



eryx

zkCity

Día 1: Introducción a la criptografía

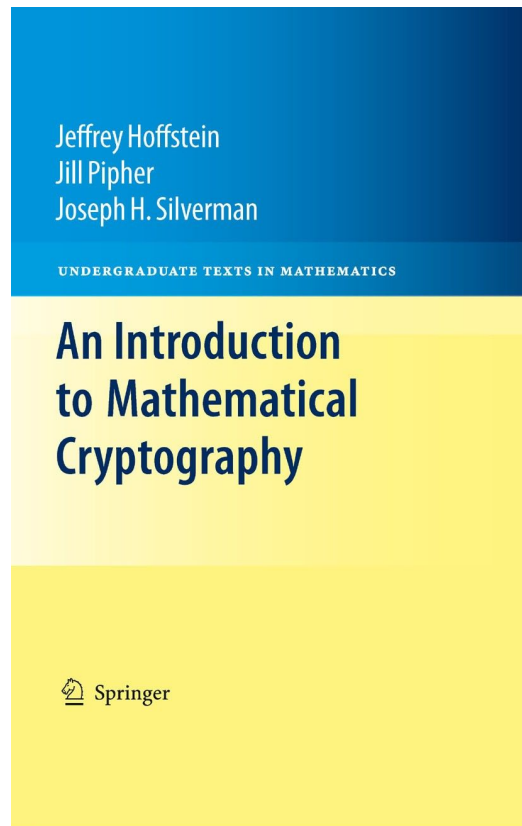
¿Qué vamos a ver?

- Introducción básica a matemática y criptografía.
- Entender qué es zero knowledge.
- Implementar un protocolo conocido de ZK.

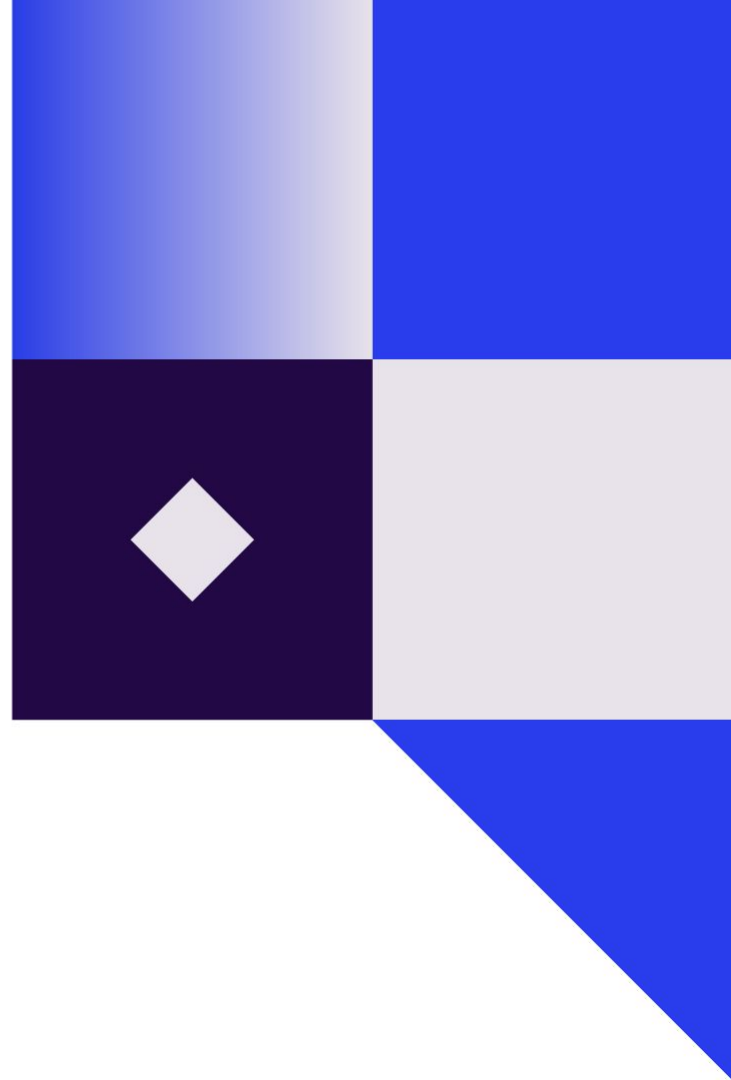


Hoy

- Introducción a criptografía y matemática.
- Implementación de un protocolo de ZK básico.



Intro





Esteganografía

Ocultar información en un objeto (físico o digital):

- Tinta invisible.
- Acrósticos.
- Imágenes y PCA.



Esteganografía

Ocultar información en un objeto (físico o digital):

- Tinta invisible.
- Acrósticos.
- Imágenes y PCA.



Problema: ¿Qué pasa si alguien encuentra el mensaje escondido?

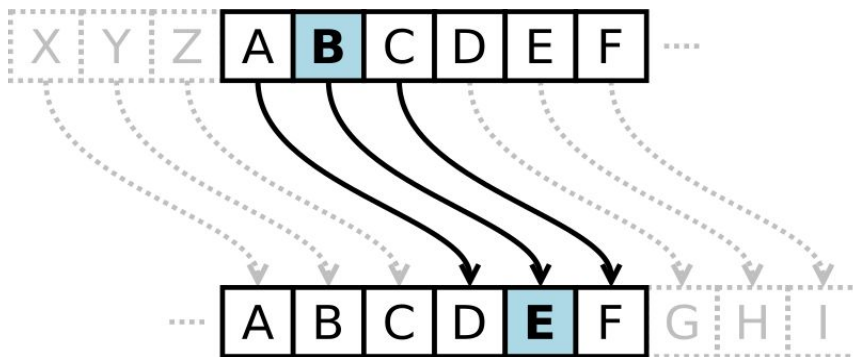
Criptografía

Inclusive si tenés acceso al mensaje, no entendés qué es.

DJXDQWH PHVVL

Cifrado César

Reemplazamos cada letra por la que está K lugares a la derecha.



Alfabeto rotado 3 posiciones.

Ejemplo

DJXDQWH PHVVL

Ejemplo

DJXDQWH PHVVL

Alfabeto: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Rotado : DEFGHIJKLMNOPQRSTUVWXYZABC

Ejemplo

DJXDQWH PHVVL

Alfabeto: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Rotado : DEFGHIJKLMNOPQRSTUVWXYZABC

Ejemplo

AGUANTE MESSI

Cifrado César

Pueden entrar a <https://www.dcode.fr/rot-cipher> y generar sus mensajes!

ROT-N ENCODER

★ PLAIN TEXT TO ROTATE ⓘ

ABCDEFGHIJKLMNOPQRSTUVWXYZ

★ ROTATION TO USE ROT-N, N=

★ ALPHABET TO USE

- ☒ ABCDEFGHIJKLMNOPQRSTUVWXYZ (IE: ROT13 / CAESAR)
- ☐ ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 (IE: ROT18)
- ☐ 94 PRINTABLE ASCII CHARACTERS FROM - ☐ FULL ASCII TABLE (128 CHARACTERS)
- ☐ CUSTOM ALPHABET (CASE INSENSITIVE)

❌
- ☐ CUSTOM ALPHABET (CASE SENSITIVE)

❌

▶ ENCRYPT

Cifrado simétrico

Alice quiere mandar un mensaje m a Bob.

