



I. Criptografía avanzada

Alumnos Ciberseguridad

1. XOR
2. RSA
3. AES
4. Introducción a Blockchain

XOR

XOR

- Cifrado que opera XOR en binario
- Utiliza una **clave secreta**.
- Como la **longitud** de la **clave** suele ser **menor al texto**, se repetirá **cíclicamente**.

Propiedades

1. Conmutativa: $A \text{ xor } B = B \text{ xor } A$



2. Asociativa: $(A \text{ xor } B) \text{ xor } C = A \text{ xor } (B \text{ xor } C)$

3. Autoinversa: $(A \text{ xor } B) \text{ xor } B = A$

<i>A</i>	<i>B</i>	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR

Reto 7 Atenea

XOR (4pts)  

Dificultad: ★☆☆☆☆

En criptografía, el cifrado XOR es, como su nombre indica, un algoritmo de cifrado basado en el operador binario XOR:

$A \text{ xor } 0 = A$
 $A \text{ xor } A = 0$
 $(B \text{ xor } A) \text{ xor } A = B$

Una cadena de texto puede ser cifrada aplicando el operador de bit XOR sobre cada uno de los caracteres utilizando una clave. Para descifrar la salida, sólo hay que volver a aplicar el operador XOR con la misma clave.

Usa la clave **encryptXOR** para descifrar el siguiente mensaje:

UGFzc3dvcmQ6IHhvFzYMACeBiAgIA==

Recuerda que la respuesta hay que ponerla en el formato correcto: flag{md5}

Referencias:
https://es.wikipedia.org/wiki/Cifrado_XOR
<http://xor.pw>
<https://conv.darkbyte.ru>



<http://xor.pw/>
<https://gchq.github.io/CyberChef/>



XOR

EJEMPLO I

¡Ayúdanos a descifrar este texto! Conocemos la correspondencia de algunas cadenas:

"cifrado muy utilizado" = 0c 2d 03 3e 00 31 3d 6a 2e 36 2d 66 16 1b 04 1c 0c 0e 08 10 06

"propiedades importantes" = 3c 13 3a 22 23 26 27 35 22 06 1c 4d 19 08 04 06 06 1d 17 01 30 00 3f

06 38 66 3b 20 3f 50 00 07 49 01 07 56 0c 2d 03 3e 00 31 3d 6a 2e 36 2d 66 16 1b 04 1c 0c
0e 08 10 06 56 0a 2a 45 20 00 75 31 38 2a 33 20 29 04 1d 0c 16 0c 15 63 20 00 13 01 21 45 3c
13 3a 22 23 26 27 35 22 06 1c 4d 19 08 04 06 06 1d 17 01 30 00 3f 6b 16 27 2b 2d 27 3b 66 17
06 08 1e 00 07 49 01 07 56 0c 2d 03 3e 00 31 3d 6a 1b 0c 06 66 1a 4f 0e 1f 0b 1b 0a 11 1a 56 1f
25 17 38 04 75 36 2f 2f 63 20 23 1b 1b 02 50 00 1a 49 17 05 17 1d 2b 45 3c 0e 31 20 ab 30 63
35 36 0f 06 0e 11 17 54 05 15 1a 56 1f 36 0a 3c 08 30 36 2b 27 26 27 66 07 0a 01 50 3d 3b 3b
54 19 17 1d 25 45 3f 00 36 33 38 63 2f 35 66 00 03 0c 06 00 58 49 54 0d 13 1c 27 0c 2a 13 34
20 26 2c 63 2d 66 00 00 03 03 00 13 1c 1d 1b 56 03 25 45 2a 0d 34 35 40 16 11 1e 05 18 37 22
22 45 11 1a 54 0f 17 0c 2d 09 31

XOR

DADO UN PAR TEXTO EN CLARO/TEXTO CIFRADO

Aplicando la propiedad autoinversa del XOR podremos descifrar la clave

$$(A \text{ xor } B) \text{ xor } B = A$$

Texto en claro: A

Texto cifrado: (A xor K)



Clave: Texto xor Cifrado

Clave: A xor (A xor K) = K

XOR

EJEMPLO II

Esta vez nos dan directamente la flag, pero parece que está cifrada:

Flag: 13 3e 2b 24 3d 3d 14 02 66 1f 04 47 34 09 11 0e 32 0d 41 0b 27 4c 02 0b 27 1a 04 47 28
03 41 02 35 4c 0c 12 3f 4c 12 02 21 19 13 08 3b

Formato de la flag: URJC{}

XOR

¿Cómo lo resolvemos?

Conocemos el formato de la flag...

CIFRADO	FLAG
13 3e	U
2b	R
24	J
3d	C
3d	{
14 02	?
66	?
...	?
	?
	...

$$(A \text{ xor } B) \text{ xor } B = A$$

Tenemos:

Cifrado = Texto[i] XOR Clave[i]

Flag = Texto[i]

Entonces:

Cifrado[i] XOR Texto[i] = Clave[i]

XOR

¿Cómo lo resolvemos?

CIFRADO	FLAG	CLAVE
13 3e	U	k[0]
2b	R	k[1]
24	J	k[2]
3d	C	k[3]
3d XOR	{	k4]
14 02	?	?
66	?	?
...	?	?
	?	?

(A xor B) xor B = A

Tenemos:

Cifrado = Texto[i] XOR Clave[i]

Flag = Texto[i]

Entonces:

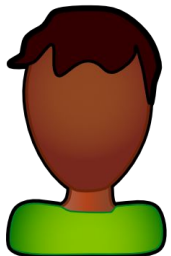
Cifrado[i] XOR Texto[i] = Clave[i]

XOR

EJEMPLO III: Alice y Bob quieren intercambiar mensajes

¡Te mando la contraseña secreta cifrada con mi clave!

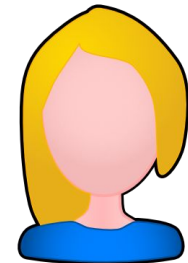
16 3e 2b 35 1e 06 11 0a 1e 0c 0e 00 43 63 08 0e 05 45 25 00 00 17 16 54 0d 50 63 0f 0d 17 13 36 45
0b 13 06 11 41 40 36 09 41 05 00 73 06 02 1c 06 11 0d 54 3e



Bob

Te la mando de vuelta cifrada con mi clave

5a 5f 78 50 79 73 7f 6e 7f 47 6b 79 0f 02 5b 6b 62 30 4b 64 61 5c 73 2d 41 31 30 6a 6a 62 7d 52 24 40
76 7f 5d 20 13 53 6e 34 6b 64 12 4d 67 65 4a 70 5e 31 59



Alice

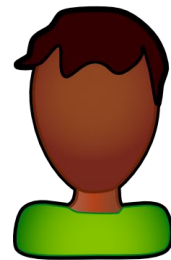
Genial, voy a volver a cifrar otra vez con mi clave

19 33 19 26 1c 20 1a 0d 0d 22 1f 18 3e 41 37 0a 14 55 18 01 02 2e 16 59 20 00 73 06 0b 14 18 01 41
23 04 1a 29 41 22 10 02 55 1d 01 41 28 04 17 2f 04 3f 00 1a



XOR

¿Qué está pasando?



