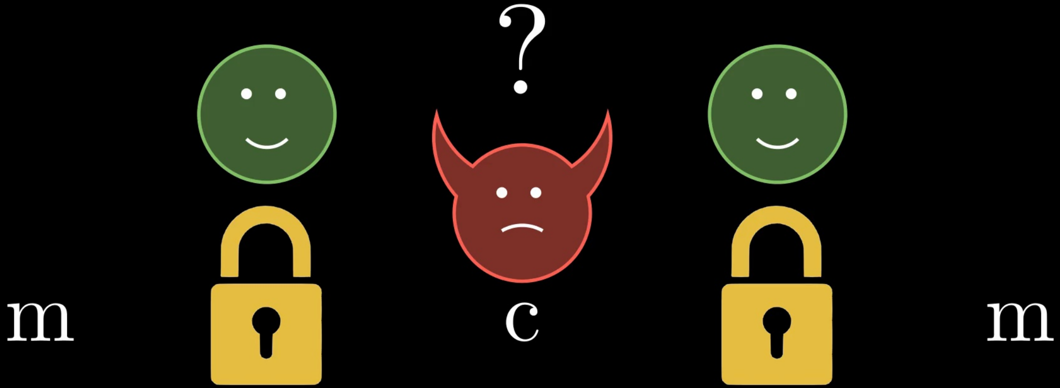


m




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 $\mathcal{M} = \mathcal{K} = \mathcal{K} = \{0,1\}^\lambda$ and $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as:

OTP

$\underline{\mathsf{Gen}(1^\lambda)}$:
  $k \xleftarrow{\$} \{0, 1\}^\lambda$
  return k

$\underline{\mathsf{Enc}(k, m)}$:
  return $k \oplus m$

$\underline{\mathsf{Dec}(k, c)}$:
  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: hard to find a good notion of security, but it seems like the adversary should choose two messages m_0 and m_1, and tell if they obtained $\mathsf{Enc}_k(m_0)$ or $\mathsf{Enc}_k(m_1)$. Still an important question:

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

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:

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots-L}} \\ \hline \underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\ \quad k \leftarrow \mathsf{Gen}(1^\lambda) \\ \quad \text{return } \mathsf{Enc}_k(m_L) \end{array}} \equiv \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots-R}} \\ \hline \underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\ \quad k \leftarrow \mathsf{Gen}(1^\lambda) \\ \quad \text{return } \mathsf{Enc}_k(m_R) \end{array}}$$

Security of OTP

Theorem

OTP is one-time secure

Proof

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$:

```
EAVESDROP(m_L, m_R ∈ M):
  k ← Gen(1^λ)
  return Enc_k(m_L)
```

$\equiv$ Def. OTP

```
EAVESDROP(m_L, m_R ∈ M):
  k ←$ {0,1}^λ
  return k ⊕ m_L
```

$\equiv$ Externalize code

```
EAVESDROP(m_L, m_R ∈ M):
  return OTENC(m_L)
```

$\diamond$

```
OTENC(m):
  k ←$ {0,1}^λ
  return k ⊕ m
```

$\equiv$ Thm OTP uniform ciphertext from first lecture

```
EAVESDROP(m_L, m_R ∈ M):
  return OTENC(m_L)
```

$\diamond$

```
OTENC(m):
  k ←$ {0,1}^λ
  return k
```

$\equiv$ Inline subroutine

```
EAVESDROP(m_L, m_R ∈ M):
  k ←$ {0,1}^λ
  return k
```

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

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:

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

- (A) Yes
- (B) No

Security of OTP

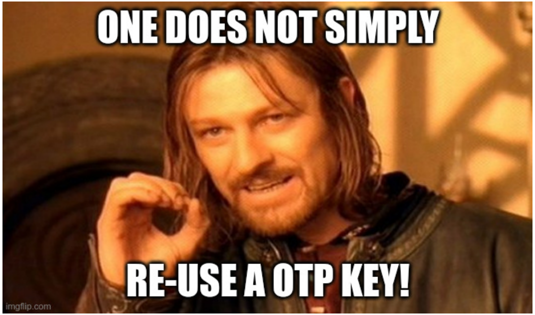
Do you think that OTP is CPA secure? If yes, sketch a proof, if not, sketch an adversary and compute its advantage.

