One Time Pad. Cifrados de Flujo (*Stream Ciphers*)

Alan Reyes-Figueroa

Criptografía y Cifrado de Información (Aula 05) 27.julio.2021

Cifrados de Shannon

Definición

*Un **cifrado** o **cifrado de Shannon** es una tripla $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, donde*

- *$\mathcal{K}$ es el espacio de claves,*
- *$\mathcal{M}$ es el espacio de mensajes,*
- *$\mathcal{C}$ es el espacio de textos cifrados,*

junto con un par de funciones "eficientes" $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ y $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$, que satisfacen

$$D(\mathbf{k}, E(\mathbf{k}, \mathbf{m})) = \mathbf{m}, \quad \textit{para todo } \mathbf{m} \in \mathcal{M},\ \mathbf{k} \in \mathcal{K}. \qquad \textbf{\textit{(ecuación de consistencia)}}$$

Observaciones:

- E es casi siempre un algoritmo aleatorio, mientras que D es determinista.
- Se espera que los algoritmos E y D sean eficientes: Por ejemplo, que corran en tiempo polinomial. O en la práctica, que no se tarden más de cierta cantidad de tiempo (*e.g.* ≤ 1 minuto en encriptar/decriptar 1G de información).

Cifrados de Shannon

Ejemplo: Mensajes en español, cifrado *Caesar*. En este caso $\mathcal{A} = \{a, b, c, \ldots, x, y, z\}$, y si $\ell > 0$ es una longitud máxima para el mensaje.

$$\mathcal{M} = \mathcal{C} = \mathcal{A}^{\ell}, \qquad \mathcal{K} = \mathcal{A},$$
$$E(\mathbf{k}, \mathbf{m}) = \mathbf{m} + \mathbf{k} \pmod{27}, \qquad D(\mathbf{k}, \mathbf{c}) = \mathbf{c} - \mathbf{k} \pmod{27}.$$

Observe que

$$D(\mathbf{k}, E(\mathbf{k}, \mathbf{m})) = E(\mathbf{k}, \mathbf{m}) - \mathbf{k} \pmod{27} = (\mathbf{m} + \mathbf{k}) - \mathbf{k} \pmod{27} = \mathbf{m} \pmod{27} = \mathbf{m}.$$

Ejemplo: Mensajes en español, a *base64*, cifrado Afín. En este caso $\mathcal{A}_1 = \{a, b, c, \ldots, x, y, z\}$, $\mathcal{A}_2 = \{A, B, \ldots, Z, a, b, \ldots, z, 0, \ldots, 9, +, /\}$ y si $\ell > 0$ es una longitud máxima para el mensaje.

$$\mathcal{M} = \mathcal{A}_1^{\ell}, \qquad \mathcal{C} = \mathcal{A}_2^{2\ell}, \qquad \mathcal{K} = \mathcal{A}_1 \times \mathcal{A}_1,$$
$$E((\mathbf{a}, \mathbf{b})\mathbf{m}) = \mathbf{am} + \mathbf{b} \pmod{27}, \qquad D((\mathbf{a}, \mathbf{b}), \mathbf{c}) = \mathbf{a}^{-1}(\mathbf{c} - \mathbf{b}) \pmod{27}.$$

También se cumple

$$D(\mathbf{k}, E(\mathbf{k}, \mathbf{m})) = \mathbf{a}\big(\mathbf{a}^{-1}(\mathbf{m} - \mathbf{b})\big) + \mathbf{b} \pmod{27} = (\mathbf{m} - \mathbf{b}) + \mathbf{b} \pmod{27} = \mathbf{m}.$$

Propiedades de XOR

Observaciones:

- La operación XOR es asociativa: $(\mathbf{a} \oplus \mathbf{b}) \oplus \mathbf{c} = \mathbf{a} \oplus (\mathbf{b} \oplus \mathbf{c})$.

  Ejemplo: $\mathbf{a} = \texttt{011100101100}$, $\mathbf{b} = \texttt{110101110101}$, $\mathbf{c} = \texttt{101101001001}$

$\mathbf{a}$	0	1	1	1	0	0	1	0	1	1	0	0	
$\mathbf{b}$	1	1	0	1	0	1	1	1	0	1	0	1	$\oplus$
$\mathbf{a} \oplus \mathbf{b}$	1	0	1	0	0	1	0	1	1	0	0	1	
$\mathbf{c}$	1	0	1	1	0	1	0	0	1	0	0	1	$\oplus$
$(\mathbf{a} \oplus \mathbf{b}) \oplus \mathbf{c}$	0	0	0	1	0	0	0	1	0	0	0	0	

