

Seminario de Criptografía

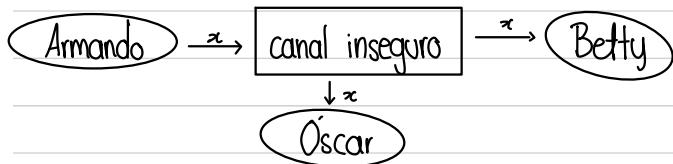
Definición

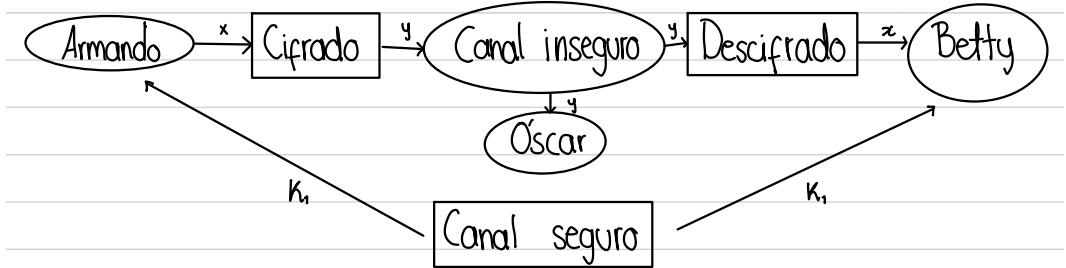
- \mathcal{A} alfabeto de definición $A = \{0, 1\}$
- M espacio de mensajes, consiste de cadenas de símbolos de \mathcal{A} . A los elementos de M se les llama **textos planos**
- C espacio de texto cifrado. A los elementos de C se les llama **textos cifrados**
- K espacio de llaves. A los elementos de K se le llaman **llaves**
- A cada elemento de K determina de forma única una función $E: M \rightarrow C$ de cifrado
- Para cada de K , $D_d: C \rightarrow M$ función de descifrado.

Principio de Kerchoffs

Comprometer el sistema en cuanto detalles no debe de ser un inconveniente para su efectividad. \hookrightarrow conocer todo menos las llaves

Criptografía simétrica





Ejemplos

① Cifrado de corrimiento \mathbb{Z}_{27}

$$A = K = \mathbb{Z}_{27}$$

Sea $k \in K$

$$e_k(x) = (x + k) \bmod 27 \quad d_k(y) = (y - k) \bmod 27$$

$$x \in M \quad y \in T$$

Cuando $k=3$ se le conoce como cifrado César

② Cifrado de sustitución

$$A = \mathbb{Z}_{27}, \quad K = S_{27}$$

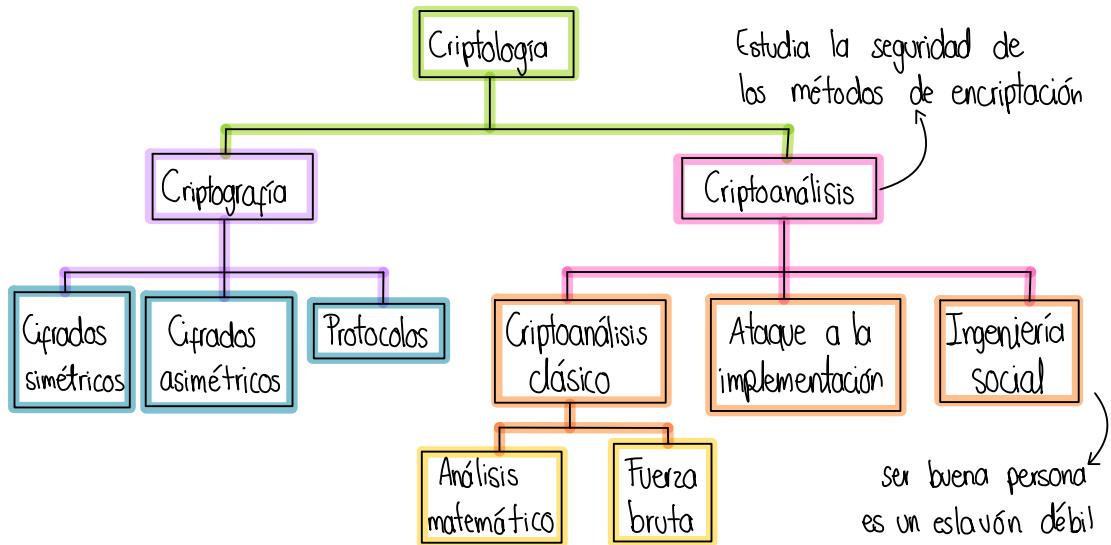
simétrico

$$\text{Sea } \pi \in K, \quad e_\pi(x) = \pi(x) \quad d_\pi(y) = \pi^{-1}(y)$$

③ Cifrado Vigenère

Stinson, D. (2002). Cryptography: Theory and Practice. 4th Ed. Chapman and Hall

Jueves 17 de septiembre del 2024



Tamaño de la llave: Sim. 80 bits ~ RSA-1024

Llave (bits)	Tiempo
56-64	Horas o días
112-128	Décadas (sin computadoras cuánticas)
256	Décadas (con computadoras cuánticas)

- Ataque de sólo texto cifrado: se conoce un texto cifrado y .
- Ataque de texto plano conocido: se conoce un texto plano x y su correspondiente cifrado y .
- Ataque de texto plano elegido: escoger un texto plano x y construir el correspondiente cifrado y .
- Ataque de texto cifrado elegido: escoger un texto cifrado y y construir el correspondiente plano x .

construir acceso temporal a la máquina de cifrado

Objetivo: obtener la llave K .

Ejemplo: Corrimiento

Fuerza bruta

Ejemplo: sustitución

Probabilidad de ocurrencia de las letras

Ejemplo: vigenére

Test de Kasiski (no estoy segura de cómo se escribe)

Ejemplo: Hill

$m=2$, $M=\mathcal{L} = (\mathbb{Z}/27)^m$, $K=GL(m, \mathbb{Z}/27)$

$K \in K$, $e_K(x) = xK$, $d_K(y) = yK^{-1}$

Suponemos que conocemos m

$X=(x_i)$, $Y=(y_i)$

$Y = XK$, $X = d_K e_K(X) \Rightarrow X^T Y = K$



Ejemplo de texto plano conocido

Cifrados secuenciales

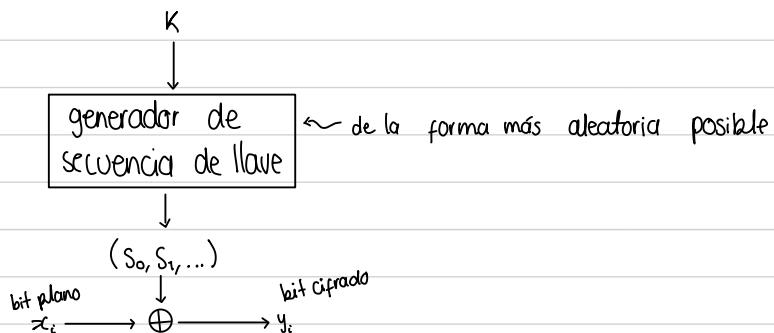
Cifrados simétricos → cifrado en secuencia

cifrado en bloques

Los cifrados en secuencia se cifran cada bit de manera independiente

Se cifran con la misma llave (bloques enteros)

un bit de texto plano no es independiente del cifrado de otros bits



Números aleatorios

La generación de números aleatorios se

2.3 Shift Register-Based Stream Ciphers (Cifrado de flujo basado en registro de desplazamiento)

2.3.1 Linear Feedback Shift Registers (LFSR) (Registros de desplazamiento con feedback lineal)

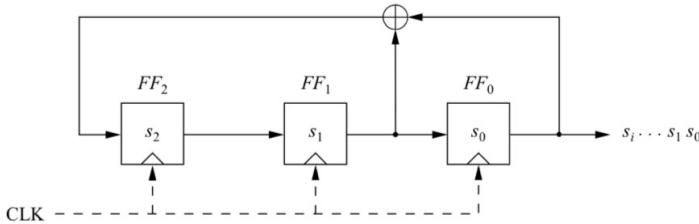
Un LFSR consiste de:

- flip-flops: elementos de almacenamiento cronometrados
- feedback path (ruta de retroalimentación)

- El grado de un LFSR es el número de flip-flops.
- la red de retroalimentación calcula el input para el último flip-flop como una XOR-suma de ciertos flip-flops en el registro de desplazamiento. (^{shift} register)

Ejemplo: LFSR simple

Consideramos un LFSR de grado 3 con flip-flops FF_2, FF_1, FF_0 con ruta de retroalimentación



Denotamos por s_i a los bits de estado interno los cuales se mueven uno hacia la derecha con cada tacto del reloj

El bit que se encuentra más hacia la derecha es el bit de salida actual.

El bit que se encuentra más a la izquierda se calcula en la ruta de retroalimentación, la cual es la suma XOR de algunos de los valores flip-flop en el periodo de tiempo previo

Como XOR es una operación lineal, a los circuitos se les llama registros de desplazamiento de retroalimentación lineal (linear feedback shift registers).

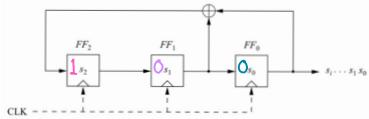
Ejemplo: Supongamos el estado inicial $s_2=1, s_1=0, s_0=0$.

clk	FF ₂	FF ₁	FF ₀ = s_i
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0

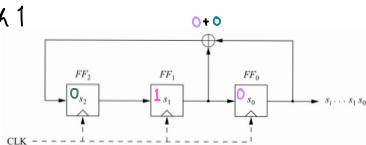
esta columna son las salidas del LFSR.

la tabla nos da una sucesión de los estados del LFSR.

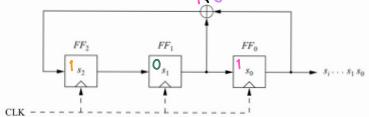
click 0



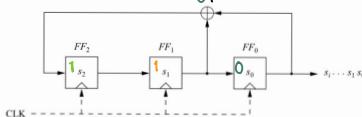
click 1



click 2



click 3



En este ejemplo podemos notar que el LFSR se empieza a repetir después de un ciclo de 6 clicks.

\Rightarrow la salida del LFSR tiene un periodo de longitud 7 de la forma
 0010111 0010111 0010111 ...

Hay una fórmula que determina el funcionamiento de este LFSR. Veamos cómo los bits de salida s_i son calculados asumiendo los bits de estado inicial s_0, s_1, s_2 :

$$s_3 \equiv s_1 + s_0 \pmod{2}$$

$$s_4 \equiv s_2 + s_1 \pmod{2}$$

$$s_5 \equiv s_3 + s_2 \pmod{2}$$

 \vdots

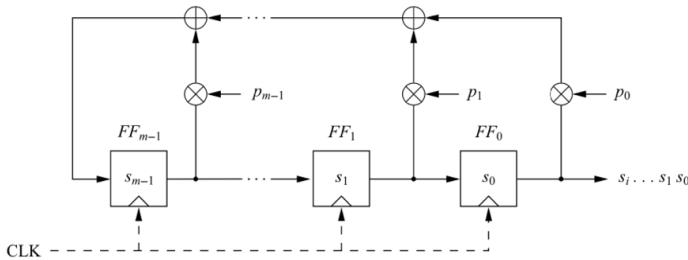
En general, el bit de salida se calcula como sigue

$$s_{i+3} \equiv s_{i+1} + s_i \pmod{2}$$

dond $i = 0, 1, 2, \dots$

Descripción matemática de un LFSR

Un LFSR general de grado m tiene la siguiente forma



tiene m flip-flops y m posibles lugares de retroalimentación (feedback) todas combinadas con la operación XOR.

Sin importar si un camino de feedback está activo o no, está definido

por los coeficientes de feedback p_0, p_1, \dots, p_{m-1} :

- Si $p_i = 1$ (interruptor cerrado) el feedback está activo.
- Si $p_i = 0$ (interruptor abierto), el flip-flop de salida no es utilizado en el feedback.

Con esto obtenemos una descripción matemática

Si multiplicamos la salida del flip-flop i por su coeficiente p_i el resultado es

- el valor de salida, si $p_i = 1$ (corresponde a un switch cerrado)
- 0, si $p_i = 0$ (corresponde a un switch abierto).

