$x \xleftarrow{\$} X$: x is obtained by sampling an element uniformly at random from the set X

$y \leftarrow A(x)$: If A is a (probabilistic) algo on a distribution, we run A on input x and store the result in x

$x \stackrel{?}{=} y$: Returns 1 (true) if x equals y, 0 (false) otherwise

$negl(\lambda)$: An arbitrary function f that is negligible (= smaller than any inverse polynomial) i.e. $\forall c \in \mathbb{N}, \lim_{\lambda \to \infty} \lambda^c f(\lambda) = 0$

$poly(\lambda)$: A function smaller than some polynomials, i.e. $\exists c \in \mathbb{N}, N \in \mathbb{N}, \forall \lambda > N, f(\lambda) \leq \lambda^c$

$negl(\lambda) + negl(\lambda) = negl(\lambda)$, $negl(\lambda) * negl(\lambda) = negl(\lambda)$, $poly(\lambda) negl(\lambda) = negl(\lambda)$

Symmetric encryption: both parties share the same secret.

Key generation $k \leftarrow Gen(1^\lambda)$

Encryption $c \leftarrow Enc_k(m)$

Decryption $m \leftarrow Dec_k(c)$

No need to share secret √ / × Stronger assumption factoring / × less efficient / No statistical security ×

Our focus: Minicrypt (one way functions exist, no public key cryptography)

An encryption must always be non-deterministic !

Never use a home-made encryption, it will be insecure !

Focus on the course: Game based security ← Challenger models → "what an adversary is allowed to do"

Blackbox or physical access / Passive (eavesdrop) or active !!! ↳ which message/function can he adv. call ← ↳ "what is considered to be bad"

Kerckhoff's principle: the adversary knows details of the protocol.

search problem: adversary needs to find a bit string

decision problem: adversary needs to find a single bit.

advantage: $Adv_{\mathcal{A}}(\lambda) = |Pr[\mathcal{A}(1^\lambda) \diamond \mathcal{L}_0 = 1] - Pr[\mathcal{A}(1^\lambda) \diamond \mathcal{L}_1 = 1]| \leq negl(\lambda)$

interchangeability def: 2 libraries $\mathcal{L}_0$ and $\mathcal{L}_1$ are interchangeable (or equal), written $\mathcal{L}_0 \equiv \mathcal{L}_1$ if any adversary $\mathcal{A}$, $Pr[\mathcal{A} \diamond \mathcal{L}_0 = 1] = Pr[\mathcal{A} \diamond \mathcal{L}_1 = 1]$

indistinguishability def: 2 libraries $\mathcal{L}_0$ and $\mathcal{L}_1$ are indistinguishable, written $\mathcal{L}_0 \approx \mathcal{L}_1$, if for any adversary $\mathcal{A}(1^\lambda)$ running in polynomial time and outputting a single bit:

$$|Pr[\mathcal{A}(1^\lambda) \diamond \mathcal{L}_0 = 1] - Pr[\mathcal{A}(1^\lambda) \diamond \mathcal{L}_1 = 1]| \leq negl(\lambda)$$

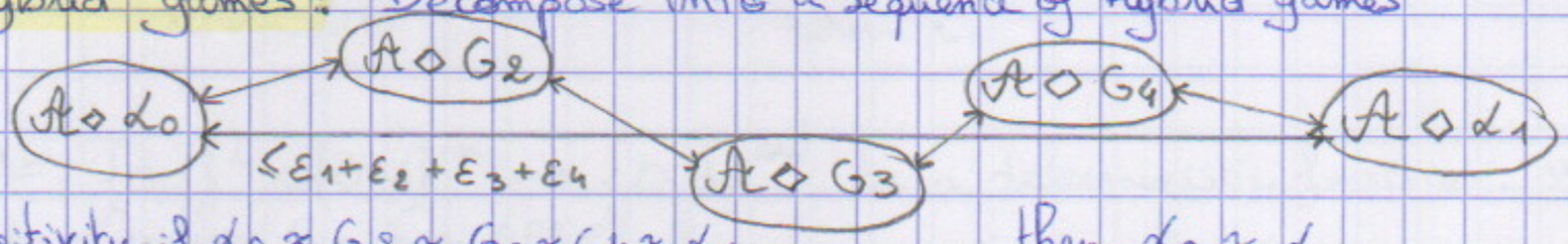
Properties:

Transitivity: $(\mathcal{L}_0 \approx \mathcal{L}_1) \wedge (\mathcal{L}_1 \approx \mathcal{L}_2) \Rightarrow \mathcal{L}_0 \approx \mathcal{L}_2$

Chaining: $(\mathcal{L}_0 \approx \mathcal{L}_1) \Rightarrow ((\mathcal{L} \diamond \mathcal{L}_0) \approx (\mathcal{L} \diamond \mathcal{L}_1))$

Reduction, 6 main methods:

1. Hybrid games: Decompose into a sequence of hybrid games



by transitivity, if $\mathcal{L}_0 \approx G_2 \approx G_3 \approx G_4 \approx \mathcal{L}_1$ then $\mathcal{L}_0 \approx \mathcal{L}_1$

2. Probabilities: explicitly compute the probability, and show equality or bound the statistical distance (statistical security only)
3. Equality: Show that the 2 games are trivially doing exactly the same thing → code externalized to a sub library, then inlined ex: TD 2
4. Reduction: show that if we can distinguish them, they $\mathcal{A}$ can be used to break a hard problem
5. Theorem / assumption: use a theorem already seen in the course or an assumption. } trivial
6. Chaining: prove $\mathcal{L}_1 = \mathcal{L}_2$, then $\mathcal{A} \diamond \mathcal{L}_1 \approx \mathcal{A} \diamond \mathcal{L}_2$ } trivial

Bad event lemma: Let $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ be 2 libraries that define a variable named bad, that is initialized to 0. If $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ have identical code except for code blocks reachable only when bad = 1 (e.g. guarded with an "if bad = 1" statement), then:

$$|Pr[\mathcal{A} \diamond \mathcal{L}_{left} = 1] - Pr[\mathcal{A} \diamond \mathcal{L}_{right} = 1]| \leq Pr[\mathcal{A} \diamond \mathcal{L}_{left} \text{ sets bad} = 1]$$

Symmetric encryption scheme:

$$\Pr_{k \leftarrow K}[Dec_k(Enc_k(m)) = m] = 1$$

One-Time Pad, OTP def: $Gen(1^\lambda)$: $k \xleftarrow{\$} \{0,1\}^\lambda$, return k

$Enc(k,m)$: return $k \oplus m$

$Dec(k,c)$: return $k \oplus c$

Correctness: $\forall k, Dec(k, Enc(k,m)) = k \oplus k \oplus m = m$

Theorem: OTP is one-time secure

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

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

$\equiv$

$\mathcal{A}^{\Sigma}_{cpa\text{-}R}$: $k \leftarrow Gen(1^\lambda)$; EAVESDROP($m_L, m_R \in \mathcal{M}$): return $Enc_k(m_R)$

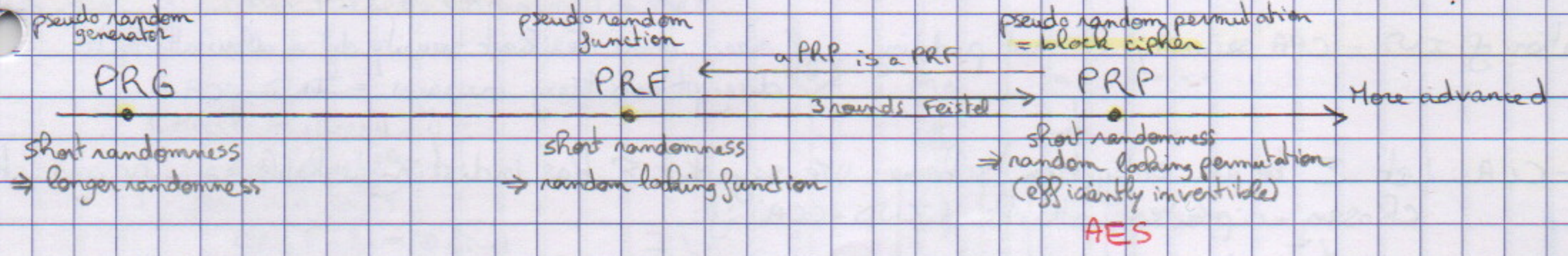
Si on veut encrypter un message, on ne voit pas la diff entre l'un et l'autre.