$\mathbf{b}$	1	1	0	1	0	1	1	1	0	1	0	1	
$\mathbf{c}$	1	0	1	1	0	1	0	0	1	0	0	1	$\oplus$
$\mathbf{b} \oplus \mathbf{c}$	0	1	1	0	0	0	1	1	1	1	0	0	
$\mathbf{a}$	0	1	1	1	0	0	1	0	1	1	0	0	$\oplus$
$\mathbf{a} \oplus (\mathbf{b} \oplus \mathbf{c})$	0	0	0	1	0	0	0	1	0	0	0	0	

Propiedades de XOR

Observaciones:

- Cada cadena es su propio inverso: $(\mathbf{a} \oplus \mathbf{a}) = \mathbf{0}$.

  Ejemplo: $\mathbf{a} = \texttt{011100101100}$

$$
\begin{array}{c|cccccccccccc}
\mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & \quad \oplus \\
\hline
\mathbf{a} \oplus \mathbf{a} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

- La cadena $\mathbf{0}$ funciona como el neutro de XOR: $(\mathbf{a} \oplus \mathbf{0}) = \mathbf{a}$.

  Ejemplo: $\mathbf{a} = \texttt{011100101100}$

$$
\begin{array}{c|cccccccccccc}
\mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \quad \oplus \\
\hline
\mathbf{a} \oplus \mathbf{0} & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0
\end{array}
$$

Cifrado One Time Pad

Recordemos el cifrado que trabajamos en el aula anterior, en donde a una cadena de bits **m** de longitud *n*, se le hace XOR con una cadena de bits aleatoria **k**, bit a bit:

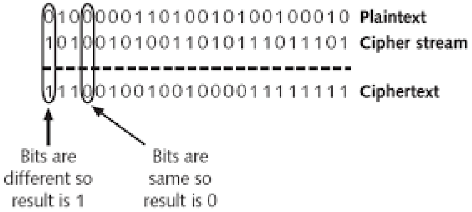


Figure 11-10 Creating ciphertext with XOR

Este método se conoce com cifrado ***One Time Pad***.

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Cifrado *One Time Pad* (OTP): (Cifrado de Vernam, 1917).

$$\mathcal{M} = \mathcal{C} = \{0,1\}^n, \qquad \mathcal{K} = \{0,1\}^n \qquad \text{(cadenas de n bits).}$$

Así, una clave será una cadena de bits, de la misma longitud del mensaje.

La funciones de encriptado y decriptado son las siguientes:

$$E(\mathbf{k},\mathbf{m}) = \mathbf{k} \oplus \mathbf{m} = \mathbf{k} + \mathbf{m} \pmod 2, \qquad E(\mathbf{k},\mathbf{c}) = \mathbf{k} \oplus \mathbf{c} = \mathbf{k} + \mathbf{c} \pmod 2,$$

donde $\mathbf{a} \oplus \mathbf{b}$ es la función XOR de las cadenas $\mathbf{a}$ y $\mathbf{b}$.

Veamos que E y D cumplen la condición de consistencia:

$$\begin{aligned} D(\mathbf{k}, E(\mathbf{k},\mathbf{m})) &= D(\mathbf{k}, \mathbf{k} \oplus \mathbf{m}) = \mathbf{k} \oplus (\mathbf{k} \oplus \mathbf{m}) = (\mathbf{k} \oplus \mathbf{k}) \oplus \mathbf{m} \\ &= \mathbf{0} \oplus \mathbf{m} = \mathbf{m}. \end{aligned}$$

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Cifrado One Time Pad

Ahora, dado el mensaje **m** y su texto cifrado **c**, fácilmente se puede determinar la clave usada en el One Time Pad. Usando las propiedades de XOR, tenemos que

$$\mathbf{c} = \mathbf{m} \oplus \mathbf{k} \quad \Rightarrow \quad \mathbf{k} = \mathbf{o} \oplus \mathbf{k} = (\mathbf{m} \oplus \mathbf{m}) \oplus \mathbf{k} = \mathbf{m} \oplus (\mathbf{m} \oplus \mathbf{k}) = \mathbf{m} \oplus \mathbf{c}.$$

Así, para recuperar la clave, es suficiente con tener un par del tipo $(\mathbf{m}, \mathbf{c})$, texto plano, texto cifrado.

Una propiedad importante del cifrado *One Time Pad* es que es muy rápido de encriptar y decriptar. Basta hacer la operación XOR a nivel de bits en dos cadenas.

Sin embargo, es difícil de usar en la práctica: Para enviar un mensaje de longitud n, se requiere una clave también de longitud n.

