Clases 5 y 6: Indistinguibilidad Computacional: PRGs, PRFs y Seguridad CPA

Fernando Krell
fekrell@uc.cl

Pontificia Universidad Católica de Chile & Dreamlab Technologies

March 21, 2018

Seguridad Perfecta

- Seguridad Perfecta es muy estricta:
  - Adversario tiene poder computacional ilimitado
  - Texto cifrado no revela información alguna
- Introduciremos el concepto de seguridad computacional:
  - Adversario tiene poder computacional limitado.
  - Adversario no puede distinguir el cifrado con *probabilidad no despreciable*.

Seguridad Computacional

- Todos los algoritmos (incluido el adversario $\mathcal{A}$) corren en *tiempo polinomial* en al largo de su entrada.
  - Es decir, para todo algoritmo Alg $\exists$ polinomio $p(\cdot)$ tal que $\text{TIME}(\text{Alg})(x)) = p(|x|)$
  - Cuando son aleatorizados, los denotamos como PPT (probabilistic polinomial time)
- Probabilidad de que el adversario "quiebre" el esquema es *negligible* en el tamaño de su entrada.
- Introducimos un parámetro λ, llamado parámetro de seguridad.
  - Algoritmos están autorizados en correr en tiempo polinomial en λ.
  - Mensajes y textos cifrados son de largo polinomial en λ.
  - Probabilidad de que el adversario gane es negligible en λ.
- Técnicamente, los algoritmos tienen que tener entrada de largo al menos λ bits, y no más que $p(\lambda)$ para algún polinomio $p(\cdot)$

¿Por qué polinomios?

- Típicamente asociados a la noción de computacion realista o "eficiente".
- Resistente a cambios en el modelo de computación (versiones de máquinas de turing, RAM)
  - Podemos escribir en pseudo-código.
- Algoritmo que hace una cantidad polinomial de llamadas a algoritmos polinomiales, es también polinomial.
  - $\text{poli}(n) \times \text{poli}(n) = \text{poli}(n)$ (Ej: $n^3 \cdot n^2 = n^5$)
- Funciones negligibles siguen siendo negligibles incluso cuando son "modificadas" por un polinomio.
  - $\text{poli}(n) \times \text{negl}(n) = \text{negl}(n)$ (Ej: $n^{10}/2^n = \text{negl}(n)$)

Generadores Pseudo-aleatorios (PRGs)

Definition (PRGs)

$G : \{0,1\}^\lambda \to \{0,1\}^n$ es un generador pseudo-aleatorio si:

- Extiende el input: $n > \lambda$ en donde $n = p(\lambda)$ para algún polinomio $p(\cdot)$.
- Pseudo-aleatoriedad: $\forall$ algoritmo probabilista de tiempo polinomial (PPT) D, $\exists$ función negligible negl tal que

$$\left| \Pr_{K \sim U_\lambda, Z \leftarrow G(K)}[D(Z) = 1] - \Pr_{Z \sim U_n}[D(Z) = 1] \right| = \text{negl}(\lambda)$$

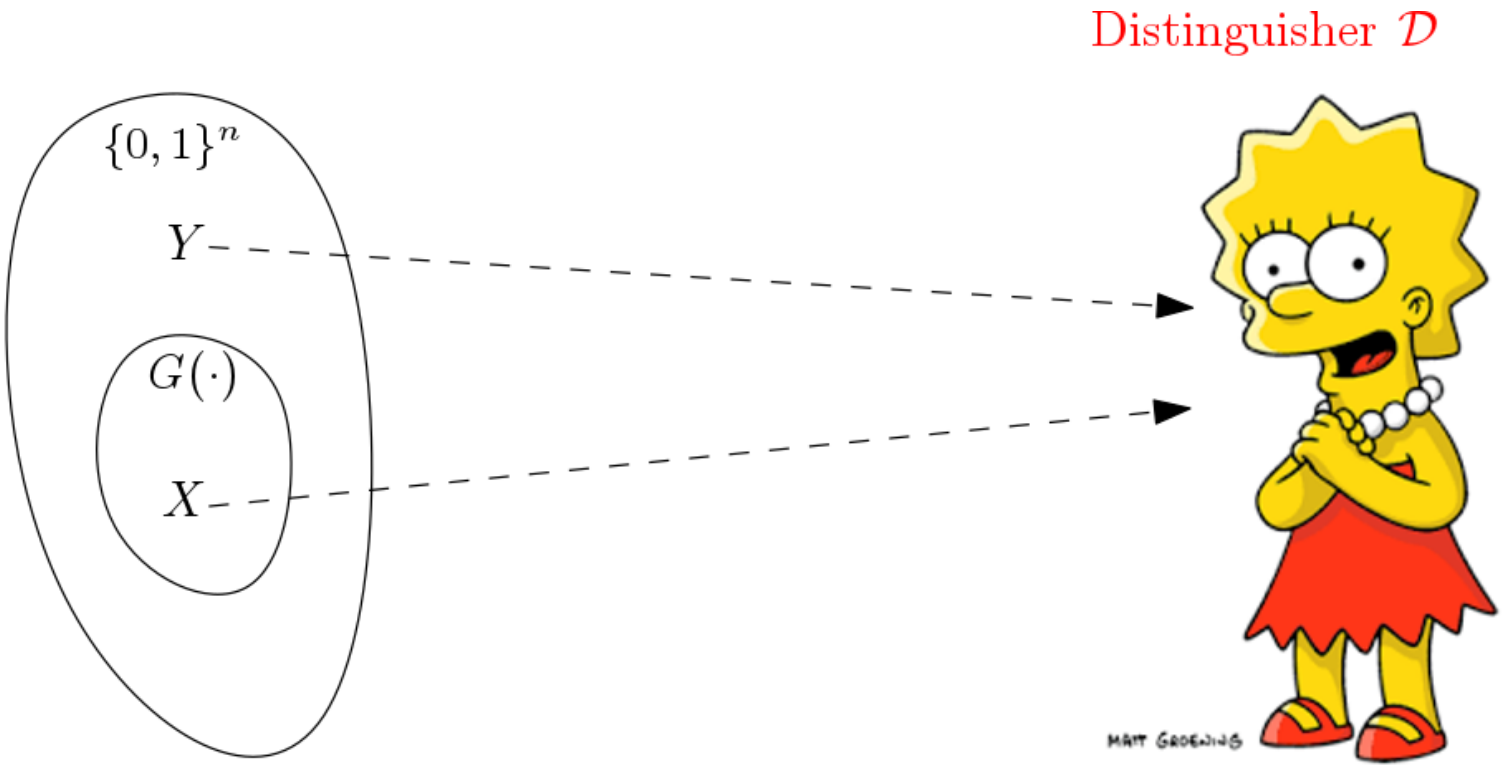
Notación:

$$\forall \text{ PPT } D, \ \left| \Pr[D(G(U_\lambda)) = 1] - \Pr[D(U_n) = 1] \right| = \text{negl}(\lambda)$$

o más simple:

$$G(U_\lambda) \stackrel{c}{\equiv} U_n \text{ (son computacionalmente indistinguibles)}$$

Distinguidor de PRG



Ejemplo

$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$:
$G(x = x_1 x_2 \dots x_\lambda) := x_1 x_2 \dots x_\lambda || (\bigoplus_{i=1}^{\lambda} x_i)$
G no es PRG: Si $Z \sim G(U_\lambda)$, entonces el último bit esta determinado por los primeros λ bits de Z. En cambio, si $Z \sim U_{\lambda+1}$ el último bit es independiente y uniforme.

Distinguidor $D(z = z_1 z_2 \dots z_\lambda z_{\lambda+1})$:

- Computa $\sigma = \bigoplus_{i=1}^{\lambda} z_i$.
- If $\sigma = z_{\lambda+1}$, output 1.
- Else, output 0.

Ejemplo

$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$:
$G(x = x_1 x_2 \dots x_\lambda) := x_1 x_2 \dots x_\lambda || (\bigoplus_{i=1}^{\lambda} x_i)$
G no es PRG: Si $Z \sim G(U_\lambda)$, entonces el último bit esta determinado por los primeros λ bits de Z. En cambio, si $Z \sim U_{\lambda+1}$ el último bit es independiente y uniforme.

Distinguidor $D(z = z_1 z_2 \dots z_\lambda z_{\lambda+1})$:

- Computa $\sigma = \bigoplus_{i=1}^{\lambda} z_i$.
- If $\sigma = z_{\lambda+1}$, output 1.
- Else, output 0.

Análisis:
$\Pr[D(G(U_\lambda)) = 1] = 1$.
$\Pr[D(U_{\lambda+1}) = 1] = 1/2$.
Por lo tanto, diferencia es $1/2 \neq$ negligible. Concluimos que $G(x) := x || \bigoplus_{i=1}^{\lambda} x_i$ no es un generador pseudo-aleatorio.

Ejemplo

$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$:
$G(x = x_1 x_2 \dots x_\lambda) := x_1 x_2 \dots x_\lambda || (\bigoplus_{i=1}^{\lambda} x_i)$
G no es PRG: Si $Z \sim G(U_\lambda)$, entonces el último bit esta determinado por los primeros λ bits de Z. En cambio, si $Z \sim U_{\lambda+1}$ el último bit es independiente y uniforme.

