



## *Crypto Blockchains - XRP Ripple as AML, KYC, and SWIFT Replacement*

Cristian TOMA, Marius POPA

IT | Cyber Security Master Program – ISM  
Applied Computer Science and Cybersecurity R&D Team – ACS  
Department of Economic Informatics & Cybernetics – DICE | DEIC  
[www.ism.ase.ro](http://www.ism.ase.ro) | [www.acs.ase.ro](http://www.acs.ase.ro) | [www.dice.ase.ro](http://www.dice.ase.ro)



# Make Blockchain Technology working for Fin-Techs



Visit us at [www.fintech-ho2020.eu](http://www.fintech-ho2020.eu)

or contact us at [info@fintech-ho2020.eu](mailto:info@fintech-ho2020.eu)

or follow us on [LinkedIn](#)

The development of the materials has also been supported by the project **A FINancial supervision and TECHnology compliance training programme**.

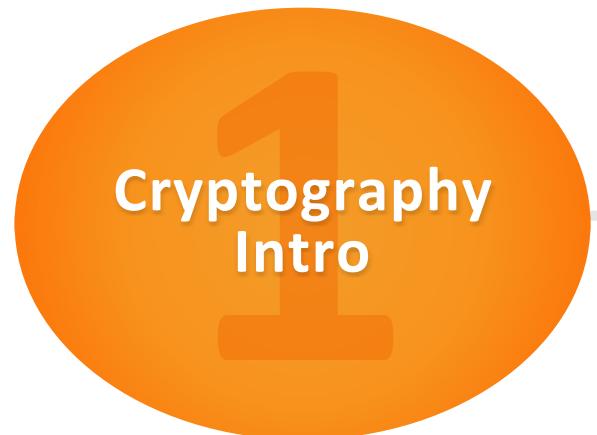
*This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825215 (Topic: ICT-35-2018 Type of action: CSA)*

All material presented here reflects only the authors' view. The European Commission is not responsible for any use that may be made of the information it contains.