Los valores de los coeficientes del feedback son cruciales para la sucesión de outputs producida por el LFSR.

Supongamos un LFSR con valores iniciales s_0, \dots, s_{m-1} .

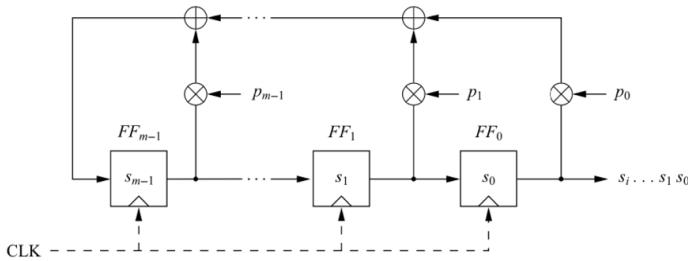
El siguiente bit de salida del LFSR s_m (que es también el input del flip-flop a la izquierda de todos) puede ser calculado como la XOR-suma de los productos de flip-flops de salida con sus coeficientes de feedback correspondientes:

$$S_m \equiv S_{m-1}p_{m-1} + \dots + S_1p_1 + S_0p_0 \pmod{2}$$

El siguiente LFSR de salida es entonces

$$S_{m+1} \equiv S_m p_{m-1} + \dots + S_2 p_1 + S_1 p_0 \pmod{2}$$

(ver el diagrama)



En general, la sucesión de bits de salida se puede describir como

$$S_{i+m} \equiv \sum_{j=0}^{m-1} p_j S_{i+j} \pmod{2}; \quad S_i, p_j \in \{0, 1\}, \quad i = 0, 1, 2, \dots \quad (2.1)$$

Los valores de salida de los LFSR están dados por una combinación de los valores de salida previos. A los LFSR a veces se les llama **recurrencias lineales**.

Observación

- Dado que se tiene un número finito de estados de recurrencia, la sucesión de salida se repite periódicamente.
- Un LFSR puede producir sucesiones de salida de diferentes longitudes (periodo) dependiendo de los coeficientes de feedback.

Teorema

La secuencia de longitud máxima generada por un LFSR de grado m tiene longitud $2^m - 1$.

(bosquejo)

Demostración todos los valores en la posición que se encuentran.

El estado de un LFSR está únicamente determinado por los m bits internos registrados.

Dado un estado, el LFSR determinísticamente asume su siguiente estado.

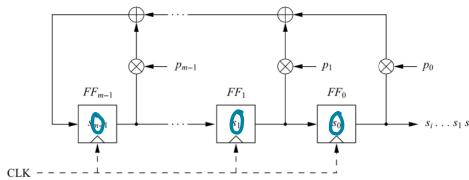
→ En el momento en el que un LFSR asume un estado previo, se comienza a repetir.

Como un estado de m bits puede asumir $2^m - 1$ estados no cero

\Rightarrow la longitud máxima de una sucesión de salida antes de comenzar a repetirse (el periodo máximo) es 2^{m-1} □

Nota: ¿por qué excluimos el estado cero?

Si un LFSR asume el estado cero \Rightarrow se queda atorado en dicho estado



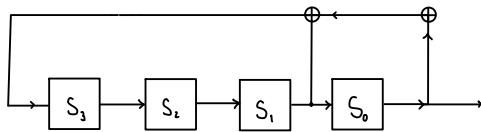
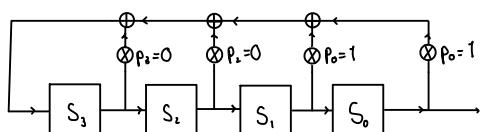
Observación

Solo ciertas configuraciones (p_0, \dots, p_{m-1}) producen longitudes máximas en un LFSR.

Ejemplos:

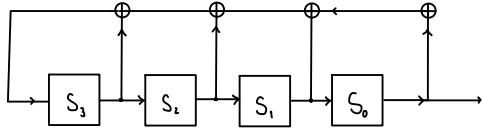
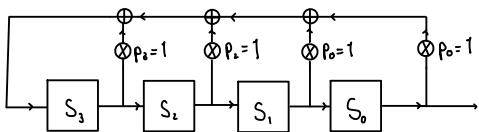
① LFSR con sucesión de salida de longitud máxima:

Dado un LFSR de grado $m=4$ y feedback path $(p_3=0, p_2=0, p_1=1, p_0=1)$ la sucesión de salida tiene periodo $2^m - 1 = 2^4 - 1 = 15$ (long máxima)



② LFSR de longitud no máxima

Dado un LFSR de grado $m=4$ y feedback path ($p_3=1, p_2=1, p_1=1, p_0=1$) la sucesión de salida tiene periodo 5 (< 15) \Rightarrow no tiene longitud máxima.



Nota

- Regulamente los LFSR se especifican con polinomios.

Un LFSR con coeficientes feedback $(p_{m-1}, \dots, p_1, p_0)$ se representa por el polinomio

$$P(x) = x^m + p_{m-1}x^{m-1} + \dots + p_1x + p_0$$

- Los LFSR de mayor longitud tienen asociado (se representan por) un **polinomio primitivo** (un tipo de polinomio irreducible)

(0,1,2)	(0,1,3,4,24)	(0,1,46)	(0,1,5,7,68)	(0,2,3,5,90)	(0,3,4,5,112)
(0,1,3)	(0,3,25)	(0,5,47)	(0,2,5,6,69)	(0,1,5,8,91)	(0,2,3,5,113)
(0,1,4)	(0,1,3,4,26)	(0,2,3,5,48)	(0,1,3,5,70)	(0,2,5,6,92)	(0,2,3,5,114)
(0,2,5)	(0,1,2,5,27)	(0,4,5,6,49)	(0,1,3,5,71)	(0,2,93)	(0,5,7,8,115)
(0,1,6)	(0,1,28)	(0,2,3,4,50)	(0,3,9,10,72)	(0,1,5,6,94)	(0,1,2,4,116)
(0,1,7)	(0,2,29)	(0,1,3,6,51)	(0,2,3,4,73)	(0,11,95)	(0,1,2,5,117)
(0,1,3,4,8)	(0,1,30)	(0,3,52)	(0,1,2,6,74)	(0,6,9,10,96)	(0,2,5,6,118)
(0,1,9)	(0,3,31)	(0,1,2,6,53)	(0,1,3,6,75)	(0,6,97)	(0,8,119)
(0,3,10)	(0,2,3,7,32)	(0,3,6,8,54)	(0,2,4,5,76)	(0,3,4,7,98)	(0,1,3,4,120)
(0,2,11)	(0,1,3,6,33)	(0,1,2,6,55)	(0,2,5,6,77)	(0,1,3,6,99)	(0,1,5,8,121)
(0,3,12)	(0,1,3,4,34)	(0,2,4,7,56)	(0,1,2,7,78)	(0,2,5,6,100)	(0,1,2,6,122)
(0,1,3,4,13)	(0,2,35)	(0,4,57)	(0,2,3,4,79)	(0,1,6,7,101)	(0,2,123)
(0,5,14)	(0,2,4,5,36)	(0,1,5,6,58)	(0,2,4,9,80)	(0,3,5,6,102)	(0,37,124)
(0,1,15)	(0,1,4,6,37)	(0,2,4,7,59)	(0,4,81)	(0,9,103)	(0,5,6,7,125)
(0,1,3,5,16)	(0,1,5,6,38)	(0,1,60)	(0,4,6,9,82)	(0,1,3,4,104)	(0,2,4,7,126)
(0,3,17)	(0,4,39)	(0,1,2,5,61)	(0,2,4,7,83)	(0,4,105)	(0,1,127)
(0,3,18)	(0,3,4,5,40)	(0,3,5,6,62)	(0,5,84)	(0,1,5,6,106)	(0,1,2,7,128)
(0,1,2,5,19)	(0,3,41)	(0,1,63)	(0,1,2,8,85)	(0,4,7,9,107)	
(0,3,20)	(0,1,2,5,42)	(0,1,3,4,64)	(0,2,5,6,86)	(0,1,4,6,108)	
(0,2,21)	(0,3,4,6,43)	(0,1,3,4,65)	(0,1,5,7,87)	(0,2,4,5,109)	
(0,1,22)	(0,5,44)	(0,3,66)	(0,8,9,11,88)	(0,1,4,6,110)	
(0,5,23)	(0,1,3,4,45)	(0,1,2,5,67)	(0,3,5,6,89)	(0,2,4,7,111)	

$$1+3x+4x^2+5x^3+112x^4$$

Algunos ejemplos de polinomios primitivos

Hay 69, 273, 666 polinomios primitivos de grado $m=31$.

2.3.2 Known-Plaintext Attack Against Single LFSRs

LFSRs son lineales (relación lineal entre inputs y outputs)

Si usamos un LFSR como stream cipher la llave secreta K es el vector de coeficiente de feedback $(p_{m-1}, \dots, p_1, p_0)$.

Un ataque es posible si el atacante conoce un fragmento del texto plano y su correspondiente texto cifrado.



Poder: Puede intentar un gran número de posibles de m valores sin problema alguno.

Con estos $2m$ pares de bits de texto plano y texto cifrado Oscar reconstruye los primeros $2m$ Key stream bits:

$$s_i \equiv x_i + y_i \pmod{2}, \quad i = 0, 1, \dots, 2m-1$$

Objetivo: encontrar la llave dada por los coeficientes de feedback p_i .

Recordemos que tenemos una forma de describir la relación entre los bits desconocidos p_i y la llave:

$$s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \pmod{2}; \quad s_i, p_j \in \{0, 1\}, \quad i = 0, 1, 2, \dots$$

Observación:

- Obtenemos una ecuación diferente para cada valor de i .
- Las ecuaciones son linealmente independientes.

Dado esto, Óscar puede generar m ecuaciones para los primeros m valores de i :

$$\begin{aligned} i=0, \quad S_m &\equiv p_{m-1}S_{m-1} + \dots + p_1S_1 + p_0S_0 \pmod{2} \\ i=1, \quad S_{m+1} &\equiv p_{m-1}S_m + \dots + p_1S_2 + p_0S_1 \pmod{2} \\ \vdots & \vdots \\ i=m-1, \quad S_{2m-1} &\equiv p_{m-1}S_{2m-2} + \dots + p_1S_m + p_0S_{m-1} \pmod{2} \end{aligned}$$

Óscar tiene un sistema de m ecuaciones lineales con m incógnitas
⇒ puede encontrar la solución

Observación

- En cuanto Óscar conozca $2m$ bits de salida de un LFSR de grado m , los coeficientes p_i pueden encontrarse resolviendo un sistema de ecuaciones lineales.

⇒ Puede construir el LFSR e ingresar cualesquiera m bits de salida consecutivas que conozca

⇒ Puede correr el LFSR y producir la secuencia de salida completa

⇒ LFSR pos si solos son **extremadamente inseguros !!!**

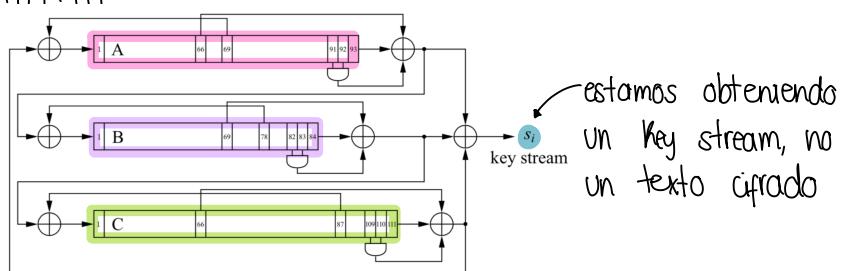
Existen stream ciphers que usan combinaciones de LFSRs para construir criptosistemas más seguros. El siguiente es un ejemplo de estos.

2.3.3 Trivium

- Relativamente nuevo
- Llave de 80-bits
- Basado en la combinación de 3 shift registers
- Hay componentes no lineares usados para derivar el output de cada registro.