Bob



K1

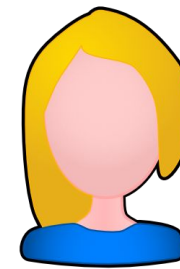
$M1 = \text{Texto} \text{ XOR } K1$



$M2 = M1 \text{ XOR } K2$



$M3 = M2 \text{ XOR } K1$



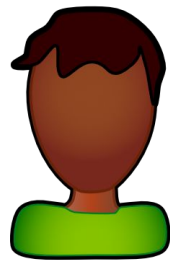
Alice



K2

XOR

¿Qué está pasando?



Bob



K1

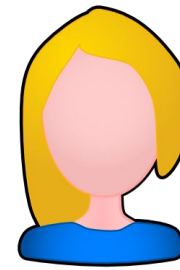
$$M1 = \text{Texto XOR } K1$$



$$M2 = (\text{Texto XOR } K1) \text{ XOR } K2$$



$$M3 = ((\text{Texto XOR } K1) \text{ XOR } K2) \text{ XOR } K1$$



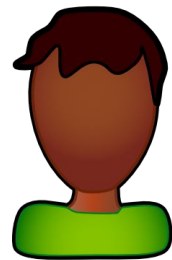
Alice



K2

XOR

¿Qué está pasando?



Bob



K1

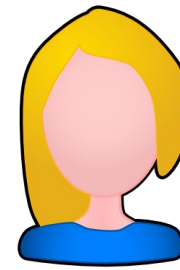
$$M1 = \text{Texto XOR } K1$$



$$M2 = (\text{Texto XOR } K1) \text{ XOR } K2$$



$$M3 = ((\text{Texto XOR } K1) \text{ XOR } K2) \text{ XOR } K1$$



Alice



K2

XOR

¿Cómo descifrarlo?

Si hacemos **M2 XOR M3** conseguiremos K1, y con la clave ya podremos descifrar el primer mensaje (M1)



Bob

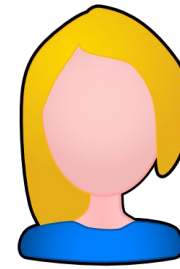


K1

$$M1 = \text{Texto} \text{ XOR } K1$$

$$M2 = (\text{Texto} \text{ XOR } K1) \text{ XOR } K2$$

$$M3 = \text{Texto} \text{ XOR } K2$$



Alice



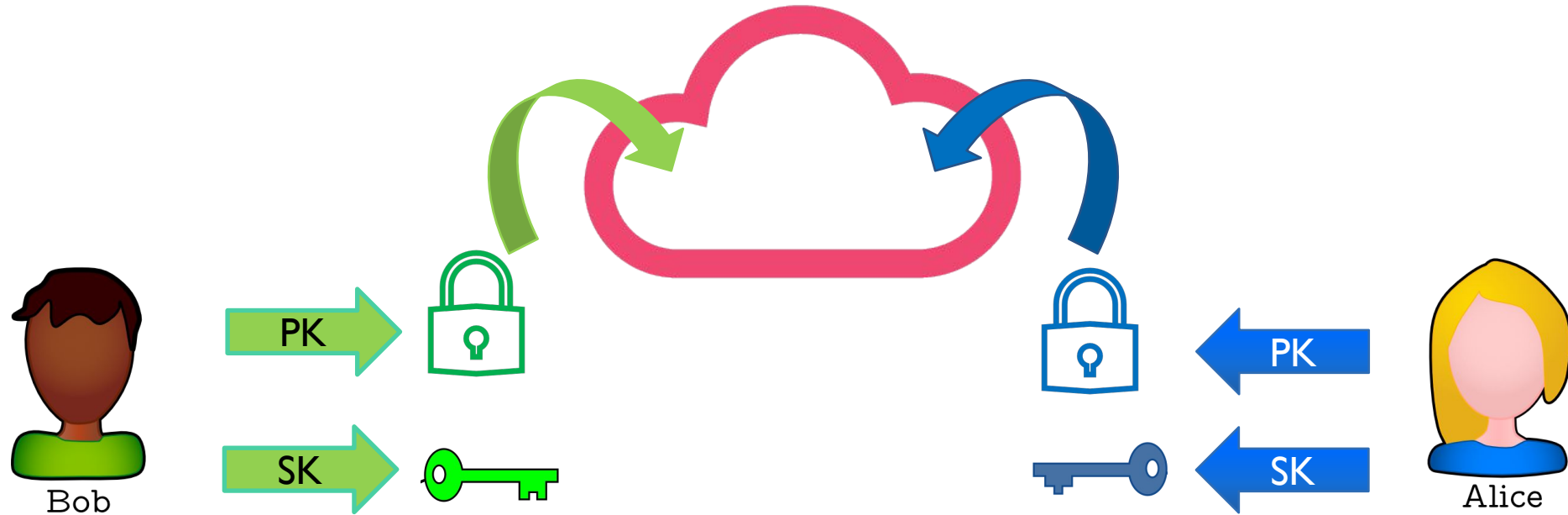
K2

CRIPTOGRAFÍA ASIMÉTRICA

¿Qué es la criptografía asimétrica?

Es un tipo de cifrado que utiliza una clave pública para cifrar y otra privada para descifrar.