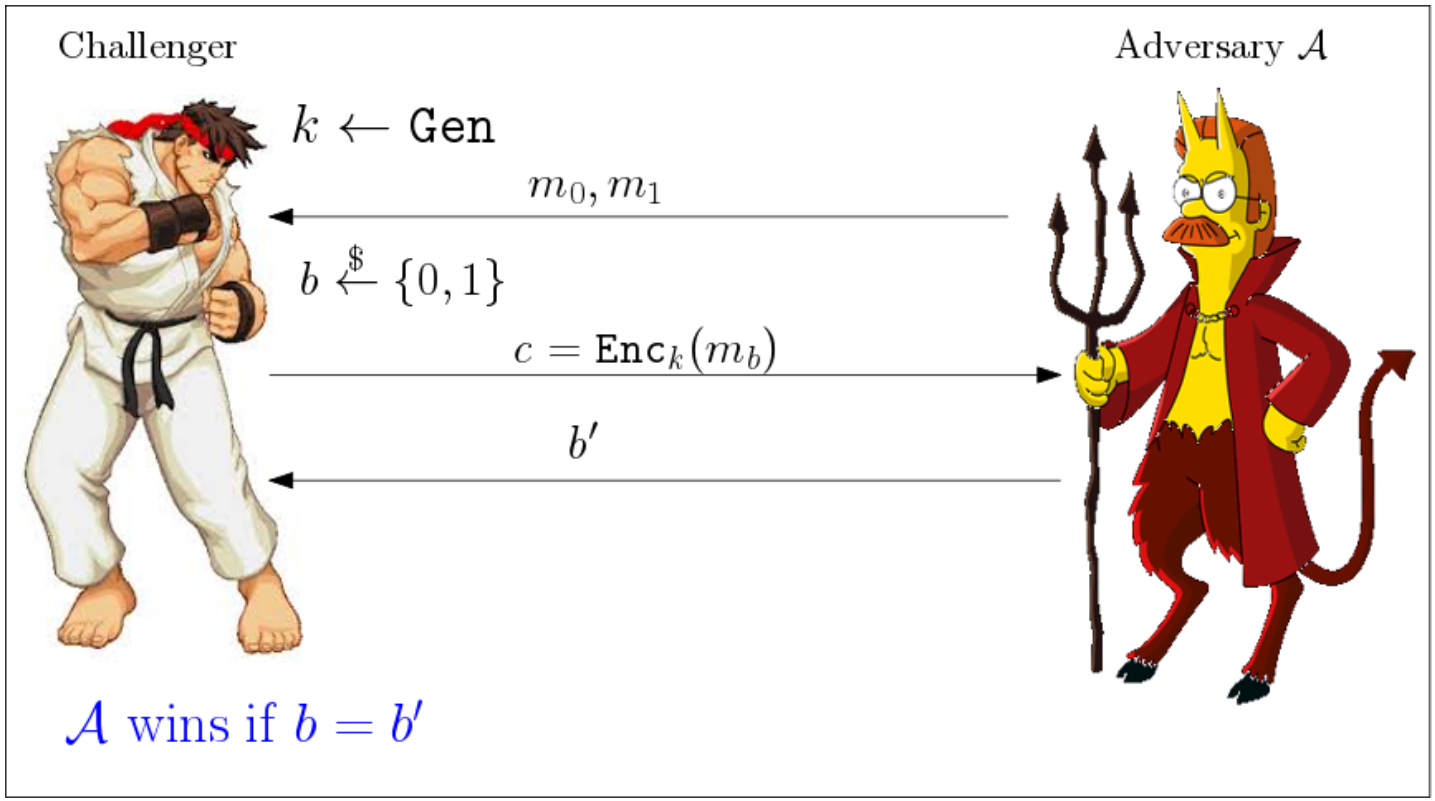
Distinguidor $D(z = z_1 z_2 \dots z_\lambda z_{\lambda+1})$:

- Computa $\sigma = \bigoplus_{i=1}^{\lambda} z_i$.
- If $\sigma = z_{\lambda+1}$, output 1.
- Else, output 0.

Análisis:
$\Pr[D(G(U_\lambda)) = 1] = 1$.
$\Pr[D(U_{\lambda+1}) = 1] = 1/2$.
Por lo tanto, diferencia es $1/2 \neq$ negligible. Concluimos que $G(x) := x || \bigoplus_{i=1}^{\lambda} x_i$ no es un generador pseudo-aleatorio.

Esquemas Indistinguibles

Sea $\Pi = \langle \mathsf{Gen}(\cdot), \mathsf{Enc}(\cdot,\cdot), \mathsf{Dec}(\cdot,\cdot) \rangle$ un esquema de cifrados.



Seguro si ningún $\mathcal{A}$ no gana con probabilidad $1/2 + 1/\mathrm{poli}(\lambda)$.

Definición: Cifrados indistinguibles

Experimento $PrivK^{\text{comp-ind}}_{\mathcal{A},\Pi}(\lambda)$:

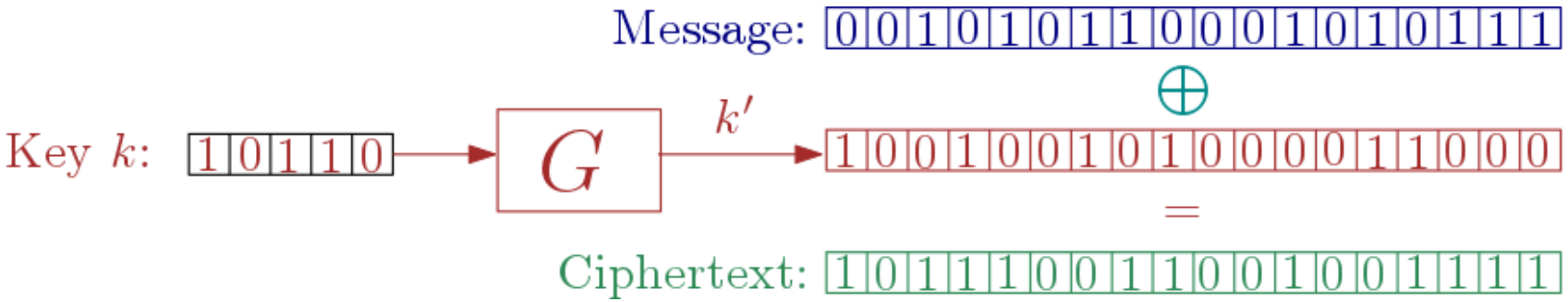
1. $k \leftarrow \text{Gen}(1^\lambda)$
2. $m_0, m_1 \leftarrow \mathcal{A}(1^\lambda)$
3. $b \xleftarrow{\$} \{0,1\}$
4. $c \leftarrow \text{Enc}_k(m_b)$
5. $b' \leftarrow \mathcal{A}(c)$
6. If $b = b'$, output 1. Else, output 0.