Seguridad en Teoría de Información

La idea de seguridad de un cifrado, utiliza conceptos e ideas de la teoría de la información (Shannon, 1949). Publicado como *Communication Theory of Secrecy Systems*), mientras trabajaba en los Bell Labs.

Idea: El texto cifrado **c** no debe revelar información sobre el mensaje original **m**.

Definición

Un cifrado de Shannon (E, D) *sobre el espacio* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ *posee* ***secreto perfecto*** *(perfect secrecy) si para cualesquiera mensajes* $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}$, *con* $len(\mathbf{m}_0) = len(\mathbf{m}_1)$, *y para cualquier texto cifraco* $\mathbf{c} \in \mathcal{C}$, *vale*

$$\mathbb{P}(E(\mathbf{k}, \mathbf{m}_0)) = \mathbf{c}) = \mathbb{P}(E(\mathbf{k}, \mathbf{m}_1) = \mathbf{c}), \quad \forall \mathbf{k} \in \mathcal{K},$$

cuando $\mathbf{k}$ *es una variable aleatoria con distribución uniforme en* $\mathcal{K}$, $\mathbf{k} \sim U(\mathcal{K})$.

Esto es, conociendo **c**, ni el adverario más poderoso puede aprender algo sobre el mensaje original **m**.

La definición anterior, dice que para un cifrado con secreto perfecto, no es posible realizar ataques de frecuencia, conociendo únicamente el texto cifrado **c**. (Sin embargo, otro tipo de ataques es posible).

Teorema

El cifrado One Time Pad posee secreto perfecto.

Prueba: Asumiento que **k** se elige de forma aleatoria con distribución uniformen en $\mathcal{K}$, para todo **m** $\in \mathcal{M}$, y todo **c** $\in \mathcal{C}$, se tiene

$$\mathbb{P}(E(\mathbf{k},\mathbf{m}) = \mathbf{c}) = \frac{|\{\mathbf{k} \in \mathcal{K} : E(\mathbf{k},\mathbf{m}) = \mathbf{c}\}|}{|\mathcal{K}|}.$$

Basta entonces mostrar que $\{\mathbf{k} \in \mathcal{K} : E(\mathbf{k},\mathbf{m}) = \mathbf{c}\}$ tiene tamaño constante.
Pero, dados **m** y **c**, entonces $E(\mathbf{k},\mathbf{m}) = \mathbf{c} \;\Rightarrow\; \mathbf{k} \oplus \mathbf{m} = \mathbf{c} \Rightarrow\; \mathbf{k} = \mathbf{m} \oplus \mathbf{c}$, de modo que sólo hay una clave posible. Así

$$\mathbb{P}(E(\mathbf{k},\mathbf{m}) = \mathbf{c}) = \tfrac{1}{|\mathcal{K}|}, \quad \forall\, \mathbf{m}, \forall\, \mathbf{c}.$$

Malas noticias:

- El cifrado OTP no es seguro, aunque tenga secreto perfecto (existen otros ataques que lo hacen vulnerable).
- La mala noticia es que en la práctica no es muy eficiente: como ya mencionamos, para transmitir un mensaje de longitud n, antes se debe transmitir una clave también de longitud n.
  (Duplica el trabajo en el envío de información).

Teorema

Secreto perfecto $\Rightarrow \quad |\mathcal{K}| \geq |\mathcal{M}|$.

Corolario

Secreto perfecto $\Rightarrow \quad len(\mathbf{k}) \geq len(\mathbf{m})$.

Veremos ahora cómo el OTP se puede hacer eficiente y que se pueda implementar en la práctica.

Cifrados de Flujo (*Stream Ciphers*):

Idea: Reemplazar la clave aleatoria, por una clave pseudo-aleatoria.

Definición

*Un **generador pseudo-aleatorio** (PRG) es una función* $G : \{0,1\}^s \to \{0,1\}^n$*, donde* $n >> s$*. El espacio* $\{0,1\}^s$ *se llama el espacio semilla.*

Ejemplo: Una función que convierte una clave de 128 bits, en una clave mucho mucho más extensa.

