Cryptography in Blockchain

Understanding the foundation of security and trust in blockchain

Fundamentals

Data integrity

Ensuring information remains unchanged

Authentication

Verifying identity and authenticity

Confidentiality

Protecting sensitive information

Cryptographic Primitives in Blockchain

Hash Functions

One-way functions for data integrity

Digital Signatures

Authenticating transactions and verifying origins

Encryption

Protecting sensitive data from unauthorized access

Hash Functions

1 **Deterministic**

Same input always produces same output

2 **Efficiently Computable**

Fast calculation for real-time use

3 **Fixed-size output**

Independent of input size, ensuring consistent representation

Hash Functions

One-way

Easy to compute hash, difficult to reverse

Collision Resistance

Minimizing chance of duplicate hash for different inputs

Avalanche Effect

Small input change drastically alters hash output

One-way Functions

The Concept

A one-way function is a mathematical operation where it's easy to compute the output (hash) from the input, but computationally infeasible to reverse the process and obtain the original input from the hash. This irreversibility is crucial for security in cryptography.

Example

What is the input of this given hash :

8b803c501ffda0bfa07ad979a7d5036d9d716c0a09a730a6f556a8123b87f923

Avalanche Effect

The Concept

A **small input** change will give a **large output** change

Example

Hello World=> a591a6d4[...]9ad9f146e

hello world => b94d27b[...]ce2efcde9

Use cases in Blockchain

1 **Block Linking**

Creating a chain of blocks through hash references

2 **Merkle Tree**

Efficiently verifying data integrity within a block

3 **Transaction ID**

Unique identifier for every transaction

4 **Address**

Derived from the public key hash

Public Key Cryptography

1

Asymmetric Keys

Public and private keys for secure communication

2

Digital Signatures

Using private keys to authenticate transactions

3

Encryption

Protecting sensitive data using public and private keys

Understanding Asymmetric Crypto

Going from Inside to Outside:

1. Sign with YOUR **inside (private)** key
2. World verifies with YOUR **outside (public)** key

Going from Outside to Inside:

1. Encrypt with THEIR **outside (public)** key
2. They decrypt with THEIR **inside (private)** key

Key Pairs

Private Key

Must be kept secret; Used to create signatures; Used to decrypt messages; Source of ownership/authority