Definition

$\Pi = \langle \text{Gen}(\cdot), \text{Enc}(\cdot,\cdot), \text{Dec}(\cdot,\cdot) \rangle$ *tiene cifrados computacionalmente indistinguibles* si $\forall$ PPT $\mathcal{A}$, $\exists$ función negligible $\mu(\cdot)$ tal que:

$$\Pr[PrivK^{\text{comp-ind}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq 1/2 + \mu(\lambda)$$

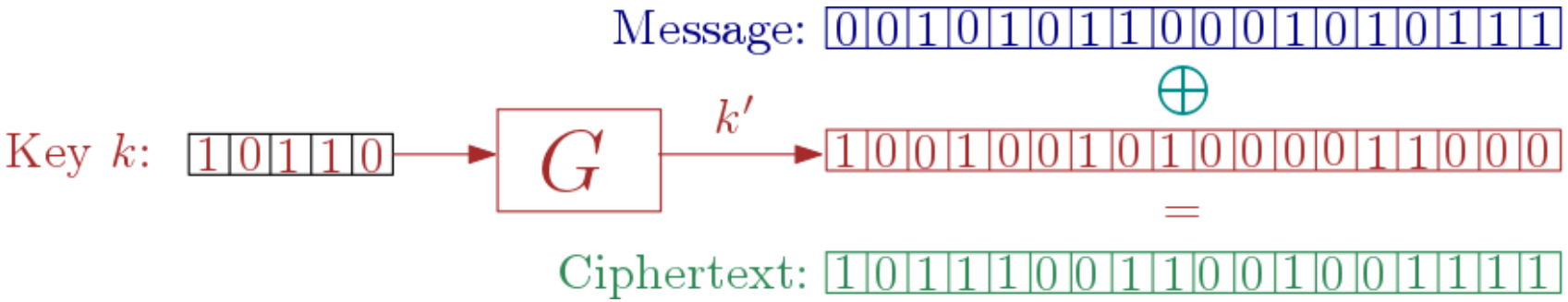
Esquema basado en PRG



Cifrador para $\mathcal{M} = \{0,1\}^n$, con $n = \text{poli}(\lambda)$. Sea $G : \{0,1\}^\lambda \to \{0,1\}^n$ una PRG:

- $\text{Gen}(1^\lambda)$: Output $k \sim U_\lambda$.
  Como Gen tiene que correr en tiempo polinomal necesita una entrada de largo λ bits (e.g. 1^λ) para poder generar un output de λ bits
- $\text{Enc}(k, m)$:
  1. $k' \leftarrow G(k)$
  2. output $c = k' \oplus m$
- $\text{Dec}(k, c)$:
  1. $k' \leftarrow G(k)$
  2. output $m = k' \oplus c$

Esquema basado en PRG



Cifrador para $\mathcal{M} = \{0,1\}^n$, con $n = \text{poli}(\lambda)$. Sea $G : \{0,1\}^\lambda \to \{0,1\}^n$ una PRG:

- $\text{Gen}(1^\lambda)$: Output $k \sim U_\lambda$.
  `Como Gen tiene que correr en tiempo polinomal necesita una entrada de largo` λ `bits (e.g.` 1^λ`) para poder generar un output de` λ `bits`
- $\text{Enc}(k, m)$:
  1. $k' \leftarrow G(k)$
  2. output $c = k' \oplus m$
- $\text{Dec}(k, c)$:
  1. $k' \leftarrow G(k)$
  2. output $m = k' \oplus c$

Esquema tiene cifrados indistinguibles

Theorem

Si G es una PRG, entonces esquema anterior tiene cifrados indistinguibles.

Demostraremos por contrapositivo:
Si Π no tiene cifrados ind., entonces G no puede ser PRG. Es decir:

$$\exists \text{ poli } p(\cdot) \text{ y PPT } \mathcal{A} \text{ tal que } \Pr[PrivK^{\text{comp-ind}}_{\mathcal{A},\Pi} = 1] = 1/2 + 1/p(\lambda)$$

$$\Longrightarrow$$

$$\exists \text{ poli } q(\cdot) \text{ y PPT } D \text{ tal que } |\Pr[D(G(U_\lambda)) = 1] - \Pr[D(U_n) = 1]| = 1/q(\lambda)$$

Esta técnica se llama *reducción*: "reduce el problema de distinguir al de quebrar el cifrador".

