



Criptografía y Seguridad

Criptografía:
Cifrado Simétrico

Criptosistema simétrico (repaso)

- Es una terna de algoritmos
 - Gen (algoritmo de generación de claves)
 - Enc (cifrado): $\text{Enc}_k(m)$
 - Dec (descifrado): $\text{Dec}_k(c)$
- Propiedades
 - La salida de Gen define K el espacio de claves.
 - La entrada de Enc define el espacio de mensajes
 - La entrada de Dec define el espacio de mensajes cifrados
 - Para todo m y k válidos: $\text{Dec}_k(\text{Enc}_k(m))=m$

Secreto perfecto (repaso)

- Definición: Un criptosistema (gen, enc, dec) posee la propiedad de secreto perfecto sobre un espacio de mensajes M si:
 - Para toda distribución de probabilidades en M , cada mensaje m y cada mensaje cifrado c tal que $\Pr[C = c] > 0$:
 - $\Pr[M=m \mid C = c] = \Pr[M=m]$
 - Es una forma de decir que el texto plano y el cifrado son probabilísticamente independientes
 - $\Pr[C=c \mid M=m_0] = \Pr[C=c \mid M=m_1]$

One Time Pad

- Atribuido a Verman (1917)

- Gen: $k \leftarrow \{0, 1\}^n$
- Enc: $e_k(m) = m \oplus k$
- Dec: $d_k(c) = c \oplus k$

El criptosistema de Verman (one time pad), define el mensaje como un xor del mensaje (en binario) con la clave.

De esta forma, tenemos funciones ENC y DEC muy faciles, ya que la forma de deshacerse de un XOR es volver a hacer el mismo XOR.

En este caso, n es la longitud del mensaje.

Esta transformacion de verman, cumple la definicion de secreto perfecto.

- Ejemplo:

Aunque el one time pad es muy bueno, tiene 3 limitaciones:

- La longitud de la clave debe ser \geq la longitud del mensaje a cifrar (shannon demuestra que esta condicion es necesaria y no suficiente para un mensaje perfecto)
- Si la clave se utiliza mas de una vez para mensajes distintos, es trivial descifrar otros textos
- La clave debe ser 100% aleatoria, si hay cualquier tipo de sesgo probabilistico ya no sirve

M=	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0
K=	0	1	1	0	0	1	1	1	0	1	0	0	1	1	0	1	0
<hr/>																	
C=	0	1	0	0	1	0	1	0	0	1	0	1	1	0	1	0	0

Ejercicio

Suponer que nuestro texto plano NO es aleatorio:

Cual es la probabilidad de tener el texto cifrado 01

$$P(p=00) = 0.60 \quad P(p=01) = 0.15$$

$$P(p=10) = 0.10 \quad P(p=11) = 0.15$$

Calcular $P(C=c)$ para cada valor de c , si se utiliza un One Time Pad

Ejercicio



Probar que el One Time Pad cumple con la propiedad de secreto perfecto

One Time Pad

- El OTP tiene secreto perfecto
 - Para cualquier par (m, c) de longitud n
 - $\Pr [C = c \mid M = m]$
 - $= \Pr [M \oplus K = c \mid M = m]$
 - $= \Pr [m \oplus K = c]$
 - $= \Pr [K = m \oplus c]$
 - $= 1/2^n$
 - ↳ una clave x en particular
 - Como $k \in \{0,1\}^n \Rightarrow \Pr[k = k] = 1/2^n$
 - Como vale para todo par, entonces:
 - $\Pr[C = c \mid M = m_0] = 1/2^n = \Pr[C = c \mid M = m_1]$

One Time Pad

- Las malas noticias

- Secreto perfecto $\Rightarrow |K| \geq |C|$

- Si una clave se utiliza 2 veces:

- $c_1 = m_1 \oplus k$

- $c_2 = m_2 \oplus k$

! jamás cifrar 2 mensajes con misma clave

- $\Rightarrow c_1 \oplus c_2 = m_1 \oplus m_2$

este msj tiene estructura \Rightarrow se puede extraer información parcial

- Se pierde el secreto perfecto

- La clave seleccionada DEBE ser aleatoria

- Esto forma parte de la prueba formal de seguridad

One Time Pad

- ¿Por que la clave debe ser aleatoria?



Ejercicio

Considerar una clave no aleatoria:

$$P(k=00) = 0.3 \text{ y } P(k=01) = 0.1$$

$$P(k=10) = 0.4 \text{ y } P(k=11) = 0.2$$

Si la distribución del texto plano es:

$$P(p=00) = 0.60 \quad P(p=01) = 0.15$$

$$P(p=10) = 0.10 \quad P(p=11) = 0.15$$

Y se obtiene un mensaje $C=01$ ¿Cuál es el texto plano más probable?

Ejercicio

- Queremos calcular $P(p=x \mid c=01)$ para todos los valores de p .
- Estadísticamente P y C no son independientes:
 - $P(p=x \mid c=y) * P(c=y) = P(c=y \mid p=x) * P(p=x)$
 - Veamos con $P(p=00 \mid c = 01)$:

$$P(p=00 \mid c=01) = \frac{P(c=01 \mid p=00) * P(p=00)}{P(c=01)}$$

$$\text{Sea } C_k = \{e(k, p) : p \in P\}$$

$$\begin{aligned} P(c=01) &= \sum_{k, c \in C_k} P(K=k) * P(P=d(k, c)) \\ &= P(K=00) * P(P=01) + P(K=01) * P(P=00) \\ &\quad + P(K=10) * P(P=11) + P(K=11) * P(P=10) \\ &= 0.3 * 0.15 + 0.1 * 0.6 + 0.4 * 0.15 + 0.2 * 0.10 \\ &= 0.185 \end{aligned}$$

Ejercicio

$$P(p=00|c=01) = \frac{P(c=01|p=00) * P(p=00)}{P(c=01)}$$

- ¿Cómo calculamos $P(c=01 | p=00)$?

$$\begin{aligned} P(C=01|P=00) &= \sum_{\{k: 00=d_k(01)\}} P(K=k) \\ &= P(K=01) = 0.1 \end{aligned}$$

- Juntando todo:

$$P(p=00|c=01) = \frac{0.1 * 0.6}{0.185} = 0.32$$

- Calcular las probabilidades de los otros textos planos y determinar el mas probable

Mas allá del OTP

- Resultado teóricos interesantes
 - Cualquier criptosistema con secreto perfecto es reducible al OTP
 - Cualquier sistema que no sea reducible al OTP no posee secreto perfecto
- Consecuencias
 - El secreto perfecto es demasiado impráctico
 - Necesitamos otras construcciones

Seguridad Computacional

Secreto perfecto = Seguridad incondicional



Seguridad Computacional

Secreto perfecto = Seguridad incondicional



- Limitar escenarios
- Limitar garantías

Seguridad computacional

Seguridad Computacional



Garantizar seguridad
solo contra
adversarios “limitados”

Asume un limite en los
recursos del atacante
(especialmente tiempo).

Obs: modelamos atacantes como algoritmos
PERO limitamos a algoritmos que no tengan tiempo
de ejecución exponencial

Aceptar una pequeña
probabilidad de éxito
para el atacante

Se deja de lado la
infalibilidad

Criptosistemas de flujo

- Intercambian la clave del OTP por la salida de un generador pseudoaleatorio:

- Gen: $s \leftarrow S$

Se genera una semilla de 128 caracteres.

La idea atrás de los sistemas de flujo es usar una clave mucho mas corta que el mensaje.