- (A) Yes
- (B) No Exploit the fact that it is **deterministic encryption**:

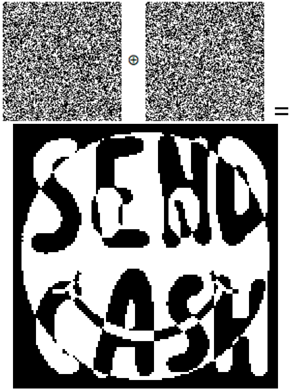
Define $\boxed{\begin{array}{l} \hfill\mathcal{A}\hfill \\ \hline x \leftarrow \text{EAVESDROP}(0^\lambda, 0^\lambda) \\ y \leftarrow \text{EAVESDROP}(0^\lambda, 1^\lambda) \\ \text{return } x = y \end{array}}$. Then, after inlining, we have

$$\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}} = \boxed{\begin{array}{l} \hfill\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}}\hfill \\ \hline k \leftarrow \text{Gen}(1^\lambda) \\ x \leftarrow 0^\lambda \oplus k \\ y \leftarrow 0^\lambda \oplus k \\ \text{return } x = y \end{array}}$$ i.e. $\Pr\left[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}} = 1\right] = 1$. But

$$\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}} = \boxed{\begin{array}{l} \hfill\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-R}}\hfill \\ \hline k \leftarrow \text{Gen}(1^\lambda) \\ x \leftarrow 0^\lambda \oplus k \\ y \leftarrow 1^\lambda \oplus k \\ \text{return } x = y \end{array}}$$ i.e. $\Pr\left[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{cpa-L}} = 1\right] = 0$. $\text{Adv} = 1 - 0 = 1 \neq \text{negl}(\lambda)$

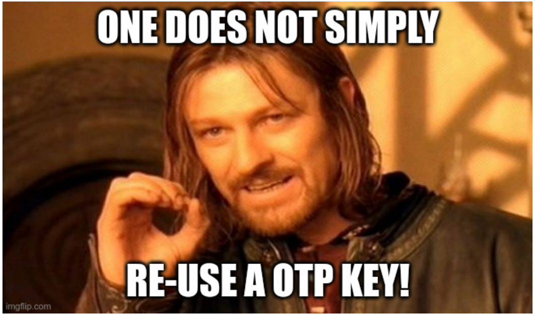


Never reuse a OTP key!!! This can lead to real attack:



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

Security of OTP

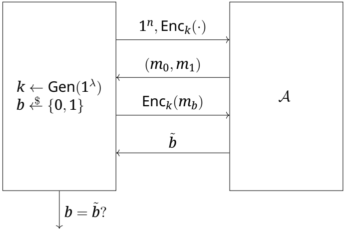


Never reuse a OTP key!!! This can lead to real attack:

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

Various definitions of IND-CPA

You might see this other **equivalent** definition of IND-CPA:



vs

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

$k \leftarrow \mathsf{Gen}(1^\lambda)$
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 $\mathsf{Enc}_k(m_R)$

- Instead of b, when $b = 0$ we play $\mathcal{L}^{\Sigma}_{\text{cpa-L}}$ otherwise $\mathcal{L}^{\Sigma}_{\text{cpa-R}}$.
- in our definition, no access to oracle $\mathsf{Enc}_k(\cdot)$, but we can **simulate it** by calling EAVESDROP(m, m) (same message twice).
- in our definition, no restriction on the number of allowed calls to EAVESDROP (= stronger notion, while in the other we have a single message $Enc_k(m_b)$). But equivalent (advantage is multiplied by the maximum number of queries done by $\mathcal{A}$, but still negligible): proof via a sequence of **hybrids on the number of queries** (cf. exercice).