RSA o el Gamal

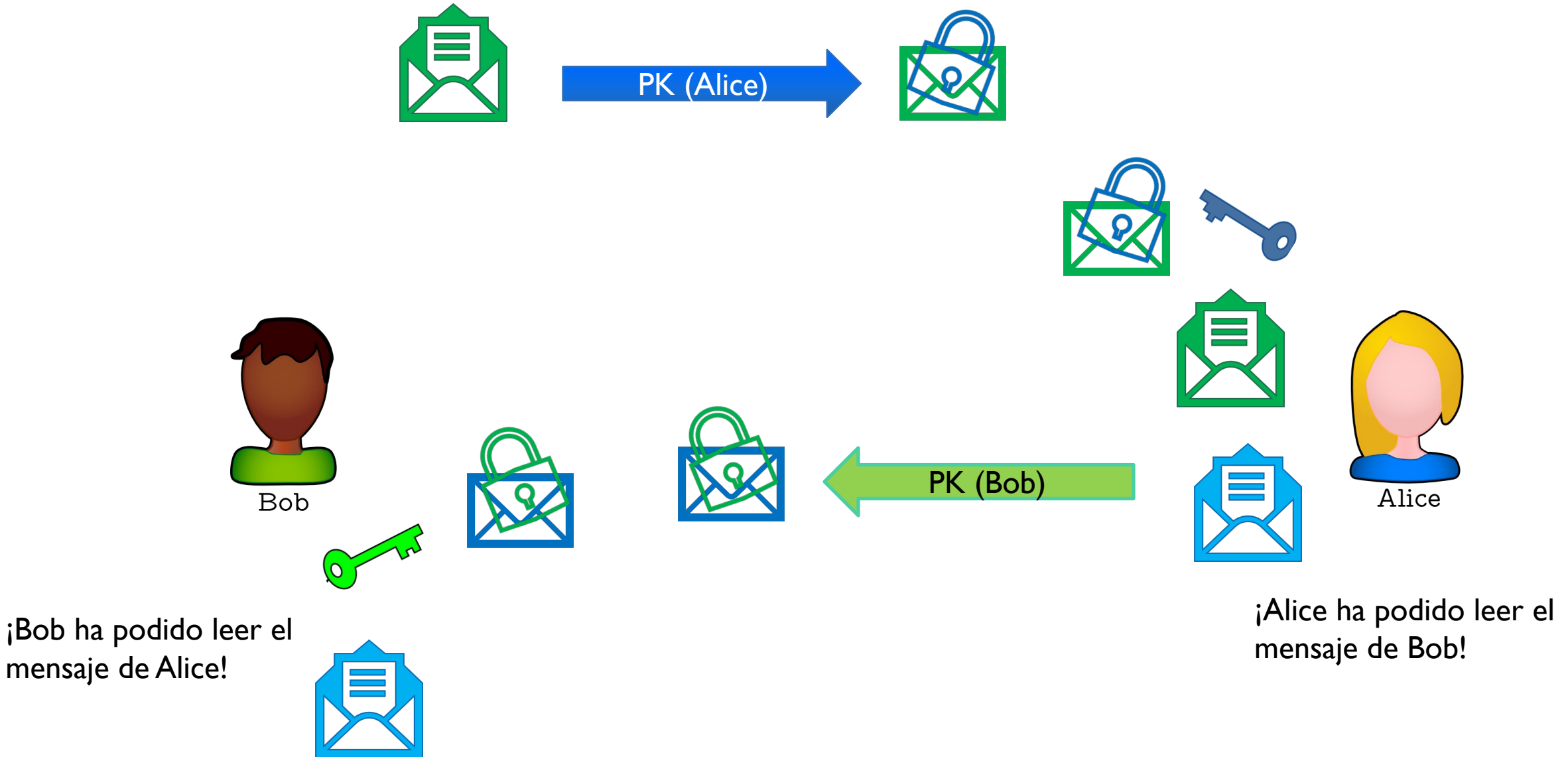


CRIPTOGRAFÍA ASIMÉTRICA

**Bob quiere mandar un mensaje seguro a Alice
pero no han acordado ninguna clave secreta
previamente**

¿Cómo lo hacen?

CRIPTOGRAFÍA ASIMÉTRICA



RSA - (Rivest – Shamir – Adleman)

Dos números primos

p y q

$$3 * 11 = 33 = N$$

e

PK

I

3 y 11 (p y q)

d (obtenido con e)

SK

IV

Texto a números
(binario)

Cuentas con
binario y 33

II

Mensaje cifrado

III

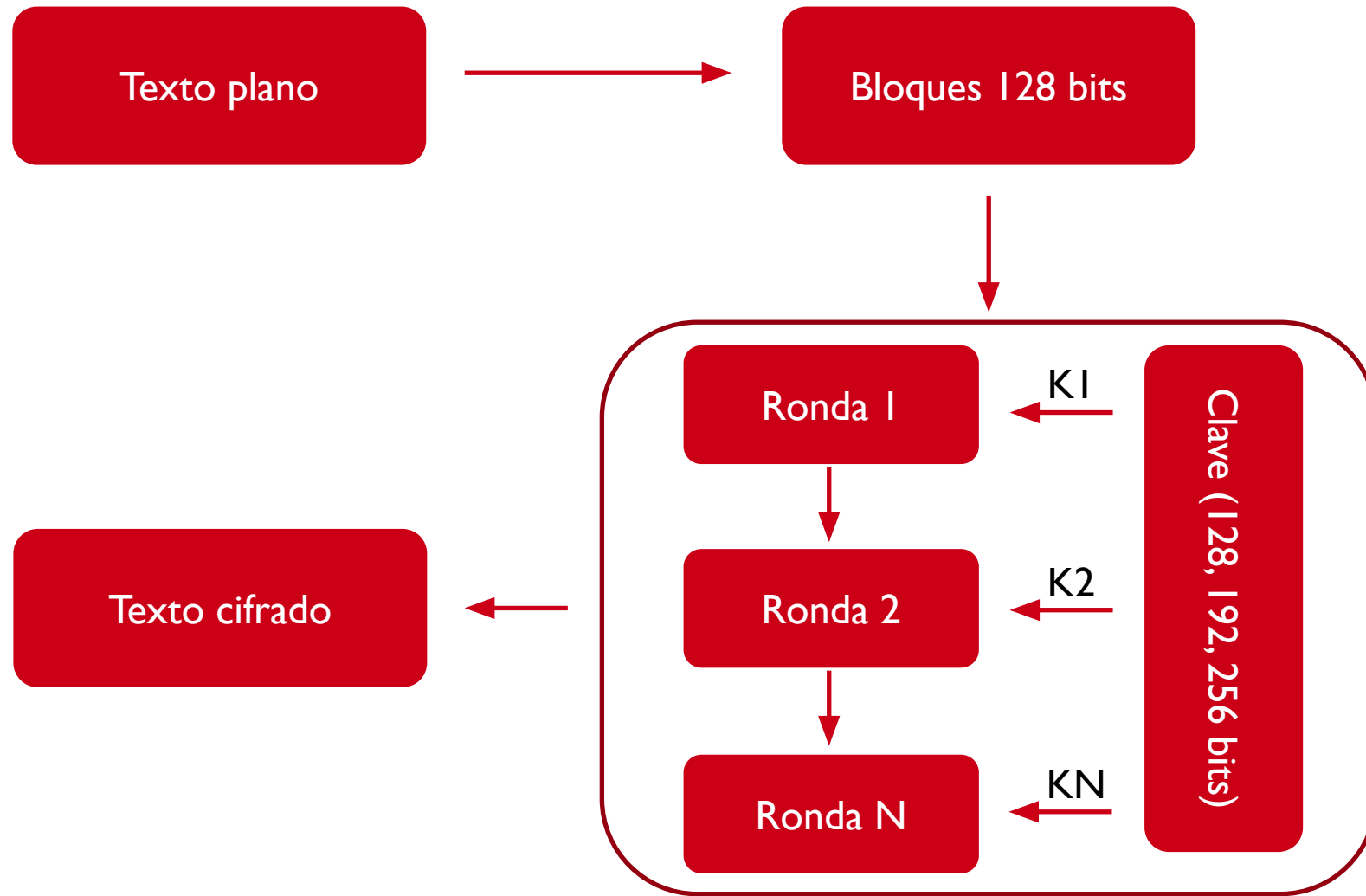
RSACtfTool

Instalación en Kali Linux (OVA de VirtualBox):

```
git clone https://github.com/Ganapati/RsaCtfTool.git  
sudo apt-get install libgmp3-dev libmpc-dev  
cd RsaCtfTool  
pip3 install -r "requirements.txt"  
python3 RsaCtfTool.py
```