# Crypto Blockchain Technology



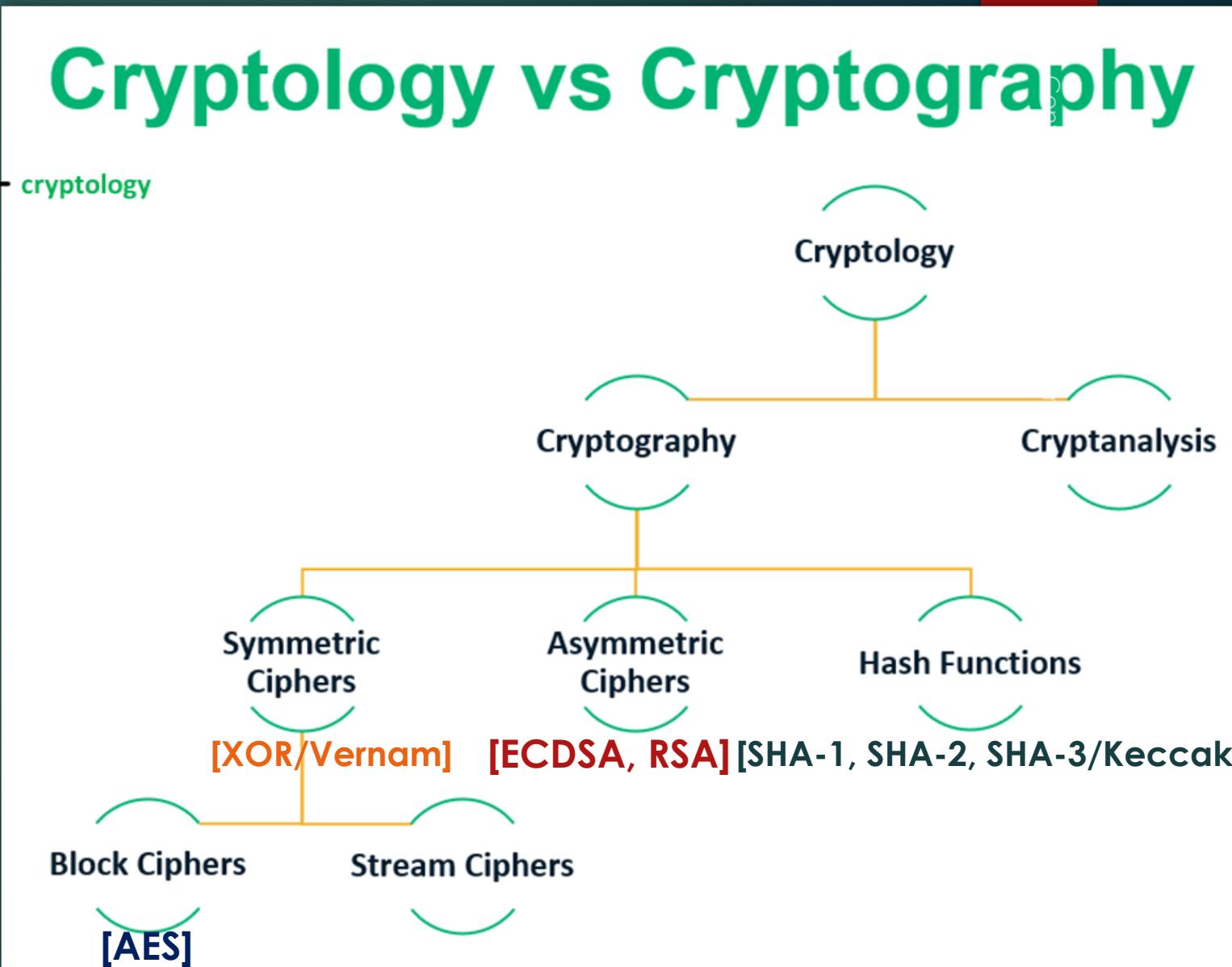
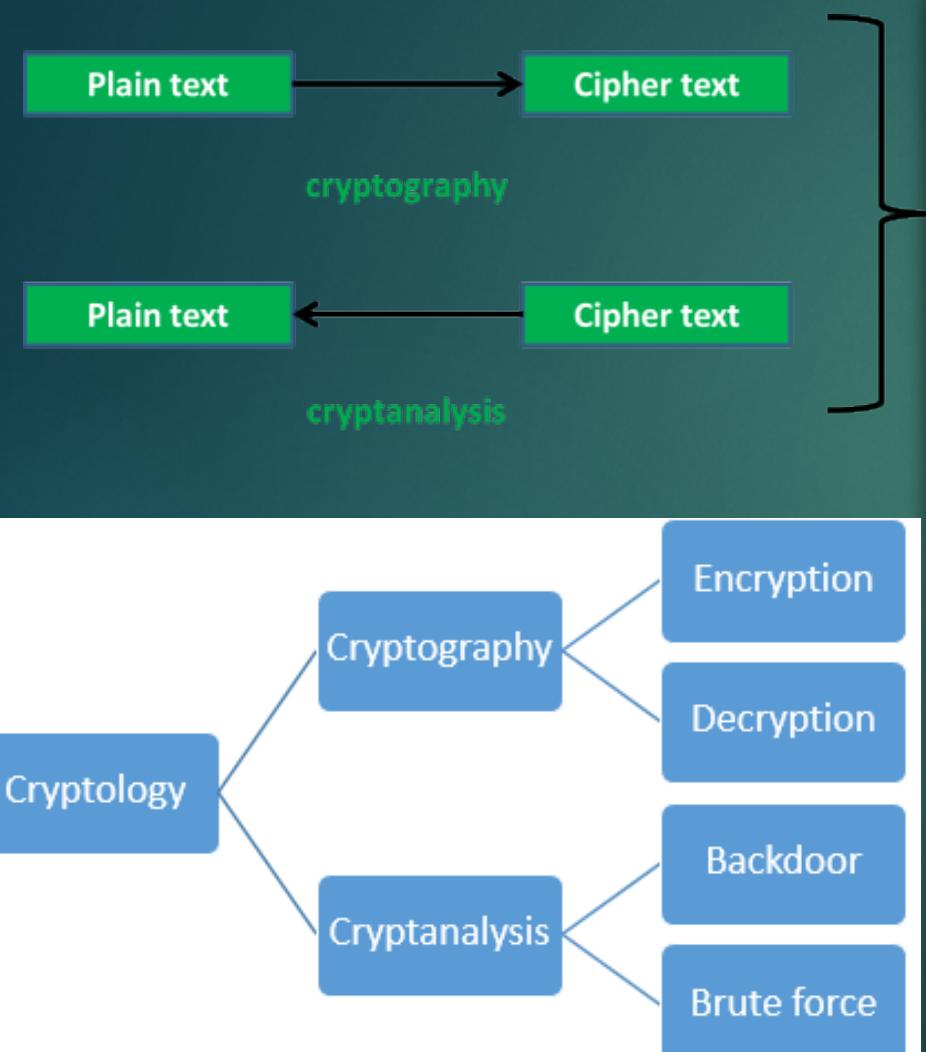
# Agenda for the Blockchain