Demostración

Sea $\mathcal{A}$ tal que $\Pr[PrivK^{\text{comp-ind}}_{\mathcal{A},\Pi}(\lambda) = 1] = 1/2 + 1/p(\lambda)$ para algún polinomio $p(\cdot)$.
Construimos distinguidor D que quiebra la pseudo-aleatoriedad de G:

Distinguidor (o reducción) $D(z)$

1. $m_0, m_1 \leftarrow \mathcal{A}(1^\lambda)$
2. $b \xleftarrow{\$} \{0,1\}$
3. $c = z \oplus m_b$
4. $b' \leftarrow \mathcal{A}(c)$
5. Output 1 si, y sólo si, $b = b'$

Análisis:

- $\Pr[D(G(K)) = 1] = \Pr[PrivK^{\text{comp-ind}}_{\mathcal{A},\Pi}(\lambda) = 1] = 1/2 + 1/p(\lambda)$
- $\Pr[D(U) = 1] = 1/2$

Por lo tanto,
$|\Pr_{K\sim U_\lambda, Z\leftarrow G(K)}[D(Z) = 1] - \Pr_{Z\sim U_n}[D(Z) = 1]| = 1/p(\lambda) \neq \text{negl}(\lambda)$

Extensión de PRGs

Sea $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$ un PRG.
Construiremos un PRG $G^{(n)} : \{0,1\}^\lambda \to \{0,1\}^n$

$G^{(n)}(k)$:

- $k_0 = k$
- For $i \in \{1 \dots n\}$ do:
- $\quad k_i || \sigma_i \leftarrow G(k_{i-1})$ $(k_i \in \{0,1\}^\lambda,\ \sigma_i \in \{0,1\})$
- Output $\sigma = \sigma_1 \sigma_2 \dots \sigma_n$.

Theorem

Si G es PRG, entonces $G^{(n)}$ también es PRG para $n \leq p(n)$ para algún polinomio $p(\cdot)$.

Argumento Híbrido

Sea $H_0 = G^{(n)}(U_\lambda)$ y $H_n = U_n$. Tenemos que demostrar que $H_0 \overset{c}{\equiv} H_n$.
Idea: Creamos *híbridos* H_i para $i \in \{1..n-1\}$ tales que
$H_i \overset{c}{\equiv} H_{i+1}\ \forall i \in \{1..n-1\}$. ¿Por qué funciona?

$$|Pr[D(G^{(n)}(K)) = 1] - \Pr[D(U_n) = 1]| = |Pr[D(H_0) = 1] - \Pr[D(H_n) = 1]|$$

$$(\text{telescópica}) = |\sum_{i=1}^{n} Pr[D(H_{i-1}) = 1] - Pr[D(H_i) = 1]|$$

$$(\text{desigualdad triangular}) \leq \sum_{i=1}^{n} |Pr[D(H_{i-1}) = 1] - Pr[D(H_i) = 1]|$$

$$(H_{i-1} \overset{c}{\equiv} H_i) = \sum_{i=1}^{n} \text{negl}_i(\lambda)$$

$$\leq n \cdot \text{negl}(\lambda) = \text{negl}(\lambda)$$

Demostración vía argumento híbrido

Sea H_i definida como: $U_i||G^{(n-i)}(U_\lambda)$. De esta manera $H_0 = G^{(n)}(U_\lambda)$ y $H_n = U_n$.
Vamos a demostrar que $\forall i\ H_{i-1} \stackrel{c}{\equiv} H_i$ (Por contrapositivo).
Sea $D^{(i)}$ que distingue entre H_i y H_{i-1} para algún $i \in \{1..n\}$, construiremos D^* que distinguie entre $G(U_\lambda)$ y $U_{\lambda+1}$:

Algoritmo distinguidor $D^*(z = z_1 z_2 \dots z_{\lambda+1})$:

1. $u \sim U_{i-1}$
2. $x = u||z_{\lambda+1}||G^{(n-i)}(z_1 z_2 \dots z_\lambda)$
3. $b \leftarrow D^{(i)}(x)$
4. output b.

Análisis: Si z es uniforme, entonces x se distribuye acorde a H_i. Si $z = G(k)$, entonces x se distribuye acorde a H_{i-1}. Por lo tanto, si $D^{(i)}$ distingue entre H_{i-1} y H_i, D^* distingue entre $G(U_\lambda)$ y $U_{\lambda+1}$

Cifradores de Flujo

- Init(s, IV): Dada una semilla s y vector de inicialización IV, retorna estado inicial st_0.
- NextBit(st_i): Output bit σ y nuevo estado st_{i+1} (como en la construción anterior)
- $\Rightarrow$ Podemos cifrar varios mensajes de largo variable:
- Para cada nuevo bit a cifrar, llamanos a NextBit y utilizamos OTP.

Cifradores de Flujo (Stream Ciphers)

- Inicialización: $st_0 \leftarrow Init(s, IV)$
- GetBits(ℓ):
  For $i = 1$ to ℓ
  - $\sigma_i, st_{j+1} \leftarrow$ NextBit(st_j)

  Output $\sigma = \sigma_1 \sigma_2 \ldots \sigma_1 \ell$
- (Para cifrar $m \in \{0,1\}^\ell$, computamos $c = m \oplus \sigma$).