- Comparten una clave k y conocen funciones D y E .
- Alice encripta: $E(m, k) = c$
- Bob desencripta: $D(c, k) = m$

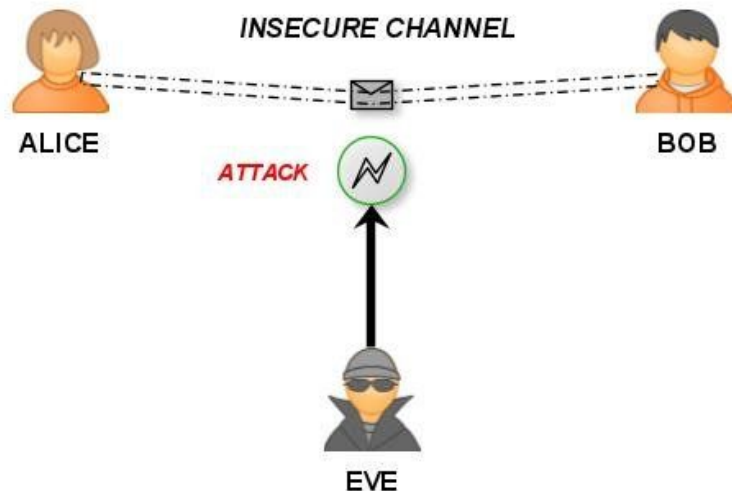


Seguridad

Depende de la mejor estrategia que tenga un adversario **Eve**.

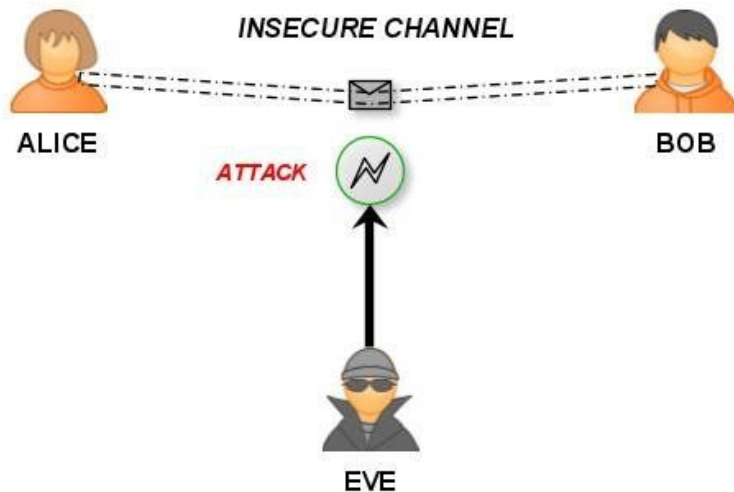
- No debe poder observar, adivinar o recuperar **k**.
- No debe obtener **m** a partir de **c**.

Se evalúa bajo distintos escenarios, por ejemplo con acceso a pares (m, c) .



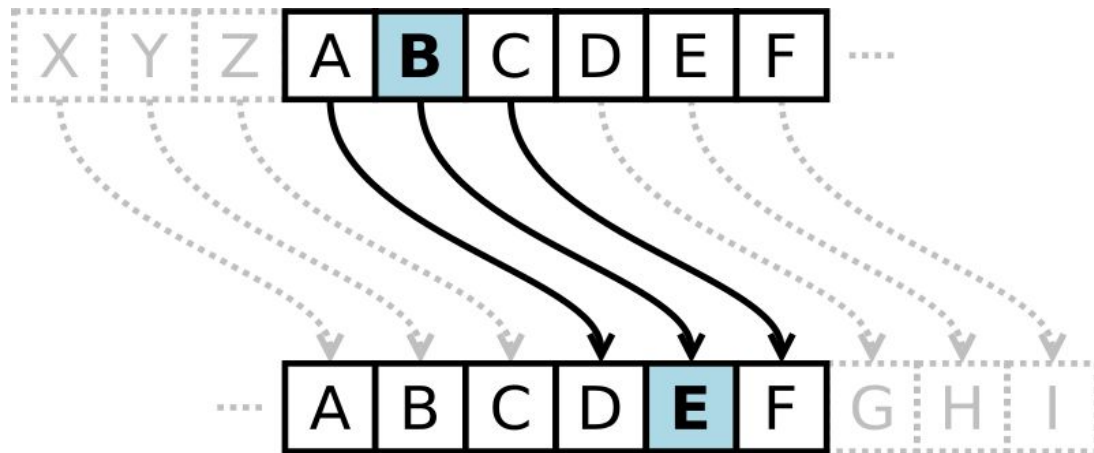
Principio de Kerckhoff

Se asume que un atacante conoce los algoritmos **E** y **D**.



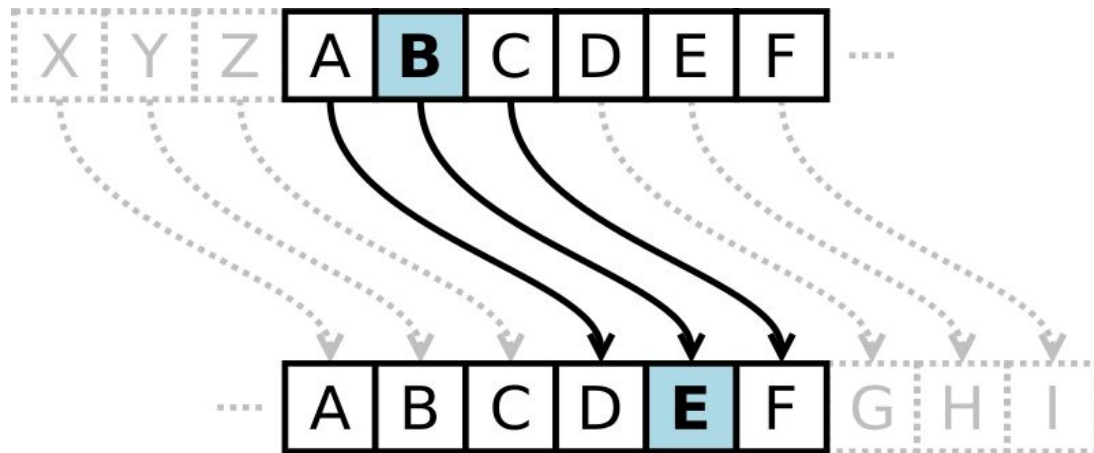
Cifrado César

Analicemos la seguridad:



Cifrado César

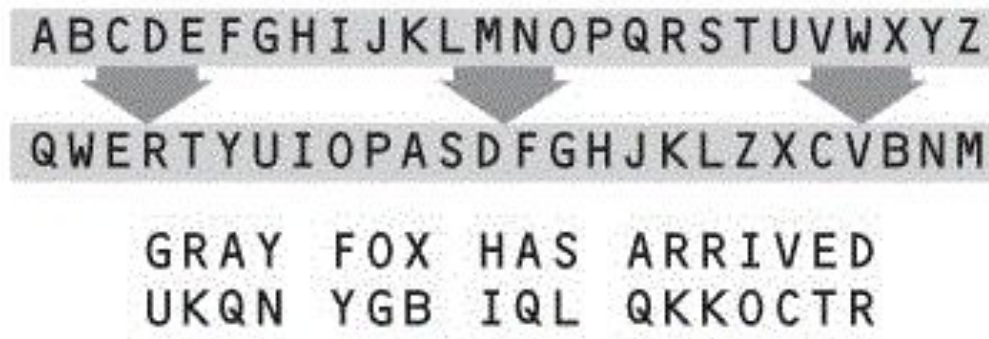
Analicemos la seguridad:



Es inseguro porque hay 26 opciones.

Cifrado César

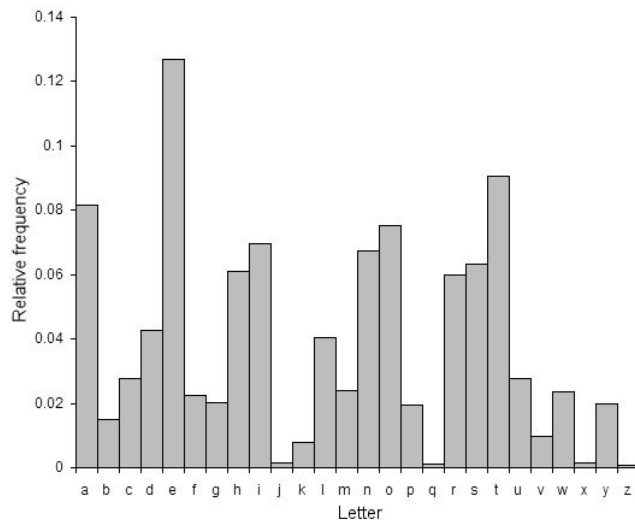
Mejora: que no sea una rotación, que sea una sustitución.