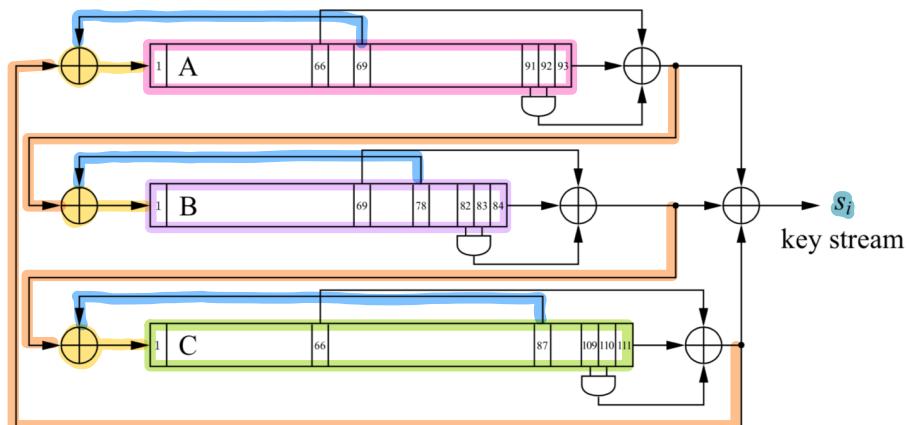
Descripción:

- Se compone de 3 shift registers: A B y C. De longitudes 93, 84 y 111, respectivamente
- La XOR-suma de los outputs de los 3 registros forman la llave s_i
- El output de cada registro está conectado al input de otro registro.
⇒ los registros pueden verse como un registro circular de longitud $288 = 93 + 84 + 111$



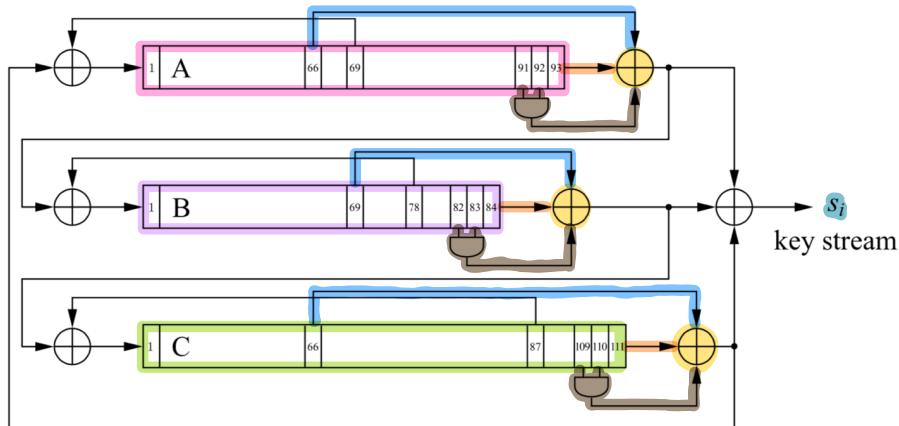
El **input** de cada registro se calcula como la **XOR**-suma de dos bits:

- El output bit de otro registro (**ejemplo**: el output del registro A es parte del input del registro B)
- Un bit de registro de una ubicación específica es llevado al input (**ejemplo**: el bit 69 del registro A es llevado a su input)



El **output** de cada registro es calculado como la xor-suma de 3 bits:

- El **bit** que se encuentra más a la derecha
- Un **bit** registrado en una ubicación específica es llevado al output (ejemplo: el bit 66 del registro A es llevado a su input)
- El **output** de una función lógica **AND** cuyo input son dos bits de registro específicos.



	register length	feedback bit	feedforward bit	AND inputs
A	93	69	66	91, 92
B	84	78	69	82, 83
C	111	87	66	109, 110

Observación:

- La operación AND es igual a la multiplicación módulo 2
- Si multiplicamos dos incógnitas, y los contenidos del registro son las incógnitas que Óscar quiere conocer
⇒ las ecuaciones resultantes ya no son lineales dado que contiene productos de incógnitas

AND		
0	0	0
0	1	0
1	0	0
1	1	1

⇒ Los caminos feedforward que involucran la operación AND son cruciales para la seguridad de Trivium puesto que preveen ataques que explotan la linealidad del cifrado.

Cifrados con Trivium

Casi todos los cifrados modernos tienen dos parámetros de entrada:

- una llave K
- un vector de inicialización IV

Llave regular usada en todo cripto sistema simétrico

El vector IV como un generador de elementos aleatorios (randomizer) y toma un nuevo valor en cada sesión de encriptación

El IV no tiene que ser un secreto, pero si cambiar en cada sesión

A los valores aleatorios que genera se les llaman **nonces** haciendo referencia a que son "number used once"

Propósito: dos Key streams producidas por el cifrado sean diferentes aún cuando la llave no haya cambiado.

Si este no es el caso un ataque que conozca el texto plano de un primer cifrado ⇒ puede calcular la Key stream correspondiente

⇒ El segundo cifrado puede ser descifrado inmediatamente usando la misma Key stream

Si el IV no cambia, el cifrado stream cipher es altamente determinístico.

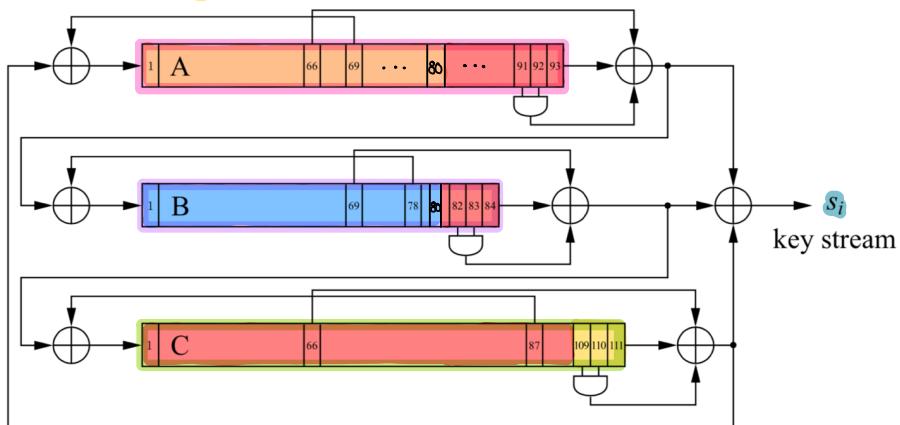
Veamos cómo es que esto funciona:

- Inicialización:

Un IV de 80 bits se carga en las 80 ubicaciones más a la izquierda del registro A.

Una llave de 80 bits se carga en las 80 ubicaciones más a la izquierda del registro B.

Todos los otros bits registrados se quedan como 0 a excepción de los 3 bits más a la derecha en el registro C (los bits C_{109} , C_{110} y C_{111}) los cuales quedan como 1.



- Fase de calentamiento:

El cifrado inicia con $4 \times 288 = 1152$ clicks del reloj. No se generan valores de salida del cifrado.

- Fase de cifrado:

Los bits producidos después de la fase de calentamiento (a partir del click 1153 del reloj) forman un Key stream

→ la fase de calentamiento es necesaria para randomizar el cifrado lo

suficiente, se asegura de que el Key stream dependa tanto de la llave K como del IV.

- Algo atractivo del Trivium es que es compacto, especialmente si se implementa en hardware.

Consta básicamente de un shift register de 288 bits y un par de operaciones Booleanas.

- Se estima que un hardware de implementación de este cifrado ocupa un área de entre 3500 y 5500 gate equivalences.

(Un gate equivalence es el área del chip ocupada por un 2-input NAND gate)

- Hasta el momento (en el que se escribió el libro) no hay ataques conocidos a este cifrado. (era relativamente nuevo en ese momento)

Muchos otros stream ciphers fueron encontrados no ser seguros.

2.4 Discusión y lectura a futuro

Stream ciphers establecidos

- La seguridad de muchos stream ciphers es desconocida y varios otros han sido rotos.
- Varios fueron usados pero debido a varios defectos actualmente no se recomienda (basados en el entendimiento actual del criptoanálisis).
- Este pesimismo cambió con el eSTREAM project

eSTREAM Project

objetivo: hacer avances en el conocimiento de diseños de stream cipher en estado del arte.

- eSTREAM fue organizado la Red Europea de Excelencia en Criptografía (CRYPT)
- Se submitieron 34 candidatos y al final del proyecto se encontraron cuatro cifrados software orientados (HC-128, Rabbit, Salsa 20/12 y SOSEMANUK) y 3 cifrados hardware orientados (Grain v1, MICKEY v2 y Trivium). Estos son relativamente nuevos (2008) y no se sabe si son realmente criptográficamente fuertes)

True random Number Generation

- Han habido varios TRNG que demuestran no comportarse de forma suficientemente aleatoria
 - ↳ Esto resulta en un problema de seguridad (dependiendo de cómo sean utilizados)

Existen estandares para evaluar formalmente los TRNG y herramientas que ponen a prueba sus propiedades estadísticas

2.5 Conclusiones

- Stream ciphers son menos populares que los cifrados de bloque en general (ejemplo: seguridad en internet). Sin embargo, tienen sus excepciones, RC4 es un stream cipher popular.

- Stream ciphers en ocasiones requieren menos recursos (ejemplo: tamaño del código o área del chip) comparado con cifrados de bloque haciendo su uso atractivo en ambientes de espacio restringido (ejemplo: celulares)
- Requieren el uso de un generador aleatorio de números criptográficamente seguro de forma más demandante.

3.7 Alternativas al DES

3.7.1 AES (Advanced Encryption Standard) y el Cifrado Finalista AES

- El algoritmo de elección para muchas aplicaciones
 - Tres llaves de long. de 128, 192 y 256 bits
 - Hasta el momento en que se publicó el libro:
 - Seguro contra ataques de fuerza bruta (por varias décadas).
 - no existen ataques analíticos conocidos que tengan oportunidad de ser exitosos.
 - Resultado de una competencia abierta donde hubieron otros 4 finalistas que también eran cifrados en bloque:
 - Mars
 - RCG
 - Serpent
 - Twofish

} criptográficamente fuertes
rápidos (especialmente en software)

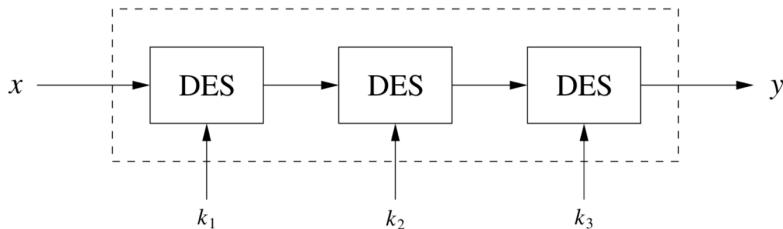
Basados en el conocimiento que se tenía hasta el momento en que se publicó el libro, todos estos son recomendables.

- Veremos su funcionamiento más adelante.

3.7.2 3DES y DESX

3DES es una alternativa al AES que consiste de 3 cifrados DES con diferentes llaves.

$$y = \text{DES}_{K_3}(\text{DES}_{K_2}(\text{DES}_{K_1}(x)))$$



- Aparenta ser seguro contra ataques de fuerza bruta y ataques analíticos.

Otra versión del 3DES:

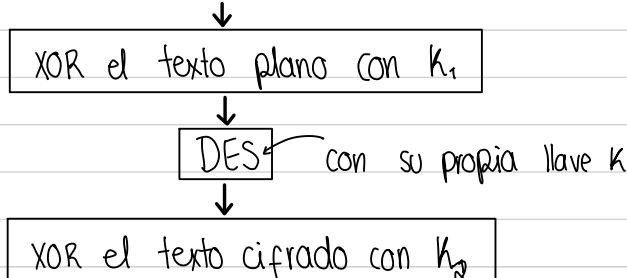
$$y = \text{DES}_{K_3}(\text{DES}_{K_2}^{-1}(\text{DES}_{K_1}(x)))$$

En esta versión si $K_1 = K_2 = K_3 \Rightarrow \text{DES}_{K_1} = 3\text{DES}_K$, } esto es una ventaja sobre la otra versión

A veces esto es deseado en implementaciones que también deben soportar a DES (una iteración) por razones "hereditarias".