- Enc: $\text{enc}_s(m) = G(s) \oplus m$

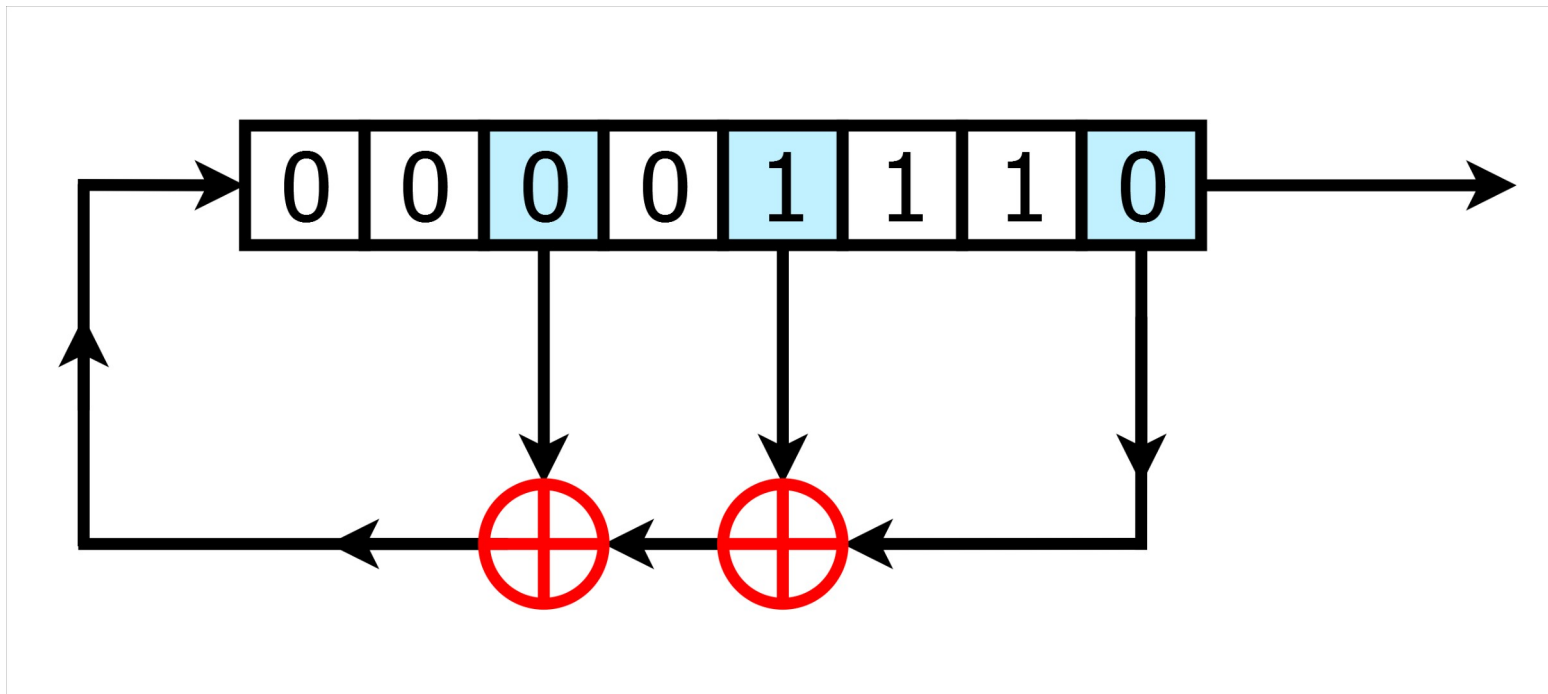
- Dec: $\text{dec}_s(c) = G(s) \oplus c$

La gran diferencia: $|S| \lll |M|$

Por ejemplo: $|S| = 2^{128}$, $|M| = |S|^{128}$

Generadores pseudoaleatorios

- Son algoritmos determinísticos
- Expanden una entrada llamada semilla (*seed*)
- La salida parece aleatoria
- Ejemplo:



Generadores pseudoaleatorios

- Formalmente: No esta demostrado que existan generadores realmente pseudoaleatorios bajo esta definicion
- Sea $D = \{ f: \{0,1\}^n \rightarrow \{0,1\} \}$ una familia de funciones
- $G: \{0,1\}^s \rightarrow \{0,1\}^n$, con $s < n$ es un generador pseudoaleatorio con respecto a D si:

Para toda f en D :

$$P(f(G(r^s)) \neq f(r^n)) = \varepsilon$$

Probabilidad generalizada

Secuencia realmente aleatoria

Valor despreciable

El generador es pseudoaleatorio si las funciones me responden de la misma manera si le doy una secuencia generada y una secuencia totalmente aleatoria. Actualmente existe un cuerpo de mas de 200 de estas funciones, aunque nada impide que alguien invente una funcion nueva y encuentre que mi secuencia no es aleatoria

Ejemplo

- Sea $s = \{ 1, \dots, 10 \}$
- $G(s) = \{ G_0 \% 2, G_1 \% 2, G_2 \% 2, \dots, G_n \% 2 \}$
 - $G_0 = s$
 - $G_i = G_{i-1} * 3 + 1 \bmod 11$
- Generar 5 bits con $s = 2$
- Generar 5 bits con $s = 6$

Criptosistemas de flujo

- Intercambian la clave del OTP por la salida de un generador pseudoaleatorio:
 - Gen: $s \leftarrow S$
 - Enc: $\text{enc}_s(m) = G(s) \oplus m$
 - Dec: $\text{dec}_s(c) = G(s) \oplus c$

La pregunta que surge es: dado que no existe el secreto perfecto, como medimos la seguridad de un criptosistema?

Las pruebas matematicas no nos sirven, ya que todas van a devolver que el criptosistema no es seguro (debido a la inexistencia del secreto perfecto)

¿Cómo medimos la seguridad del criptosistema ?

Pruebas de seguridad

- Prueban características de un criptosistema
- Son una serie de pasos que prueban un algoritmo (que representa un ataque)
- El atacante gana o pierde la prueba
- Se puede repetir múltiples veces
 - Interesa la probabilidad de éxito del atacante

Prueba de indistinguibilidad

- Eavesdropping Indistinguishability test: $E_{av_{A,\Pi}}$
- Dado un adversario A , y un Criptosistema Π :

1) A emite m_0 y m_1 a su criterio

el atacante crea dos mensajes

2) Se genera una clave $k \leftarrow K$

El criptosistema genera una clave secreta

3) Se genera $b \leftarrow \{0, 1\}$

El criptosistema elige un numero entre 0 y 1

4) Se calcula $c \leftarrow \text{Enc}_k(m_b)$ y se le envía a A

El criptosistema le manda uno de los mensajes (m_0 o m_1) ya cifrado al atacante

5) A emite $b' = \{0, 1\}$

El atacante tiene que elegir si el mensaje recibido viene de m_0 o de m_1

- $E_{av_{A,\Pi}} = 1$ si $b = b'$ (A gana)

El adversario gana cuando decide que la probabilidad de que el mensaje recibido sea el mensaje 0 (o el mensaje 1, es lo mismo) se acerca al 50%. Para esto, epsilon debe ser cercano a 0.

Si $\Pr[E_{av_{A,\Pi}}=1] = 0,5 + \epsilon \Rightarrow \Pi$ es indisting.

La idea de esta prueba es someter a escrutinio la idea de que si yo consigo obtener un texto cifrado, me es imposible ver el mensaje original.

La idea es reducir el problema a donde todas las ventajas estan del lado del adversario.

No nos sirve decir que el sistema pasa la prueba si el atacante tiene probabilidad 0 de elegir la opcion correcta, porque si el atacante se da cuenta va a adivinar siempre.

La desventaja de este criptosistema es que dependemos de los algoritmos atacantes ya existentes. Si alguien inventa uno nuevo, podemos tener problemas.

Sin embargo, hay miles de expertos que pasaron años tratando de romper los criptosistemas que usamos y no pudieron.

Nivel de seguridad

- Nivel de seguridad

Supongamos que tenemos dos criptosistemas que pasan la prueba de indistinguibilidad y queremos ver cual es mas seguro. El parametro n (nivel de seguridad) esta asociado al tamaño del espacio de claves. Mientras mayor sea este, mas seguro es el criptosistema.

- Está dado por una variable
- Relaciona la cota en el poder de un adversario y la probabilidad de éxito que tendrá

- Dado un nivel de seguridad n se espera que:

- Un adversario corra algoritmos de orden $PPT(n)$ (Probabilistic Polynomial Time)
- La probabilidad de éxito sea una función despreciable en n :

- $\epsilon(n)$ es despreciable $\leftrightarrow \lim \epsilon(n) < 1 / n^k$

No vamos a aceptar un adversario con capacidad de computo exponencial. Porque esto sería hacer fuerza bruta, e introduce el concepto de un atacante con capacidad de computo infinita, que es un requerimiento que relajamos.

Prueba de indistinguibilidad

- Eavesdropping Indistinguishability test: $\text{Eav}_{A,\Pi}$
- Dado un adversario $A(n)$, y un Criptosistema $\Pi(n)$:

- 1) A emite m_0 y m_1 a su criterio
- 2) Se genera una clave $k \leftarrow K$
- 3) Se genera $b \leftarrow \{0, 1\}$
- 4) Se calcula $c \leftarrow \text{Enc}_k(m_b)$ y se le envía a A
- 5) A emite $b' = \{0, 1\}$

- $\text{Eav}_{A,\Pi} = 1$ si $b = b'$ (A gana)

Si $\Pr[\text{Eav}_{A,\Pi}=1] = 0,5 + \varepsilon(n) \Rightarrow \Pi$ es indisting.