[RsaCtfTool: https://github.com/Ganapati/RsaCtfTool](https://github.com/Ganapati/RsaCtfTool)

AES – (Advanced Encryption Standard)



AES – (Advanced Encryption Standard)

- En cada ronda, se calcula una **nueva clave** a partir de la **original**
- El **número de rondas** depende de la **longitud de la clave**

LONGITUD DE LA CLAVE (bits)	RONDAS
128	10
192	12
256	14

- Lo que ocurre en cada una de esas rondas, queda a vuestra curiosidad

INTRODUCCIÓN A BLOCKCHAIN



¿QUÉ ES UNA BLOCKCHAIN?

- Una blockchain es un “libro” mayor **compartido** donde se registran transacciones de forma **inmutable**.

- Las transacciones se unen para formar una **cadena irreversible**:

Cada bloque adicional refuerza la verificación del bloque anterior haciéndolo a prueba de manipulaciones

- El primer ejemplo de blockchain nació en **1991** de mano de Stuart Haber y W. Scott Stornetta.

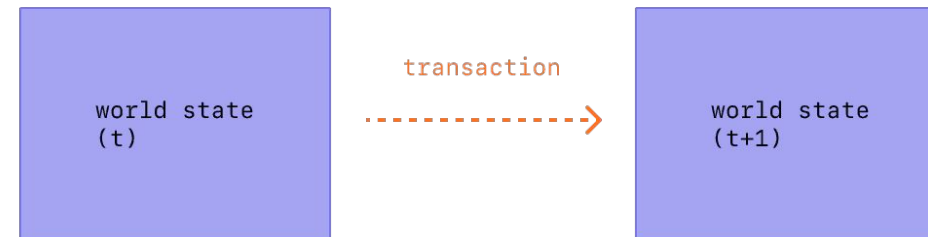
- El primer gran impulso para la blockchain vino en **2008** de mano de *Satoshi Nakamoto* con la aplicación de esta tecnología para crear una red distribuida de pago



ETHEREUM

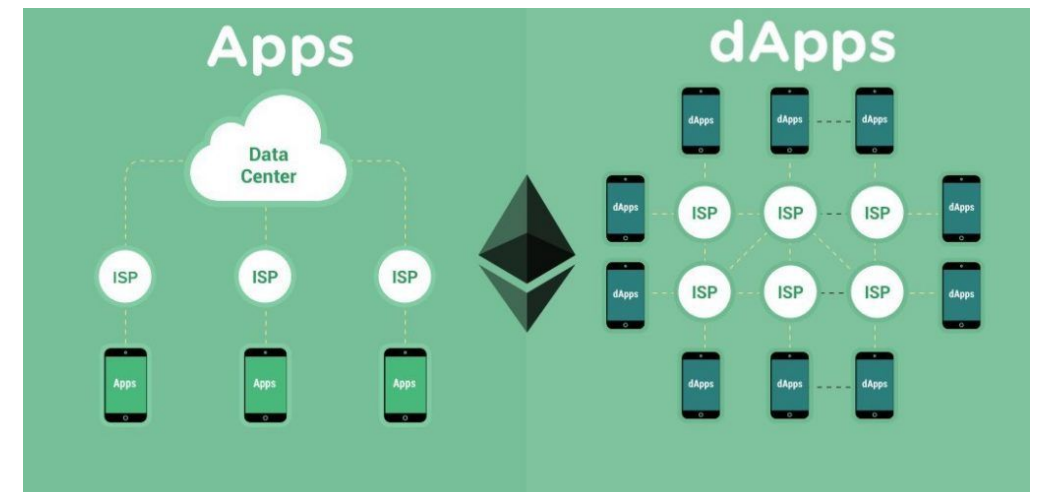
- **Ethereum** no nace únicamente como sistema de pago, sino como un “ordenador” gigante descentralizado capaz de procesar **100.000** transacciones por segundo.

Por transacción entendemos cualquier cambio de balance en dos carteras, ya sea Ether o cualquier otro activo digital

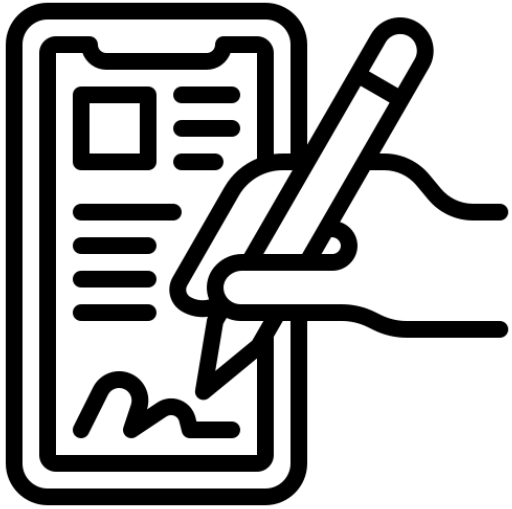


- Ethereum introduce los **smart contracts** que son piezas de código (*funcionalidad + datos*) que residen en la cadena de Ethereum.