Ahora sí debe ser seguro! Hay 403291461126605635584000000 opciones.

Cifrado César

No es seguro, porque el lenguaje no es aleatorio, y esto se puede explotar usando análisis de frecuencia:



Cifrado de Vigenere

También llamado el **código indescifrable**.

- Es un cifrado polialfabético.
- En cada posición del mensaje **m** se utiliza una sustitución distinta.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	S
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y

Cifrado de Vigenere

También llamado el **código indescifrable**.

- Es un cifrado polialfabético.
- En cada posición del mensaje **m** se utiliza una sustitución distinta.

También fue descifrado.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	S
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y

Comunicación moderna



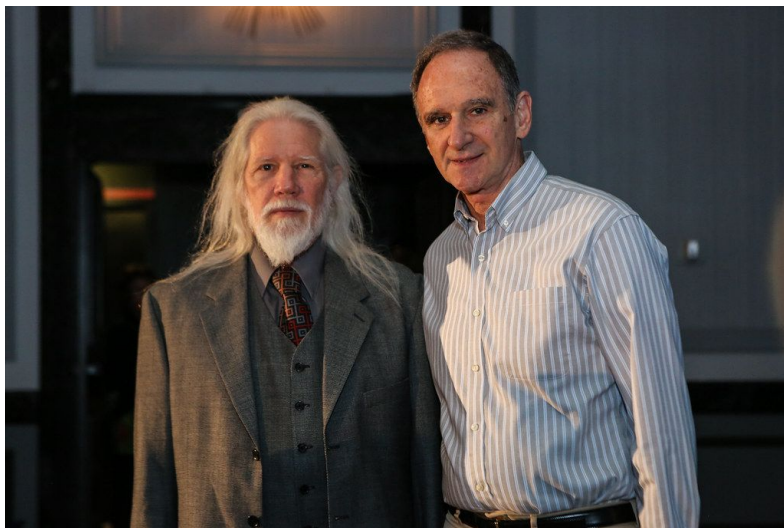
Distribución de claves

Muy costoso (e inseguro) distribuir la clave para mandar mensajes.



Diffie-Hellman

En la década de los 70 publicaron un paper fundacional donde resolvieron este problema y propusieron ideas para varios otros.

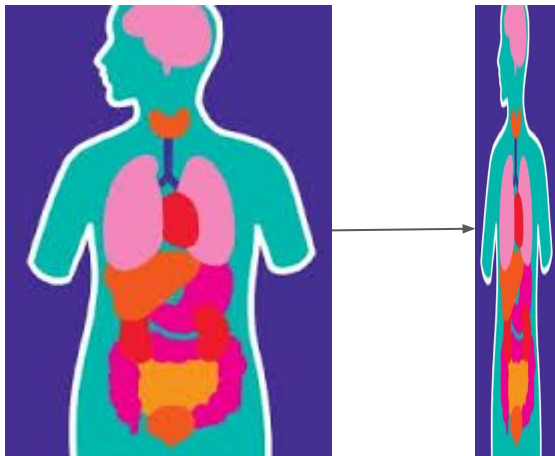


Mates



Cuerpos finitos

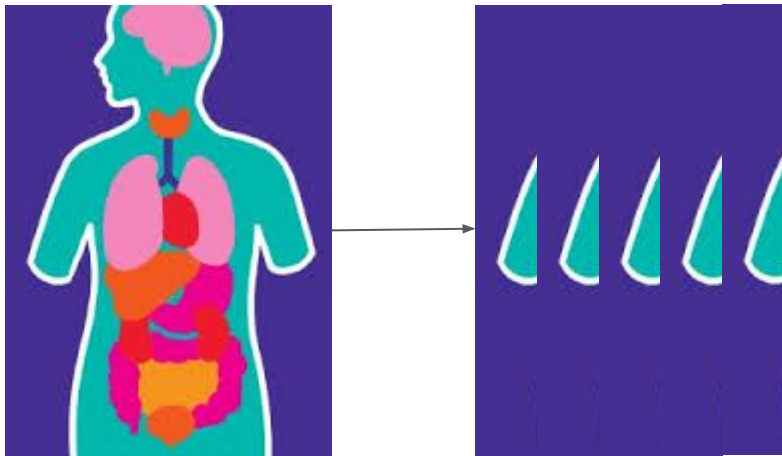
Cuerpos finitos



Z

Z_p

Cuerpos finitos



\mathbb{Z}

\mathbb{Z}_p

Repaso, qué es un cuerpo

Conjunto de elementos con operaciones definidas (+ y *).

Las operaciones “caen” siempre dentro del cuerpo.

Tienen elementos neutros (el 0 y el 1), e inversos siempre dentro del cuerpo (salvo el inverso multiplicativo del 0).

El cuerpo ejemplo: restos módulo un primo. F_p para los amigos.

El grupo de unidades de F_p

Un grupo es como un cuerpo pero sólo con una única operación.

Vamos a definir el grupo F_p^* a partir de F_p , donde la operación va a ser el producto de números (módulo p), y los elementos son todos los que tienen inverso multiplicativo.

Entonces, hay $p-1$ y son los mismos que F_p sin el 0 (¿por qué?)

Problema del logaritmo discreto

¿Qué es elevar a una potencia en F_p^* ? ¿Y encontrar un logaritmo?

Fermatito: $a^p \equiv a \pmod{p}$

Corolario: el logaritmo es un número entre 0 y $p-2$ inclusive.

¿Cómo es el algoritmo para encontrar un logaritmo?

Problema del logaritmo discreto

¿Qué es elevar a una potencia en F_p^* ? ¿Y encontrar un logaritmo?

Fermatito: $a^p \equiv a \pmod{p}$

Corolario: el logaritmo es un número entre 0 y $p-2$ inclusive.

¿Cómo es el algoritmo para encontrar un logaritmo?

A priori no hay nada mejor que probar todo* (lineal en p).
¡Eso es bueno!

*: Hay mucho trabajo en desarrollar formas de romper el problema del logaritmo discreto para algunos primos en particular, y no es verdad que nunca hay nada mejor que probar todo, pero no hay una forma de romper logaritmo discreto (por ahora) para el caso general.

Raíces de la unidad

Algo es una raíz de la unidad si operar con él una cierta cantidad de veces da 1.

Corolario de Fermatito: $a^{p-1} \equiv 1 \pmod{p}$

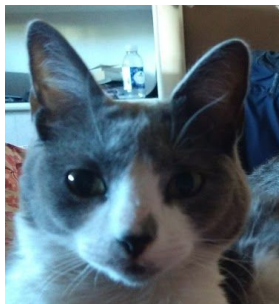
¡Todos los a son raíces!

El **orden de una raíz** es la primera potencia que da 1.

El orden siempre divide a $p-1$. En los ejercicios, $p=2^{16}+1$, con lo cual el orden siempre divide a 2^{16} (o sea, todos los órdenes son potencias de 2)

Diffie-Hellman (protocolo)

Objetivo: acordar una clave simétrica por un canal inseguro (para qué?)