- 3DES es muy eficiente en hardware pero no particularmente en software.
- Popular en aplicaciones financieras y para protección de información biométrica en pasaportes electrónicos.
- Otra forma de reforzar a DES es usar blanqueamiento de llaves:

Consideramos dos llaves adicionales K_1 y K_2 de 64-bits cada una



Tenemos entonces el siguiente esquema de encriptación:

$$y = \text{Des}_{K, K_1, K_2}(x) = \text{DES}_K(x \oplus K_1) \oplus K_2$$

Esta modificación hace a DES más resistente contra búsqueda exhaustiva de llaves.

3.7.3 Cifrado ligero PRESENT

Baja complejidad de implementación (especialmente en hardware)

PRESENT es un cifrado en bloque diseñado específicamente para aplicaciones como RFID tags. usa campos electromagnéticos para identificar o seguir etiquetas pegadas a objetos
(Radio-frequency identification)

Uno de los autores del libro participó en el diseño de este cifrado.

- Basado en una red de sustitución y permutación (SP-network) y consta de 31 rondas.

Long. del block: 64 bits

Núm. de llaves: 2

Long. de llaves: 80 y 128 bits

Cada ronda consiste de:

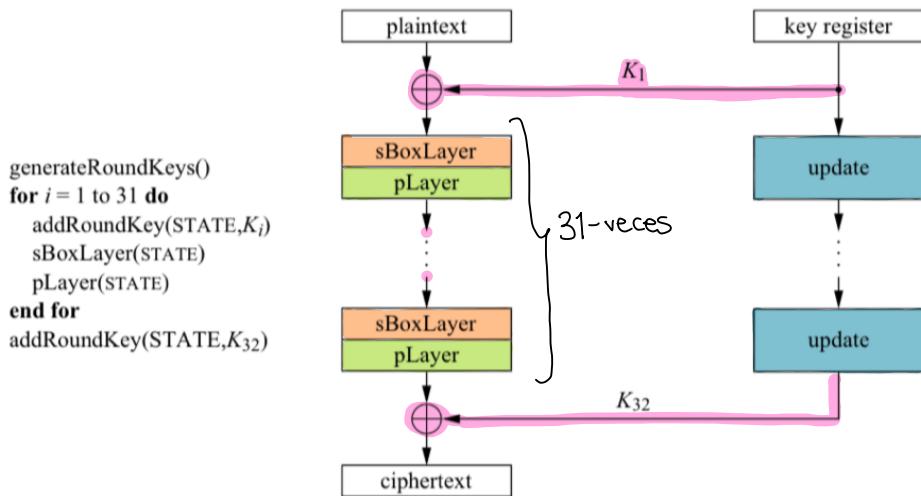
- una operación XOR para introducir una llave de ronda K_i : $(1 \leq i \leq 32)$
Son 31 rondas ↳ 32?

K_{32} se utiliza después de la ronda 31

- una capa de sustitución no lineal (s Boxlayer)
- permutación de bits (lineal en bits) (p layer)

usa una única S-box S de 4 bits que se aplica 16 veces en cada ronda

¿Cómo funciona?



- **add RoundKey** : Al inicio de cada ronda la llave de ronda K_i se XOR ea con el estado actual.
- **sBoxLayer** : Usa una única S-box de 4 bits (con la intención de ser eficiente para su uso en hardware)

las entradas de la S-box están dadas en notación hexadecimal y la S-box funciona como indica la siguiente tabla:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Ejemplo: $S(A1B2) = F586$

El camino de 64 bits $b_{63} \dots b_0$ es llamado **estado**.

Para la sBoxLayer el estado actual es considerado como 16 palabras

de 4 bits donde $w_{15} \dots w_0$ separa los bits

$$w_i = b_{4*i+3} \parallel b_{4*i+2} \parallel b_{4*i+1} \parallel b_{4*i}$$

para $0 \leq i \leq 15$ y las salidas son las 16 palabras $S[w_i]$.

- **Player:** se utiliza la siguiente permutación donde el bit i del estado es movido al bit en la posición $P(i)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

La permutación de los bits puede expresarse de la siguiente forma:

$$P(i) = \begin{cases} i \cdot 16 \bmod 63, & i \in \{0, \dots, 62\} \\ 63, & i = 63 \end{cases}$$

- **Key Schedule:** Generación de llaves por ronda comenzando con una llave de 80 bits

Normalmente una llave de este tamaño es suficiente (pues las principales aplicaciones son de bajo costo pero también se puede usar una de 128 bits).

La llave dada por el usuario K y se representa como
 $K_{79} K_{78} \dots K_0$.

En la ronda i la llave de ronda K_i está dada como

$$K_i = K_{63} K_{62} \dots K_0$$

esto es, los 64 bits más a la izquierda del contenido actual del K registrado.

La primer subllave K_1 es una copia directa de 64 bits de la llave original.

Para las siguientes subllaves K_2, \dots, K_3 , la llave registrada K se actualiza de la siguiente manera.

$$1. [K_{79} K_{78} \dots K_0] = [K_{18} K_{17} \dots K_{20} K_{19}]$$

rotamos la llave registro 61 bits a la izquierda \nwarrow

$$2. [K_{79} K_{78} K_{77} K_{76}] = S[K_{79} K_{78} K_{77} K_{76}]$$

los 4 bits más a la izquierda de K son pasados por la S-box

$$3. [K_{19} K_{18} K_{17} K_{16} K_{15}] = [K_{19} K_{18} K_{17} K_{16} K_{15}] \oplus (\text{contador de ronda})$$

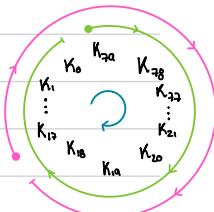
el contador de ronda de valor i es XOR-eado con los bits

$K_{19} K_{18} K_{17} K_{16} K_{15}$ de K donde el bit menos significante del contador de ronda está a la derecha.

Este contador de ronda es un entero que toma valores:

$\underbrace{00001}_1, \underbrace{00010}_2, \dots, \underbrace{11111}_{31}$

Notemos que para K_2 usamos 00001, para K_3 usamos 00010, ..., para K_{32} usamos 11111.



Implementación:

- Diseño agresivamente optimizado para hardware.
- Su rendimiento en software no es muy competitivo comparado con cifrados modernos como AES.
- PRESENT-80 puede ser implementado en hardware con requerimiento de área de aprox. 1,600 gate equivalences, donde la encriptación de un texto plano requiere 32 clock cycles
- Una buena implementación de este cifrado alcanza un throughput de 64 bits por clock cycle (esto se traduce a más de 50 Gbit/s) de encryption throughputs.
- Hasta el momento de la escritura del libro el cifrado era nuevo y no se conocían ataques hacia este.

3.8 Discusión y lectura recomendada.

Ataque e historia del DES

- DES es "actualmente" utilizado en heredadas (legacy applications).
- Los dos ataques analíticos principales desarrollados contra DES:
 - criptoanálisis diferencial,
 - criptoanálisis linealsiguen siendo los métodos más poderosos contra cifrados en bloque
- DES ya no debería ser usado dado que puede ser roto con fuerza bruta en poco tiempo

Aternativas a DES

- La mayoría de los cifrados en bloque exitosos tomaron prestadas ideas del DES. Algunos otros están basados en redes Feistel.
- Un cifrado en bloque notoriamente distinto a DES es IDEA, el cual utiliza aritmética en 3 estructuras algebraicas y operaciones atómicas.
- Además de PRESENT otros cifrados pequeños son Clefia, HIGHT y mCrypton.

Aprendizajes

- DES fue el algoritmo de encriptación simétrico dominante de mediados de los 70s a mediados de los 90s.
Cuando las llaves de 56 bits dejaron de ser seguras
⇒ se creó AES.
- DES puede romperse con fuerza bruta de forma más o menos rápida.
- Es muy complicado romper DES con criptoanálisis diferencial y/o criptoanálisis lineal.
- DES es razonablemente eficiente en software y muy rápido y pequeño en hardware.
- 3DES no tiene ataques prácticos conocidos
- El cifrado simétrico por default es el AES. Los otros finalistas también parecen ser seguros y eficientes.

El criptosistema RSA

Ronald Rivest

Adi Shamir

Leonard Adleman



1976: Whitfield Diffie y Martin Hellman introducen criptografía de llave pública.



Se comienzan a buscar métodos con los cuales se pueda utilizar encriptación de llave pública



1977: Ronald Rivest, Adi Shamir y Leonard Adleman proponen lo que resultó ser el esquema criptográfico asimétrico más utilizado (RSA)

- RSA fue patentado en USA (pero no en el resto del mundo) hasta 2000.
- Es mayormente utilizado para:
 - Encriptación de pedazos pequeños de información (especialmente transporte de llaves).
 - Firmas electrónicas

- Algunas veces es más lento que cifrados como el AES.
- En la práctica RSA se utiliza frecuentemente en conjunto con un cifrado simétrico (por ejemplo el AES).

Cifrado y descifrado

Estos se realizan en el anillo $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$

Cifra
 RSA ~~encripta~~ texto plano x considerándolo como un elemento en \mathbb{Z}_n
 \Rightarrow el valor binario de x debe ser menor que n .

Cifrado RSA: Dada la llave pública $(n, e) = K_{\text{pub}}$ y el texto plano x , la función encriptación es

$$y = e_{K_{\text{pub}}}(x) \equiv x^e \pmod{n}$$

donde $x, y \in \mathbb{Z}_n$

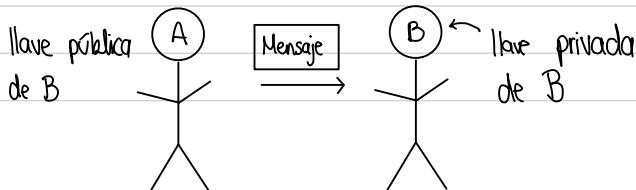
Descifrado RSA: Dada la llave privada $d = K_{\text{pr}}$ y el texto cifrado y , la función de descifrado es

$$x = d_{K_{\text{pr}}}(y) \equiv y^d \pmod{n}$$

donde $x, y \in \mathbb{Z}_n$

x, y, n, d suelen ser números muy grandes (~ 1024 bits o más)

A valor e se le llama exponente de encriptación o exponente público, y al valor d exponente de descifrado o exponente privado.



Requisitos para un criptosistema RSA.

1. Debe ser computacionalmente no viable determinar la llave privada d dados los valores n y e de la llave pública. (De lo contrario, conoceríamos la llave pública y la llave privada \Rightarrow somamente inseguro)
2. No encriptar más de l bits con una encriptación RSA, donde l es la longitud en bits de n . (Esto dado que x sólo es único hasta el tamaño del módulo n)
3. Deber relativamente fácil calcular $x^e \text{ mod } n$ y $y^d \text{ mod } n$ (para cifrar y descifrar). (Necesitamos un método para calcular exponentiación de números grandes de forma rápida).
4. Para n dado deben existir varios pares de llave privada y llave pública. (De lo contrario es susceptible a un ataque mediante fuerza bruta)

7.3 Generación de llaves y prueba de correctitud

- La generación de llaves puede ser compleja, esto depende del esquema de llave pública.

Algunos pasos involucrados en el cálculo de llaves privadas y públicas para un Criptosistema RSA:

1. Elegir dos números primos p, q grandes
(Ejemplo: $p=11, q=13$)
2. Calcular $n=p \cdot q$
(Ejemplo: $n=(11) \cdot (13) = 143$)

3. Calcular $\phi(n) = (p-1)(q-1)$

(Ejemplo: $\phi(143) = (11-1)(13-1) = (10)(12) = 120$)

4. Seleccionar el exponente público $e \in \{1, 2, \dots, \phi(n)-1\}$ tal que
 $\gcd(e, \phi(n)) = 1$

(Ejemplo: $\gcd(e, 120) = 1 \Rightarrow e \in \{1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59, 61, 67, 71, 73, 77, 79, 83, 89, 91, 95, 97, 101, 103, 107, 109, 113, 115, 119\}$)
Seleccionamos $e = 71$)

5. Seleccionar la llave privada d tal que

$$d \cdot e \equiv 1 \pmod{\phi(n)}$$

(Ejemplo: $d \cdot 71 \equiv 1 \pmod{120} \Rightarrow d \cdot 71 = 120 \cdot x + 1, x \in \mathbb{N}$
Notemos que si $d = 71 \Rightarrow (71)(71) = 5041 = 5040 + 1 = (120)(42) + 1 \equiv 1 \pmod{120}$)
Seleccionamos $d = 71$)

⇒ Obtenemos la llave pública $K_{\text{pub}} = (n, e)$ y la llave privada $K_{\text{pr}} = (d)$.