but the OTP is a deterministic encryption ← so OTP is not CPA secure

Never reuse a OTP key

How to build IND-CPA secure schemes

↳ try to build encryption from simpler, more tested primitives



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

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

$\approx$

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

PRG ≠ random number generator

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

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

$\approx$

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

PRP: Let $F: \{0,1\}^\lambda \times \{0,1\}^{blen} \to \{0,1\}^{blen}$ be a deterministic function. We say that F is a secure Pseudo Random Permutation (PRP) aka a block cipher, if F is invertible, ie if there exists an efficient function F^{-1} s.t

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

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

$\mathcal{A}^{F}_{prp\text{-}real}$: $k \xleftarrow{\$} \{0,1\}^\lambda$; LOOKUP($x \in \{0,1\}^{blen}$): return $F(k,x)$

$\approx$

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

Birthday paradox: $\frac{1}{\sqrt{N}}$

One to many / Many to many

Theorem (Asymptotic birthday paradox):

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

$\approx$

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

→ The birthday paradox does not mean asymptotic security ($\sqrt{negl(\lambda)} = negl(\lambda) \Rightarrow$ in real life : size of the key may need be doubled to prevent this attack

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

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

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, m) : $m = F(k,r) \oplus x$; return m

Theorem : the PRF pseudo-OTP is IND-CPA secure ← but limitations (long message) ← solution : use block cipher modes

Block cipher modes : Block cipher (PRP) —block cipher mode→ Encryption scheme

↳ allow us in some sense what we want

Enc : $\{0,1\}^k$ (key) $\times \{0,1\}^*$ (message) $\to \{0,1\}^*$

Never use DES ← broken

Today : most widely used cipher is AES (Advanced Encryption Standard)

IND-CPA for variable-length plaintexts : when messages can have various length, we need to update the def°

in EAVESDROP : if $|m_L| \neq |m_R|$ return err ; return $Enc_k(m_L)$

Is the length an issue ? Sometimes → Google Maps, possible to know where file is displayed only by looking by traffic

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

→ CTR mode : simple, just truncate the ciphertext

→ CBC mode : need to add padding

Limitation of IND-CPA security : not padding // need more resilient security def → 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 :

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

$k \leftarrow Gen(1^\lambda)$

$S = \emptyset$

EAVESDROP($m_L, m_R \in \mathcal{M}$):
  if $|m_L| \neq |m_R|$ return err
  $c := Enc_k(m_L)$
  $S := S \cup \{c\}$
  return c