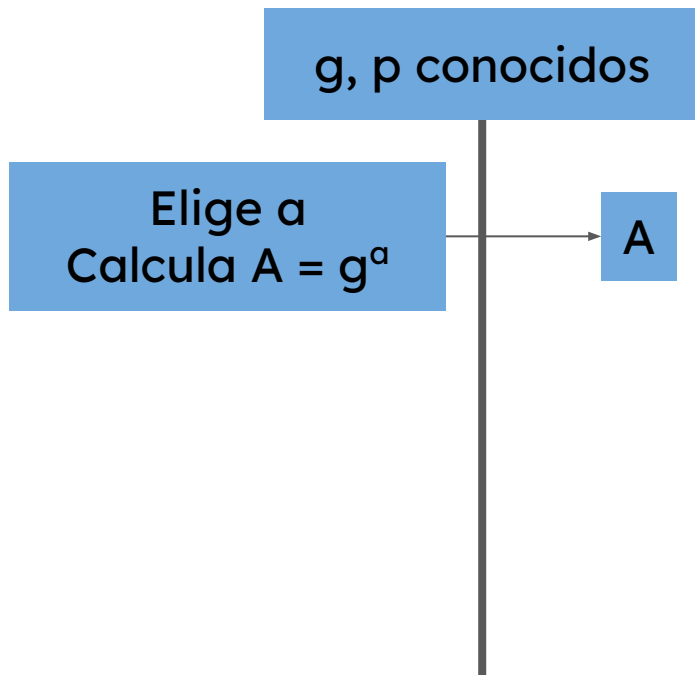
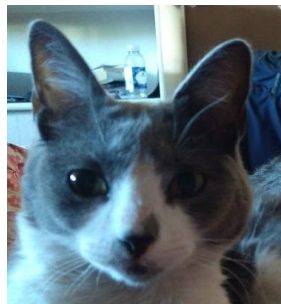
g, p conocidos

p primo,
 g generador
(raíz de orden
alto)



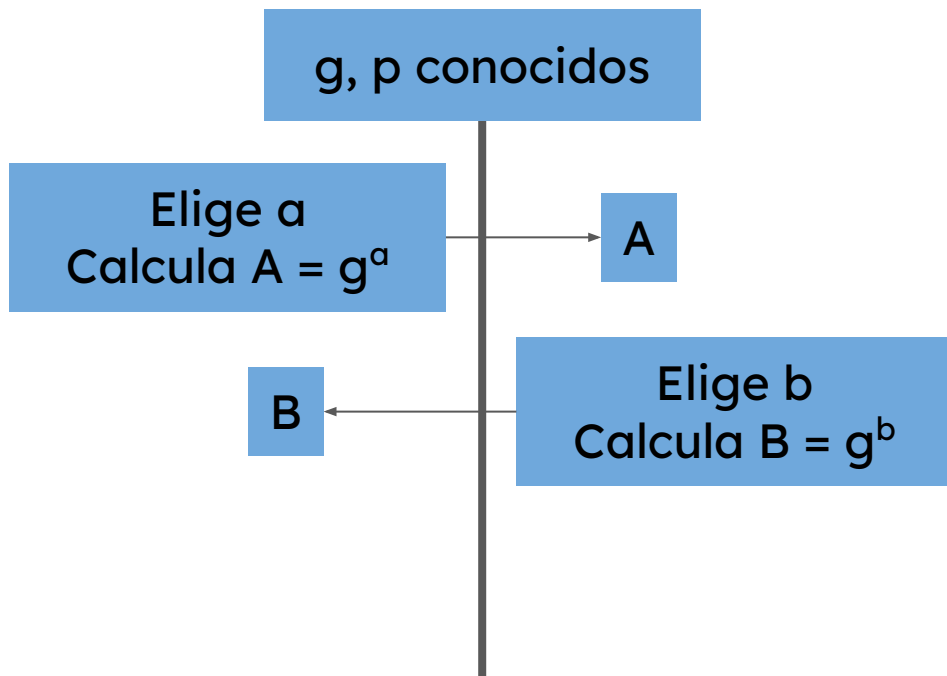
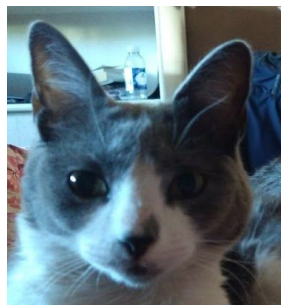
Diffie-Hellman (protocolo)

Objetivo: acordar una clave simétrica por un canal inseguro (para qué?)



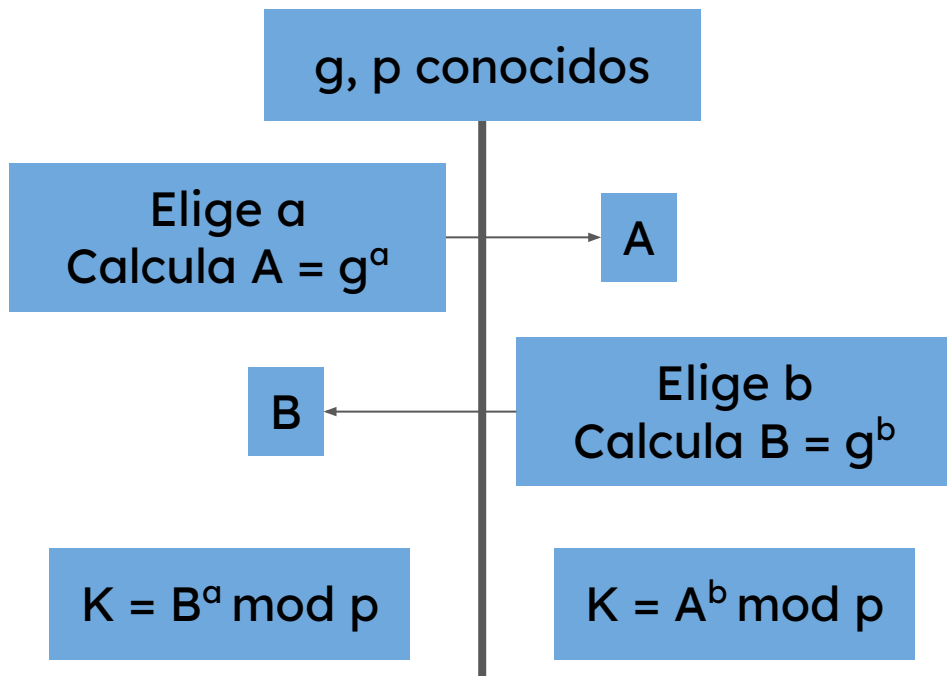
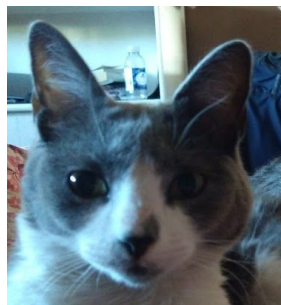
Diffie-Hellman (protocolo)

Objetivo: acordar una clave simétrica por un canal inseguro (para qué?)



Diffie-Hellman (protocolo)

Objetivo: acordar una clave simétrica por un canal inseguro (para qué?)



Curvas elípticas

Son pares de números (x,y) que satisfacen una ecuación de la forma:

$$y^2 = x^3 + ax + b \quad **$$

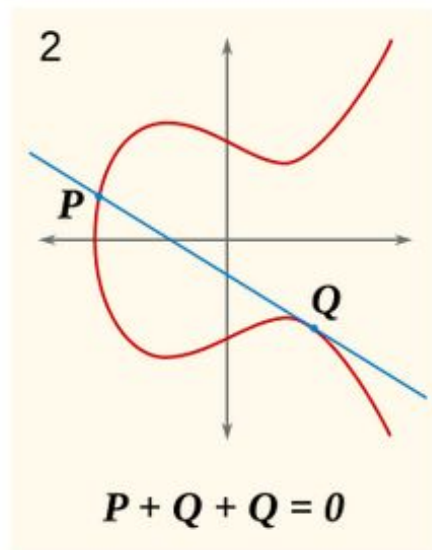
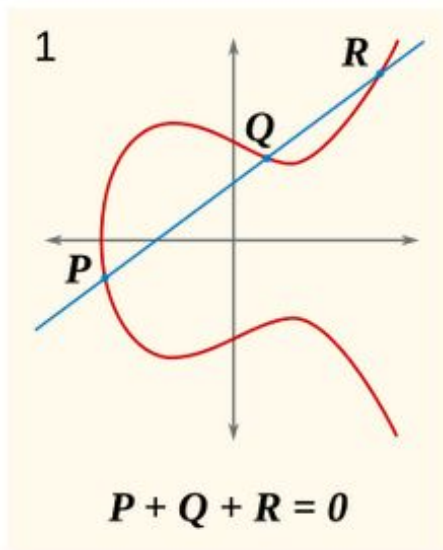
con a y b fijos.