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
$\Rightarrow$ longer randomness

Pseudo-random function

PRF

short randomness
$\Rightarrow$ random looking function

Pseudo-random permutation
= block-cipher

PRP

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

AES

More advanced

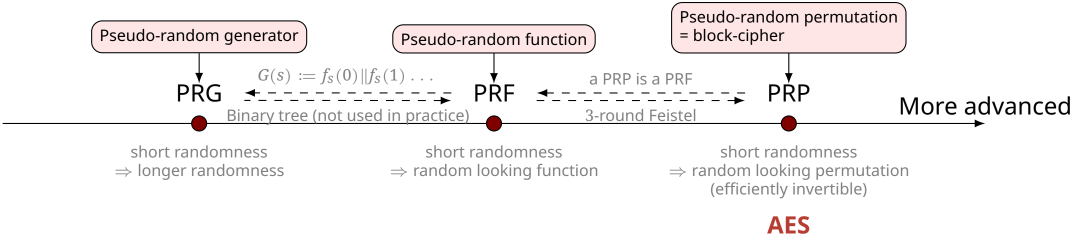
Encryption from simpler primitives

How to build an encryption:

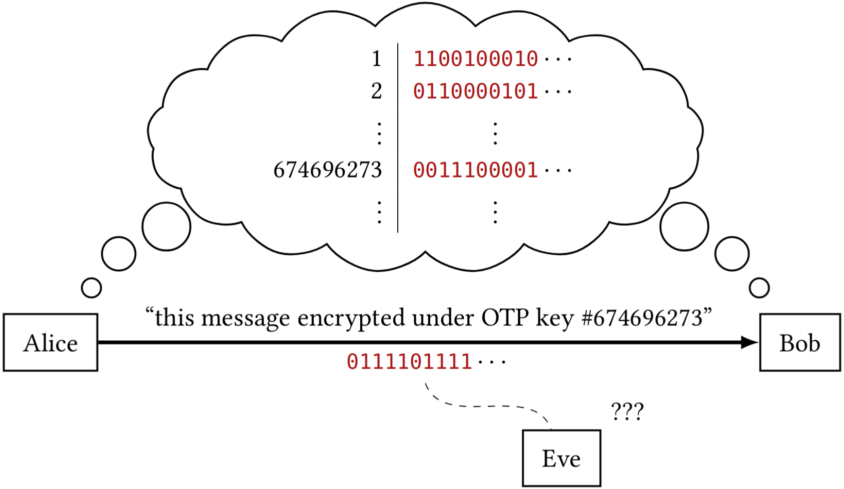
- Approach 1: start from scratch. Less guarantees it will be secure.
- Approach 2: try to build encryption from simpler, more tested, primitives.

Our approach

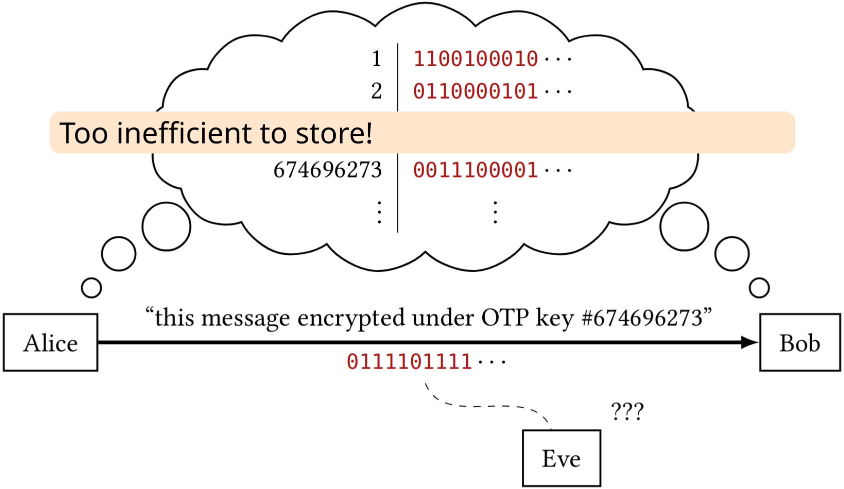
But which more fundamental primitive can we use?