Criptosistemas de flujo

- Intercambian la clave del OTP por la salida de un generador pseudoaleatorio:
 - Gen: $s \leftarrow S$
 - Enc: $\text{enc}_s(m) = G(s) \oplus m$
 - Dec: $\text{dec}_s(c) = G(s) \oplus c$
- Teorema:
 - Si $G(s)$ es un generador pseudoaleatorio entonces el criptosistema es indistinguible ante observadores

Si G cumple la definicion matematica de generador pseudoaleatorio, entonces el criptosistema pasa la prueba de indistinguibilidad.

Ejercicio

∃ una función D /
 $D(G(s)) = 1$ y $D(\text{rnd}) = 0$



Demostrar que si es posible distinguir $G(s)$ de una secuencia aleatoria un criptosistema de flujo basado en $G(s)$ no pasa la prueba EAV

Criptosistemas de flujo

- Quedan definidos por el Generador utilizado
- Ejemplos reales:

- RC4 (usado en https y WEP)
- CSS (usado en DVDs)
- A51 (GSM)
- E0 (Bluetooth)

Todos con problemas

- Salsa20: $\{0,1\}^{128 \text{ o } 256} \times \{0,1\}^{64} \rightarrow \{0,1\}^n, n=2^{64} \times 2^9$

Estos son los mas usados

- Rabbit: $\{0,1\}^{128} \times \{0,1\}^{64} \rightarrow \{0,1\}^n, n=2^{128}$

Nonce (lo veremos más adelante)

Estado de un criptosistema

- Criptosistema seguro
 - Cumple con las expectativas de su modelo de seguridad



Estado de un criptosistema

- Criptosistema debilitado
 - Existen adversarios con probabilidades no despreciables de éxito
 - Pero el esfuerzo es muy alto (ej. decadas) o las condiciones muy dificiles (ej. disponer de 2^{80} mensajes)

Se considera practico un ataque que necesita un año de ejecucion o menos.
Si se cumple esa condicion, se dice que el criptosistema esta quebrado.



Cuando un criptosistema esta debilitado, no es necesario correr a cambiarlo pero hay que ir pensando en implementar uno nuevo.
Para un criptosistema quebrado, hay que ir corriendo a cambiarlo.

Estado de un criptosistema

- Criptosistema quebrado
 - Existen adversarios con probabilidades no despreciables de éxito en tiempos practicables



Escenarios

- Un criptosistema puede ser seguro y estar quebrado al mismo tiempo
 - Hay múltiples pruebas de seguridad
 - Cada una prueba un escenario diferente

La seguridad es relativa a un escenario de ataque.



Múltiples cifrados

- Multiple message eavesdropping test: $\text{Mul}_{A,\Pi}$
- Dado un adversario A , y un Criptosistema Π :

1) A emite $(m_{00}, m_{01}, \dots, m_{0i})$ y $(m_{10}, m_{11}, \dots, m_{1i})$

2) Se genera una clave $k \leftarrow K$

3) Se genera $b \leftarrow \{0, 1\}$

4) Se calculan $c_i \leftarrow \text{Enc}_k(m_{bi})$ y se le envían a A

5) A emite $b' = \{0, 1\}$

↓
vectores de mensajes
cifrados con misma
clave

- $\text{Mul}_{A,\Pi} = 1$ si $b = b'$ (A gana)

Si $\Pr[\text{Mul}_{A,\Pi}=1] = 0.5 + \varepsilon \Rightarrow \Pi$ es indisting.

Es muy parecido al eavesdropping común. La única diferencia es que el atacante en vez de generar dos mensajes, genera dos listas de mensajes. El atacante recibe todos los mensajes cifrados de una de esas listas. Si pasa esta prueba, podemos distinguir que el criptosistema puede reutilizar una clave sin problemas. Sin embargo, se puede demostrar de forma general que NINGUN criptosistema de flujo (así como el One Time Pad) funciona con múltiples cifrados.

Una forma interesante de hacer que un criptosistema no pase esta prueba es que la lista 1 tenga todos los mensajes iguales, y la lista 2 tenga todos los mensajes distintos. Entonces el atacante puede saber fácilmente cual de las dos listas recibió, demostrando que es inseguro que el criptosistema reutilice la clave. Esta es la forma que se usa para demostrar que el One Time Pad no pasa esta prueba.


Ejercicio


- Los criptosistemas de flujo NO son seguros bajo múltiples cifrados
 - $c_1 = m_1 \oplus G(s)$
 - $c_2 = m_2 \oplus G(s)$
 - $\Rightarrow c_1 \oplus c_2 = m_1 \oplus m_2$

Utilizar este hecho para definir un ataque que gane la prueba Mul

Solución

Definir un Algoritmo A que gane la prueba de múltiples cifrados

- $A \rightarrow (m_{00}=0\dots 0, m_{01}=0\dots 0), (m_{10}=0\dots 0, m_{11}=1\dots 1)$
- A obtiene c_1, c_2
 - $X = c_1 \oplus c_2 = m_1 \oplus m_2$  aplicamos criptosistema de flujo
 - Si $x = 0\dots 0 \rightarrow b=0$
 - Si no $\rightarrow b=1$

 2 mensajes iguales
y 2 distintos

Necesidad de cifrado probabilístico

- Si una función de cifrado es determinística, NO es segura bajo múltiples cifrados
- El adversario anterior aplica a cualquier criptosistema donde $e_k(x)$ es constante



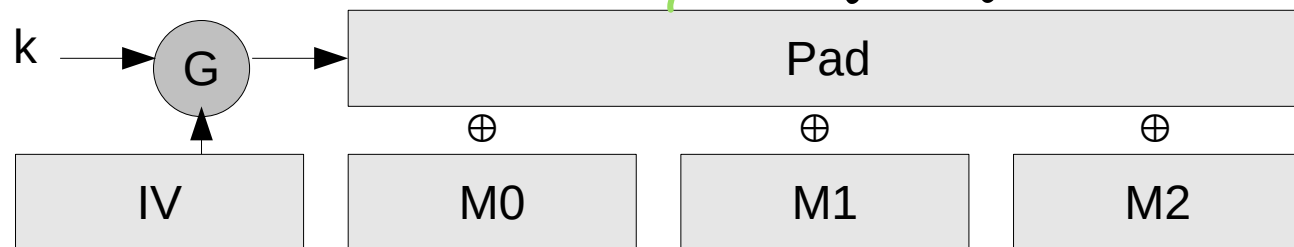
Evitar reutilizar la clave

- Se agrega un valor a la función generadora

- Dicho valor no deberá repetirse para una misma clave (se lo llama **nonce** o **IV**)

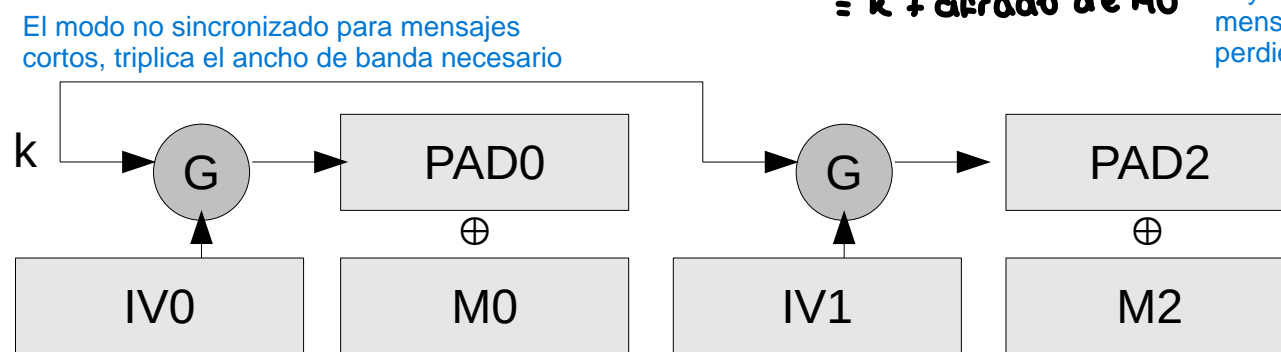
- Dos formas:

- Modo sincronizado



sec aleatoria no secreta
entrada de generador = (k, IV)
(no comprate la seguridad que sea publica)
me tengo que guardar nuevo estado = bit donde estoy parado

- Modo no sincronizado



clave de $M1$
= $k + \text{cifrado de } M0$

PROBLEMA: el IV es largo.
Si yo quiero mandar muchos mensajes cortitos, estoy perdiendo espacio

longitud determinado x el criptosistema

⚠ Guardar el IV para descifrar

Cada vez que quiero reutilizar una clave, genero un IV y hago un XOR entre la clave y el IV.
El IV tiene la misma longitud que la clave, entonces es como si le aplicáramos un One Time Pad a la clave (porque hacemos un XOR bit a bit entre el IV y la clave).
Esto no compromete la seguridad, porque estamos cambiando la entrada del generador.
El IV no viaja cifrado, viaja en plano. La seguridad radica en que yo no repito este IV, por lo que no hay problema en que el atacante la conozca..