[www.dice.ase.ro](http://www.dice.ase.ro)



[www.ism.ase.ro](http://www.ism.ase.ro)



# 01. Crypto Intro – XOR with symmetric key

6

XOR = Exclusive or or exclusive disjunction is a logical operation that outputs true only when inputs differ (one is true, the other is false).

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

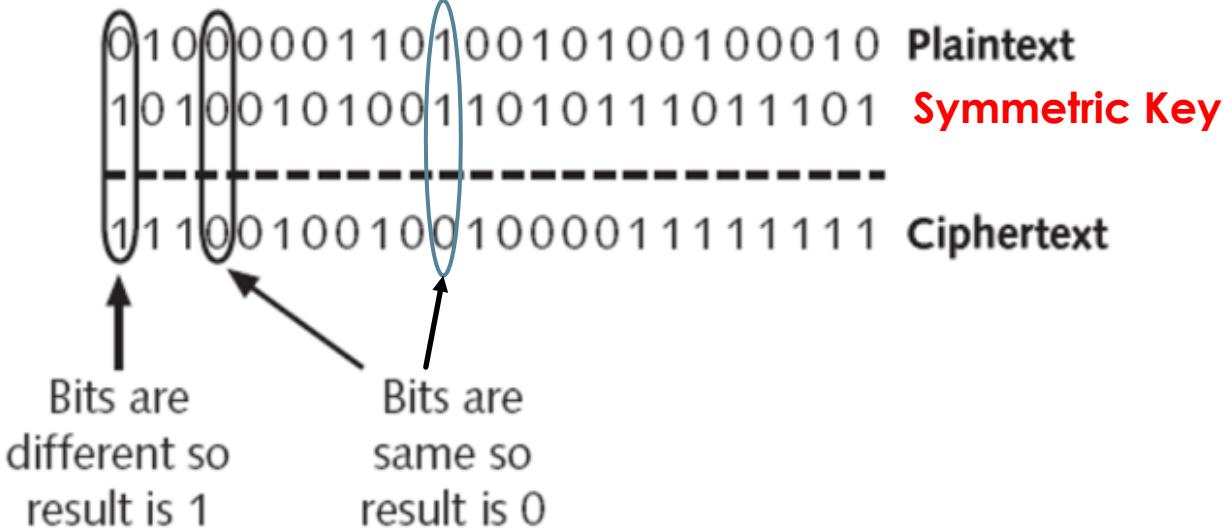
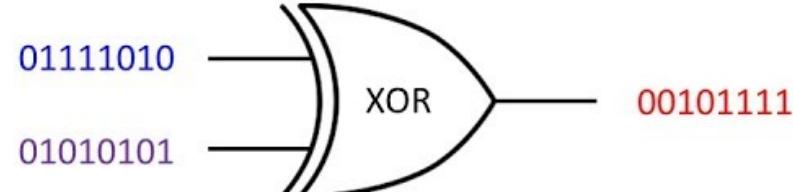
XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