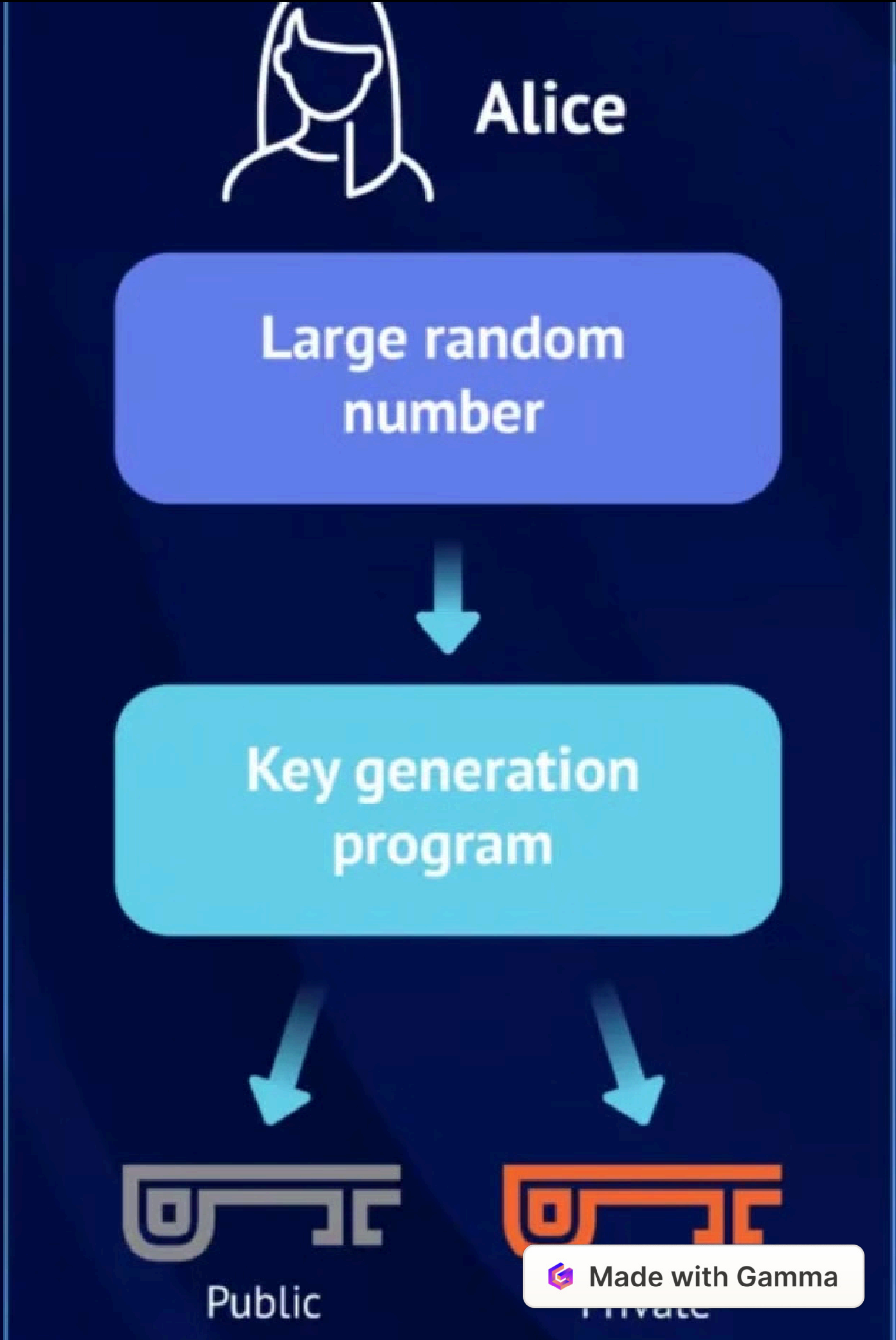
Public Key

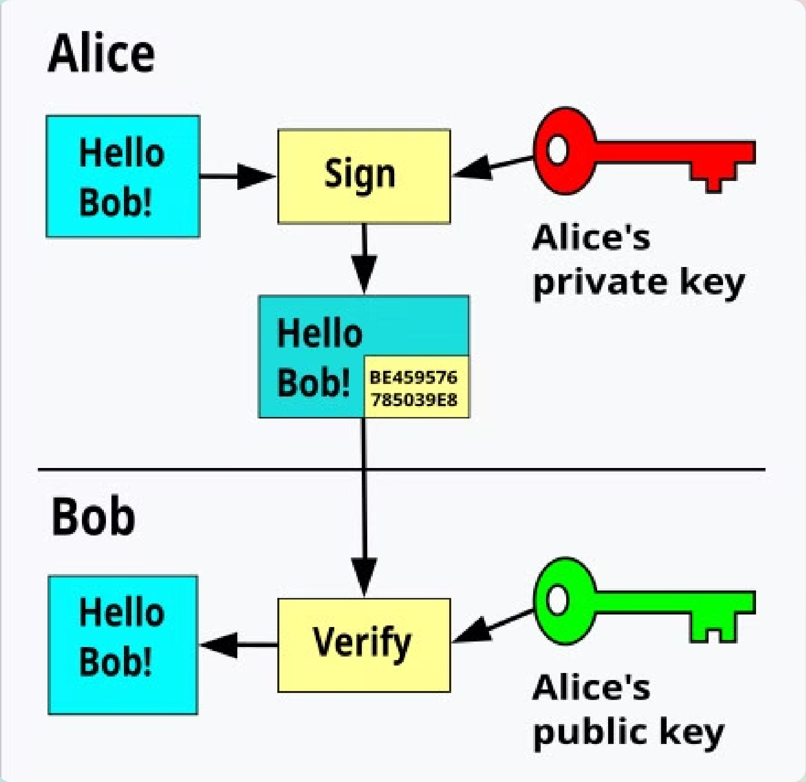
Can be freely shared; Used to verify signatures; Used to encrypt messages