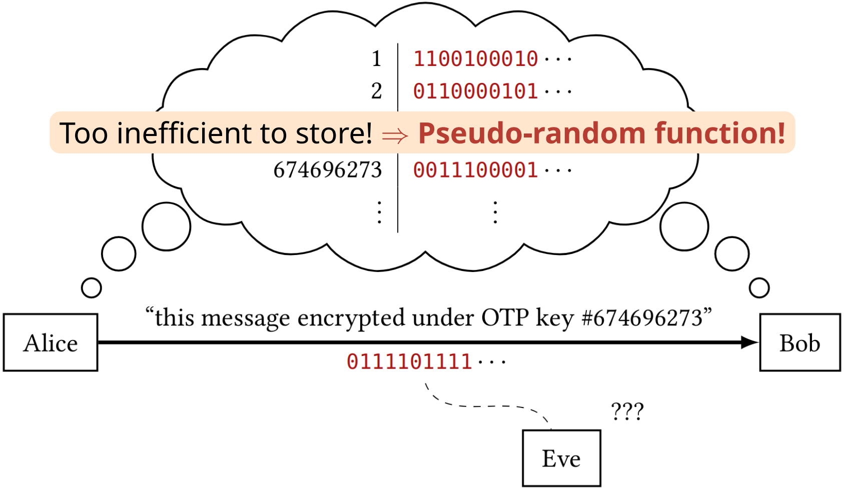
Motivation PRF



Motivation PRF



Motivation PRF



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

Pseudo-Random Generator (PRG)

PRG

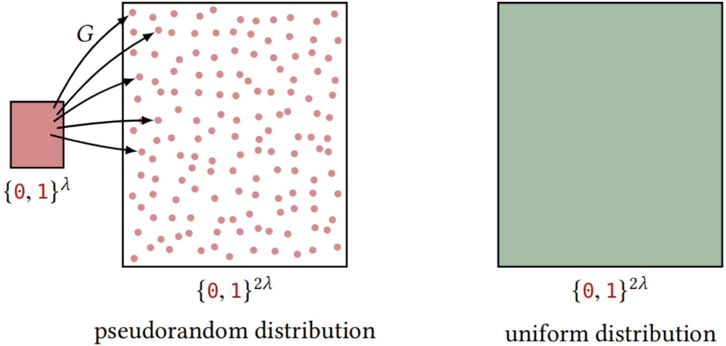
Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{\lambda+l}$ be a deterministic function with $l > 0$. We say that G is a *secure Pseudo-Random Generator (PRG)* if:

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

$\approx$

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

Pseudo-Random Generator (PRG)



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

Pseudo-Random Function (PRF)

PRF

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

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

$\approx$

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

$$\boxed{\begin{array}{l} \mathcal{L}^{F}_{\text{prp-real}} \\ \hline k \xleftarrow{\$} \{0,1\}^\lambda \\ \underline{\text{LOOKUP}(x \in \{0,1\}^{\mathtt{blen}}):} \\ \quad \text{return } F(k,x) \end{array}} \approx \boxed{\begin{array}{l} \mathcal{L}^{F}_{\text{prp-rand}} \\ \hline T := \text{ empty assoc. array} \\ \underline{\text{LOOKUP}(x \in \{0,1\}^{\mathtt{blen}}):} \\ \quad \text{if } T[x] \text{ undefined:} \\ \quad\quad T[x] \xleftarrow{\$} \{0,1\}^{\mathtt{blen}} \setminus T.\text{values} \\ \quad \text{return } T[x] \end{array}}$$

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?

- (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 sample, p = proba collision:

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

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

The birthday paradox

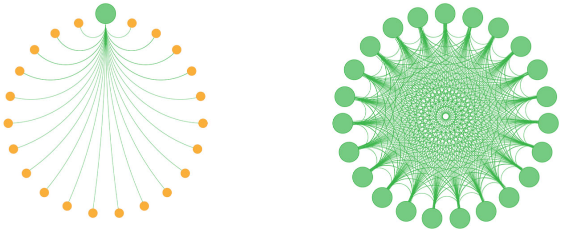
Let's try! Type your birthday (one per line, in format "DD/MM" with zeros, e.g. `02/08`) at:

`https://mensuel.framapad.org/p/crypto-aafw`

Use `echo '...' | sort | uniq -D` to find duplicates

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 ✓ $\sqrt{2^{128}} = 2^{128/2} = 2^{64}$.
  $\Rightarrow$ 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

$\mathcal{L}_{\text{samp-L}}$
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. Bad-event lemma: $\mathcal{A}$ is polynomial, so $\Pr[\text{bad} = 1] = \text{poly}(\lambda) \times \frac{\text{poly}(\lambda)}{2^\lambda} = \text{negl}(\lambda)$

$= |R|$

Number of calls to SAMP

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

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

Let $F\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a secure PRP (with $\mathsf{blen} = \lambda$). 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
- $\underline{\text{LOOKUP}(x \in \{0,1\}^{\mathsf{blen}})\text{:}}$
  - if $T[x]$ undefined:
    - $T[x] \xleftarrow{\$} \{0,1\}^{\mathsf{blen}} \setminus T.\text{values}$
  - return $T[x]$

$\equiv$

$\mathcal{L}^F_1$

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

$\diamond$

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

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

$\overset{\text{2 slides above}}{\approx}$

$\mathcal{L}^F_1$

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

$\diamond$

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

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

$\equiv$

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

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

$\square$

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:

$\Sigma_{\text{prf-pseudo-OTP}}$

$\underline{\mathsf{Gen}():}$
- $k \xleftarrow{\$} \{0,1\}^\lambda$
- return k

$\underline{\mathsf{Enc}(k,m):}$
- $r \xleftarrow{\$} \{0,1\}^\lambda$
- $x := F(k,r) \oplus m$
- return (r,x)

$\underline{\mathsf{Dec}(k,c):}$
- $m := F(k,r) \oplus c$
- 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)

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

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

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

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

Def. Enc $\equiv$

$\mathcal{L}_1$

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

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

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

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

  return (r, x)

Externalize $\equiv$

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

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

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

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

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

Def. Enc

$\equiv$

$\mathcal{L}_1$

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

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

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

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

  return (r, x)

Externalize

$\equiv$

$\mathcal{L}_1$

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

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

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

  return (r, x)

$\diamond$

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

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

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

  return $F(k, x)$

$\mathcal{L}_1$

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

$\mathcal{L}_1$

Def. Enc $\equiv$

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

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

$\quad r \overset{\$}{\leftarrow} \{0,1\}^\lambda$

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

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

Externalize $\equiv$

$\mathcal{L}_1$

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

$\quad r \overset{\$}{\leftarrow} \{0,1\}^\lambda$

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

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

$\diamond$

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

$k \overset{\$}{\leftarrow} \{0,1\}^\lambda$

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

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

Def PRF $\approx$

$\mathcal{L}_1$

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

$\quad r \overset{\$}{\leftarrow} \{0,1\}^\lambda$

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

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

$\diamond$

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

$T :=$ empty assoc. array

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

$\quad \text{if } T[x] \text{ undefined:}$

$\quad\quad T[x] \overset{\$}{\leftarrow} \{0,1\}^{\mathtt{out}}$

$\quad \text{return } T[x]$

$\mathcal{L}_2$

Externalize $\equiv$

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

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

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

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

$\diamond$

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

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

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

Def PRF $\approx$

$\mathcal{L}_1$

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

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

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

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

$\diamond$

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

$T :=$ empty assoc. array

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

$\quad$ if $T[x]$ undefined:

$\quad\quad T[x] \xleftarrow{\$} \{0,1\}^{\texttt{out}}$

$\quad \text{return } T[x]$

Inline $\equiv$

$\mathcal{L}_2$

$T :=$ empty assoc. array

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

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

$\quad$ if $T[r]$ undefined:

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

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

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

return (r, x)