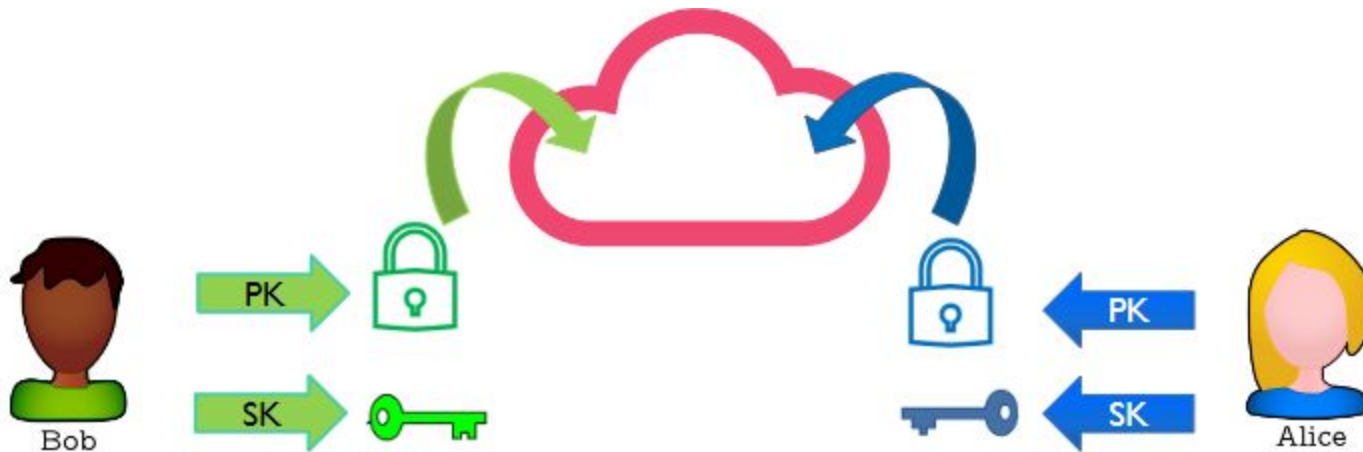
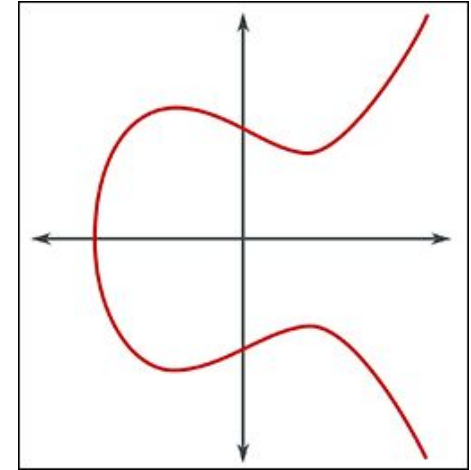
Estos permitirán el desarrollo de Dapps



ETHEREUM



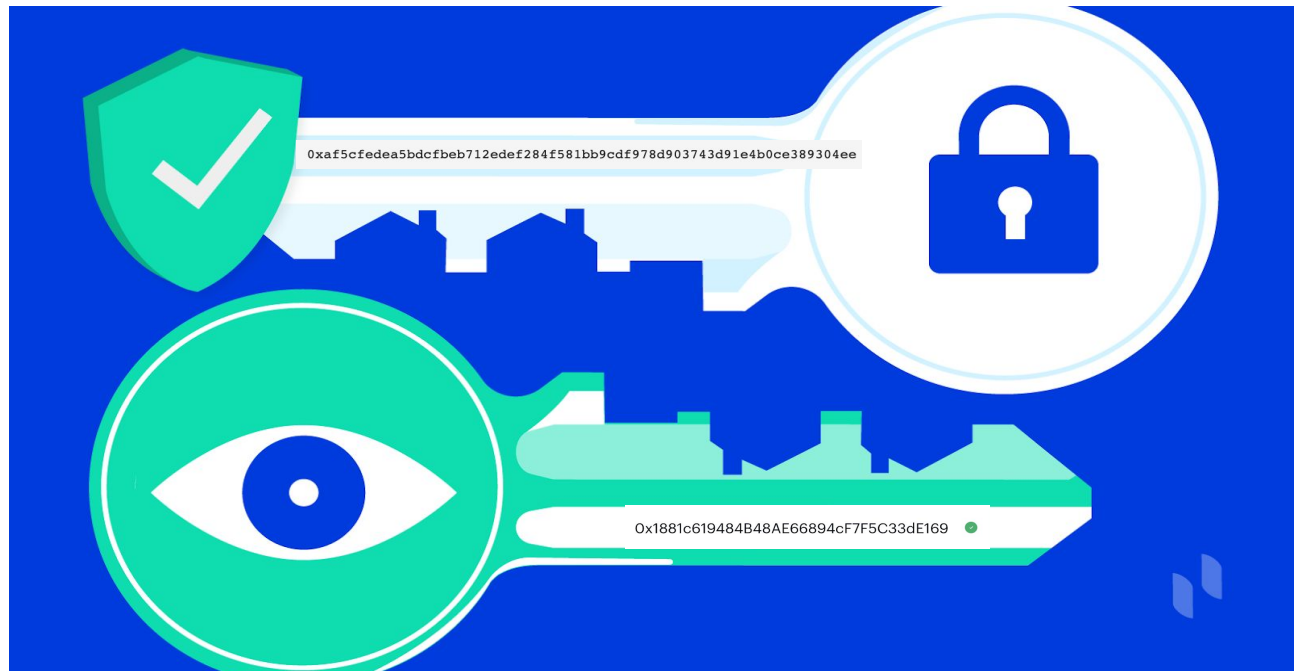
Cifrado curva elíptica
Criptografía clave pública
Firmas Digitales
Hashes