Together, they form a cryptographic key pair used for secure communication, signing and data protection

Key Generation

- Generate a random number
- Use a cryptographic algorithm (like ECC or RSA)
- Calculate the private and public key pair





Digital Signatures

Signing (Private Key)

1. Hash the message
2. Sign the hash with the private key
3. Result: signature + original message

Verification (Public Key)

1. Hash the received message
2. Verify the signature using the public key
3. Confirms: Message integrity and Sender authenticity

Some Approach to Public Key Crypto

RSA

Traditional & widely used

Security relies on difficulty of factoring large numbers

ECC

Based on discrete logarithm problem on **elliptic curves**

Currently used in most **blockchains**

Latice

Post-quantum resistant and an **emerging standard** for future use.

Based on short-vector problem

Why E(lliptic) C(urve) C(ryptography)

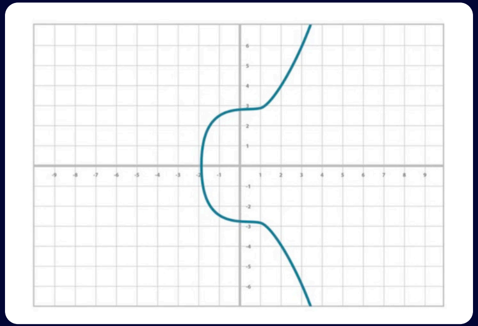
ECC vs. RSA

For similar security, ECC uses much smaller keys than RSA.

Benefits of ECC

- Smaller keys
- Faster computation
- Lower bandwidth
- Better for mobile devices
- Yeah it is plus mieux

Elliptic what ?



$$y^2 = x^3 + a * x + b$$

- **Non-singular:**

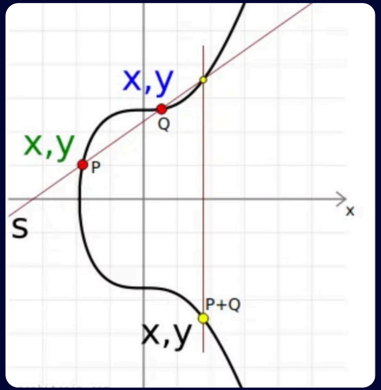
$$4 * a^3 + 27 * b^2 != 0$$

- **Symmetry on the X-axis:** if (x,y) is on the curve then (x,-y) is on the curve
- **Group Law:** Adding 2 points on the curve give us a third point on the curve

Secp256k1 : https://neuromancer.sk/std/secg/secp256k1

Operation on Elliptic Curve

Two core operations are fundamental to elliptic curve cryptography:



Point Addition (P + Q = R): Connect points P and Q with a line. Find the third intersection of this line with the curve, and reflect it across the x-axis to obtain point R.

Scalar Multiplication (nP): Add point P to itself n times. When adding for first time, make the line tangent

These operations are **performed modulo N**, ensuring the results remain within the defined **elliptic curve group**.

[1] https://curves.xargs.org/

ECC: Modular Arithmetic

Modular Arithmetic (mod P): Operations on the curve are performed modulo a large **prime number** (**P**), ensuring points remain within a **finite field**.