Este metodo soluciona el ataque del EavesDropping multiple, dado que por mas que le mande una lista de mensajes con el mismo mensaje repetido, todos estos se crigran distintos (porque tienen la misma clave pero distintos IV)

Ataques de texto plano escogido

- Chosen Plain Text indistinguishability: $\text{CPA}_{A,\Pi}$
- Dado un adversario A , y un Criptosistema Π :

En este ataque, le vamos a dar la capacidad al atacante de cifrar la cantidad de mensajes que el quiera. Además de eso, la prueba es parecida a EavesDropping simple: el atacante emite dos mensajes, recibe dos mensajes cifrados y tiene que elegir cual es. No hay ninguna restricción en los mensajes que el atacante puede cifrar, entonces puede mandar a cifrar los mismos mensajes que envía.

- 1) Se genera una clave $k \leftarrow K$
- 2) A obtiene $f(x) = \text{Enc}_k(x)$ y emite (m_0, m_1)
- 3) Se genera $b \leftarrow \{0, 1\}$
- 4) Se calcula $c \leftarrow \text{Enc}_k(m_b)$ y se le envía a A
- 5) A emite $b' = \{0, 1\}$

↳ genera mensajes arbitrarios y obtiene los cifrados

- $\text{CPA}_{A,\Pi} = 1$ si $b = b'$ (A gana)

Conclusion: si el criptosistema es determinístico, es trivial romperlo con esta prueba. El atacante manda a cifrar los dos mensajes que luego envía al sistema.

Si $\Pr[\text{CPA}_{A,\Pi}=1] = 0.5 + \varepsilon \Rightarrow \Pi$ es indisting.

⇒ ningún criptosist determinístico es indistinguible en este ataque

Este tipo de ataques aparece en protocolos de ida y vuelta donde hay cifrados. Por ejemplo, en un sistema de envío de emails cifrados, en donde el atacante puede enviarse mails y verlos tanto cifrados como descifrados.

Propiedades CPA

- Un criptosistema determinístico no puede ser CPA-Secure
- Si un criptosistema es CPA-Secure para un mensaje también lo es para multiples
- Un criptosistema que es CPA-Secure, pero de tamaño limitado (cifra mensajes de hasta n bits) puede ser extendido arbitrariamente
 - $m = m_0 || m_1 || \dots || m_i$ ($|m_j| = n$ bits)
 - $enc_k(m) = enc_k(m_0) || enc_k(m_1) || \dots || enc_k(m_i)$
 - Esto da origen a los criptosistemas de bloque

Esto es porque no es deterministico, aunque tiene una demo media complicada

Basicamente dividimos al mensaje en n partes y ciframos cada parte del mensaje con nuestro cifrado. Esto se conoce como criptosistema de bloque. Además esto nos ayuda con que los procesadores no son buenos multiplicando numeros muy grandes (de 256 bits por ejemplo)

Primitivas de cifrado en bloque

- Están definidos para mensajes de tamaño FIJO
 - $K = \{0, 1\}^n$, $C=P = \{0, 1\}^b$
 - Obs: son determinísticas
⇒ NO usar como criptosistemas
 - Gen: $k \leftarrow K$
 - Enc: $\text{enc}_k(m)=c$
 - Dec: $\text{dec}_k(c)=m$
- Son primitivas y no criptosistemas
 - Forman criptosistemas al combinarse en diferentes modos

Extensión y encadenamiento

- ¿Que ocurre el mensaje a cifrar es más chico que el tamaño de bloque?

Aca se hace extension

- Se lo extiende sistemáticamente:

- Simple Pad:

- Completar con ceros

- Es necesario conocer el tamaño real del mensaje

Hay dos formas de hacer extension. Simple pad (padding ambiguo), en donde se completa al final del mensaje con 0s para llegar al tamaño. Es ambiguo porque yo estoy modificando el mensaje que entra.

Para encontrar el mensaje original, empiezo a sacar 0s. Pero si el mensaje original terminaba en 0, voy a tener problemas.

Es por esto que Simple Pad solo se puede usar cuando yo ya se la longitud del mensaje.

Por ejemplo, en el protocolo SSH ya se sabe que se esta mandando una sola letra, entonces es sabido que son 8bits.

Cuando no se sabe la longitud del mensaje, recurrimos al padding no ambiguo. El ejemplo mas basico de este se llama Des Pad, y la idea es agregar un 1 y despues todos 0.

Para obtener el mensaje original, voy sacando todos los 0s del final hasta encontrar el primer bit en 1.

El problema con este es un caso borde, en el cual la longitud del mensaje es la misma que el tamaño del bloque, en donde tengo que agregar un bloque entero de padding.

- Des Pad:


- Agregar un bit 1 y luego bits en 0

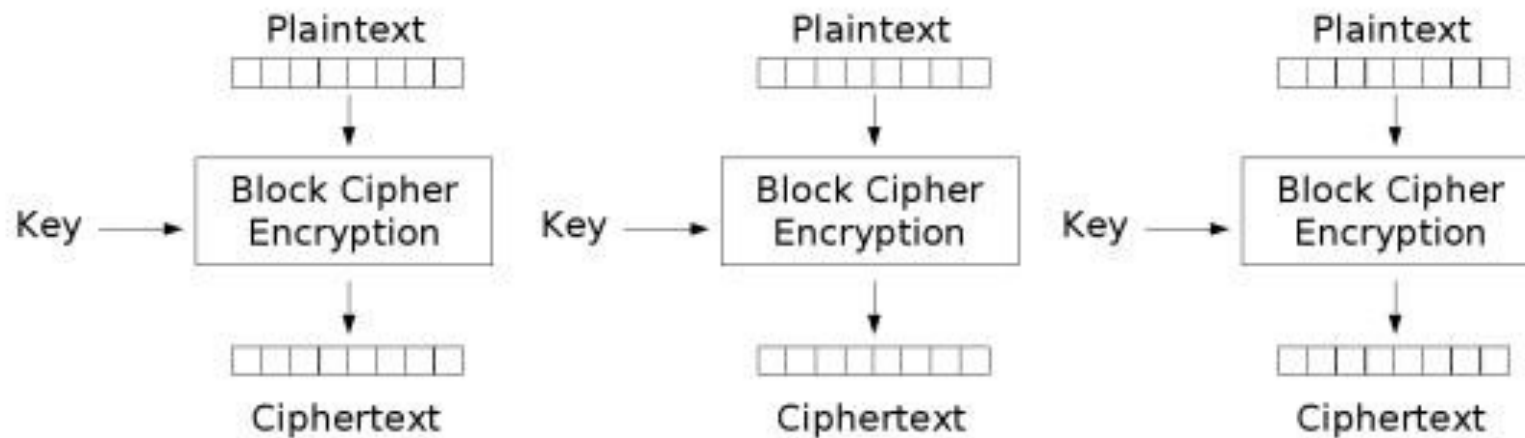
- Puede agregar un bloque completo de padding

Extensión y encadenamiento

- ¿Que ocurre el mensaje a cifrar es más grande que el tamaño de bloque?
 - Se divide el mensaje en bloques
 - Se extiende el ultimo bloque
 - Se transforma cada bloque según algún modo de encadenamiento
- Modos de encadenamiento
 - Objetivo → extender una primitiva de cifrado a bloques mayores a su tamaño
 - Hay diversos modos con diferentes propiedades
 - No todos son aplicables a cada problema

Encadenamiento ECB

- Trata al mensaje original como un conjunto de bloques independientes  Este mecanismo transmite mucho info*



Electronic Codebook (ECB) mode encryption

Yo podría pensar que cada una de estas unidades es un mensaje distinto que estoy cifrando con la misma clave. Entonces estaría usando la misma clave con un cifrado determinístico, lo que termina siendo inseguro.

**NO es CPA-Secure
(NO UTILIZAR)**



Encadenamiento CBC

Este encadenado se usa y es seguro.