Se puede definir una operación para construir un grupo (la “suma de puntos de curva elíptica”), módulo un par de detalles.

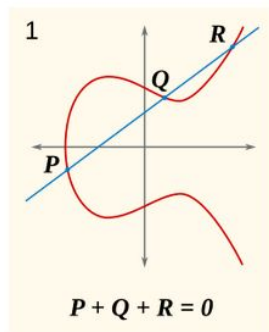
**: Le pedimos que $4a^3+27b^2$ no sea múltiplo de p para que no sea una curva degenerada pero las curvas que van a usar en general ya están fijas de antes así que no importa.

Curvas elípticas (suma)

Idea geométrica:



Curvas elípticas (suma)

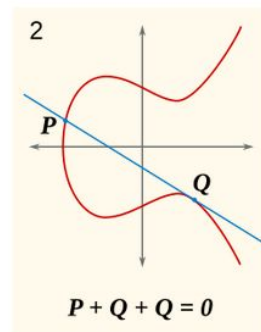


Cuando $X_P \neq X_Q$:

$$s = (y_P - y_Q)/(x_P - x_Q)$$

$$x_R = s^2 - x_P - x_Q$$

$$y_R = -y_P + s(x_P - x_R)$$



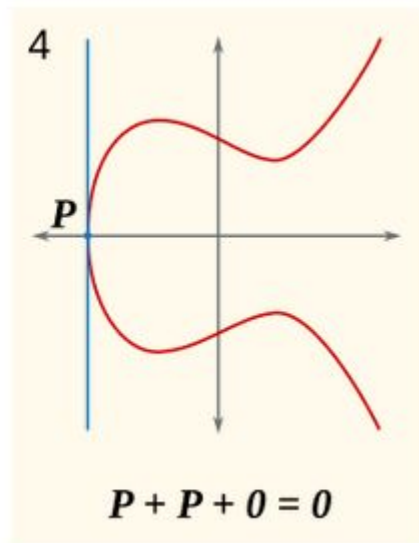
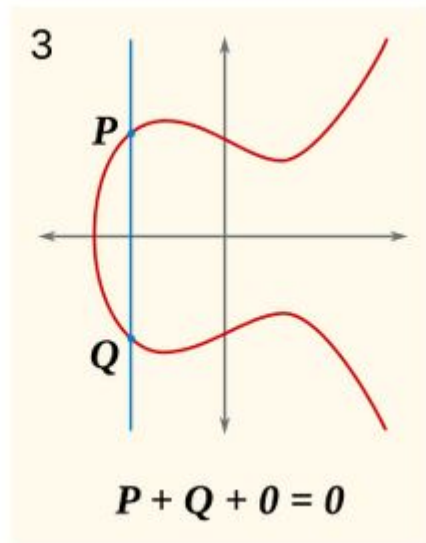
Cuando $X_P = X_Q$:

$$s = (3x_Q - b)/(2y_Q)$$

$$x_R = s^2 - 2x_Q$$

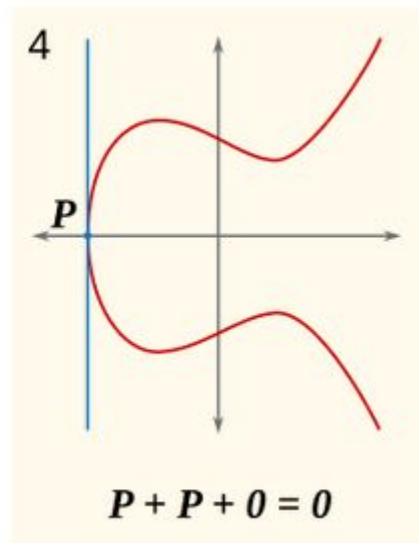
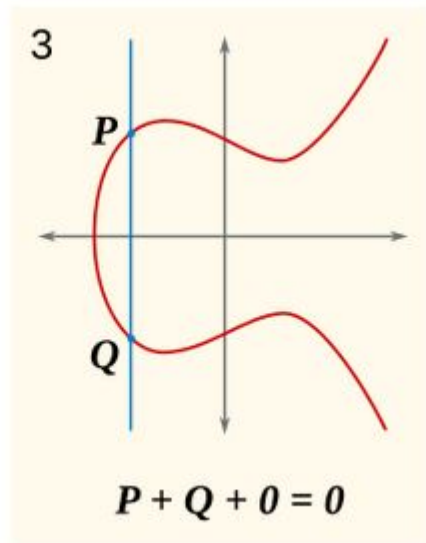
$$y_R = -y_Q + s(x_Q - x_R)$$

Curvas elípticas (el cero)



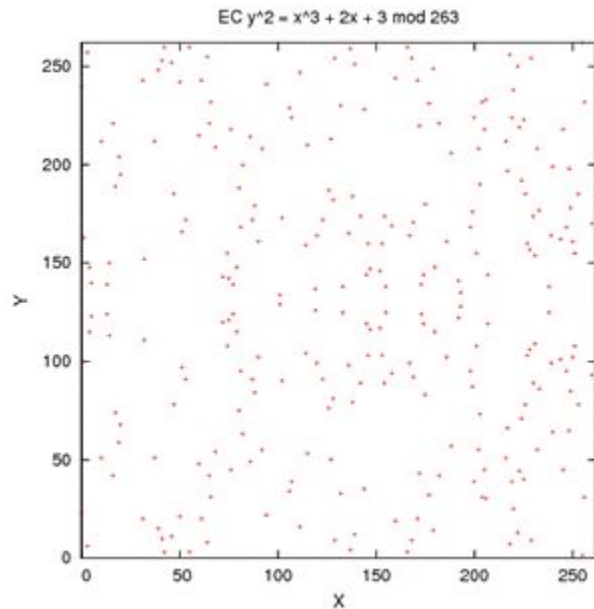
¿Cuál es la tercera intersección en estos casos?

Curvas elípticas (el cero)



¿Cuál es la tercera intersección en estos casos? ¡El “punto infinito”!

Curvas elípticas (módulo p)



Problema del... ¿factor discreto?

Hay un análogo del problema del logaritmo discreto para las curvas elípticas:

Dados dos puntos P y Q , encontrar un número n tal que:

$$\underbrace{P+P+\dots+P}_{n \text{ veces}} = Q$$

Es útil porque no existen las estrategias particulares que existen para el caso del logaritmo discreto.

¿Qué es un buen hash?

Un hash es una **función**:

- Determinística (si hasheo lo mismo dos veces da lo mismo)
- Resistente a colisiones (si hasheo dos cosas distintas da distinto con mucha probabilidad)
- Resistente a buscar la preimagen (no se puede volver de hashear)
- Sensible a cambios de input (cosas parecidas NO tienen hashes parecidos)
- Tiene un output de tamaño fijo
- Se calcula eficientemente

¿Cómo usamos hashes?

El hash es una versión “corta” y “oculta” de un dato. No es lo mismo que “encriptada” porque no se puede desencriptar.

- Verifican integridad de un conjunto de datos
- El hash sirve como identificador de una estructura más grande
- Para comprometerte a un dato en específico (es decir, asegurar que no lo vas a cambiar)

La regla de oro de los hashes

**NO USEN HASHES
CREADOS POR USTEDES.**

ZK



¿Qué es ZK?

Mostrar que una afirmación es cierta sin revelar información secreta.

Conozco secreto s tal que $\text{Hash}(s) = y$.

Convencer a alguien de que esa afirmación es cierta sin revelar s .

Intuición

Problema:

- Una persona **P** puede distinguir el rojo del verde y otra **V** no puede.



Intuición

Problema:

- Una persona **P** puede distinguir el rojo del verde y otra **V** no puede.
- **P** quiere probar la afirmación “puedo distinguirlos”, sin revelar nada extra, como cuál es cuál.



Intuición

Problema:

- Una persona **P** puede distinguir el rojo del verde y otra **V** no puede.
- **P** quiere probar la afirmación “puedo distinguirlos”, sin revelar nada extra, como cuál es cuál.