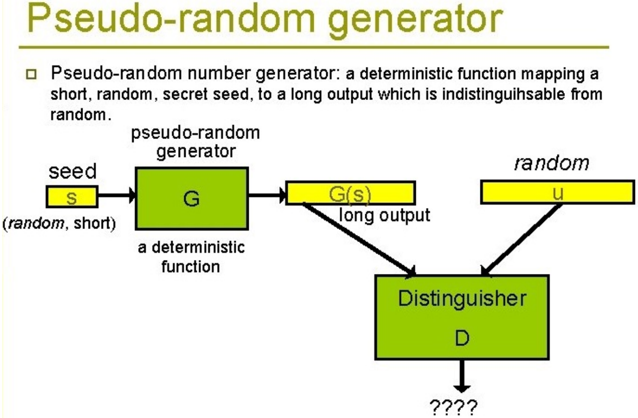
$$G : \texttt{1011001011111010} \longrightarrow \texttt{1011101011} \ldots \texttt{1010011010}.$$

Se espera que G cumpla con algunas propiedades:

- $n >> s$.
- $G(\mathbf{k})$ debe verse aleatoria.
- G debe calcularse de forma eficiente.

Cifrados de Flujo

Podemos pensar a un generador pseudo-aleatorio como una función G que transforma cadenas cortas $\mathbf{s}$ en cadenas de bits $G(\mathbf{s})$ mucho más largas, con propiedades estadísticas que las hacen indistinguibles de una cadena aleatoria:



Cifrados de Flujo

Obs! Los cifrados de flujo son seguros, sin embargo, no tienen la propiedad de secreto perfecto (porque el tamaño del espacio clave es menor que el tamaño del espacio de mensajes $|\mathcal{K}| < |\mathcal{M}|$).

Entonces, necesitamos una noción diferente de seguridad.

secret Denotamos por $G(\mathbf{k})\big|_{1:i} = \mathbf{msb}_i(G(\mathbf{k}))$ a los primeros i bits de la salida $G(\mathbf{k})$.

Definición

Sea $G : \{0,1\}^s \to \{0.1\}^n$ un generador pseudo-aleatorio. G es ***predecible****, si existe i, con $1 \leq i < n$, tal que*

$$\textit{conocer } G(\mathbf{k})\big|_{1:i} \;\Rightarrow\; \textit{conocer } G(\mathbf{k})\big|_{i+1:n}.$$

Obs Un PRG predecible conduce a brechas de seguridad.
(Por ejemplo, en un caso donde se conozca un prefijo, se podría predecir el resto del mensaje original).

Definición

*Decimos que un generador pseudo-aleatorio $G : \mathcal{K} \to \{0,1\}^n$ es **predecible**, si existe un algoritmo eficiente $\mathcal{A}$, y existe i, con $1 \leq i < n$, tal que*

$$\mathbb{P}\big(\mathcal{A}(G(\mathbf{k}))\big|_{1:i} = G(\mathbf{k})\big|_{i+1}\big) \geq \tfrac{1}{2} + \varepsilon,$$

para alguna $\varepsilon > 0$ no despreciable. (e.g. $\varepsilon \geq 2^{-30}$), para $\mathbf{k} \in \mathcal{K}$ aleatorias con distribución uniforme sobre $\mathcal{K}$.

Definición

*Decimos que un generador pseudo-aleatorio $G : \mathcal{K} \to \{0,1\}^n$ es **inpredecible**, si no es predecible. Esto es, si no existe un algoritmo eficiente $\mathcal{A}$, e i, con $1 \leq i < n$, tal que*

$$\mathbb{P}\big(\mathcal{A}(G(\mathbf{k}))\big|_{1:i} = G(\mathbf{k})\big|_{i+1}\big) \geq \tfrac{1}{2} + \varepsilon,$$

para alguna $\varepsilon > 0$ no despreciable.

Generadores Lineales (*Linear Congruence Generator*).

Algoritmo: (PRG lineal).
Inputs: $a, b, p \in \mathbb{Z}^+$, con p primo, $(a, p) = 1$.
Outputs: $\mathbf{k} \in \{0, 1\}^n$, cadena de bits.
`Initialize` $\mathbf{r}[0] = seed$.
`for i = 1 to n:`
    $\mathbf{r}[i] = a\mathbf{r}[i-1] + b \pmod p$
    Append first bits of $\mathbf{r}[i]$ to $\mathbf{k}$

- Tiene propiedades estadísticas bonitas.
- Sin embargo, es fácil de predecir.

La función *random()* en la librería estándar *glibc* de C.

c
glibc.

Algoritmo: (glibc).
Inputs: n el tamaño de la cadena de bits.
Outputs: $\mathbf{k} \in \{0, 1\}^n$, cadena de bits.
`Initialize` $\mathbf{r}[0 : 32] = seed$.
`for i = 1 to n:`

$$\mathbf{r}[i] = \mathbf{r}[i-3] + \mathbf{r}[i-32] \pmod{2^{32}}$$

Append $\mathbf{r}[i] >> 1$.