#

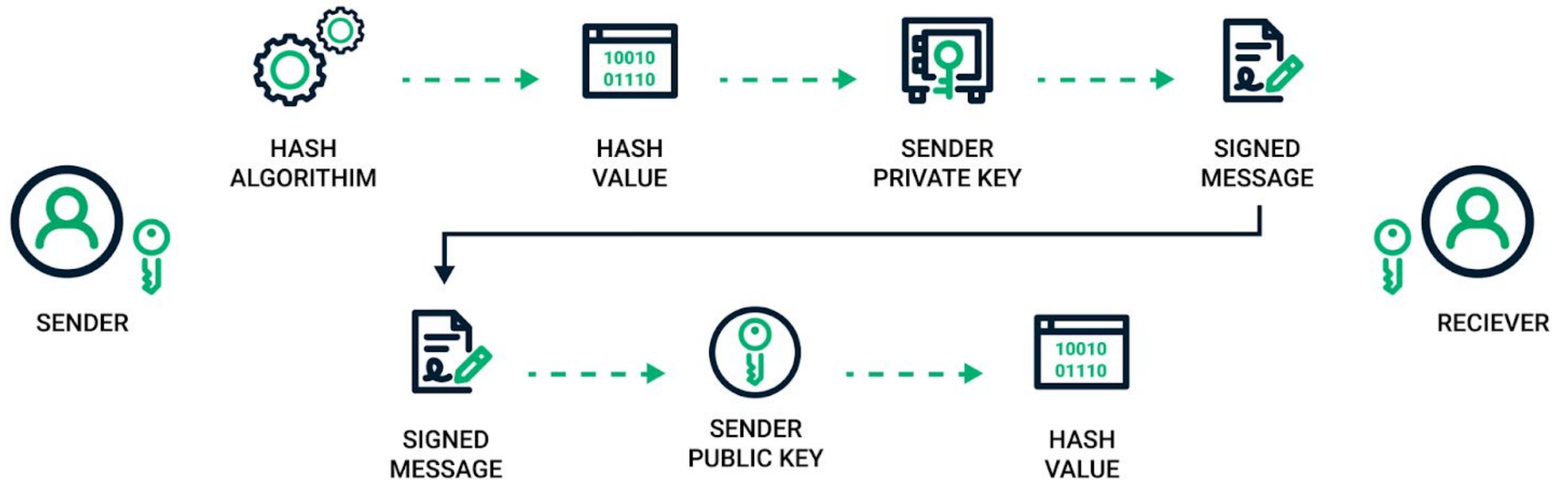
CLAVE PÚBLICA & ETHEREUM

- Utiliza un esquema de clave pública de forma que:
 - Se genera un **par** de claves **público-privada** en el que la clave pública **deriva** de la privada
 - Se utiliza la **clave pública** para **cifrar transacciones**, y se verifica la autoría de esta clave pública mediante **firmas digitales**.



FIRMAS DIGITALES

¿Cómo funciona una firma digital?



CLAVE PÚBLICA Y PRIVADA

- En Ethereum, nuestra clave privada se genera de forma **aleatoria**. Muchas veces nos pedirá un input del usuario para aumentar la aleatorización.

El espacio de generación de semillas para claves privadas es de $2^{256} \sim 10^{77}$

- Mediante operaciones matemáticas sobre **curva elíptica** se deriva la clave privada para obtener la pública.

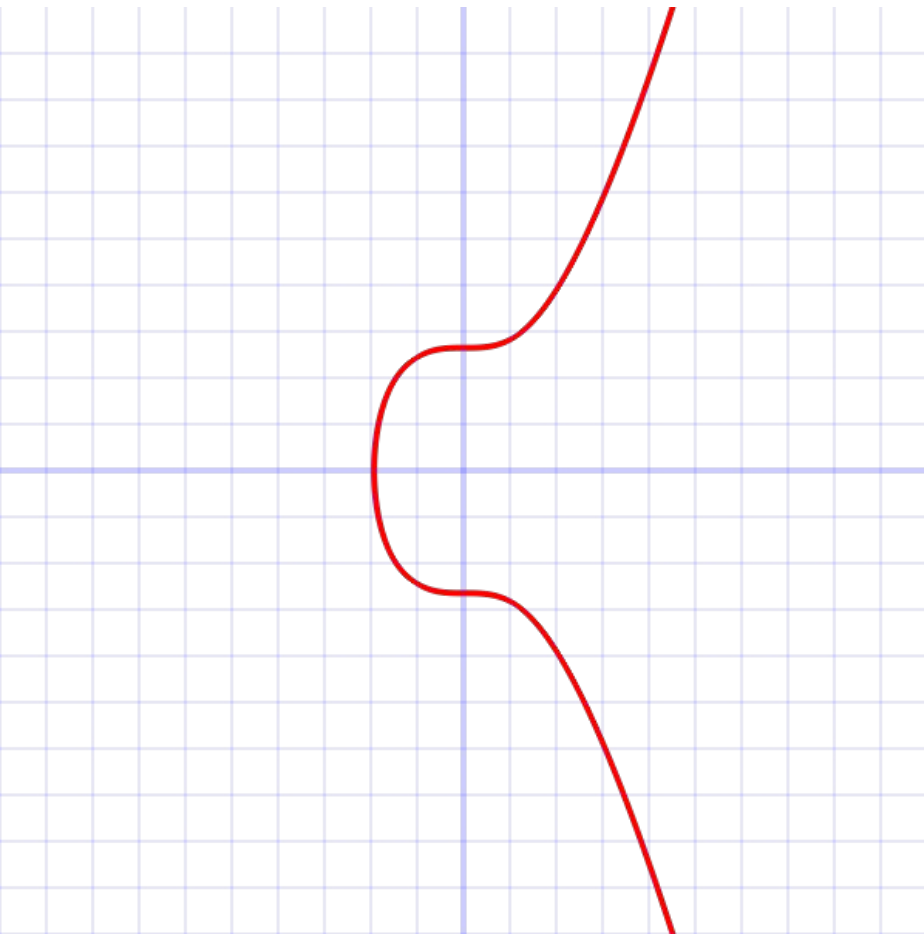


Bienvenido a MetaMask

CURVA ELÍPTICA Y HASH

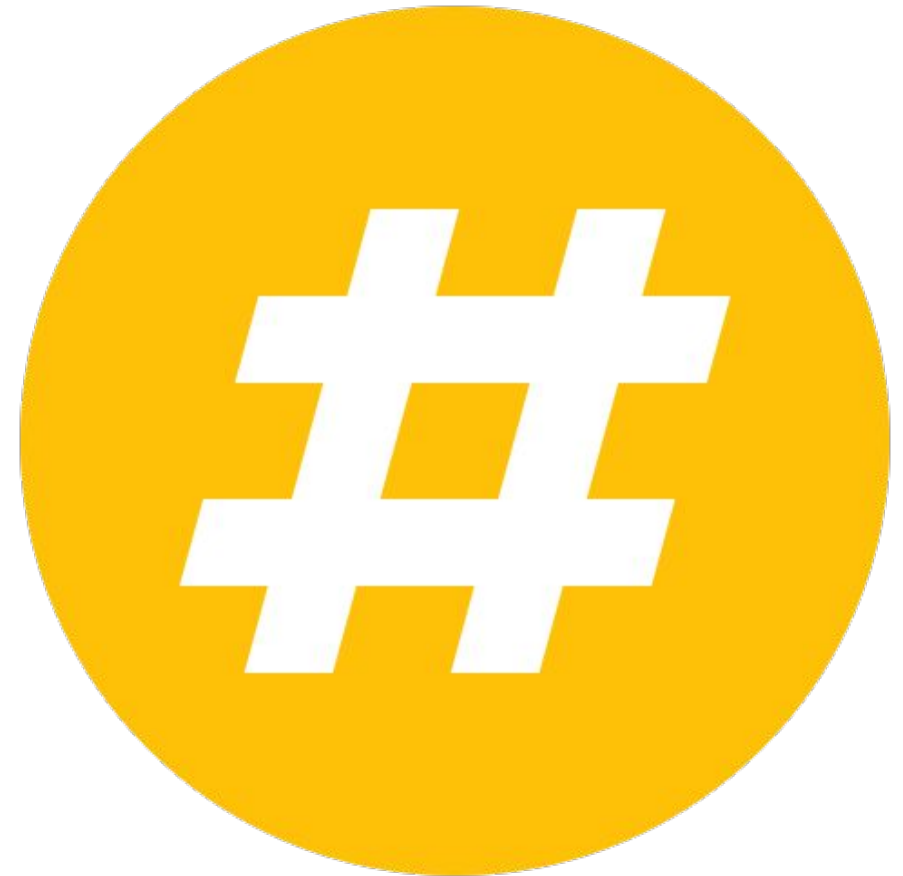
CURVA ELÍPTICA

SECP256k1



HASH

KECCAK-256



CÁLCULO DE DIRECCIONES

Cuando creamos una cartera en Ethereum, se nos asigna una clave privada, como por ejemplo k :