Protocolo:

1. **V** mezcla los marcadores sin que **P** vea.



Intuición

Problema:

- Una persona **P** puede distinguir el rojo del verde y otra **V** no puede.
- **P** quiere probar la afirmación “puedo distinguirlos”, sin revelar nada extra, como cuál es cuál.

Protocolo:

1. **V** mezcla los marcadores sin que **P** vea.
2. **P** le dice si los cambio o no de orden.



Intuición

Problema:

- Una persona **P** puede distinguir el rojo del verde y otra **V** no puede.
- **P** quiere probar la afirmación “puedo distinguirlos”, sin revelar nada extra, como cuál es cuál.

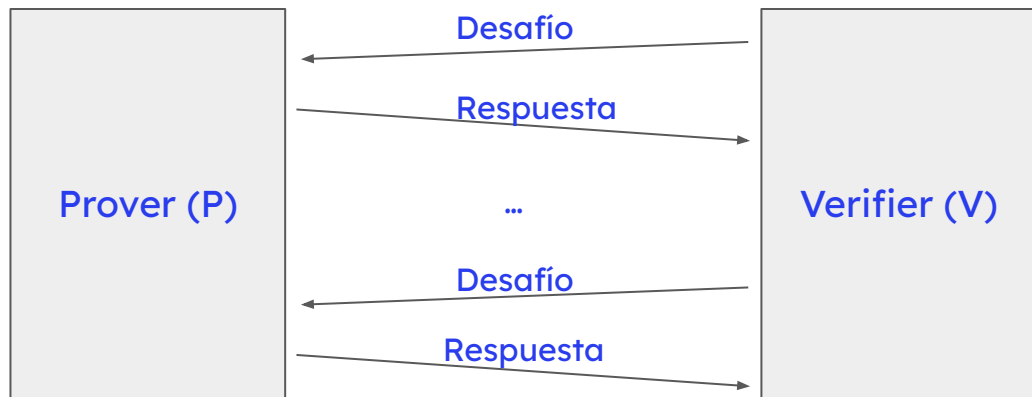
Protocolo:

1. **V** mezcla los marcadores sin que **P** vea.
2. **P** le dice si los cambio o no de orden.
3. Lo repiten hasta que **V** se convence.



Proving system

Es un protocolo interactivo, donde una parte **V** le presenta desafíos a **P**, hasta que se convence de que **P** conoce algo que cumple determinada propiedad.



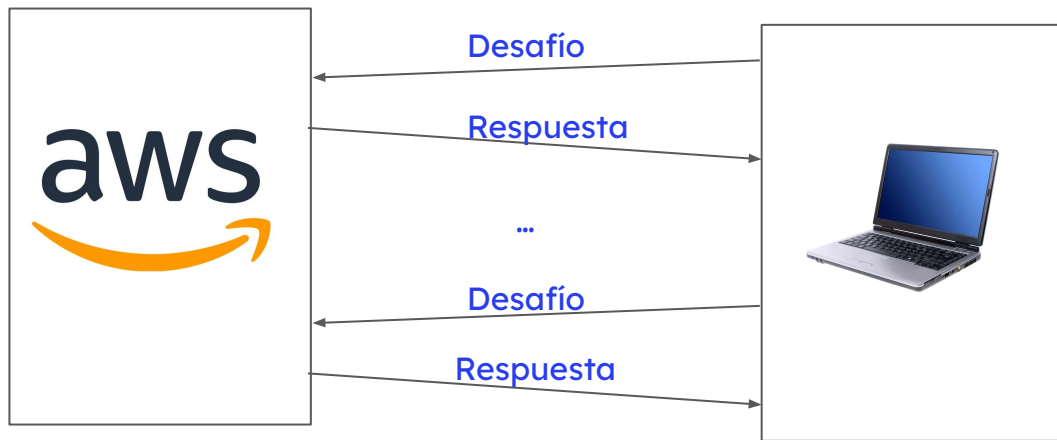
Proving system

Hay ciertas propiedades deseables:

- **Completeness**: toda afirmación “válida” puede ser probada.
- **Soundness**: un prover malicioso no puede convencer a un verifier de una afirmación falsa.
- **Succinct**: una prueba es “chica” y se puede verificar “rápido”.
- **Zero knowledge**: no revela más información que la afirmación que se está probando.

Proving system

Zero knowledge es **opcional**, un caso de uso es la delegación de cómputo.



Amazon me convence de que ejecutó mi programa.

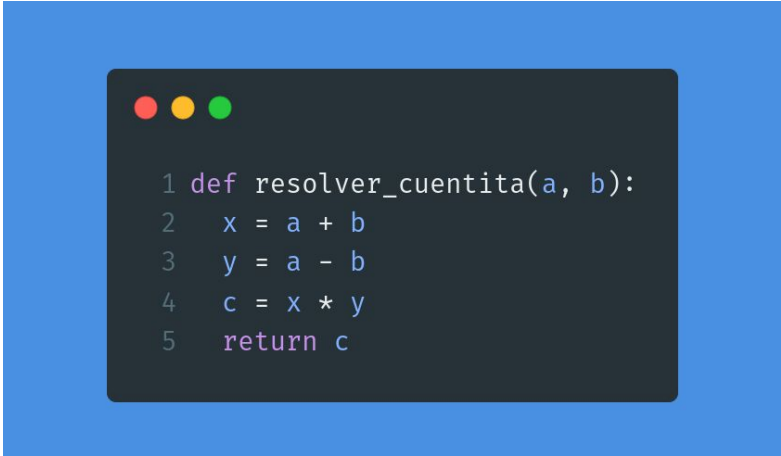
Provable programs

Demostrar cómputo arbitrario parece magia.



Provable programs

Cuando ejecutamos un programa con un input específico, todas sus variables intermedias quedan **determinadas**:



```
1 def resolver_cuentita(a, b):  
2     x = a + b  
3     y = a - b  
4     c = x * y  
5     return c
```

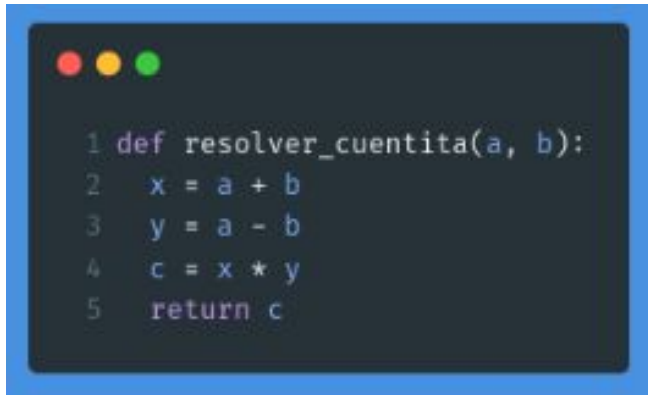
¿Pueden **x** e **y** valer cualquier cosa? ¿Puedo distinguir **(x, y)** “válidos”?

Provable programs

El prover va a demostrar que conoce un **witness**, compuesto por el valor de todas las variables intermedias al ejecutar un programa.

Para $a = 4$ y $b = 2$:

- Witness **válido**: $(x=6, y=2, c=12)$
- Witness **inválido**: $(x=1, y=1, c=1)$

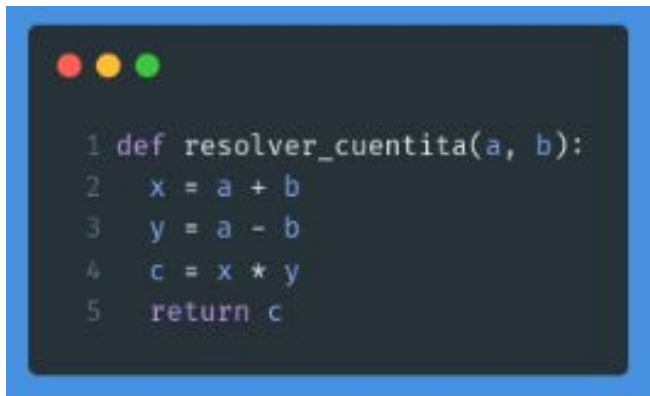


```
1 def resolver_cuentita(a, b):  
2     x = a + b  
3     y = a - b  
4     c = x * y  
5     return c
```

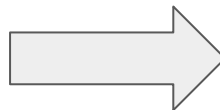
Si conoce todas las variables intermedias, nos convencemos de que ejecutó el programa.