Ejemplo: $K_{\text{pub}} = (143, 71)$ y $K_{\text{pr}} = 71$

Observación

- La condición $\gcd(e, \phi(n)) = 1$ asegura que el inverso de e existe módulo $\phi(n) \Rightarrow$ siempre existe una llave privada d .
- El cálculo de las llaves d y e puede realizarse al mismo tiempo usando la versión extendida del algoritmo de Euclides
- En la práctica, se realiza el siguiente proceso:
 - (1) Se selecciona un parámetro público e con $0 < e < \phi(n)$
Nota: Si $e = 1 \Rightarrow y = z^e \pmod{n} = z \pmod{n}$
función de encriptación
 $\Rightarrow z = y$. Debemos excluir el caso $e = 1$.

(2) Se aplica el algoritmo de Euclides extendido conociendo n y e para obtener la relación

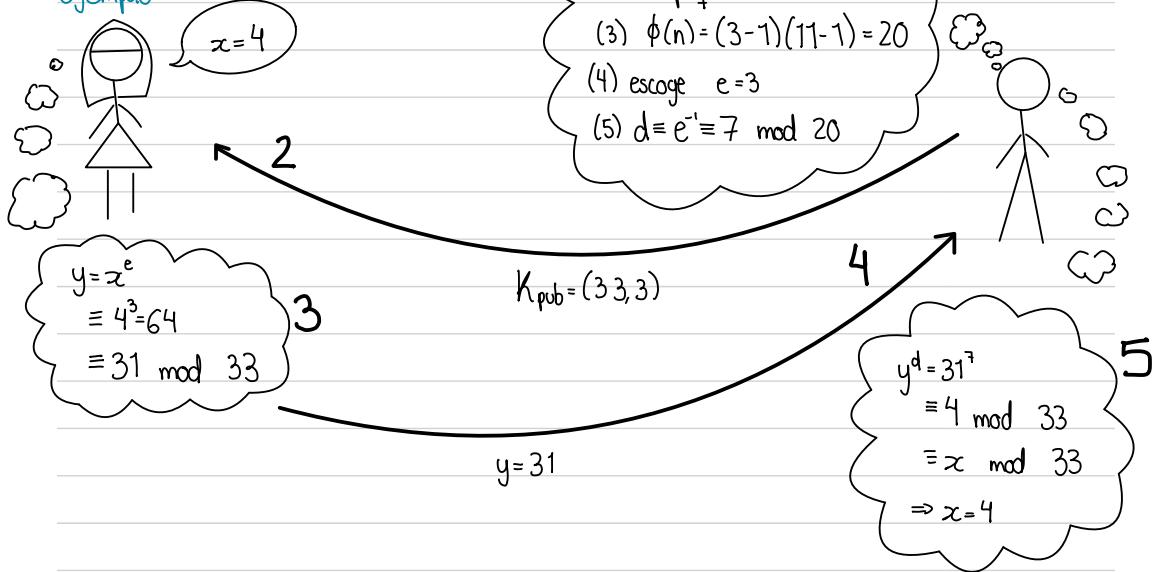
$$\gcd(\phi(n), e) = s \cdot \phi(n) + t \cdot e$$

(3) Evaluamos: si $\gcd(e, \phi(n)) = 1 \Rightarrow e$ es una llave pública válida
De lo contrario, volvemos al paso (1).

(4) Si e es llave pública $\Rightarrow t$ es el inverso de e

Observación: Nunca usamos s .

Ejemplo:



Nota: En la práctica, RSA utiliza parámetros más grandes.

Un RSA módulo n debe ser de por lo menos 2048 bits de largo, lo cual hace que p y q tengan longitud de 1024 bits

Ejemplo: Si n es de 1024 bits se pueden usar, por ejemplo, los siguientes parámetros.

- $p = E0DFD2C2A288ACEBC705EFAB30E4447541A8C5A47A37185C5A9$
- $CB98389CE4DE19199AA3069B404FD98C801568CB9170EB712BF$
- $10B4955CE9C9DC8CE6855C6123_h$
- $q = EBE0FCF21866FD9A9F0D72F7994875A8D92E67AEE4B515136B2$
- $A778A8048B149828AEA30BD0BA34B977982A3D42168F594CA99$
- $F3981DDABFAB2369F229640115_h$
- $n = CF33188211FDF6052DBBB1A37235E0ABB5978A45C71FD381A91$
- $AD12FC76DA0544C47568AC83D855D47CA8D8A779579AB72E635$
- $D0B0AAC22D28341E998E90F82122A2C06090F43A37E0203C2B$
- $72E401FD06890EC8EAD4F07E686E906F01B2468AE7B30CBD670$
- $255C1FEDE1A2762CF4392C0759499CC0ABECFF008728D9A11ADF_h$
- $e = 40B028E1E4CCF07537643101FF72444A0BE1D7682F1EDB553E3$
- $AB4F6DD8293CA1945DB12D796AE9244D60565C2EB692A89B888$
- $1D58D278562ED60066DD8211E67315CF89857167206120405B0$
- $8B54D10D4EC4ED4253C75FA74098FE3F7FB751FF5121353C554$
- $391E114C85B56A9725E9BD5685D6C9C7EED8EE442366353DC39_h$
- $d = C21A93EE751A8D4FBFD77285D79D6768C58EBF283743D2889A3$
- $95F266C78F4A28E86F545960C2CE01EB8AD5246905163B28D0B$
- $8BAABB959CC03F4EC499186168AE9ED6D88058898907E61C7CC$
- $CC584D65D801CFE32DFC983707F87F5AA6AE4B9E77B9CE630E2$
- $C0DF05841B5E4984D059A35D7270D500514891F7B77B804BED81_h$

Observación

Para cifrar y descifrar el mensaje usamos que

$$d_{K_{pr}}(y) = d_{K_{pr}}(e_{K_{pub}}(x)) \equiv (x^e)^d \equiv x^{de} \equiv x \pmod{n} \quad (*)$$

En esto se basa RSA. Hay que verificar que funciona

Prueba de correctitud

Queremos verificar que el descifrado es la inversa de la función de cifrado.
(i.e. $d_{K_{priv}}(e_{K_{pub}}(x)) = x$)

Comenzaremos construyendo la regla entre las llaves privada y pública
 $d \cdot e \equiv 1 \pmod{\phi(n)}$

Notemos que

$$d \cdot e \equiv 1 \pmod{\phi(n)} \iff d \cdot e = 1 + t \cdot \phi(n) \text{ para cierto } t \in \mathbb{Z}.$$

Sustituyendo esto en (*) tenemos que

$$d_{K_{priv}}(y) \equiv x^{de} \equiv x^{1+t\phi(n)} \equiv x^{t\phi(n)} \cdot x = (x^{\phi(n)})^t \cdot x \pmod{n} \quad (*)$$

Entonces debemos probar que $x \equiv (x^{\phi(n)})^t \cdot x \pmod{n}$

Observación

Teorema (Euler)

Si $\gcd(x, n) = 1$ entonces $1 \equiv x^{\phi(n)} \pmod{n}$.

De esto, $1 \equiv 1^t \equiv (x^{\phi(n)})^t \pmod{n}$ para todo $t \in \mathbb{Z}$

Dividiremos la prueba en dos casos

$$\begin{cases} \text{Caso 1: } \gcd(x, n) = 1 \\ \text{Caso 2: } \gcd(x, n) = \gcd(x, p \cdot q) \neq 1 \end{cases}$$

Caso 1: $\gcd(x, n) = 1$

Podemos usar el teorema previo
 $\Rightarrow 1 \equiv (x^{\phi(n)})^t \pmod{n}$

Sustituimos esto en la ecación (*) y obtenemos

$$d_{K_p}(y) \equiv (x^{\phi(n)})^t \cdot x \equiv 1 \cdot x \equiv x \pmod{n} \quad \checkmark$$

Caso 2: $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$

Como q y p son primos entonces uno de ellos debe de dividir a x , esto es

$$x = r \cdot p \quad \text{ó} \quad x = s \cdot q$$

con $r, s \in \mathbb{Z}$ tales que $r \nmid q$ y $s \nmid p$.

Sin pérdida de generalidad podemos suponer $x = r \cdot p$
 $\Rightarrow \gcd(x, q) = 1$

Usamos ent. el teorema previo obteniendo

$$1 \equiv 1^t \equiv (x^{\phi(q)})^t \pmod{q}$$

donde t es cualquier entero.

Luego,

$$(x^{\phi(n)})^t \equiv (x^{(q-1)(p-1)})^t \equiv ((x^{\phi(q)})^t)^{p-1} \equiv 1^{(p-1)} = 1 \pmod{q}$$

$$\Rightarrow (x^{\phi(n)})^t = 1 + u \cdot q, \text{ para cierto } u \in \mathbb{Z}$$

De esto,

$$\begin{aligned} x \cdot (x^{\phi(n)})^t &= x(1 + u \cdot q) = x + x \cdot u \cdot q \\ &= x + (r \cdot p) \cdot u \cdot q, \text{ pues } x = r \cdot p \\ &= x + r \cdot u \cdot (p \cdot q) \\ &= x + r \cdot u \cdot n \end{aligned}$$

$$\Rightarrow x \cdot (x^{\phi(n)})^t \equiv x \pmod{n}$$

Sustituyendo esto en (*) tenemos que

$$d_{K_{pr}}(y) = (x^{d(n)})^t \cdot x \equiv x \pmod{n}$$

7.4 Cifrado y descifrado: Exponenciación rápida

Los algoritmos de llave pública están basados en aritmética con números muy largos

⇒ puedes fácilmente terminar con esquemas muy lentos para uso práctico

Recordemos que en RSA

$$\text{cifrado: } e_{K_{pub}}(x) \equiv x^e \pmod{n}$$

$$\text{descifrado: } x = d_{K_{pr}}(y) \equiv y^d \pmod{n}$$

y basado en exponenciación

Una forma directa de realizar exponenciación es la siguiente:

$$x \xrightarrow[\text{SQ}]{\substack{\text{elevar al cuadrado}}} x^2 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^3 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^4 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^5 \longrightarrow \dots$$

Como los exponentes e y d suelen ser muy grandes (≥ 2048 bits), hacer esto requiere de alrededor de 2^{2048} multiplicaciones o más

Esto es muchísimo (hay un estimado de 2^{300} átomos en el universo visible)

¿Hay formas más rápidas de hacer exponenciación? Si ☺

Ejemplo: square-and-multiply algorithm

Calcular x^8

$$\text{método directo: } x \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^2 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^3 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^4 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^5 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^6 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^7 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^8$$

$$\text{método s-m: } x \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^2 \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^4 \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^8$$

Este algoritmo funciona bien pero solo puede utilizarse para exponentes de la forma 2^i (potencias de 2)

Ejemplo: Calcular x^{26} (26 no es potencia de 2)

método directo: $x \xrightarrow{\text{SQ}} x^2 \xrightarrow{\text{MUL}} x^3 \xrightarrow{\text{MUL}} x^4 \rightarrow \dots \xrightarrow{\text{MUL}} x^{26}$

otro método: $x \xrightarrow{\text{SQ}} x^2 \xrightarrow{\text{MUL}} x^3 \xrightarrow{\text{SQ}} x^6 \xrightarrow{\text{SQ}} x^{12} \xrightarrow{\text{MUL}} x^{13} \xrightarrow{\text{SQ}} x^{26}$

El segundo método es más rápido y nos da una forma de usar el square-and-multiply algorithm.

¿En qué orden usar SQ y MUL?

Hay un algoritmo que nos ayuda a determinar esto mediante los siguientes pasos:

1: Escanear los bits del exponente de izquierda a derecha.