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

$\diamond$

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

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

Externalize $\equiv$

$$\begin{array}{l}
\mathcal{L}_2 \\
\hline
T := \text{empty assoc. array} \\
\underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\
\quad r \xleftarrow{\$} \text{SAMP}() \\
\quad \text{if } T[r] \text{ undefined:} \\
\quad\quad T[r] \xleftarrow{\$} \{0,1\}^{\texttt{out}} \\
\quad x := T[r] \oplus m_L \\
\quad \text{return } (r, x)
\end{array}
\diamond
\begin{array}{l}
\mathcal{L}_{\text{samp-L}} \\
\hline
\underline{\text{SAMP}():} \\
\quad r \xleftarrow{\$} \{0,1\}^{\lambda} \\
\quad \text{return } r
\end{array}$$

Asymptotic birthday paradox $\approx$

$$\begin{array}{l}
\mathcal{L}_2 \\
\hline
T := \text{empty assoc. array} \\
\underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\
\quad r \xleftarrow{\$} \text{SAMP}() \\
\quad \text{if } T[r] \text{ undefined:} \\
\quad\quad T[r] \xleftarrow{\$} \{0,1\}^{\texttt{out}} \\
\quad x := T[r] \oplus m_L \\
\quad \text{return } (r, x)
\end{array}
\diamond
\begin{array}{l}
\mathcal{L}_{\text{samp-R}} \\
\hline
R := \emptyset \\
\underline{\text{SAMP}():} \\
\quad r \xleftarrow{\$} \{0,1\}^{\lambda} \setminus R \\
\quad R = R \cup \{r\} \\
\quad \text{return } r
\end{array}$$

$$\mathcal{L}_3$$

Asymptotic birthday paradox $\approx$

$\mathcal{L}_2$

$T :=$ empty assoc. array

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

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

$\quad$ if $T[r]$ undefined:

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

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

$\quad$ return (r, x)

$\diamond$

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

$R := \emptyset$

$\underline{\text{SAMP}():}$

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

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

$\quad$ return r

Inline $\equiv$

$\mathcal{L}_3$

$T :=$ empty assoc. array

$R := \emptyset$

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

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

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

$\quad$ if $T[r]$ undefined:

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

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

$\quad$ return (r, x)

$\mathcal{L}_4$

$\mathcal{L}_3$

$T :=$ empty assoc. array

$R := \emptyset$

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

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

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

if $T[r]$ undefined:

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

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

return (r, x)

Inline

$\equiv$

$\mathcal{L}_4$

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

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

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

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

$x := r' \oplus m_L$

return (r, x)

$T[r]$ always undefined

$\equiv$

$\mathcal{L}_5$

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

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

return (r, x)

T[r] always undefined

$\equiv$

$\mathcal{L}_4$

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

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

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

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

$x := r' \oplus m_L$

return (r, x)

Externalize

$\equiv$

$\mathcal{L}_5$

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

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

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

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

return (r, x)

$\diamond$

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

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

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

return $k \oplus m$

$\mathcal{L}_5$

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

$r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
$R = R \cup \{r\}$
$r' \xleftarrow{\$} \{0,1\}^{\texttt{out}}$
$x := r' \oplus m_L$
return (r, x)

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

$\mathcal{L}_5$

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

$r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
$R = R \cup \{r\}$
$x \leftarrow \text{OTENC}(m_L)$
return (r, x)

$\diamond$

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

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

$k \xleftarrow{\$} \{0,1\}^\lambda$
return $k \oplus m$

Externalize $\equiv$

$\mathcal{L}_5$

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

$r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
$R = R \cup \{r\}$
$x \leftarrow \text{OTENC}(m_L)$
return (r, x)

$\diamond$

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

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

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

OTP uniform ciphertext $\equiv$

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

return $k \oplus m$

OTP uniform ciphertext $\equiv$

$\mathcal{L}_5$

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

$\diamond$

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

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

Inline $\equiv$

$\mathcal{L}_5$

$\text{EAVESDROP}(m_L, m_R \in \mathcal{M})$:
- $r \xleftarrow{\$} \{0,1\}^\lambda \setminus R$
- $R = R \cup \{r\}$
- $x \xleftarrow{\$} \{0,1\}^\lambda$
- 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 $\{\mathtt{0}, \mathtt{1}\}^{\text{out}}$, 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**!