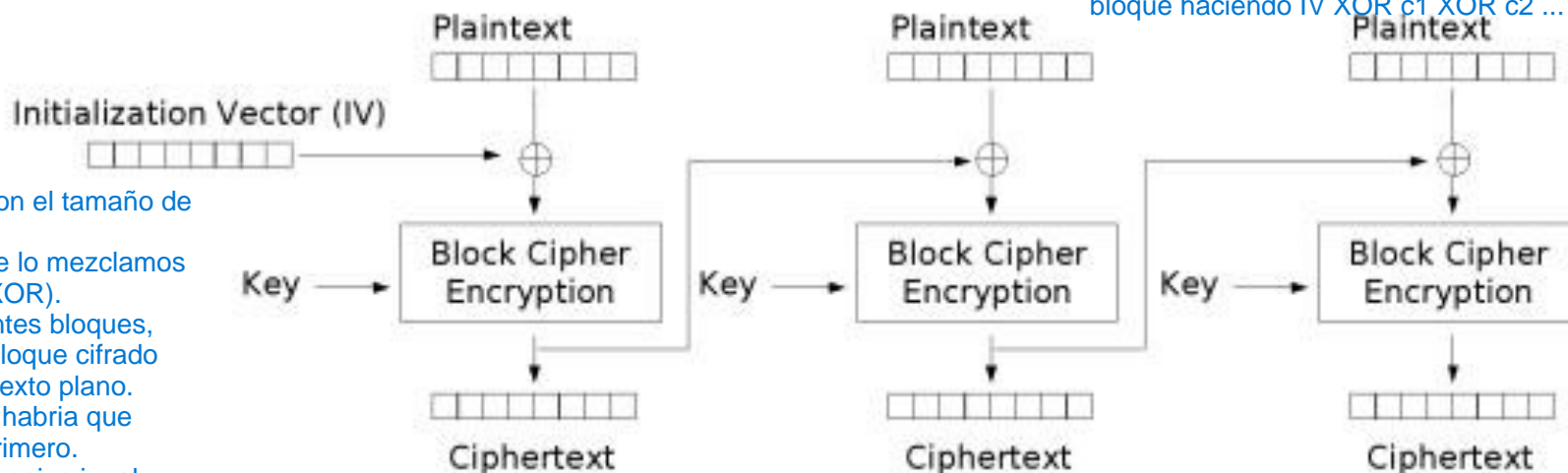
- Utiliza la salida de un bloque como entrada para el próximo
 - Disminuye el traspaso de información
 - Requiere un valor inicial (IV) ALEATORIO

IV = vector de inicializacion

Obs: si 2 textos son iguales \Rightarrow cifrados distintos

El problema con CBC es que es cifrado secuencial. Para cifrar el bloque N, yo necesito usar el bloque N-1 ya cifrado. Sin embargo, el descifrado no es secuencial (entonces se puede paralelizar), porque yo puedo calcular el IV de cada bloque haciendo $IV \text{ XOR } c_1 \text{ XOR } c_2 \dots$

Genero un IV con el tamaño de mi bloque.
El primer bloque lo mezclamos con el IV (con XOR).
Para los siguientes bloques, mezclamos el bloque cifrado anterior con el texto plano.
El unico IV que habria que generar es el primero.
Si tuviera n mensajes iguales, siguen teniendo cifrados distintos.
Con esto nos ahorramos el problema de generar un IV distinto por cada bloque que mando.



Cipher Block Chaining (CBC) mode encryption

\Rightarrow es CPA Secure

Si se cifra mal un bloque, el error no se propaga al resto de los bloques cifrados, ya que luego los podre descifrar correctamente

Encadenamiento CFB

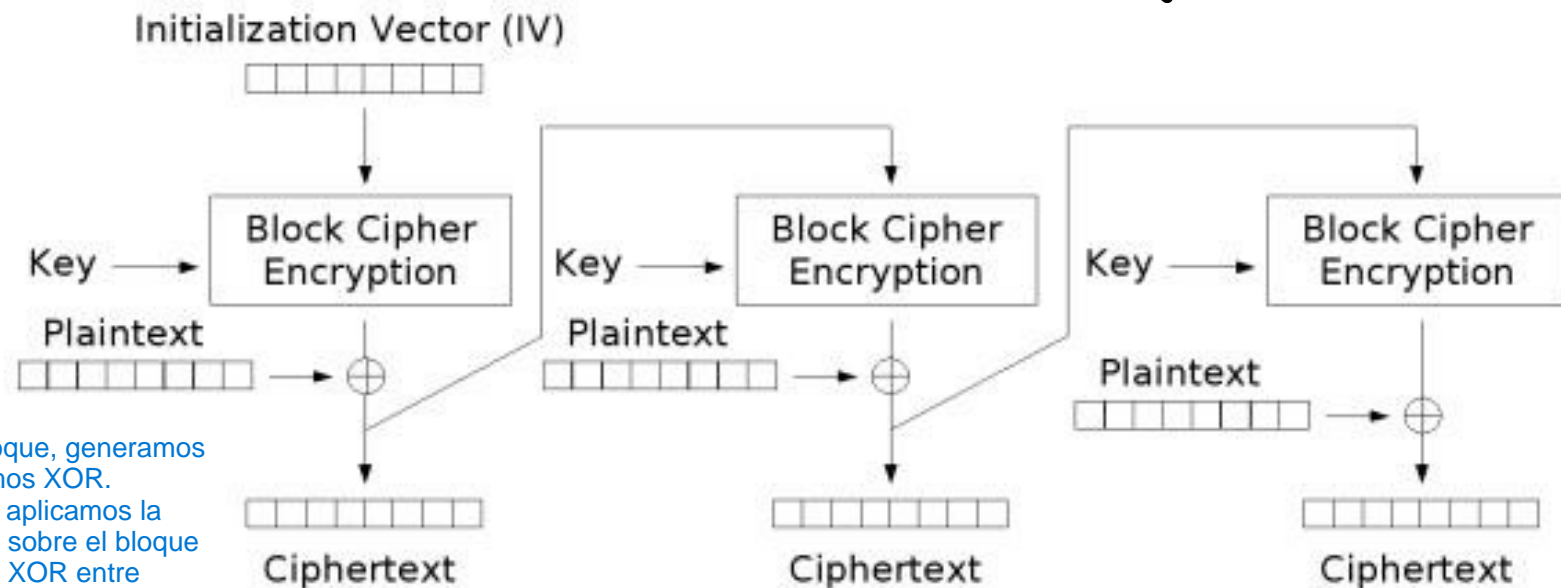
Este encadenado no se usa mucho, solo en legacy

- Utiliza solo la función Enc

- Menor complejidad para dispositivos embebidos
- Permite generar una transformación de flujo a partir de una de bloque

1. Se cifra IV

2. XOR con msj



Cipher Feedback (CFB) mode encryption

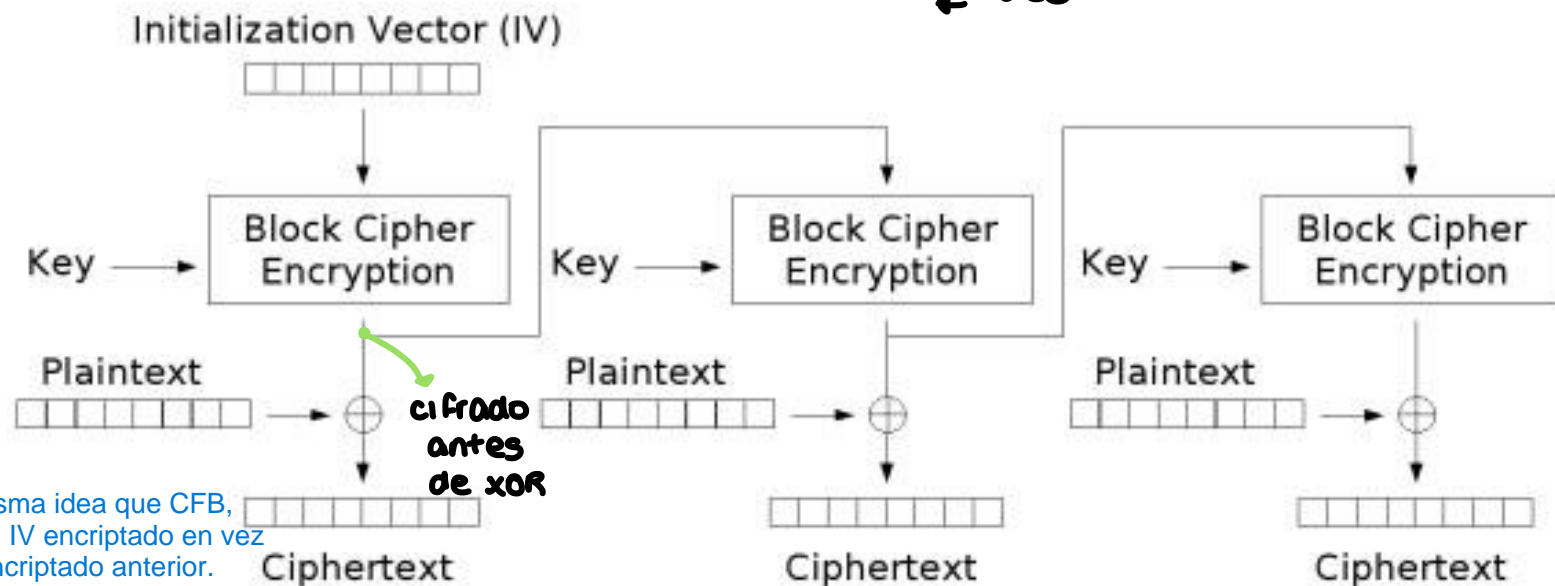
En este cifrado:
Para cifrar el primer bloque, generamos un IV aleatorio y hacemos XOR.
Para cifrar el bloque N, aplicamos la función de cifrado ENC sobre el bloque cifrado N-1, y hacemos XOR entre ambos.
Desaparece la necesidad de tener una función DEC, dado que para descifrar el mensaje, solo necesito el IV, y la función ENC. Esto era muy eficiente para textos embebidos.