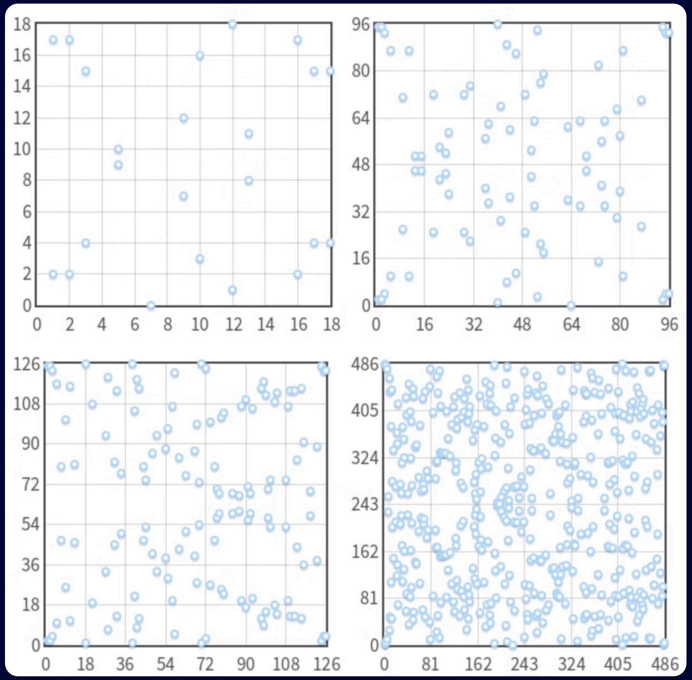
For SECP256k1, **P = 2^256 - 2^32 - 977.**

Operations wrap around like a clock but with P hours instead of 12

[1] https://www.mathsisfun.com/numbers/modulo.html

ECC: Curve Order & Generator Point

Curve Order (n): Total number of points on the curve; a large prime number crucial for security. In SECP256k1, n is slightly smaller than P.
For example: 19, 97, 127, 487



Generator Point (G): Starting point for public key generation. Repeated addition of G (private key * G) produces all other points.

On secp256k1:

G=(
0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798;
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)

https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/

https://neuromancer.sk/std/secg/secp256k1

ECDSA

1

Key Generation

Private key: random number. Public key: result of multiplying the generator point by the private key.

2

Signing Process

Generating a random nonce, computing curve points, and combining with the message hash.

3

Verification

Uses the public key to check the mathematical relationship between the signature and the message.

ECDSA: Key Generation

Key Generation involves two steps:

- Private key: a randomly chosen integer s such that 0 < s < n, where n is the order of the elliptic curve.
- Public key: calculated as sG (where G is the generator point) using modular arithmetic modulo n

$$PrivKey = s$$

$$PubKey = s * Gmod(n)$$

G=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798;
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)

P = 2^256 - 2^32 - 977

ECDSA: Signing Process

Step 1: Generate a nonce. Generate a random number k, where $0 < k < n$. This number, k, is the nonce and must be unique for each signature.

Step 2: Compute the curve point. Compute $(i, j) = k * G \bmod n$, where G is the generator point.

Step 3: Calculate *x*. Compute $x = i \bmod n$. If $x = 0$, repeat Step 2 (generate a new k).

Step 4: Calculate *y*. Compute $y = k$^-1$(H(m) + x* s) \bmod n$, where $H(m)$ is the hash of the message m, and s is the private key.
If $y = 0$, repeat Step 2 (generate a new k).

Step 6: The Signature. The signature is the pair (x, y), with the public key and the message

ECDSA: Verification

1. Verify that the public key is not the point at infinity and that it lies on the curve.
2. Verify that *n* * PubKey = 0 (the point at infinity).
3. Check that *x* and *y* are integers between 1 and *n* - 1.
4. Compute (*i*, *j*) = (*H*(*m*) * *y*-1 mod *n*) * G + (*x* * *y*-1 mod *n*) * PubKey.
5. The signature is valid if *x* = *i* mod *n*.

$$
\begin{aligned}
&\left(H(m)y^{-1} \mod n\right)\ G + \left(xy^{-1} \mod n\right)\ Q \\
&= \left(H(m)y^{-1} \mod n\right) G + \left(xy^{-1} \mod n\right) s\ G \\
&= \left((H(m) + sx)y^{-1}\right) \mod n\ G \\
&= \left((H(m) + sx)\, k(H(m) + sx)^{-1}\right) \mod n\ G \\
&= (k \mod n)\ G = k\, G = (i,\ j)
\end{aligned}
$$

Demonstration:

IF : x = i

The signature is valid (we can be sure that the private has been involved in the process).

Steps :

Q = s*G (public key)

Factorisation by G

y = k(H(m)+sx)^-1

delete H(m) + sx because of ^-1

XRP LEDGER