Decrypt($c \in C$):
  if $c \in S$ return err
  return $Dec(k, c)$

$\approx$

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

↓ parcil que-L

Malleability

In conclusion

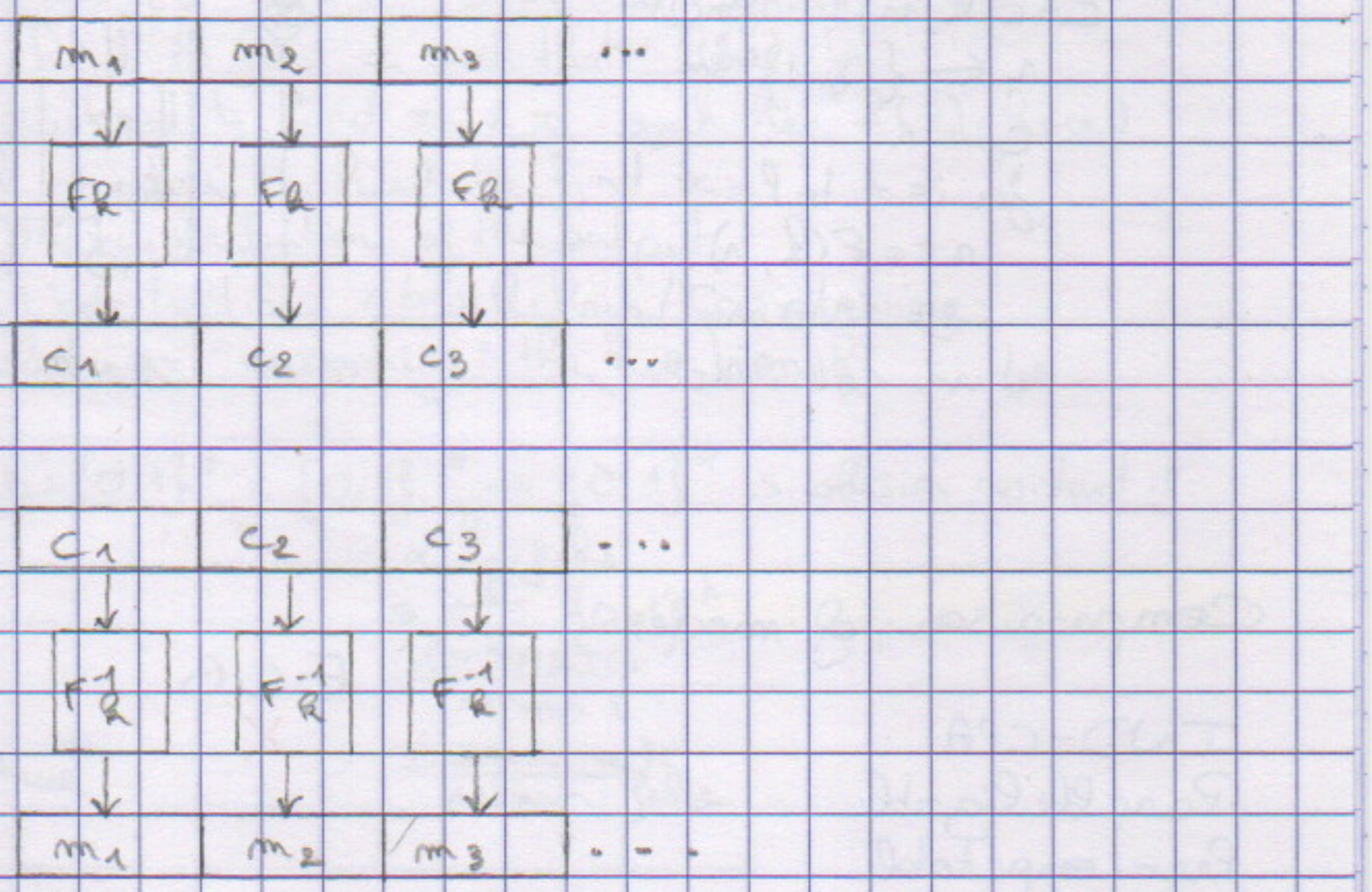
- OTP is statistically secure if used once
- A first notion of security against passive adversary is IND-CPA
- PRF $\Rightarrow$ IND-CPA secure schemes
- Birthday Paradox = may need to double the size of Key
- Block cipher modes = encrypt efficiently arbitrarily long messages (padding sometimes necessary)
- 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 (padding oracle / timing attacks)
- Authentication will help us !

Cryptography Engineering: Symmetric cryptography
Block cipher modes

ECB (Electronic Codebook) mode def: Never use it.

Enc(k, m1||...||mℓ):
  for i = 1 to ℓ:
    ci := F(k, mi)
  return c1||...||cℓ

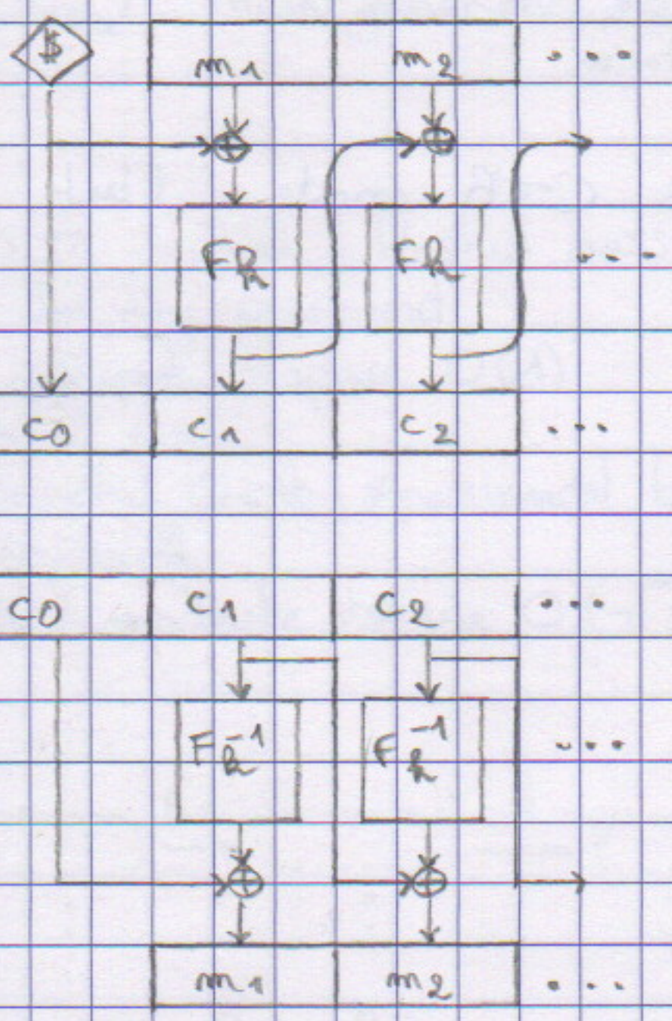
Dec(k, c1||...||cℓ):
  for i = 1 to ℓ:
    mi = F^-1(k, ci)
  return m1||...||mℓ