$k = \text{f8f8a2f43c8376ccb0871305060d7b27b0554d2cc72bccf41b2705608452f315}$

Y, mediante cálculos, se autogenera una clave pública Q derivada de la clave privada:

$Q = \text{6e145ccefd1033dea239875dd00dfb4fee6e3348b84985c92fd103444683bae07b83b5c38e5e...}$

El siguiente paso será calcular el hash de la clave pública:

$\text{Keccak256}(Q) = \text{2a5bc342ed616b5ba5732269001d3f1ef827552ae1114027bd3ecf1f086ba0f9}$

Nuestra dirección de cartera se corresponde con los últimos 40 bytes del hash:

$\text{2a5bc342ed616b5ba5732269001d3f1ef827552ae1114027bd3ecf1f086ba0f9}$

Mediante el prefijo 0x denotaremos que su codificación es hexadecimal:

$\text{0x001d3f1ef827552ae1114027bd3ecf1f086ba0f9}$

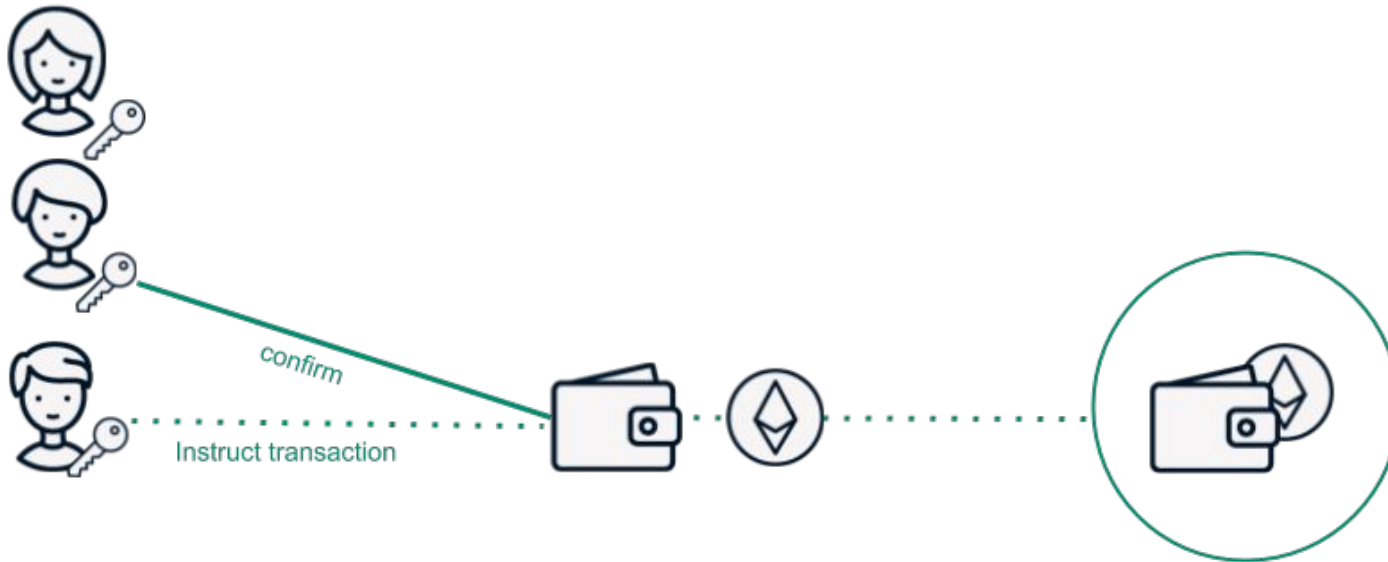
CARTERAS



EOAs (Externally Owned Accounts)

Multi-Signature Accounts

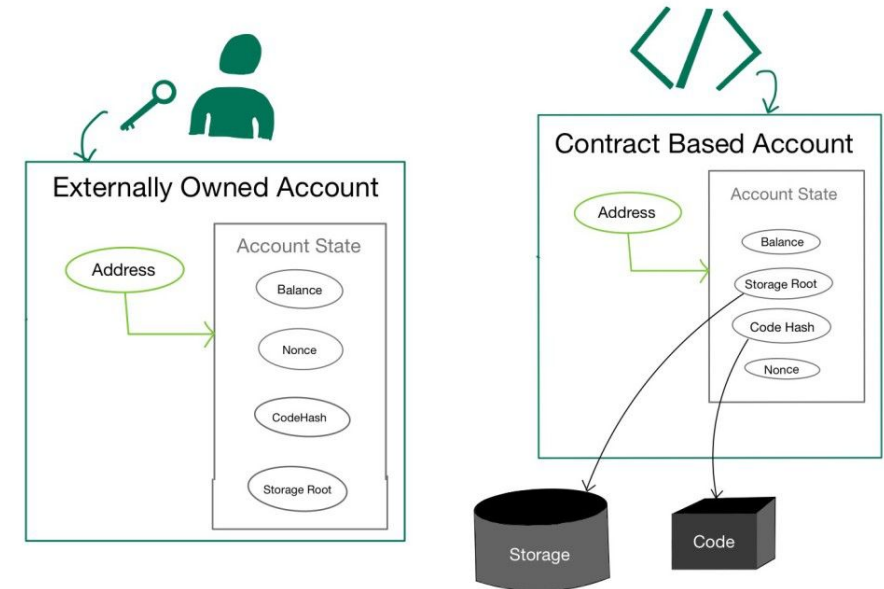
Contract Accounts



1 2 out 3 confirmations
by wallet's owners are
needed for a transaction.