2: En cada iteración de dicho escaneo

si el bit es $1 \Rightarrow \text{SQ}$ y MUL $\quad \quad \quad$ y después tomamos mod n
si el bit es $0 \Rightarrow \text{SQ}$

Ejemplo:

$$\text{Calculemos } x^{26} = x^{(11010)_2} = x^{(h_4 h_3 h_2 h_1 h_0)_2}$$

Empezamos de izquierda a derecha

Paso 0: $x = x^{(1)}$ (bit $h_4 = 1$)

Paso 1: SQ: $(x^1)^2 = x^2 = x^{(10)_2}$ (bit $h_3 = 1$)

MUL: $x^2 \cdot x = x^3 = x^{(10)_2} \cdot x^{(1)_2} = x^{(11)_2}$

Paso 2: SQ: $(x^3)^2 = x^6 = (x^{10_2})^2 = x^{1100_2}$ (bit $h_2=0$)

Paso 3: SQ: $(x^6)^2 = x^{12} = (x^{10_2})^2 = x^{1100_2}$ (bit $h_1=1$)
MUL: $x^{12} \cdot x = x^{13} = (x^{100_2}) \cdot x^{1_2} = x^{1101_2}$

Paso 4: SQ: $(x^{13})^2 = x^{26} = (x^{101_2})^2 = x^{11010_2}$ (bit $h_0=0$)

Nota: SQ agrega 0 a la derecha del exponente

MUL cambia el último dígito del exponente de 0 a 1

Con estas dos operaciones vamos escribiendo el exponente en binario.

Pseudocódigo para el square-and-multiply algorithm para exponenciación modular

Paso 0: Tenemos el elemento base x , el exponente $H = \sum_{i=0}^t h_i 2^i$ con $h_i \in \{0, 1\}$ y $h_t=1$; y el módulo que estamos tomando n .

Paso 1: Tomamos $r = x$

Paso 2: Para $i = t-1$ hasta 0

$$r = r^2 \text{ mod } n$$

$$\text{si } h_i = 1$$

$$r = r \cdot x \text{ mod } n$$

Paso 3: Devolver $r = x^H$

¿Cuál es la complejidad de este algoritmo?

Sea H un exponente de longitud $t+1$ ($\log_2 H = t+1$)

- El número de SQ usados es independiente del valor de H (depende únicamente de la longitud de H)

- El número de MUL es igual al peso de Hamming de H (el número de 1's usados en la representación binaria)

Entonces, $\#SQ = t$

$$\# MUL \approx 0.5t \text{ (aproximación/promedio)}$$

¿Cuántas operaciones en promedio se necesitan para una exponenciación con un exponente de 2048 bits?

Usando el método directo: $2^{2048} \approx 10^{600}$ multiplicaciones !Esto es imposible

Usando el square-and-multiply algoritmo:

$$1.5(2048) = 3072$$

elevaciones al cuadrado y multiplicaciones en promedio

complejidad logarítmica

complejidad lineal

8. Criptosistemas basados en el problema del logaritmo discreto

Recordatorio:

Definición

Una función es de una dirección si es sencillo calcular $f(x)=y$ pero computacionalmente no viable calcular $f^{-1}(y)=x$.

Nota: El problema de factorización de enteros es una función de una dirección de RSA.

¿Hay otras funciones de una dirección para construir sistemas criptográficos asimétricos?

Sí

El problema del logaritmo discreto

La mayoría de los algoritmos de llave pública distintos a RSA están basados en esto

8.1 Intercambio de llaves Diffie-Hellman (DHKE)

- Propuesto por Whitfield Diffie y Martin Hellman en 1976.
- Primer esquema asimétrico publicado en la "literatura abierta".
- Usando en muchos protocolos criptográficos como Secure Shell (SSH), Transport Layer Security (TLS) e Internet Protocol Security (IPSec).

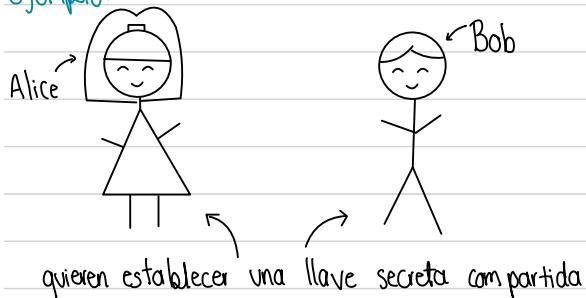
Idea general:

Para un primo p , la exponenciación en \mathbb{Z}_p^* es una función de una dirección y es conmutativa, i.e.

$$k \equiv (\alpha^x)^y \equiv (\alpha^y)^x \pmod{p}$$

la clave del porqué funciona
Este es el secreto del DHKE

Ejemplo:



Probablemente hay una tercera persona involucrada que elige y publica los parámetros públicos que se necesitan para el intercambio de llaves.

También lo pueden hacer únicamente Alice y Bob.

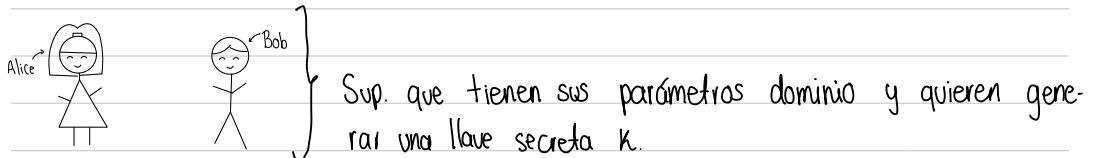
- DHKE consta de dos protocolos:

- el set-up
- el protocolo principal

Diffie-Hellman setup:

- 1: Elige un primo p grande.
- 2: Elige un entero $\alpha \in \{2, 3, \dots, p-2\}$.
- 3: Publicar p y α .

parámetros dominio

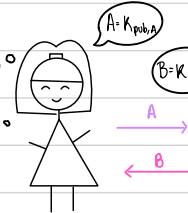


Intercambio de llaves Diffie-Hellman

Alice:

- Elige $a = K_{pr,A} \in \{2, \dots, p-2\}$
- Calcula $A = K_{pub,A} \equiv a^a \pmod{p}$

$$K_{AB} = K_{pub,B}^{K_{pr,A}} \equiv B^a \pmod{p}$$



Bob:

- Elige $b = K_{pr,B} \in \{2, \dots, p-2\}$
- Calcula $B = K_{pub,B} \equiv a^b \pmod{p}$

$$K_{AB} = K_{pub,A}^{K_{pr,B}} \equiv A^b \pmod{p}$$

Afirmación
 $K_{AB} = K_{AB}$

La llave conjunta K_{AB} puede ser utilizada para establecer una comunicación segura entre Bob y Alice con algoritmos como AES, 3DES o un stream cipher.

Observación

$$\begin{aligned} B^a &\equiv (a^b)^a \pmod{p} = a^{ba} \pmod{p} = (a^a)^b \pmod{p} \equiv A^b \\ &\Rightarrow B^a \pmod{p} = A^b \pmod{p} \Rightarrow K_{AB} = K_{AB} \end{aligned}$$

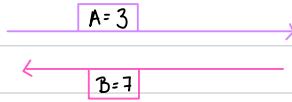
Ejemplo:

Consideremos los parámetros dominio $p=29$ y $a=2$

Alice

$$p-2 = 29-2$$

- Escoge $a = K_{pr,A} = 5 \in \{2, \dots, 27\}$
- $A = K_{pub,A} = 2^5 \equiv 3 \pmod{29}$



$$K_{AB} = B^a = 7^5 \equiv 16 \pmod{29}$$

Bob

- Escoge $b = K_{pr,B} = 12 \in \{2, \dots, 27\}$
- $B = K_{pub,B} = 2^{12} \equiv 7 \pmod{29}$

$$K_{AB} = A^b = 3^{12} \equiv 16 \pmod{29}$$

- Los aspectos computacionales de DHKE son similares a los de RSA.
 \Rightarrow DHKE debe tener una longitud similar a la de RSA mod n para ser seguro (≥ 2048 bits)
- El entero α debe ser primitivo (Definición: Sea G un grupo cíclico, un elemento $a \in G$ es **primitivo** si tiene orden máximo)
 \Leftrightarrow (i.e. $\text{ord}(a) = |G|$)
- K_{AB} tiene la misma longitud de bits que p .
- a y b deben provenir de un generador de números aleatorios (para que no se puedan adivinar fácilmente)
- El protocolo se puede generalizar (ej. para grupos de curvas elípticas \rightsquigarrow criptografía de curvas elípticas)

8.2 Álgebra Abstracta

- Grupos. (Nota: se busca trabajar con grupos finitos)
- Grupos cíclicos
- Teoremas de grupos cíclicos que son relevantes en la criptografía.

Teorema

Para todo primo p , (\mathbb{Z}_p^*, \cdot) es un grupo cíclico abeliano finito.



Relevante puesto que casi todo buscador web usa un criptosistema sobre \mathbb{Z}_p^* .

Teorema

Sea G un grupo cíclico finito. Entonces para todo $a \in G$ se tiene que

1: $a^{|G|} = 1$

2: $\text{ord}(a)$ divide a $|G|$

Teorema

Sea G un grupo cíclico finito, entonces

1- El número de elementos primativos de G es $\phi(|G|)$.

2- Si $|G|$ es primo entonces todos los elementos $a \neq 1 \in G$ son primativos.

• Subgrupos

Teorema

Sea G un grupo cíclico finito de orden n y sea α un generador de G . Entonces para todo entero k que divide n existe exactamente un subgrupo cíclico H de G de orden k . Este subgrupo está generado por $\alpha^{\frac{n}{k}}$. H consiste exactamente de los elementos $a \in G$ tales que $a^k = 1$. No hay otros subgrupos.

8.3 El problema del logaritmo discreto en campos primos

Sea p primo

Definición (El problema del logaritmo discreto (DLP) en \mathbb{Z}_p^*)

Dados el grupo cíclico \mathbb{Z}_p^* , un elemento primitivo $\alpha \in \mathbb{Z}_p^*$ y algún otro elemento $\beta \in \mathbb{Z}_p^*$. El DLP es el problema de encontrar el entero $1 \leq x \leq p-1$ tal que

$$\alpha^x \equiv \beta \pmod{p}$$

Observación

Como α es primitivo \Rightarrow dicho x debe de existir

• A x se le llama el **logaritmo discreto** de β base α y escribimos

$$x = \log_{\alpha} \beta \pmod{p}$$

Nota: Para parámetros suficientemente grandes, calcular logaritmos discretos módulo un primo es un problema muy difícil.

Observación

Como $\alpha^x \equiv \beta \pmod{p}$ es computacionalmente fácil (exponenciación)
⇒ tenemos una función de una dirección.

Nota: Como $|\mathbb{Z}_p^*| = p-1$ no es primo, a menudo se usa DLPs en subgrupos de \mathbb{Z}_p^* de orden primo

Ejemplo:

Consideremos el grupo \mathbb{Z}_{47}^* de orden 46
⇒ los subgrupos de \mathbb{Z}_{47}^* tienen orden 23, 2 o 1

Como 23 es primo ⇒ todos los elementos en el grupo de cardinalidad 23 son generadores.

Escogemos el 2 de dicho subgrupo

Un DLP posible está dado por $p=36$:

$$2^x \equiv 36 \pmod{47}$$

Usando fuerza bruta se obtiene $x=17$.

8.3.2 El problema del logaritmo discreto generalizado

Nota: Algo que hace a DLP muy útil es que puede definirse sobre cualquier grupo cíclico.

Definición (Problema del logaritmo discreto generalizado)

Dado un grupo cíclico finito G con operación de grupo \circ y cardinalidad n . Consideraremos un elemento primitivo $\alpha \in G$ y algún otro elemento $\beta \in G$. El problema del logaritmo discreto es encontrar el entero x , con $1 \leq x \leq n$, tal que

$$\beta = \underbrace{\alpha \circ \alpha \circ \dots \circ \alpha}_{x\text{-veces}} = \alpha^x$$

Observación

Como α es primitivo $\Rightarrow x$ debe existir

iNOTA! Existen grupos cíclicos donde el problema del logaritmo discreto no es difícil \Rightarrow No pueden usarse en criptosistemas de llave pública.

Ejemplo: Consideremos el grupo aditivo $G = (\mathbb{Z}_{11}, +)$.