CBC (Cipher Block Chaining) mode:

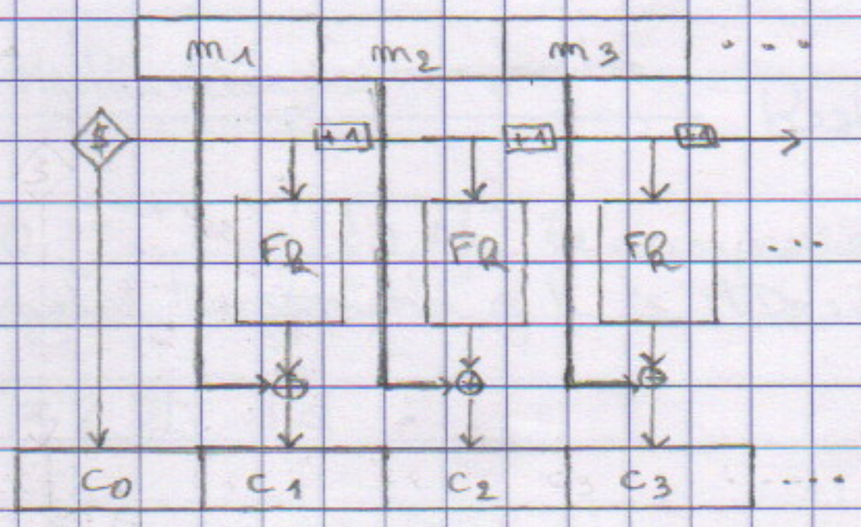
Enc(k, m1||...||mℓ):
  c0 ← {0,1}^blen
  for i = 1 to ℓ:
    ci = F(k, mi ⊕ ci-1)
  return c0||c1||...||cℓ

Dec(k, c0||...||cℓ):
  for i = 1 to ℓ:
    mi := F^-1(k, ci) ⊕ ci-1
  return m1||...||mℓ



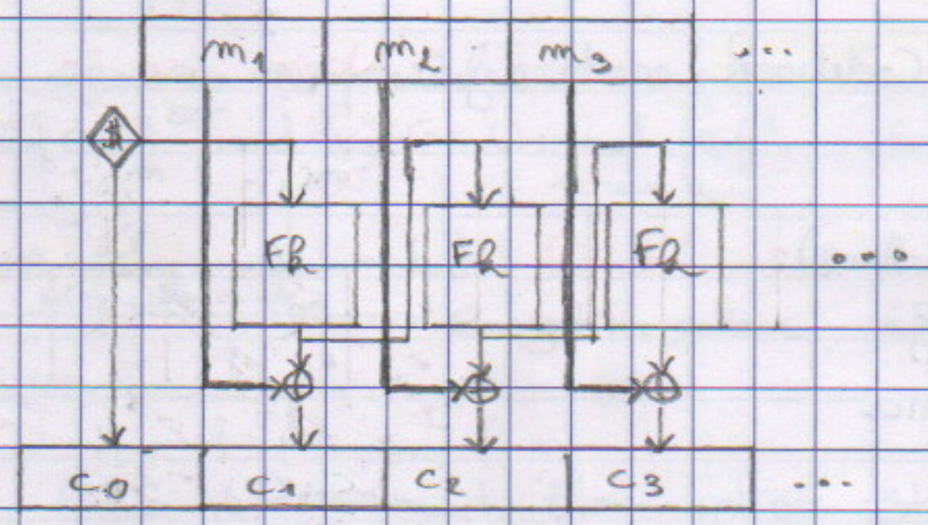
CTR (Counter):