Modos de Operación para cifradores de flujo

Modo sincronizado

- Alice y Bob sincronizan estado del cifrador st_i
- Inicialización: $st_0 \leftarrow Init(k, 0^\lambda)$ (no necesitamos IV)
- $\text{Enc}_{st}(m)$: $k', st' \leftarrow \text{Getbits}(|m|)$, output $c = k' \oplus m$, guardamos nuevo estado st' para próximo mensaje a cifrar.
- $\text{Dec}_{st}(c)$: $k', st' \leftarrow \text{Getbits}(|c|)$, output $m = k' \oplus c$, guardamos nuevo estado st' para próximo mensaje a decifrar.

Modo Asíncrono

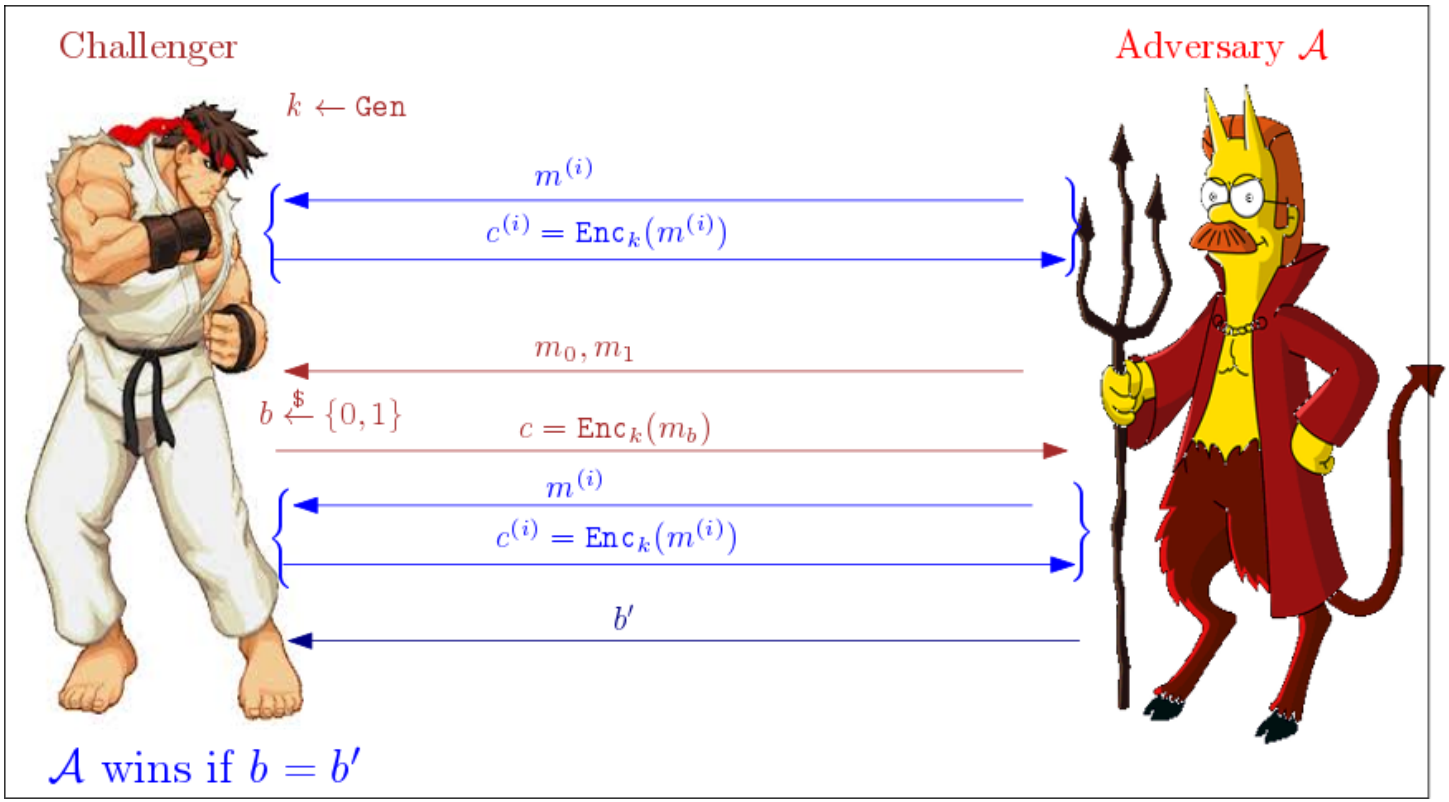
- Alice y Bob *no mantienen estado*.
- Inicialización nueva para cada mensaje.
- $\mathsf{Enc}_k(m)$:
  1. $IV \sim U_\lambda$
  2. $st \leftarrow \mathsf{Init}(k, IV)$
  3. $k' \leftarrow \mathsf{GetBits}(|m|)$
  4. Output $c = \langle k' \oplus m, IV \rangle$
- $\mathsf{Dec}_k(c = \langle c_0, IV \rangle)$:
  1. $st \leftarrow \mathsf{Init}(k, IV)$
  2. $k' \leftarrow \mathsf{GetBits}(st, |m|)$
  3. Output $m = k' \oplus c_0$

Seguridad CPA

Seguridad bajo ataque de textos planos escogidos:

- Captura seguridad en el caso de cifrar varios mensajes.
- Captura situaciones en donde el adversario puede influir en los mensajes a enviar.
  - Adversario tiene acceso a oráculo de encriptación.

$$PrivK^{(ind-cpa)}_{\mathcal{A},\Pi} :$$



Definición Seguridad CPA

Sea $\Pi = \langle \text{Gen}(\cdot), \text{Enc}(\cdot,\cdot), \text{Dec}(\cdot,\cdot) \rangle$ un esquema de cifrados.