Multiple modes of operation (= variants):

Block cipher (PRP) $\xrightarrow{\text{Block cipher mode}}$ Encryption scheme

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

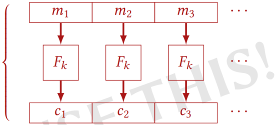
Key

Message

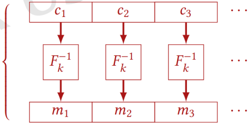
Definition (ECB mode: NEVER USE THIS)

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

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



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



NEVER USE THIS!

?

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:

$\underline{\mathrm{Enc}(k, m_1 \| \cdots \| m_\ell)\text{:}}$

$c_0 \leftarrow \{0,1\}^{blen}$;

for $i = 1$ to ℓ:

$\quad c_i := F(k, m_i \oplus c_{i-1})$

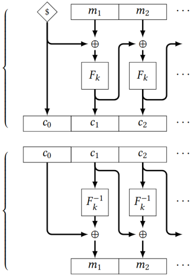
return $c_0 \| c_1 \| \cdots \| c_\ell$

$\underline{\mathrm{Dec}(k, c_0 \| \cdots \| c_\ell)\text{:}}$

for $i = 1$ to ℓ:

$\quad m_i := F^{-1}(k, c_i) \oplus c_{i-1}$

return $m_1 \| \cdots \| m_\ell$



?

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

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

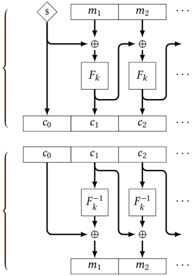
Common modes

Definition (CBC mode)

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

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

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



?

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

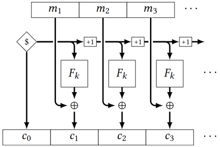
- (A) It acts like a OTP on the message, hence hides it ✗ IV is public, so cannot be a OTP key!
- (B) Used to have a non-deterministic encryption ✓

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:

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



?

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

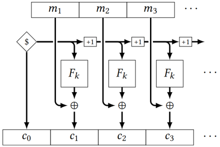
- A Yes
- B No

Common modes

Definition (CTR mode)

The counter (CTR) mode is defined as:

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



?

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

A Yes

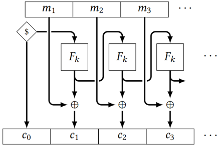
B No No need to have a PRP, PRF is enough (but in practice, most efficient PRF are PRP anyway)

Common modes

Definition (OFB mode)

The output feedback (OFB) mode is defined as:

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



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

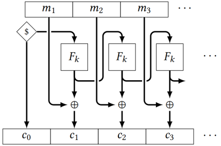
- (A) Yes
- (B) No

Common modes

Definition (OFB mode)

The output feedback (OFB) mode is defined as:

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



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

- (A) Yes
- (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 ✗
- (B) No ✓ Bad idea, because OFB is not parallelizable. Hence to decrypt the last byte of the drive, we need to decrypt the whole drive!

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 $= \lambda$ for simplicity)?

- (A) No
- (B) Yes, with

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

- (C) Yes, with

$\mathcal{A}$
$c :=$ EAVESDROP$(\texttt{0}^\lambda, \texttt{0}^{2\lambda})$ return $\lvert c \rvert \stackrel{?}{=} 2\lambda$

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

✗

Ⓒ Yes, with

$\mathcal{A}$
$c :=$ EAVESDROP$(\mathtt{0}^\lambda, \mathtt{0}^{2\lambda})$ return $\lvert c\rvert \stackrel{?}{=} 2\lambda$

✓ 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:

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

$\approx$

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

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...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 `10...0`): if you pad a message m of size $k$blen into m', what is the size of m'?