G es un grupo cíclico finito y $\alpha = 2 \in G$ es un elemento primitivo del mismo

Tratemos de resolver el DLP para $\beta = 3$; esto es, busquemos x tal que $1 \leq x \leq 11$ y

$$x \cdot 2 = \underbrace{2 + 2 + \dots + 2}_{x\text{-veces}} \equiv 3 \pmod{11}$$

$$2^x \equiv 6 \pmod{11} \Rightarrow x \equiv 2^{-1} \cdot 3 \pmod{11} = 6 \cdot 3 \pmod{11} = 18 \pmod{11} \equiv 7 \pmod{11}$$

Listado de DLPs propuestos para ser usados en criptografía:

- 1: Grupo multiplicativo (o un subgrupo) el campo primo \mathbb{Z}_p .
- 2: Grupo cíclico formado por una curva elíptica (Capítulo 9).
- 3: El grupo multiplicativo de un campo de Galois $GF(2^m)$ o un subgrupo de este. **Nota:** No son tan populares porque los ataques hacia estos son más poderosos que los de DLP en $\mathbb{Z}_p \Rightarrow$ se requiere una llave de mayor longitud

4. Curvas hiperelípticas o variedades algebraicas (raramente usadas)

8.3.3 Ataques en contra del DLP

Los algoritmos para calcular logaritmos discretos que veremos se pueden clasificar en **algoritmos genéricos** y **algoritmos no genéricos**.

Algoritmos genéricos

- ↑ usan únicamente la operación del grupo
- no se analizan las propiedades específicas de los grupos
 - ⇒ funcionan el cualquier grupo cíclico

Se pueden subdividir en dos clases:

- los que su tiempo de ejecución depende del tamaño del grupo cíclico.
Por ejemplo: fuerza bruta, el algoritmo baby-step giant-step y el método de Pollard's rho.
- los que su tiempo de ejecución depende del tamaño de los factores primos del orden del grupo.
Ejemplo: algoritmo Pohlig-Hellman.

Fuerza bruta

Se calculan las potencias de α hasta que coincide con β .

$$\alpha^1 \stackrel{?}{=} \beta$$

$$\alpha^2 \stackrel{?}{=} \beta$$

⋮

Usando este método se espera, para un exponente aleatorio x , se espera encontrar una solución después de verificar con la mitad de todos los posibles valores de x .

$$\alpha^x \stackrel{?}{=} \beta$$

⇒ Esto nos da una complejidad de $\mathcal{O}(|G|)$ pasos, donde $|G|$ es la cardinalidad de G .

⇒ La cardinalidad $|G|$ debe ser suficientemente grande para evitar ataques de fuerza bruta en criptosistemas DLP-basados.

Ejemplo: Para el grupo \mathbb{Z}_p^* con p primo se requieren $\frac{(p-1)}{2}$ intentos (en promedio) para calcular un logaritmo discreto.

⇒ Para evitar ataques de fuerza bruta $|G|=p-1$ debe estar en el rango de 2^{128} .

El método Baby-step Giant-step de Shank

La idea se basa en reescribir el logaritmo discreto $x = \log_a \beta$ en una representación de dos dígitos:

$$x = x_g m + x_b \quad \text{para } 0 \leq x_g, x_b < m$$

El entero m se elige como la raíz cuadrada del orden del grupo
 $m = \sqrt{|G|}$.

$$\Rightarrow \beta = a^x = a^{x_g m + x_b}$$

La idea del algoritmo es encontrar (x_g, x_b) (\Rightarrow tenemos el valor de x)

¿Cómo?

• Baby-step:

Calcular y almacenar todos los valores a^{x_b} , con $0 \leq x_b < m$.

Aquí estamos haciendo $m \approx \sqrt{|G|}$ pasos y almacenando $m \approx \sqrt{|G|}$ elementos

• Giant-step:

Verificar para todos los x_g con $0 \leq x_g < m$ si se satisface
 $\beta \cdot (\alpha^{-m})^{x_g} = \alpha^{x_b}$
 para algún α^{x_b} calculado en la fase Baby-step.

- Si se satisface

$$\Rightarrow \beta \cdot (\alpha^{-m})^{x_{g,0}} = \alpha^{x_{b,0}} \text{ para algún par } (x_{g,0}, x_{b,0})$$

$$\Rightarrow x = x_{g,0}m + x_{b,0}$$

Este método requiere $O(\sqrt{|G|})$ pasos y la misma cantidad de memoria.

Ejemplo: En un grupo G con $|G| = 2^{128}$ el atacante necesita aprox. $2^{64} = \sqrt{2^{128}}$ cálculos y espacios de memoria (lo cual actualmente es posible de realizar)

\Rightarrow Para tener un ataque de complejidad 2^{128} el grupo debe tener cardinalidad $|G| \geq 2^{256}$

Si $G = \mathbb{Z}_p^*$ $\Rightarrow p$ debe tener longitud de al menos 256 bits.

El método Rho de Pollard \leftarrow método probabilístico

A igual que el método previo, tiene un tiempo de ejecución de $O(\sqrt{|G|})$ pero utiliza menos memoria

Idea general: generar un grupo de elementos de la forma $\alpha^i \beta^j$ (de manera pseudoaleatoria) teniendo en cuenta los valores i y j (rastreándolos)

Se generan estos elementos hasta obtener

$$\alpha^{i_1} \beta^{j_1} = \alpha^{i_2} \beta^{j_2}$$

Si sustituimos $\beta = \alpha^x$ y comparamos los exponentes a cada lado de la ecuación previa tenemos

$$i_1 + x j_1 \equiv i_2 + x j_2 \pmod{16}$$

$$\Rightarrow x \equiv \frac{i_2 - i_1}{j_1 - j_2} \pmod{16}$$

- Actualmente es el mejor algoritmo conocido para calcular logaritmos discretos en grupos de curvas elípticas.

Algoritmo Pohlig-Hellman

Basado en el Teorema Chino del Residuo

Normalmente se utiliza en conjunto con otro ataque DLP.

Sea $|G| = p_1^{e_1} \cdot p_2^{e_2} \cdots p_r^{e_r}$ la factorización en primos de $|G|$.

Se calculan los logaritmos discretos

$$x_i \equiv x \pmod{p_i^{e_i}}$$

en cada subgrupo de orden $p_i^{e_i}$

} con el algoritmo que quieras
(método Rho ó Baby-step
Giant step)

Se obtiene x de todos los x_i usando el Teorema Chino del Residuo.

El tiempo de ejecución del algoritmo depende de los factores primos de $|G|$.

Para prevenir el ataque, el grupo debe tener un orden tal que el factor primo más grande esté en el rango de 2^{256}

Este algoritmo necesita conocer una factorización de $|G|$. En el caso de criptosistemas de curvas elípticas, calcular el orden del grupo cíclico no siempre es sencillo.

Algoritmos no genéricos

El método de cálculo de índice

Explota propiedades especiales del grupo en específico

Es un algoritmo muy eficiente para calcular logaritmos discretos en \mathbb{Z}_p^* y $GF(2^n)^*$.

Tiempo de ejecución subexponencial

Propiedad: Una porción significativa de elementos en G puede ser expresada como un producto de elementos de un subconjunto pequeño de G .

Ejemplo: En \mathbb{Z}_p^* muchos elementos pueden expresarse como un producto de primos pequeños

Nota: No se sabe si esta propiedad se satisface para grupos de curvas elípticas.

Para garantizar seguridad ante este ataque de 128 bits (i.e. el atacante debe hacer 128 pasos), el primo p en un DLP en \mathbb{Z}_p^* debe tener al menos 3072 bits de longitud.

9. Criptosistemas de curvas elípticas (ECC)

- Provee el mismo nivel de seguridad que RSA o sistemas de logaritmos complejos con operandos considerablemente más cortos.
- Basado en el problema de logaritmo discreto generalizado.
- RSA con llaves cortas es mucho más rápido que ECC

9.1 Cómo hacer cálculos con curvas elípticas

Paso 1: Encontrar un grupo cíclico en el cual construir un criptosistema
¡No sólo debe existir un grupo cíclico!

Debemos encontrar un grupo cíclico en el que el problema del logaritmo discreto sea computacionalmente complicado

9.1.1 Definición de curvas elípticas

→ Por curva nos referimos a puntos que satisfacen una ecuación.

Definición

La **curva elíptica** sobre \mathbb{Z}_p , $p > 3$, es el conjunto de pares $(x, y) \in \mathbb{Z}_p^2$ que satisfacen

$$y^2 \equiv x^3 + a \cdot x + b \pmod{p}$$

en conjunto con un punto imaginario de infinito \mathcal{O} donde $a, b \in \mathbb{Z}_p$ y la condición

$$4 \cdot a^3 + 27 \cdot b^2 \neq 0 \pmod{p} \quad (*)$$

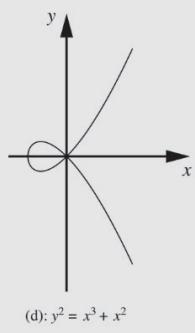
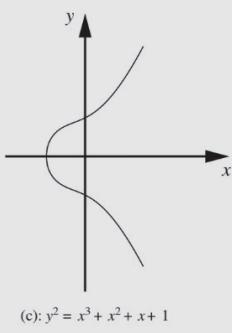
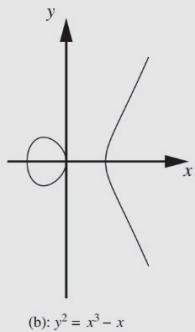
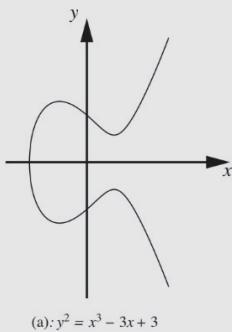
La condición (*)

$$4 \cdot a^3 + 27 \cdot b^2 \neq 0 \pmod{p} \Leftrightarrow \underbrace{-16(4a^3 + 27b^2)}_{\text{determinante de la curva}} \neq 0 \quad (p > 3 \text{ primo})$$

\Leftrightarrow la imagen no tiene puntos de autointersección

Observación

Podemos graficar sobre \mathbb{R} para obtener algo que se ve como una curva y notar algunas propiedades:



(1) Las curvas elípticas son simétricas respecto al eje x .

$$\begin{aligned} y^2 &\equiv x^3 + a \cdot x + b \pmod{p} \\ \Rightarrow y_i &= \sqrt{x_i^3 + ax_i + b} \\ y'_i &= -\sqrt{x_i^3 + ax_i + b} \end{aligned}$$

(2) Hay 0, 1, 2 o 3 intersecciones con el eje x

$$\begin{aligned} &\left(\begin{array}{l} \text{las soluciones a la ecuación} \\ 0 = x^3 + ax + b \end{array} \right) \end{aligned}$$

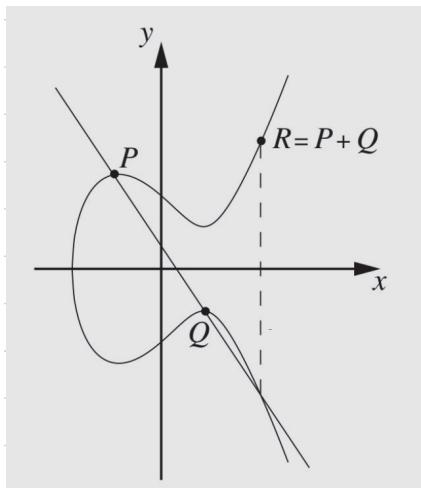
Recordemos:

Paso 1: Encontrar un grupo cíclico en el cual construir un criptosistema

Conjunto: soluciones a un curva elíptica, Operación: \circ ?

9.1.2 Operaciones de grupo en curvas elípticas

→ Suma de puntos distintos $P+Q$



1: Dibujar una línea recta que pasa por P y Q

2: Obtener un tercer punto de intersección entre la linea recta y la curva elíptica.

3: Tomar la reflexión de dicho punto sobre el eje x

4: Definimos $R := P+Q$ como dicho punto.

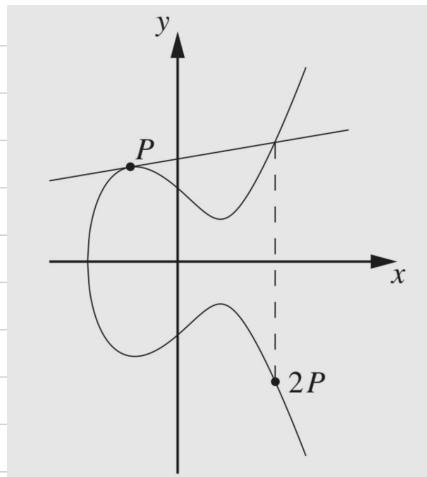
→ Suma de puntos iguales $P+P$

1: Dibujamos la línea tangente a la curva elíptica en el punto P .