Experimento $PrivK^{\text{ind-cpa}}_{\mathcal{A},\Pi}(\lambda)$:

1. $k \leftarrow \text{Gen}(1^\lambda)$
2. $m_0, m_1 \leftarrow \mathcal{A}(1^\lambda)^{\text{Enc}_k(\cdot)}$
3. $b \xleftarrow{\$} \{0,1\}$, $c \leftarrow \text{Enc}_k(m_b)$
4. $b' \leftarrow \mathcal{A}(c)^{\text{Enc}_k(\cdot)}$
5. If $b = b'$, output 1. Else, output 0.

Definition

$\Pi = \langle \text{Gen}(\cdot), \text{Enc}(\cdot,\cdot), \text{Dec}(\cdot,\cdot) \rangle$ *es CPA-seguro* si $\forall$ PPT $\mathcal{A}$, $\exists$ función negligible $\mu(\cdot)$ tal que:

$$\Pr[PrivK^{\text{ind-cpa}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq 1/2 + \mu(\lambda)$$

Nueva Herramienta: Funciones Pseudo-aleatorias

- Idea: Generar un nuevo "pad" para cada mensaje con la misma llave.
- Fácil!: utilizamos una función aleatoria! ¿mmmm?
- ¿Qué es una función aleatoria?
  - Para cada nuevo input x , output $f(x)$ esta uniformemente distribuido.

x	f(x)
00...0	01011
...	...
11...1	01011

- ¿Cómo obtenemos una función aleatoria? No es tan fácil.
- Necesitamos funciones pseudo-aleatorias:
  - Fáciles de obtener: obtenemos una llave $k \sim U_\lambda$
  - Se comportan como si fueran aleatorias si no sabemos k. $F_k(x)$ "parece uniforme".

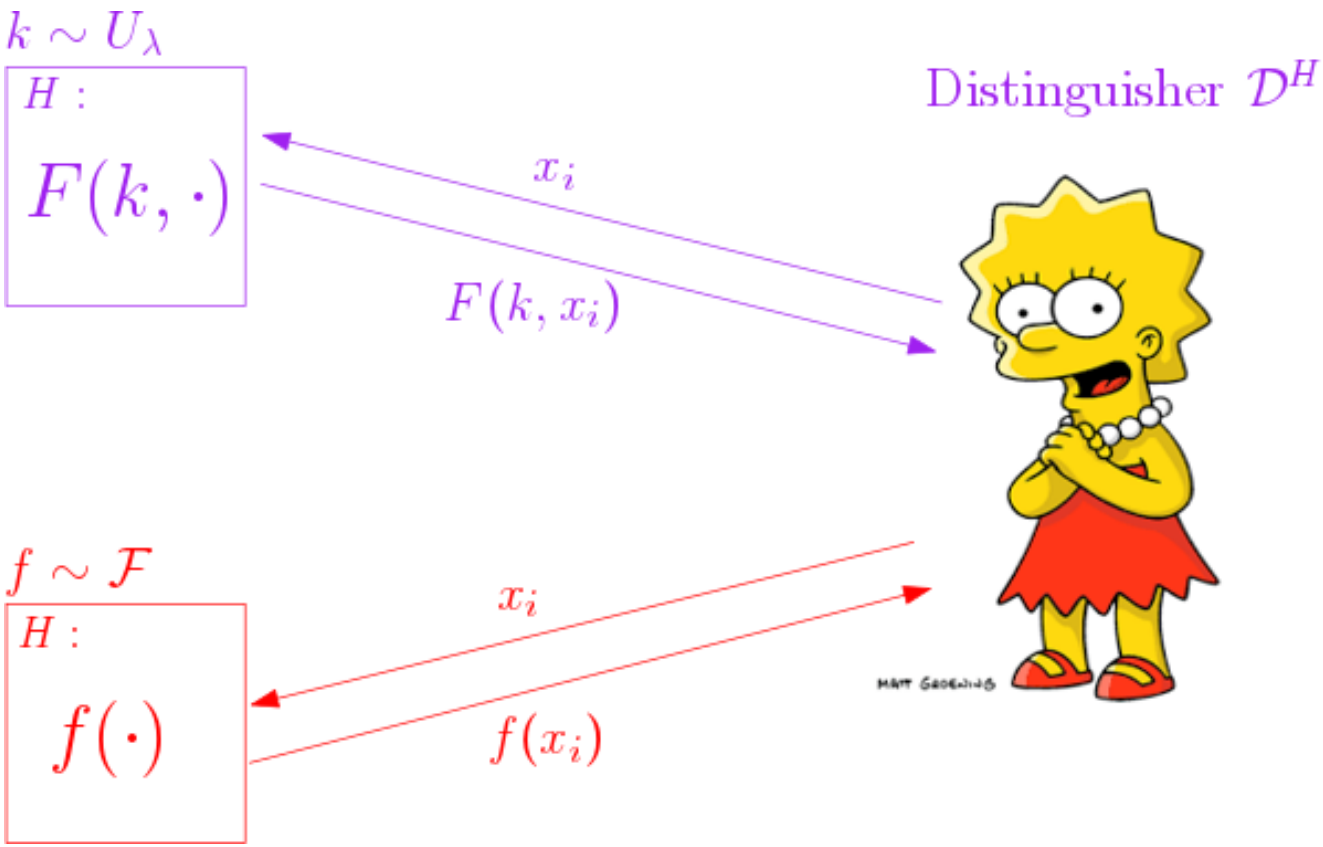
Nueva Herramienta: Funciones Pseudo-aleatorias