Enc(k, m1||...||mℓ):
  r ← {0,1}^blen
  c0 := r
  for i = 1 to ℓ:
    ci = F(k, r) ⊕ mi
    r = r + 1 % 2^blen
  return c0||...||cℓ



OFB (Output FeedBack)

Enc(k, m₁ || ... || mℓ):
  r ← {0,1}^blen
  c₀ := r
  for i = 1 to ℓ:
    r := F(k, r)
    cᵢ := r ⊕ mᵢ
  return c₀ || ... || cℓ



Comparison of modes

	ECB	CBC	CTR	OFB
IND-CPA	✗	✓	✓	✓
Parallelizable		✗	✓	✗
Pre-computable		✗	✓	✓
Can avoid padding		✗	✓	✓
Safe with no permutation cycle		✗	✓	✗
Slightly safer against IV re-use ↳ so bad implementation		✓	✗	✗

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

Hash function

Hash function: a Hash function is a function $R: \{0,1\}^* \longrightarrow \{0,1\}^n$

Multiple properties:

Collision resistance: Hard to find a collision, i.e $x \neq x'$ such that $R(x) = R(x')$

First-preimage resistance: (one way) given y, Hard to find x such that $R(x) = y$

Second-preimage resistance: given a random x, Hard to find $x' \neq x$ such that $R(x) = R(x')$

Hiding: given $R(r \| x)$ for a long enough random r, Hard to find x.

Universality: weaker assumption about the distribution of the output.

Some applications: Authentication / IND-CCA constructions / blockchain / Coin tossing

Salt = random publicly known value sampled to "customize" the function R.

Often: $R(s,x) = R(s \| x)$

Collision resistant (flavor 1): A Hash function $R: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is collision resistant if:

$\alpha^R_{cr\text{-}real}$

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

GETSALT():
return s

TEST(x, x'):
if $x \neq x'$ and $R(s,x) = R(s,x')$: return true
else: return false

$\alpha^R_{cr\text{-}fake}$

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

GETSALT():
return s

TEST(x, x'):
return false

You should always Hash the passwords you store in a database!

If we can't "decrypt" the password (s_{Alice}, R_{Alice}) (Hash function), How can we check if the password p is correct ?

$\Rightarrow$ Check if $R(s_{Alice}, p) = R_{Alice}$!

Salt limit pre-computation attacks. How to recover a Hashed password with no salt ?

Method 1: Brut force, restart from scratch for any password

$\Rightarrow$ inefficient in time $O(\#passwords)$, efficient in space $O(1)$

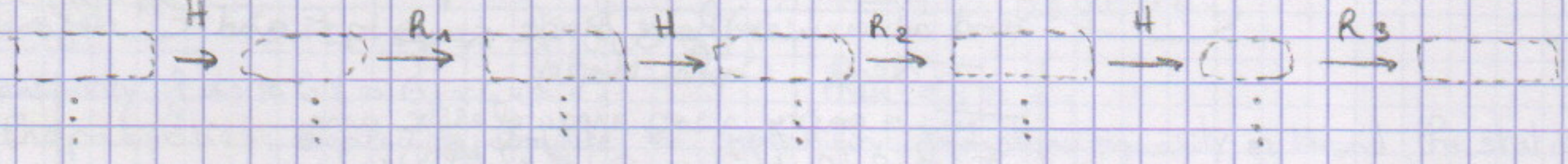
Method 2: Brut force & store for re-use next time $O(\#passwords)$

$\Rightarrow$ efficient in time once the table is generated $O(\log \#passwords)$, but needs HUGE storage

Method 3: rainbow tables = time / space tradeoff

$\Rightarrow$ e.g. moderate time $O(\sqrt{\#passwords})$, moderate storage $O(\sqrt{\#passwords})$