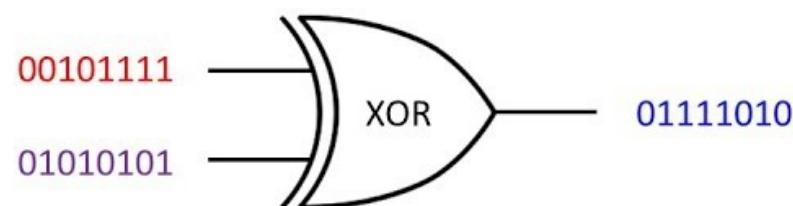
NOT Truth Table

A	B
0	1
1	0

## ENCRYPTION



## DECRIPTION



Home Insert Draw Design Transitions Anim Mute Start Video Security Participants New Share Pause Share Annotate Remote Control More

Cut Copy Layout Reset Calibri (Body) You are screen sharing Stop Share

Paste Format New Slide Section Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

# 01. Crypto Intro – XOR with symmetric key

5

XOR = Exclusive or or exclusive disjunction is a logical operation that outputs true only when inputs differ (one is true, the other is false).

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

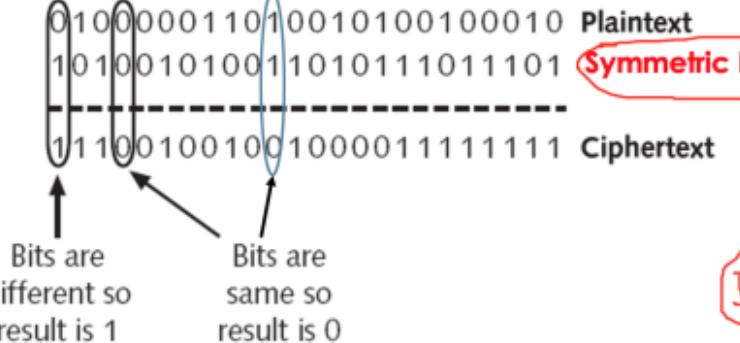
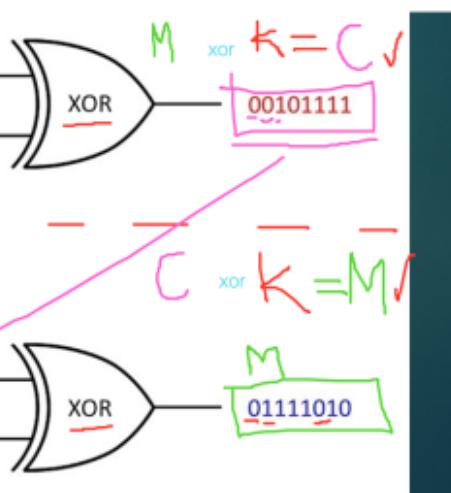
XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

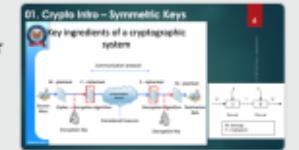
NOT Truth Table

A	B
0	1
1	0

Exclusive disjunction essentially means 'either one, but not both nor none'. In other words, the statement is true if and only if one is true and the other is false. For example, if two horses are racing, then one of the two will win the race, but not both of them.