- Idea: Generar un nuevo "pad" para cada mensaje con la misma llave.
- Fácil!: utilizamos una función aleatoria! ¿mmmm?
- ¿Qué es una función aleatoria?
  - Para cada nuevo input x , output $f(x)$ esta uniformemente distribuido.

x	f(x)
00...0	01011
...	...
11...1	01011

- ¿Cómo obtenemos una función aleatoria? No es tan fácil.
- Necesitamos funciones pseudo-aleatorias:
  - Fáciles de obtener: obtenemos una llave $k \sim U_\lambda$
  - Se comportan como si fueran aleatorias si no sabemos k. $F_k(x)$ "parece uniforme".

Definición Funciones Pseudo-aleatorias



Definición Funciones Pseudo-aleatorias

Definition (PRFs)

$F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ es pseudo-aleatoria si:

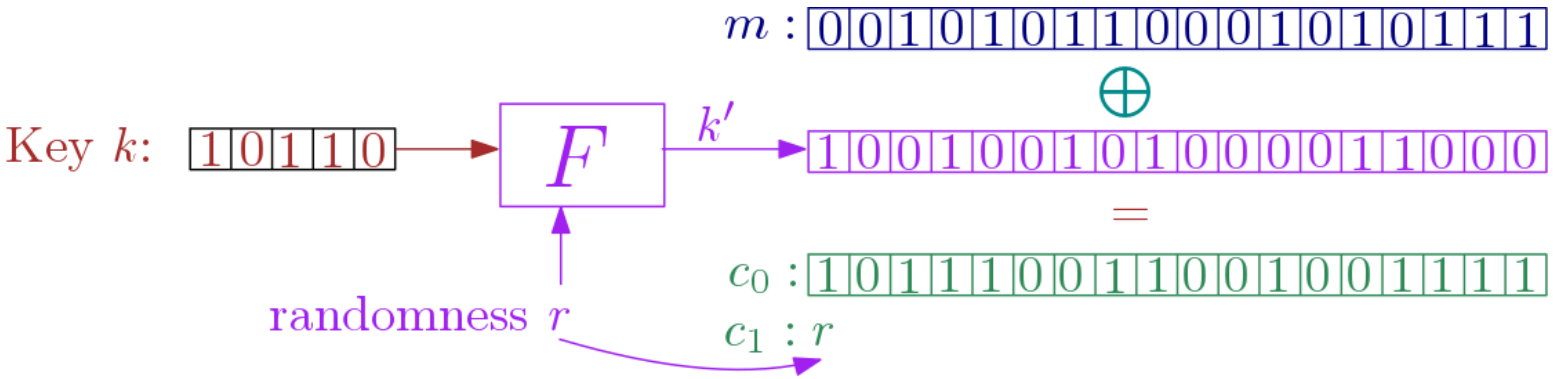
- Dado k y x, $F_k(x)$ es eficientemente computable $\forall x \in \{0,1\}^n$
- Existe función negligible $\mu(\cdot)$ tal que $\forall$ PPT $\mathcal{D}$

$$|\Pr_{k\sim U_\lambda}[\mathcal{D}^{F_k}(\cdot)(1^\lambda)] = 1 - \Pr_{f\sim \mathcal{F}_n^m}[\mathcal{D}^f(1^\lambda) = 1]| = \mu(\lambda)$$

En donde $\mathcal{F}_n^m$ denota distribución uniforme en el conjunto de todas las funciones de $\{0,1\}^n$ a $\{0,1\}^m$.

Cifrador CPA-seguro

Sea $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^n$ una PRF.



$\Pi = \langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$:

- $\mathsf{Gen}(1^\lambda)$: output $k \sim U_\lambda$
- $\mathsf{Enc}_k(m)$:
  1. $r \sim U_\lambda$
  2. $k' \leftarrow F_k(r)$
  3. Output $c = \langle k' \oplus m, r \rangle$
- $\mathsf{Dec}_k(c = \langle c_0, c_1 \rangle)$:
  1. $k' \leftarrow F_k(c1)$
  2. Output $m = k' \oplus c_0$

Demostración de Seguridad (Reducción)

Theorem

Si F es PRF, entonces esquema anterior es CPA-seguro.

Reducción $D^H(1^\lambda)$

Ejercicio ...