Encadenamiento OFB

- Construye una transformación de flujo a partir de una de bloque
 - Permite calcular los bits de la transformación por adelantado

⇒ el **cifrado** es **independiente del mensaje**

⇒ enc ↔ dec



Output Feedback (OFB) mode encryption

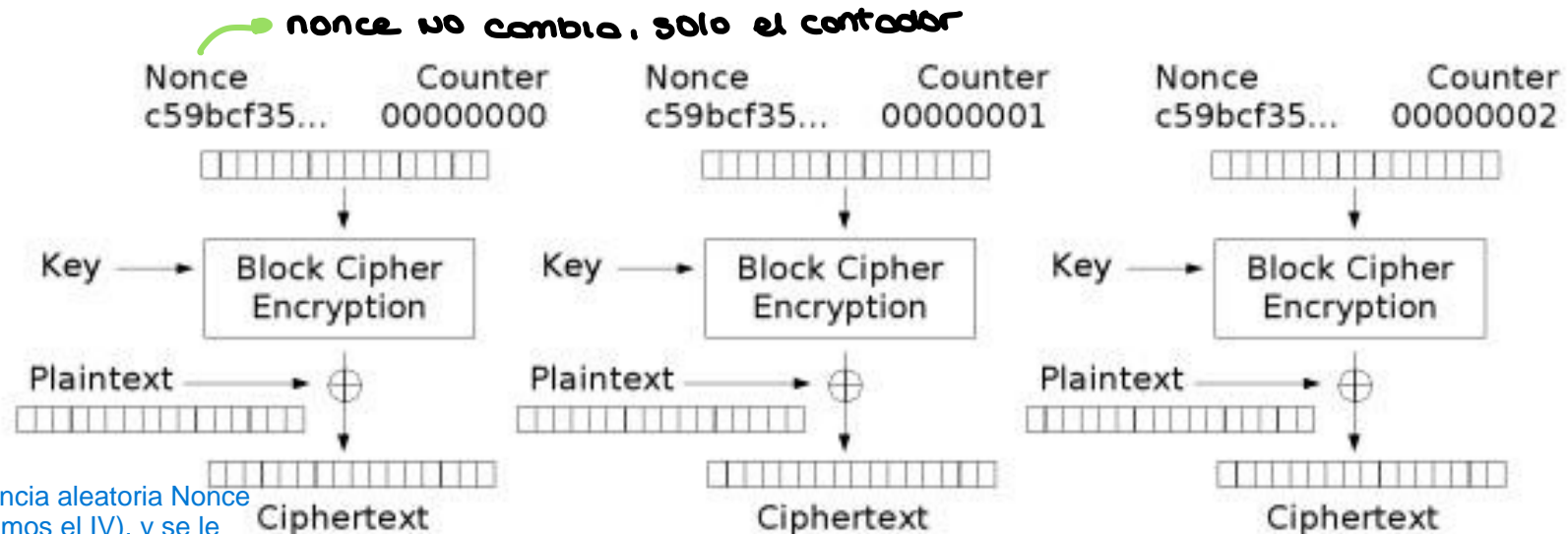
Es practicamente la misma idea que CFB, pero en este usamos el IV encriptado en vez de la salida del texto encriptado anterior. En este modo, nosotros podriamos precalcular estas claves de antemano aplicando la funcion de ENC N veces sobre el IV. Esto de precalcular las claves sumaba en eficiencia, pero ahora no se usa porque tenemos mucha capacidad de computo.

Encadenamiento Counter

- Utiliza un contador para generar secuencias de bits de clave

El gran problema de los metodos anteriores es la secuencialidad, donde tengo que cifrar todo lo anterior para cifrar algo. El counter me deja cifrar y descifrar de manera aleatoria (por ejemplo, para cifrar un dvd en donde todo el tiempo accedo de manera aleatoria).

- Permite acceso aleatorio
- Muy útil en ambientes con capacidad de procesamiento paralelo



Counter (CTR) mode encryption

Se genera una secuencia aleatoria Nonce (lo que antes llamabamos el IV), y se le suma un counter. Luego se cifra ese Nonce + Counter, lo que nos asegura que siempre tendremos una clave pseudoaleatoria. Una vez que ya tenemos esta nueva clave, aplicamos XOR con el bloque a cifrar. Esto permite acceso aleatorio, dado que puedo cifrar y descifrar en paralelo.

Este metodo depende mucho de la funcion de encriptacion, porque necesitaria que con solo cambiar un poquito el counter (y por ende el nonce), tengo que tener una clave completamente aleatoria.

Seguridad de cifrado por bloques

- Las pruebas son por reducción a propiedades de la primitiva subyacente
 - Requieren que la primitiva se comporte como una función pseudoaleatoria
 - Esto significa que no es posible distinguir la función $f(x) = \text{Enc}_k(x)$ de una función tomada al azar del conjunto de funciones del mismo dominio.

Seguridad de cifrado por bloques

Si la primitiva es una función pseudoaleatoria

- CBC es CPA-Secure con IVs aleatorios
- OFB, CFB son CPA-Secure con IVs aleatorios
- Counter es CPA-Secure si no se repite (k, nonce)



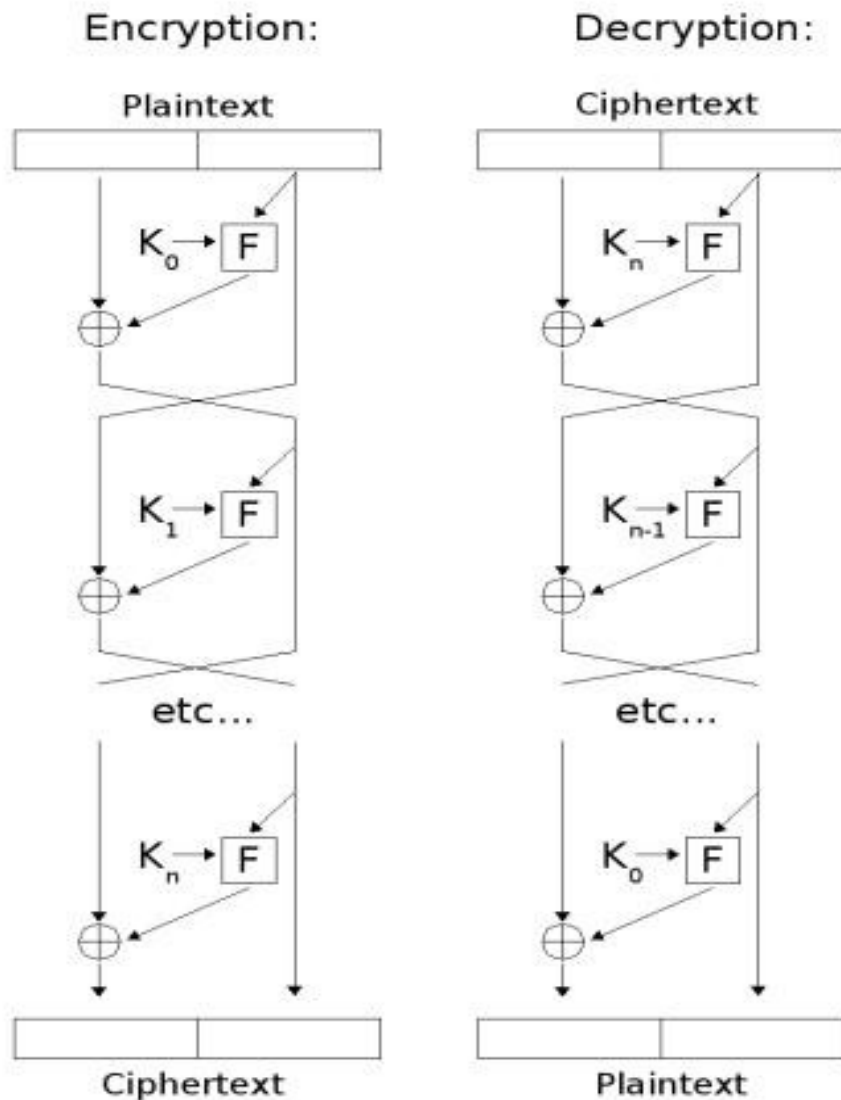
No esta demostrado que existan las funciones pseudoaleatorias