Rainbow tables

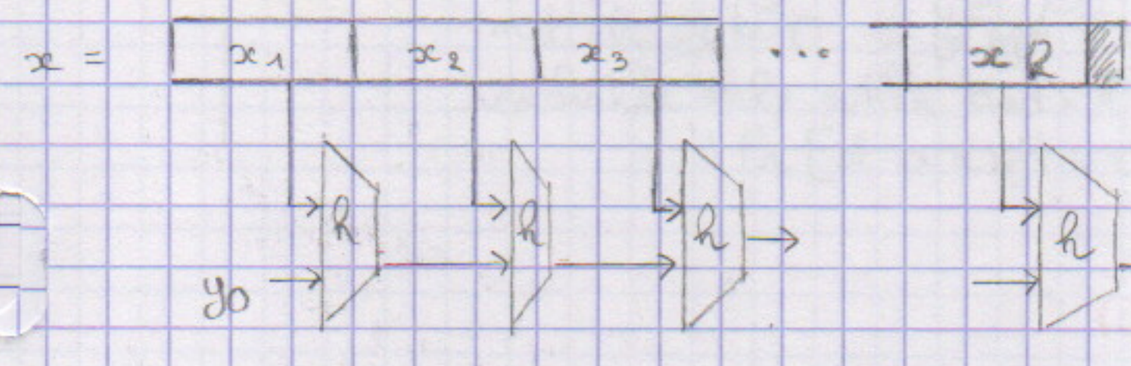


$\neq$ reduction function for each column = avoid long chains of collision

Good to add pepper.

Fixed-size compression function $\xrightarrow{\text{Merkle-Damgård transformation}}$ Hash Function

Merkle-Damgård construction: Let $R: \{0,1\}^{n+t} \to \{0,1\}^n$ be a compression function, Then the Merkle-Damgård transformation of R is $MD_R: \{0,1\}^* \longrightarrow \{0,1\}^n$ where:



MDPAD$_t(x)$
$\ell := |x|$ as length-t binary number
while $|x|$ not a multiple of t:
$x := x \| 0$
return $x \| \ell$

$MD_R(x)$
$x_1 \| \cdots \| x_{R+1} := MDPAD_t(x)$ // each x_i is t bits
$y_0 = 0^n$
for $i = 1$ to $R+1$:
$y_i := R(y_{i-1} \| x_i)$
output y_{R+1}

Length extension attack : Goal = obtain Message Authentication Code (MAC = authentication) from hash function via $R(k \| m)$

Issue : Length extension attack : With the MD construction, possible to get $R(k \| m)$ from $R(k \| m) \leftarrow$ sign $\neq m$ without knowing k.

How ? Observation : "Knowing $H(x)$, allows to predict the hash of any strings that begin with $MD\text{PAD}(x)$" :

Mitigate length-extension attack → wide pipe construction
→ sponge construction

Compression function $R : \{0,1\}^{n+t} \longrightarrow \{0,1\}^n$

Random Oracle Model (ROM) : A protocol is said to be defined in the Random-Oracle Model if all parties (including honest parties when defining the protocol and adversaries) have oracle access to H defined as :

Random Oracle
T = empty assoc. array
$H(x \in \{0,1\}^*)$:
  if $T[x]$ undefined
    $T[x] \xleftarrow{\$} \{0,1\}^{out}$
  return $T[x]$

Lazy sampling
ROM ≠ PRF

Ideal Cipher Model (ICM) : A protocol is said to be defined in the Ideal Cipher Model if all parties (including honest-parties when defining the protocol and adversaries) have oracle access to F and F^{-1} defined as :

Ideal Cipher Model :
T := empty assoc. array of assoc. array
$F(k \in \{0,1\}^\lambda, x \in \{0,1\}^{blen})$:
  if $T[k][x]$ undefined
    $T[k][x] \xleftarrow{\$} \{0,1\}^{blen} \setminus T[k].values$
  return $T[k][x]$
$F^{-1}(k \in \{0,1\}^\lambda, y \in \{0,1\}^{blen})$:
  if $\exists x$ s.t. $T[k][x] = y$:
    return x
  else :
    $x \xleftarrow{\$} \{0,1\}^{blen} \setminus T[k].keys$
    $T[k][x] = y$
    return x

Use SHA-3 or SHA-2 (but not for MAC)

Never use MD5, SHA-0, SHA-1.