Provable programs

Los witness válidos se van a distinguir por medio de **sistemas de ecuaciones**.



```
1 def resolver_cuentita(a, b):  
2     x = a + b  
3     y = a - b  
4     c = x * y  
5     return c
```



$$\begin{cases} x = a + b \\ y = a - b \\ c = xy \end{cases}$$

Va a haber un truquito **probabilístico** para mostrar que conocemos una solución sin darla explícitamente.

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.
- Parte **pública**: inputs (a, b), constantes, outputs (c).

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.
- Parte **pública**: inputs (a, b), constantes, outputs (c).
- Parte **secreta**: variables intermedias (x, y), inputs secretos.

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.
- Parte **pública**: inputs (a, b), constantes, outputs (c).
- Parte **secreta**: variables intermedias (x, y), inputs secretos.

Dispara otros problemas:

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.
- Parte **pública**: inputs (a, b), constantes, outputs (c).
- Parte **secreta**: variables intermedias (x, y), inputs secretos.

Dispara otros problemas:

- Mostrar que conocés **solución** sin revelarla (succinct, ZK).

Resumen

Para probar la ejecución de un programa:

- Vamos a tener **sistemas de restricciones**.
- Parte **pública**: inputs (a, b), constantes, outputs (c).
- Parte **secreta**: variables intermedias (x, y), inputs secretos.

Dispara otros problemas:

- Mostrar que conocés **solución** sin revelarla (succinct, ZK).
- Expresar cómputo como restricciones (if, for)

Pero antes...



Protocolo de Schnorr

Utiliza un generador g del grupo de unidades F_p^* .

- Prover: conoce un secreto s tal que $g^s = S$.
- Verifier: se quiere convencer de que el prover conoce s .

Datos públicos: p, g, S

Protocolo de Schnorr

Prover

Verifier

Samplea r random y
calculo $R = g^r$

R



```
graph LR; Prover[Prover] -- R --> Verifier[Verifier];
```


Protocolo de Schnorr

Prover

Samplea r random y
calcula $R = g^r$

R



```
graph LR; Prover -- R --> Verifier; Verifier -- b --> Prover;
```

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Protocolo de Schnorr

Prover

Samplea r random y
calculo $R = g^r$

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Si b es 0 , devuelvo r .

R

b

r

Protocolo de Schnorr

Prover

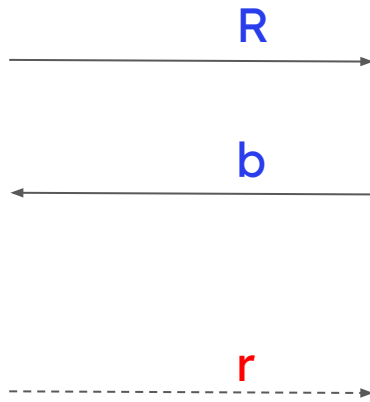
Samplea r random y
calculo $R = g^r$

Si b es 0 , devuelvo r .

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Verifica $R = g^r$



Protocolo de Schnorr

Prover

Samplea r random y
calculo $R = g^r$

R

b

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Si b es 0 , devuelvo r .

Si b es 1 , devuelvo $u = r + s$

r

u

Verifica $R = g^r$

Protocolo de Schnorr

Prover

Samplea r random y
calculo $R = g^r$

R

b

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Si b es 0, devuelvo r .

Si b es 1, devuelvo $u = r + s$

r

u

Verifica $R = g^r$

Verifica $g^u = RS$

Protocolo de Schnorr

Prover

Samplea r random y
calculo $R = g^r$

Verifier

Desafío: sampleo un bit
random $b = \{0, 1\}$

Si b es 0, devuelvo r .

Si b es 1, devuelvo $u = r + s$

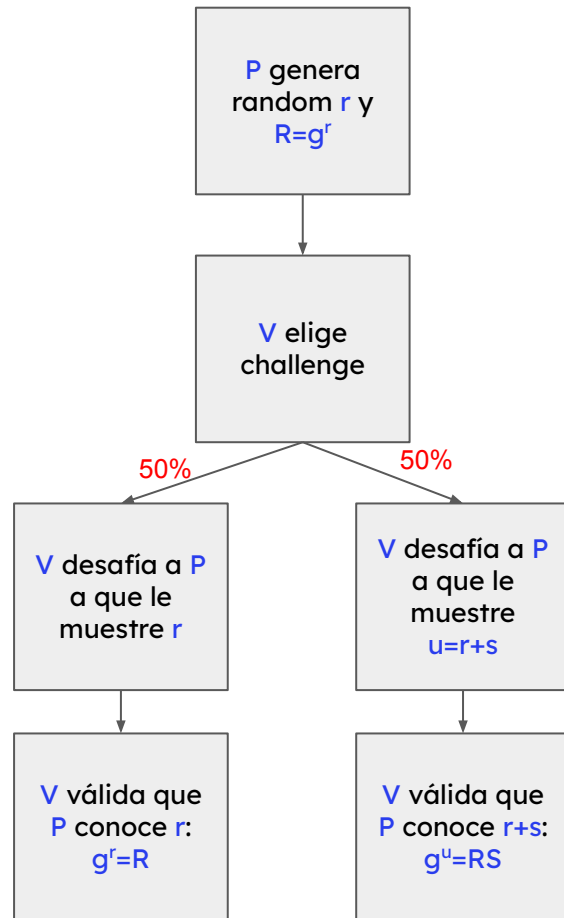
Verifica $R = g^r$

Verifica $g^u = RS$

En esta versión, hay que repetir muchas veces!

Protocolo de Schnorr

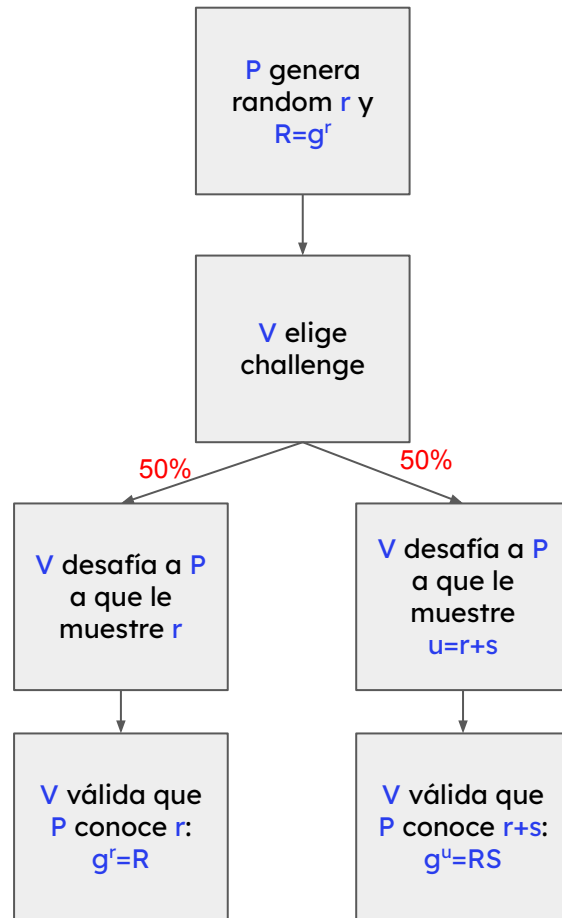
Intuición de que es correcto:



Protocolo de Schnorr

Intuición de que es correcto:

- No revela s :
 - En un caso devuelve r , y en el otro $s+r$, pero nunca ambos.