DES – Data Encryption Standard

- Método desarrollado por IBM
- Adoptado por el gobierno de EE.UU. como estandard para usos no militares
 - Fue la primera función de cifrado pública apoyada por un gobierno
 - Alcanzó uso comercial masivo
- Características
 - Trabaja con textos planos binarios
 - Entrada: 64 bits
 - Clave: 64 bits (56 bits efectivos)
 - Salida: 64 bits

El espacio de claves es 2^{56} , porque en ese momento se usaban 7 bits por cada 8 (el octavo era de paridad, porque las computadoras eran malas en ese momento)

DES – Data Encryption Standard

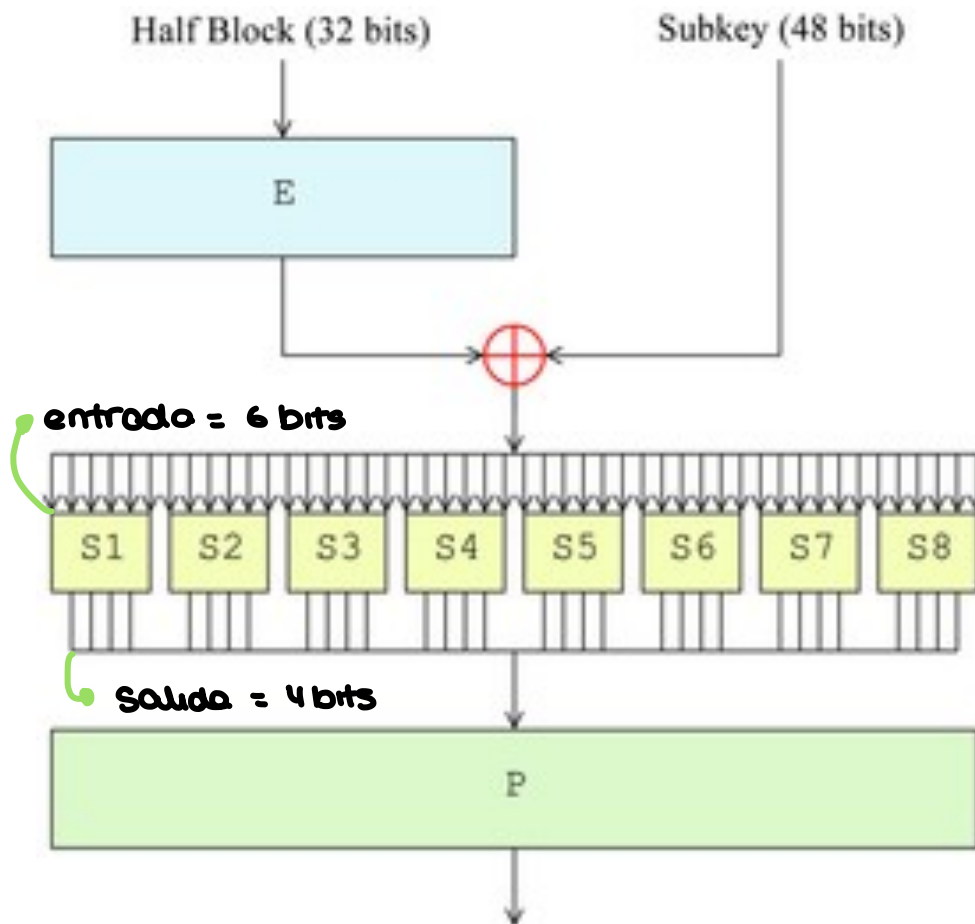


Feistel Cipher

- Se parte el mensaje en dos mitades
- Se transforma una mitad con parte de la clave
- Se intercambian las dos mitades de lugar
- Se repiten los tres pasos anteriores (una ronda) 16 veces

Siempre tiene que se un numero par de rondas

La función de transformación



La función F es una primera transformación que mezcla la mitad del bloque con la subclave (la mitad de la clave). Como tienen tamaños distintos, aparece una función de transformación.

$E \rightarrow$ Expansión

32 a 48 bits

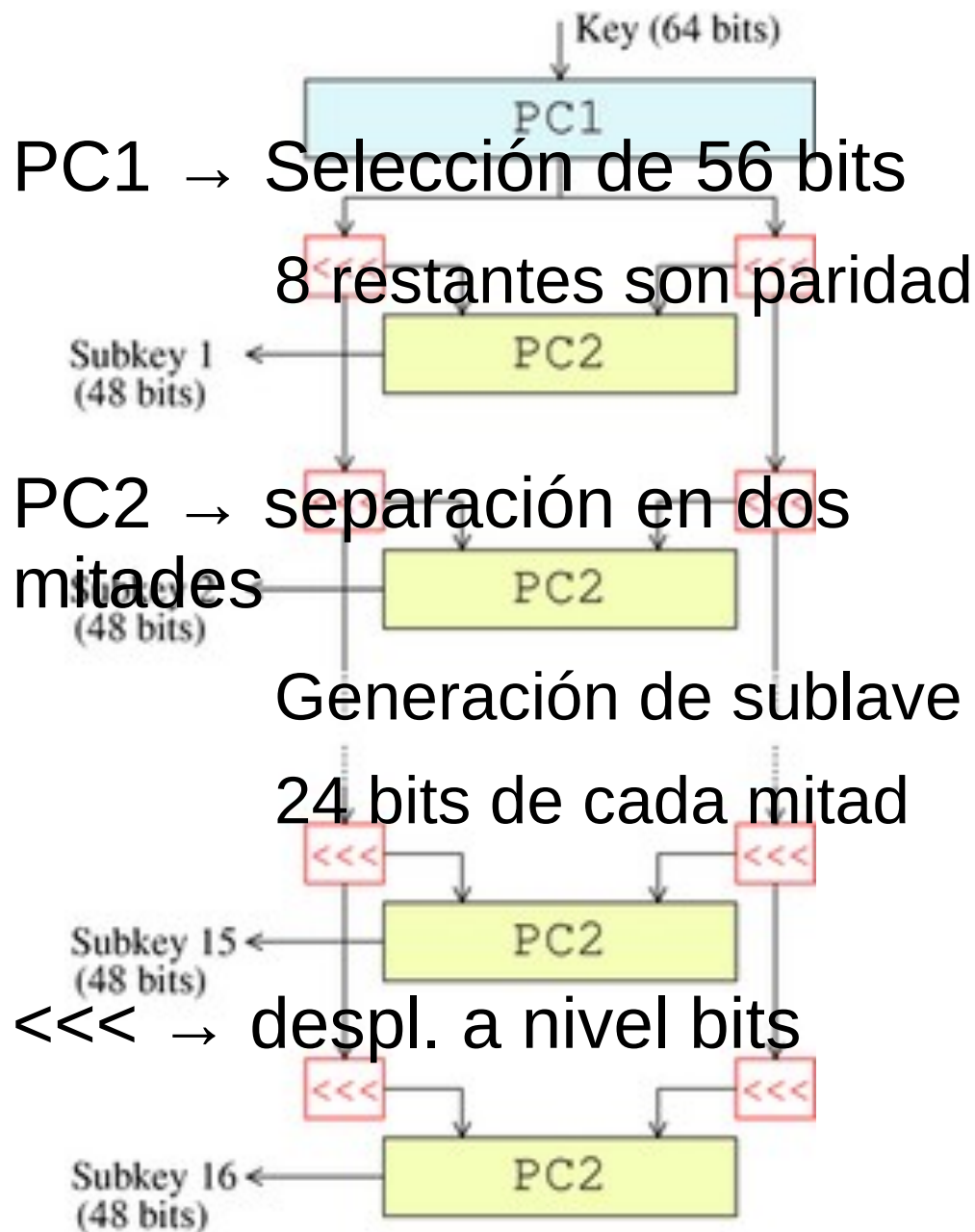
$S_x \rightarrow$ Sustituciones

6 a 4 bits

Sustitución
monoalfabética con
pérdida de información

$P \rightarrow$ Permutación

Generación de subclaves



Evolución

- Nivel teórico de seguridad: 56 bits
 - Fuerza bruta: probar 2^{56} claves
- Generó desconfianza debido a cuestiones de diseño confidenciales (sustituciones)
- 1990 - Criptoanálisis diferencial
 - Permite atacar a DES en 2^{47} intentos
 - Gran parte del diseño de DES disminuye el impacto de este ataque (otros sistemas fueron inmediatamente quebrados)
- 1992 – Criptoanálisis lineal
 - Permite atacar a DES en 2^{43} intentos