2 Transaction is pending
as the other owners need to
confirm

3 Approved transaction
will be executed

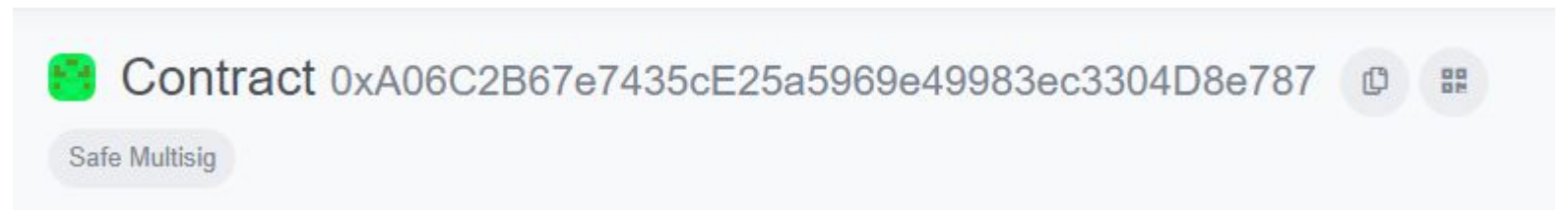
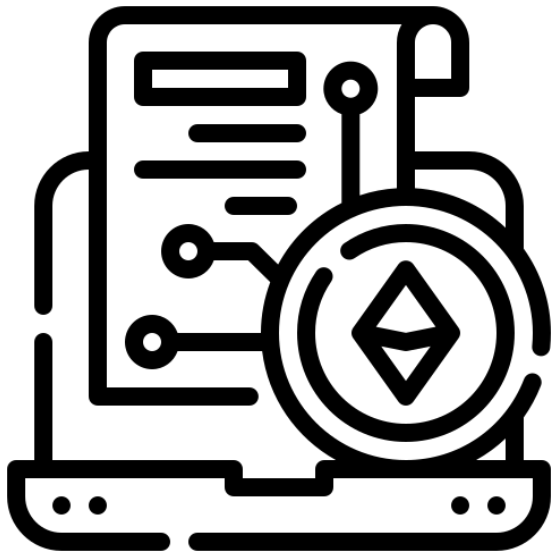


SMART CONTRACTS

Programa que se ejecuta en la blockchain de Ethereum y contiene 2 cosas:

- Funciones, que es el código funcional.
- Datos, que almacenan el estado del Smart Contract.

Cada Smart Contract se almacena en una dirección específica de la blockchain de Ethereum.



CONTENIDOS DE SMART CONTRACTS I

Los datos se pueden guardar en 2 sitios:

- Storage (Almacenamiento)
- Memory (Memoria)

Storage

Es más costoso guardar, ya que se quedan almacenados permanentemente en la blockchain. Por eso se tiene que declarar una variable que almacena la cantidad de datos que tiene el Smart Contract.

```
contract SimpleStorage {  
    uint datosAlmacenados; // variable de estado  
    // ...  
}
```

CONTENIDOS DE SMART CONTRACTS II

Memory

Valores que son almacenados durante el tiempo de ejecución que se llaman variables de memoria.

Environment Variables

Son variables “especiales” que tienen que estar en el contrato, se usan para dar información sobre la blockchain o la transacción actual.

· Algunos ejemplos son:

block.timestamp
msg.sender

uint256
address

Marca de tiempo actual
Remitente del mensaje



SMART CONTRACTS: FUNCIONES

Las funciones permiten al Smart Contract recolectar información de la transacción que las usa, o proporcionarle información a la misma.

Hay 2 tipos de funciones:

- **Internal**

Las funciones del tipo Internal son aquellas que solo se pueden llamar de forma local, es decir, por el propio contrato o contratos que lo contienen.

- **External**

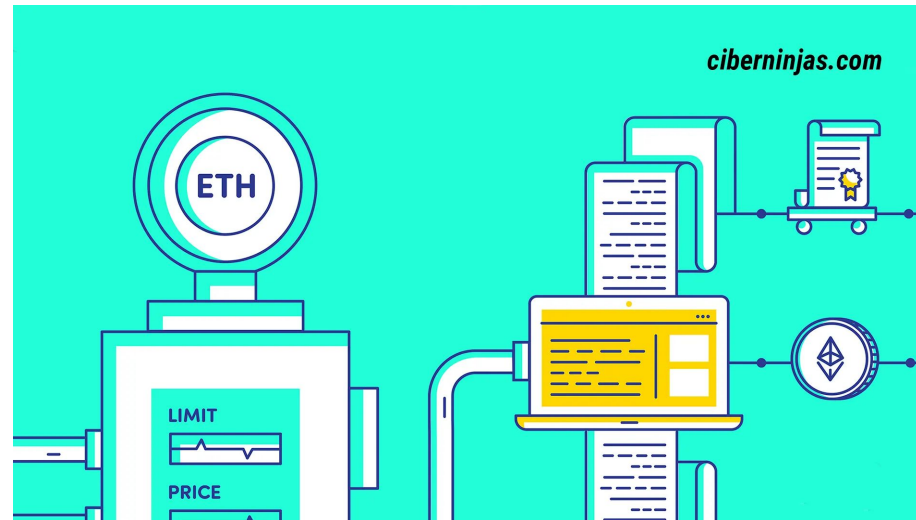
Son parte de la interfaz pública del contrato y pueden ser llamadas desde otros contratos que no lo contienen / transacciones.

Los contratos tienen entre ellos una relación de composición

Como una API pública en un programa

GAS

- El **gas** es la unidad de medida de esfuerzo computacional requerido para ejecutar operaciones en una red de Ethereum.
- Este gas es una “**comisión**” que se cobra en gwei que su equivalente es 10^{-9} ETH.
- Existen para **evitar bucles** infinitos tanto hostiles como accidentales.
- Los mineros se aprovechan de esto para su beneficio. Existe una relación de gas / instrucción. Cuando se realiza una transacción se cobra un precio por el gas utilizado.



PARA SEGUIR APRENDIENDO...

Recursos de consulta y práctica

- CryptoHack. Retos de criptografía:

<https://cryptohack.org/challenges/>

- CrypTool. RSA paso a paso:

<https://www.cryptool.org/en/cto/rsa-step-by-step.html>

BIBLIOGRAFÍA BLOCKCHAIN

<https://github.com/ethereumbook/ethereumbook/blob/develop/04keys-addresses.asciidoc>

<https://ethereum.org/en/developers/docs/transactions/>

<https://ethereum.org/en/developers/docs/dapps/>

<https://ethereum.org/en/developers/docs/smart-contracts/>

<https://legacy.ethgasstation.info/blog/ethereum-block-size/>

https://www.cert.fnmt.es/content/pages_std/html/tutoriales/tuto7.htm

<https://medium.com/mycrypto/the-magic-of-digital-signatures-on-ethereum-98fe184dc9c7>

<https://www.bitstamp.net/learn/security/what-are-private-and-public-keys/>

https://keccak.team/keccak_specs_summary.html

<https://es.bitcoin.it/wiki/Secp256k1>

https://en.wikipedia.org/wiki/Trapdoor_function

https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

https://en.wikipedia.org/wiki/Discrete_logarithm



I. Criptografía avanzada

Alumnos Ciberseguridad