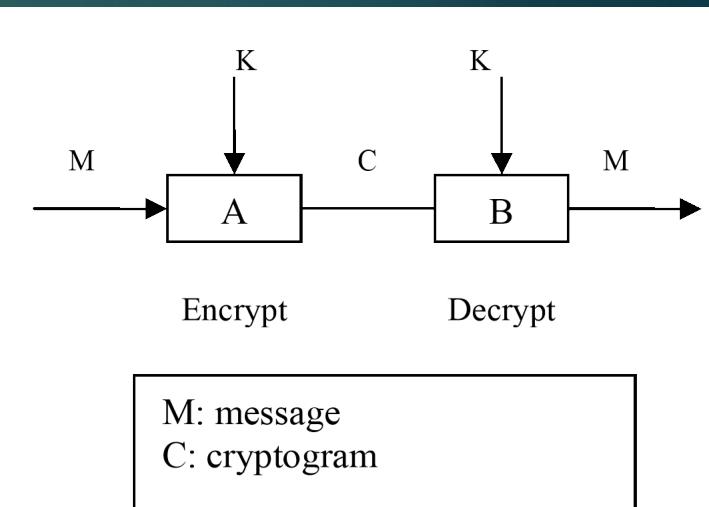
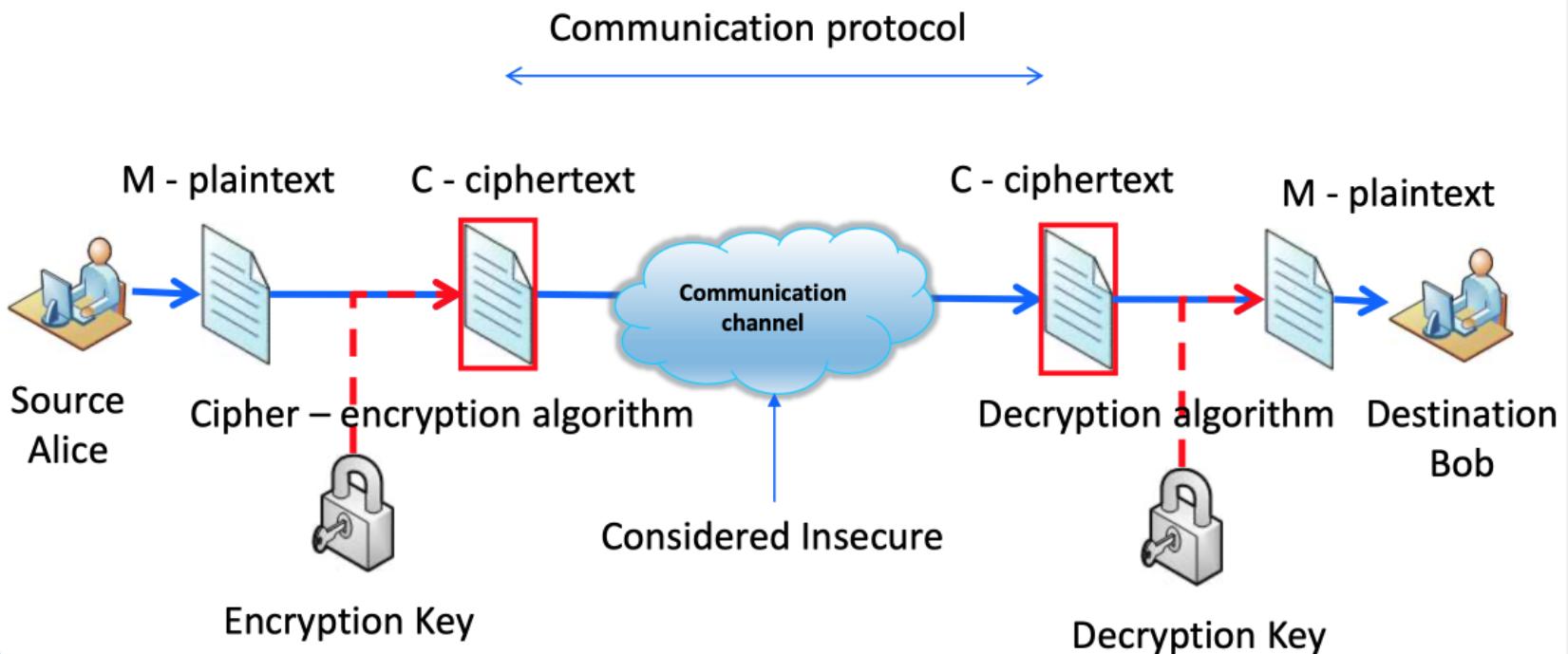
Click to add notes