2: Obtenemos un segundo punto de intersección entre esta linea y la curva elíptica.

3: Tomar la reflexión de este punto sobre el eje x .

4: Definimos $R := 2P$ como dicho punto.



Nota: A este proceso se le llama el método tangente y cuerda y cumple los axiomas de grupo.

→ Pasemos de los reales a un campo primo $GF(p)$ y traduzcamos esta suma a algo más aritmético /analítico y menos geométrico.

Adición y multiplicación por 2 en curvas elípticas

$$x_3 \equiv s^2 - x_1 - x_2 \pmod{p}$$

$$y_3 \equiv s(x_1 - x_3) - y_1 \pmod{p}$$

$$P = (x_1, y_1)$$

$$Q = (x_2, y_2)$$

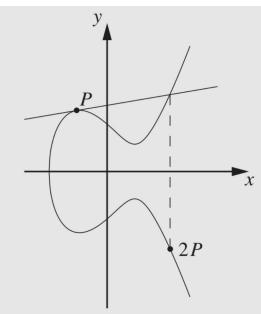
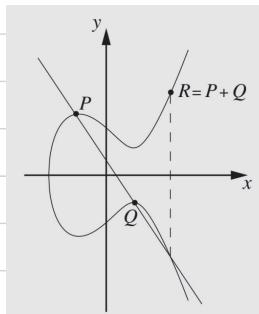
$$R = (x_3, y_3)$$

donde

$$s \equiv \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \pmod{p}, & \text{si } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \pmod{p} & \text{si } P = Q \end{cases}$$

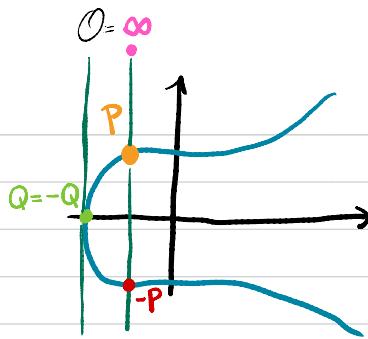
Observación

- En la adición, s es la pendiente de la curva que pasa por P y Q
- En la multiplicación por 2, s es la pendiente de la recta tangente en el punto P



Ya tenemos una operación **¿Cuál es el neutro?**

! No hay un punto (x,y) que se comporte como neutro



Neutro = punto al infinito

¿Cuáles son los inversos?

$$\text{Sea } P = (x, y) \Rightarrow -P = (x, p-y)$$

Encontrar el inverso es fácil: $-y \equiv p-y \pmod{p}$

$$\Rightarrow -P = (x, p-y)$$

Esto nos da un grupo $(GF(p), +)$

9.2 Construyendo un problema del logaritmo discreto con curvas elípticas

Teorema

Los puntos de una curva elíptica en conjunto con el punto al infinito tiene subgrupos cíclicos. Bajo ciertas condiciones todos los puntos de una curva elíptica son un grupo cíclico.

Ejemplo: El grupo cíclico de una curva elíptica

Queremos encontrar todos los puntos de la curva

$$E: y^2 = x^3 + 2 \cdot x + 2 \pmod{17}$$

Sucede que el número de puntos en la curva y el orden del grupo son $\#E=19 \Rightarrow$ El grupo es cíclico

Veamos cuáles son estos elementos

Comenzamos con el elemento primitivo $P = (5, 1)$. Calculando las potencias de P obtenemos los siguientes elementos

$$2P = (5, 1) + (5, 1) = (6, 3)$$

$$3P = 2P + P = (10, 6)$$

$$4P = (3, 1)$$

$$5P = (9, 16)$$

$$6P = (16, 13)$$

$$7P = (0, 6)$$

$$8P = (13, 7)$$

$$9P = (7, 6)$$

$$10P = (7, 11)$$

$$11P = (13, 10)$$

$$12P = (0, 11)$$

$$13P = (16, 4)$$

$$14P = (9, 1)$$

$$15P = (3, 16)$$

$$16P = (10, 11)$$

$$17P = (6, 14)$$

$$18P = (5, 16)$$

$$19P = \emptyset$$

Nota: Para construir un criptosistema de logaritmo discreto es importante conocer la cardinalidad del grupo con el que estamos trabajando.

El sig. teo. nos da un aproximado

Teorema (Hasse's Theorem)

Dada una curva elíptica E módulo p , el número de puntos en la curva se denota por $\#E$ y satisface lo sig

$$p+1-2\sqrt{p} \leq \#E \leq p+1+2\sqrt{p}$$

⇒ el orden del grupo $\approx p$

- Si queremos usar una curva elíptica con 2^{256} elementos
⇒ necesitamos un primo de longitud ≈ 256 bits

Definición (Problema del logaritmo discreto para curvas elípticas (ECDLP))

Dada una curva elíptica E , consideramos un elemento primitivo P y algún otro elemento T . El problema DL es encontrar el entero d , con $1 \leq d \leq \#E$, tal que

$$\underbrace{P + P + \dots + P}_{d-\text{veces}} = d \cdot P = T$$

En criptosistemas:

- $d \leftarrow$ llave privada
- $T \leftarrow$ llave pública

→ punto de la curva elíptica de coordenadas $T = (x_T, y_T)$

Observación: Cuando hacíamos DL en \mathbb{Z}_p^* ambas llaves eran enteros (aquí ya no)

Ejemplo:

→ Resolveremos el problema DL en la curva

$$y^2 \equiv x^3 + 2x + 2 \pmod{17}$$

→ Elementos dados: $P = (5, 1)$ elemento primitivo
 $T = (16, 4)$ llave pública

Recordemos los cálculos dados en el ejemplo previo

$$\begin{aligned} 2P &= (5, 1) + (5, 1) = (6, 3) \\ 3P &= 2P + P = (10, 6) \\ 4P &= (3, 1) \\ 5P &= (9, 16) \\ 6P &= (16, 13) \\ 7P &= (0, 6) \\ 8P &= (13, 7) \\ 9P &= (7, 6) \\ 10P &= (7, 11) \end{aligned}$$

$$\begin{aligned} 11P &= (13, 10) \\ 12P &= (0, 11) \\ 13P &= (16, 4) \\ 14P &= (9, 1) \\ 15P &= (3, 16) \\ 16P &= (10, 11) \\ 17P &= (6, 14) \\ 18P &= (5, 16) \\ 19P &= \mathcal{O} \end{aligned}$$

$$\Rightarrow 13 \cdot P = (16, 4) = T$$

$$\Rightarrow d = 13$$

↑ llave privada

¿Cómo realizamos estos cálculos de forma eficiente?

↓ "análogo" al algoritmo elevar al cuadrado - multiplicar

Algoritmo Doble-y-Suma para Multiplicación de puntos

Input: - curva elíptica E

- punto P de la curva

- número $d = \sum_{i=0}^t d_i 2^i$ (con $d_i \in \{0, 1\}$ y $d_t = 1$)

Output: $T = d \cdot P$

Algoritmo:

$$T = P$$

Para $i = t-1$ hasta 0 (va disminuyendo)

$$T = T + T$$

$$\left[\begin{array}{l} \text{Si } d_i = 1 \\ T = T + P \end{array} \right]$$

Return (T)

- Para un escalar con longitud de $t+1$ bits, el algoritmo requiere $1.5t$ multiplicaciones por 2 y sumas (en promedio)

Ejemplo: Multiplicación escalar 26P

$$26P = (11010_2)P = (d_4 d_3 d_2 d_1 d_0)_2 \cdot P$$

Step

$$\#0 \quad P = 1_2 P$$

initial setting, bit processed: $d_4 = 1$

$$\#1a \quad P + P = 2P = \mathbf{10}_2 P$$

DOUBLE, bit processed: d_3

$$\#1b \quad 2P + P = 3P = 10_2 P + 1_2 P = \mathbf{11}_2 P$$

ADD, since $d_3 = 1$

$$\#2a \quad 3P + 3P = 6P = 2(11_2 P) = \mathbf{110}_2 P$$

DOUBLE, bit processed: d_2

$$\#2b$$

no ADD, since $d_2 = 0$

$$\#3a \quad 6P + 6P = 12P = 2(110_2 P) = \mathbf{1100}_2 P$$

DOUBLE, bit processed: d_1

$$\#3b \quad 12P + P = 13P = 1100_2 P + 1_2 P = \mathbf{1101}_2 P$$

ADD, since $d_1 = 1$

$$\#4a \quad 13P + 13P = 26P = 2(1101_2 P) = \mathbf{11010}_2 P$$

DOUBLE, bit processed: d_0

$$\#4b$$

no ADD, since $d_0 = 0$

9.3 Intercambio de llaves Diffie-Hellman (DHKE) con curvas elípticas

Parámetros dominio ECDH

1: Escoger un primo p y la curva elíptica

$$E: y^2 \equiv x^3 + a \cdot x + b \pmod{p}$$

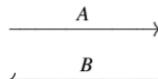
2: Escoger un elemento primitivo $P = (x_p, y_p)$

Parámetros: $\begin{cases} p \\ a, b \text{ (que determinan la curva } E\text{)} \\ P \end{cases}$

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Alice

choose $k_{prA} = a \in \{2, 3, \dots, \#E - 1\}$
compute $k_{pubA} = aP = A = (x_A, y_A)$



compute $aB = T_{AB}$

Bob

choose $k_{prB} = b \in \{2, 3, \dots, \#E - 1\}$
compute $k_{pubB} = bP = B = (x_B, y_B)$

compute $bA = T_{AB}$

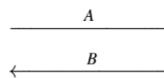
¿Porqué funciona?

Alice calcula $aB = a(bP)$
Bob calcula $bA = b(aP)$

) = por asociatividad y commutatividad
de la suma ✓

Ejemplo:

Alice
choose $a = k_{pr,A} = 3$
 $A = k_{pub,A} = 3P = (10, 6)$



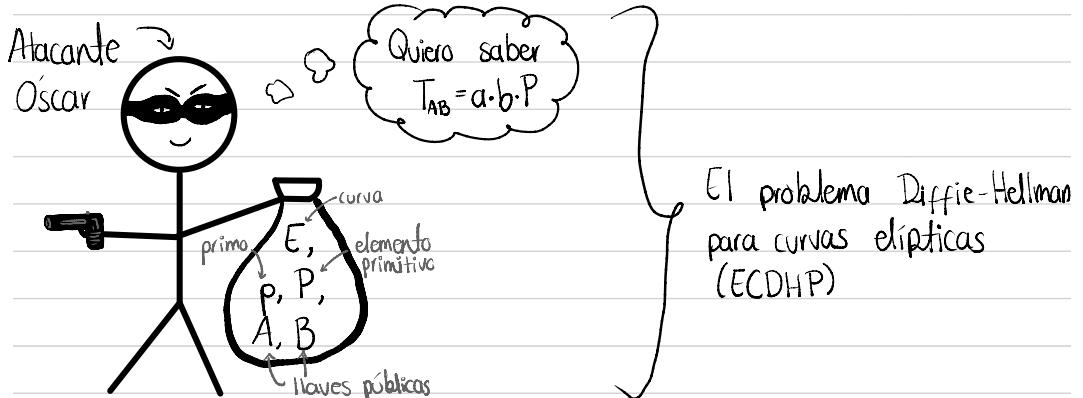
$$T_{AB} = aB = 3(7, 11) = (13, 10)$$

Bob
choose $b = k_{pr,B} = 10$
 $B = k_{pub,B} = 10P = (7, 11)$

$$T_{AB} = bA = 10(10, 6) = (13, 10)$$

9.4 Seguridad

- Buenas propiedades de una dirección



⇒ necesita resolver alguno de los problemas de logaritmo discreto
 $a = \log_p A$ ó $b = \log_p B$

Si la curva elíptica se elige con cuidado, entonces los mejores ataques conocidos contra ECDLP son considerablemente más débiles que los mejores algoritmos para resolver el problema DL módulo p y los mejores algoritmos de factorización para ataques contra RSA.

La seguridad se logra sólo si se usan curvas elípticas criptográficamente fuertes.