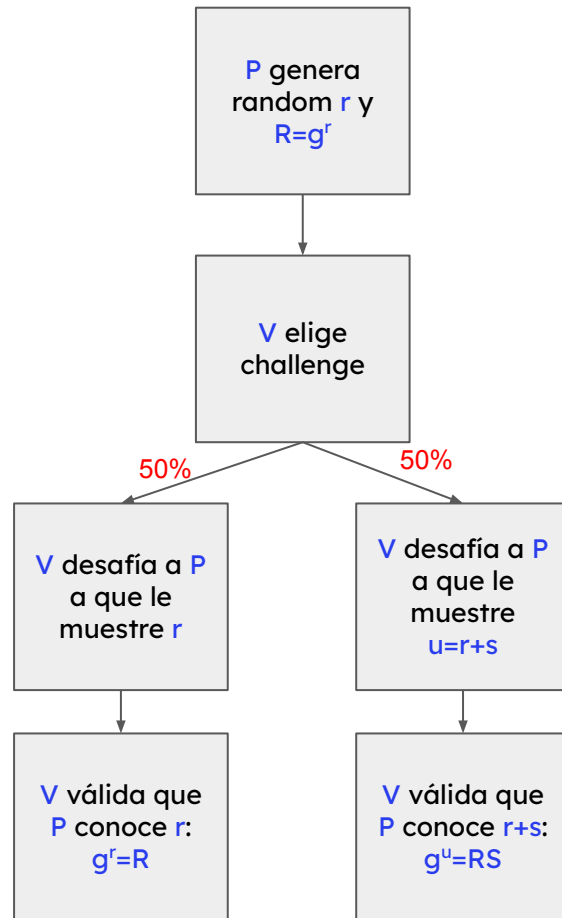


Protocolo de Schnorr

Intuición de que es correcto:

- No revela s :
 - En un caso devuelve r , y en el otro $s+r$, pero nunca ambos.
- Convince al verifier:
 - $b=0$ prueba que conoce r .
 - $b=1$ prueba que conoce $s+r$.
 - Entonces conoce s .

¿Cómo evitamos repetir varias veces?



Versión optimizada

Prover

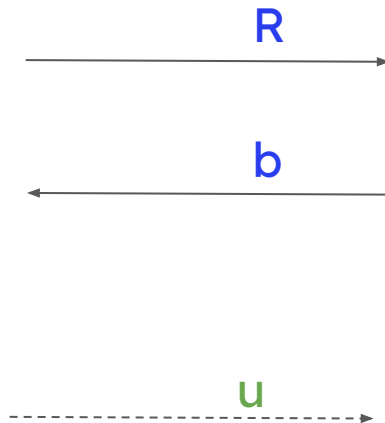
Samplea r random y
calcula $R=g^r$

Calcula $u=r+s*b$

Verifier

Desafío: sampleo un b en
 $[0, p-1)$

Verifica $g^u = RS^b$



Prueba no interactiva

En la práctica los protocolos no son interactivos. Se usa una heurística llamada **Fiat-Shamir**.

Prueba no interactiva

En la práctica los protocolos no son interactivos. Se usa una heurística llamada **Fiat-Shamir**.

- En lugar de interactuar almacenan datos en un objeto llamado **transcript**.

Prueba no interactiva

En la práctica los protocolos no son interactivos. Se usa una heurística llamada **Fiat-Shamir**.

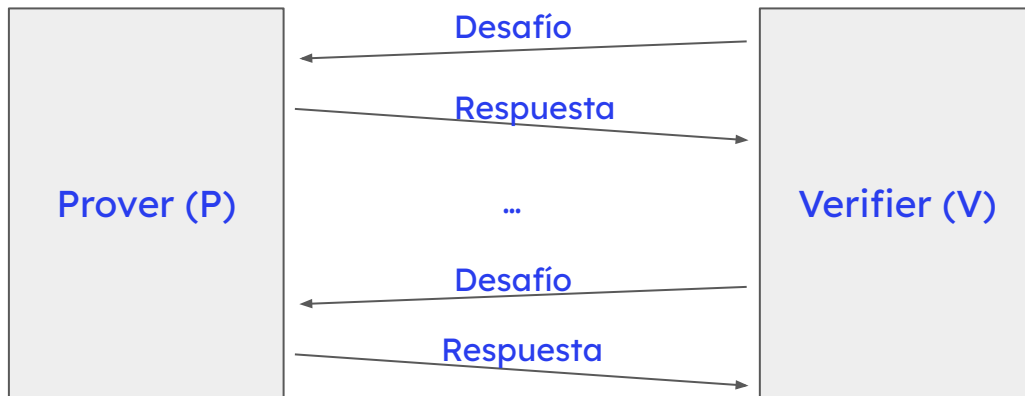
- En lugar de interactuar almacenan datos en un objeto llamado **transcript**.
- El **transcript** tiene un estado interno y se puede hacer digest, como una función de hash.

Prueba no interactiva

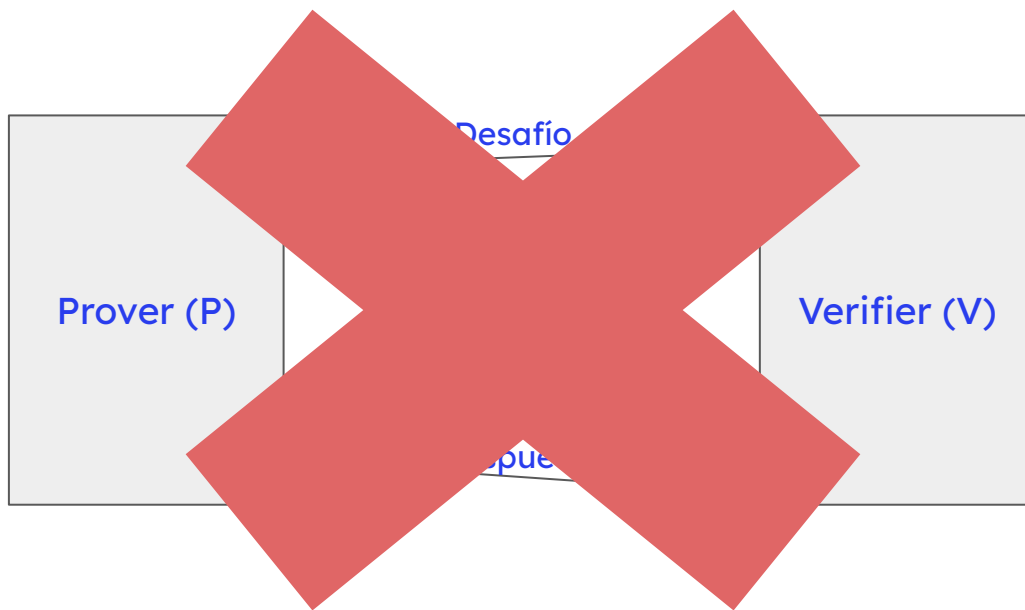
En la práctica los protocolos no son interactivos. Se usa una heurística llamada **Fiat-Shamir**.

- En lugar de interactuar almacenan datos en un objeto llamado **transcript**.
- El **transcript** tiene un estado interno y se puede hacer digest, como una función de hash.
- Cada vez que el Prover necesita un challenge random, se lo pide al **transcript**.

Fiat-Shamir



Fiat-Shamir



Fiat-Shamir

```
1 def prove( ... ):
2     t = Transcript()
3
4     # En lugar de enviar cosas al verifier
5     # hago append al transcript.
6     t.append( ... )
7
8     # En lugar de recibir un desafío del
9     # verifier, lo sampleo del transcript.
10    b = t.sample()
11
12    return c
```

Fiat-Shamir

```
1 def prove( ... ):
2     t = Transcript()
3
4     # En lugar de enviar cosas al verifier
5     # hago append al transcript.
6     t.append( ... )
7
8     # En lugar de recibir un desafío del
9     # verifier, lo sampleo del transcript.
10    b = t.sample()
11
12    return c
```

El verifier tiene todos los elementos para reconstruir el transcript.

Seguridad

Hay que tener varias consideraciones.

- Weak Fiat-Shamir.
- Schnorr en su primera versión, no sería seguro!

Codear!