# 01. Crypto Intro – Symmetric Keys



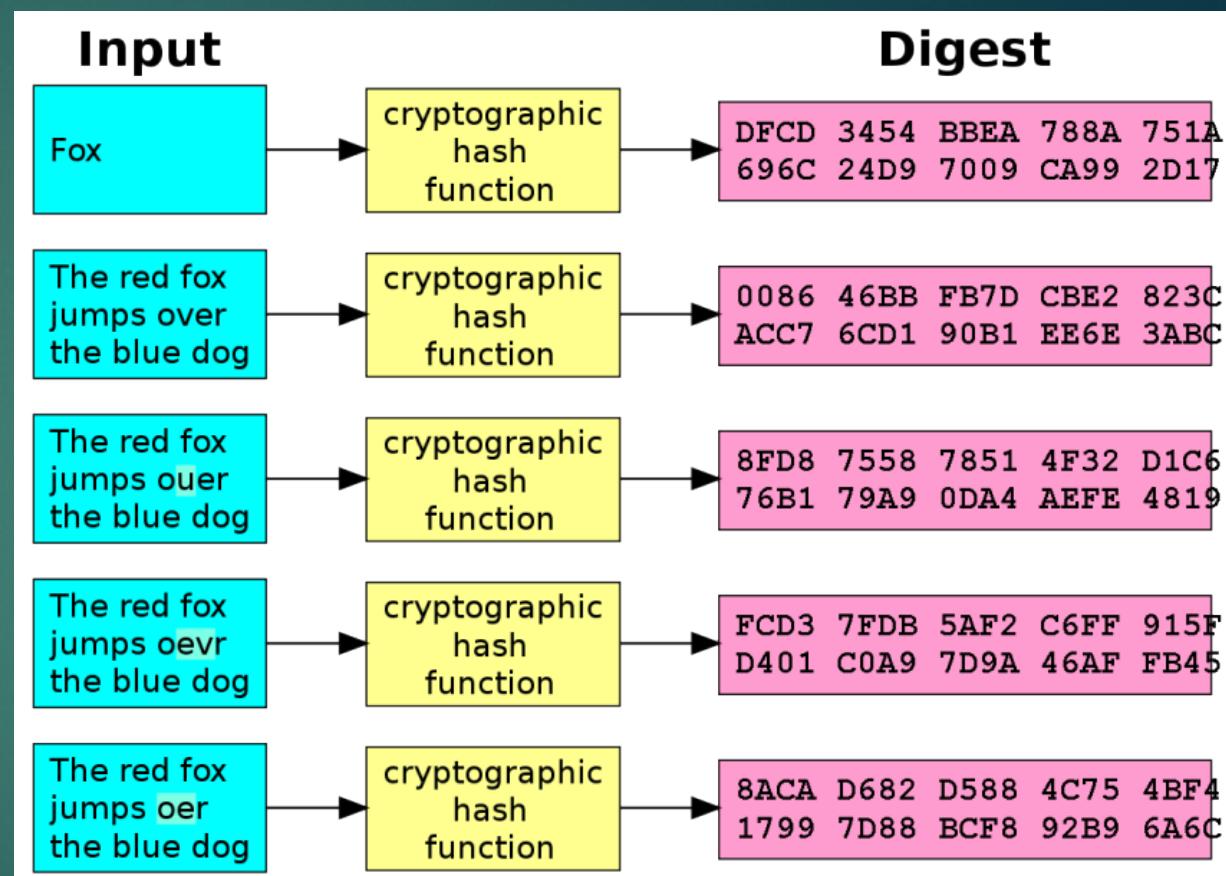