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

$$\boxed{Alice} \longleftrightarrow \boxed{Eve} \longleftrightarrow \boxed{Bob}$$

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

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

- step 0: realize that in CTR mode, $\mathsf{Enc}_k(m) \oplus (0^{\mathsf{blen}}, x) = \mathsf{Enc}_k(m \oplus x)$. 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 $\mathsf{Enc}_k(m)$).
- 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)
- second step: once you know the length of the padding p, you know that m_0 looks like $m_{\mathsf{unpad}}0^{8p}$Byte(p). Xor to the last byte of c_0 the byte Byte(p) $\oplus$ Byte(p+1). Thanks to step 0 you now have an encryption of $m_{\mathsf{unpad}}0^{8p}$Byte(p+1). Since m_{unpad} does not (a-priori) ends with a zero-byte, PADDINGORACLE will return an error. Now we iterate over $x \in \{0, \ldots, 255\}$ 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…)

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

IND-CCA

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

$$
\begin{array}{|l|}
\hline
\multicolumn{1}{|c|}{\mathcal{L}^{\Sigma}_{\text{cpa-L}}} \\
\hline
k \leftarrow \mathsf{Gen}(1^\lambda) \\
\mathcal{S} := \emptyset \\
\underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\
\quad \text{if } |m_L| \neq |m_R| \text{ return } \mathtt{err} \\
\quad c := \mathsf{Enc}_k(m_L) \\
\quad \mathcal{S} := \mathcal{S} \cup \{c\} \\
\quad \text{return } c \\
\underline{\text{DECRYPT}(c \in \mathcal{C}):} \\
\quad \text{if } c \in \mathcal{S} \text{ return } \mathtt{err} \\
\quad \text{return } \mathsf{Dec}(k, c) \\
\hline
\end{array}
\approx
\begin{array}{|l|}
\hline
\multicolumn{1}{|c|}{\mathcal{L}^{\Sigma}_{\text{cpa-R}}} \\
\hline
k \leftarrow \mathsf{Gen}(1^\lambda) \\
\mathcal{S} := \emptyset \\
\underline{\text{EAVESDROP}(m_L, m_R \in \mathcal{M}):} \\
\quad \text{if } |m_L| \neq |m_R| \text{ return } \mathtt{err} \\
\quad c := \mathsf{Enc}_k(m_R) \\
\quad \mathcal{S} := \mathcal{S} \cup \{c\} \\
\quad \text{return } c \\
\underline{\text{DECRYPT}(c \in \mathcal{C}):} \\
\quad \text{if } c \in \mathcal{S} \text{ return } \mathtt{err} \\
\quad \text{return } \mathsf{Dec}(k, c) \\
\hline
\end{array}
$$

Malleability

$\underline{\mathrm{Enc}(k, m_1 \| \cdots \| m_\ell):}$

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

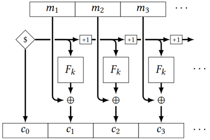
$c_0 := r$

for $i = 1$ to ℓ:

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

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

return $c_0 \| \cdots \| c_\ell$



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

A. No

B. Yes, with

$\mathcal{A}$

$(c_0, c_1) \leftarrow$ EAVESDROP$(0^{\mathsf{blen}}, 1^{\mathsf{blen}})$

$m \leftarrow$ DECRYPT$((c_0, c_1 \oplus (10\ldots0)))$

return $m \stackrel{?}{=} 10\ldots0$

C. Yes, with

$\mathcal{A}$

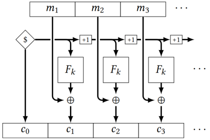
$(c_0, c_1) \leftarrow$ EAVESDROP$(0^{\mathsf{blen}}, 1^{\mathsf{blen}})$

$m \leftarrow$ DECRYPT$((c_0, c_1))$

return $m = 0^{\mathsf{blen}}$

Malleability

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



?

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

A No

B Yes, with

$\mathcal{A}$

$(c_0, c_1) \leftarrow \text{EAVESDROP}(0^{\text{blen}}, 1^{\text{blen}})$
$m \leftarrow \text{DECRYPT}((c_0, c_1 \oplus (10\ldots0)))$
return $m \stackrel{?}{=} 10\ldots0$

C Yes, with

$\mathcal{A}$

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

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** = common PRP (hence PRF) used in block-cipher modes
- **Malleable** encryption $\Rightarrow$ attacks against active adversaries (e.g. padding oracle/timing attacks)
- Authentication will help us!