Cuando IBM presenta DES, publica todo el algoritmo pero nunca explica por que eligieron los valores para la parte de la permutacion. Muchos investigadores se dan cuenta que si se eligen otros valores, se pierde la seguridad del criptosistema.

2^{43} es algo que se puede ejecutar en una computadora de entrecasa. Por este motivo, DES no se usa mas.

3-DES

- Variante fuerte de DES
- Consisten en seleccionar 3 claves independientes y calcular

- $c = \text{Enc}_{k1}(\text{Dec}_{k2}(\text{Enc}_{k3}(p)))$

Para invertir esto, hay que encontrar la clave $k1$, $k2$ y $k3$, por lo que es mucho mas difícil. Ciframos, desciframos y ciframos en lugar de cifrar 3 veces por una cuestion ingenieril. Esto es porque si ponemos $k2=k3$ podemos hacer que DES sea compatible con 3DES.

- Brinda una seguridad del orden de 112 bits

Es solo 2/3 del espacio de claves, esto es suboptimo

- Notar que es menor a 168 bits!
 - Debido a un ataque conocido como meet-in-the middle

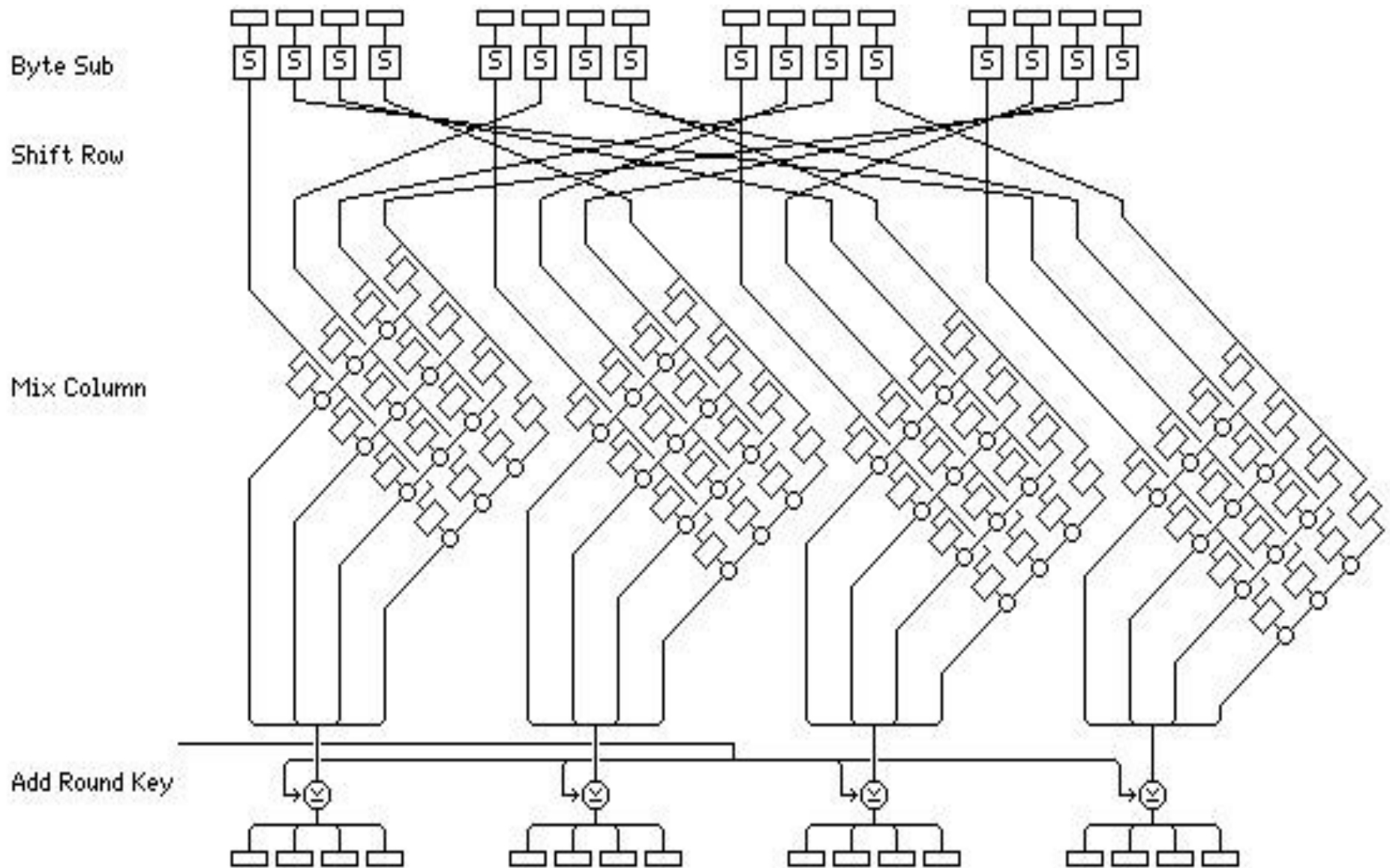
La idea de este ataque es tomar el texto plano y cifrarlo con todas las posibles $k3$ (que son 2^{56}). Despues voy descifrando con $k2$, y si encuentre una coincidencia, entonces encuentre un par $k2, k3$ que me llevan al resultado.

- Es inmune a criptoanálisis diferencial y lineal
- Es mucho más lenta que des (3 veces!)

AES

- Reemplazo de DES
- Fue seleccionado mediante un concurso internacional abierto
- Orientado a bloques de 128 bits
- Clave de 128, 192 o 256 bits
- Es la primitiva de cifrado **recomendada** en la actualidad

Esquema de funcionamiento



Esquema de funcionamiento

- Byte Sub: Sustitución de valores según una tabla derivada de invertir una matriz en campo $GF(2^8)$
- Shift Row: Permutación de bits
- Mix Column: transformación lineal invertible de cada byte en función de los 4 que forman el mismo grupo
- Add round key: Xor con la parte de la clave derivada para la ronda en cuestión

Esquema de funcionamiento

- 1ra Ronda
 - Solo tiene el paso Add Round Key
- Rondas intermedias
 - Byte sub
 - Shift Row
 - Mix Column
 - Add Round key
- Ultima ronda
 - Byte sub
 - Shift Row
 - Add round key

Round keys (128 bits)

- Los primeros 16 bytes:
 - corresponden a la clave original
- 16 bytes más:
 - Generar mascara
 - Tomar los últimos 4 bytes y rotarlos 8 bits a la derecha
 - Aplicar ByteSub a cada byte
 - Al byte menos significativo: xor con 2^i , siendo i el número de grupo de bytes a generar (1 los primeros 16)
 - Generar grupos de 4 bytes
 - 1^{ros} 4 bytes: mascara xor key bytes generados 16 posiciones antes
 - Siguientes: xor entre los 4 bytes generados y los generados 16 posiciones antes

Cifrado de bloque - tamaños

- Espacios de clave:
 - > 64 bits. AES tiene 128, 192 o 256 bits

Si la información es efímera, 128 bits alcanza, si se quiere almacenar la info por 30 años, se recomienda usar 256.

- Espacio de mensajes:
 - ≥ 128 bits

- Para comparar:
 - Partículas en el universo: $\sim 2^{84}$ part.
 - Edad estimada del universo: $\sim 2^{58}$ seg.

Lectura Recomendada

Capítulos 2 y 3

Introduction to Modern Cryptography
Katz & Lindell