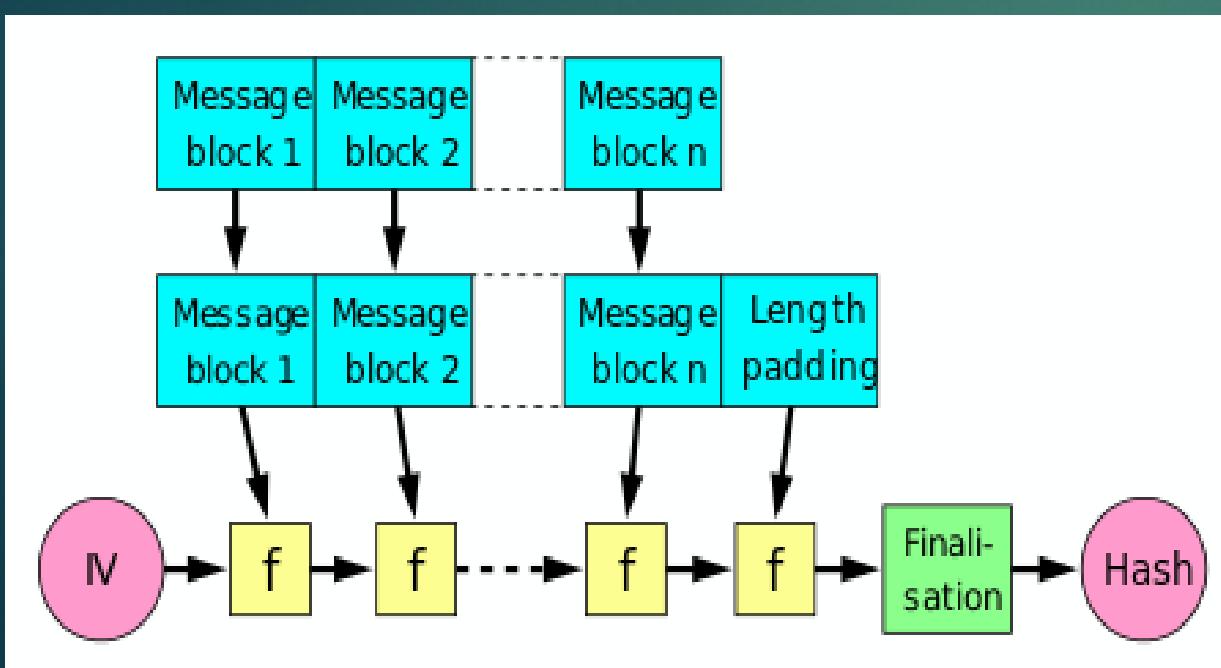
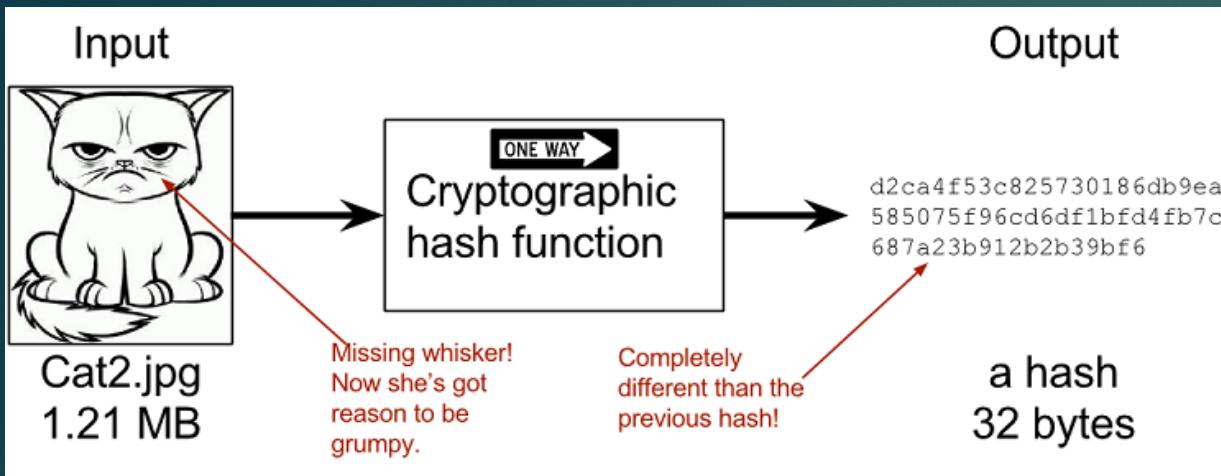
## Key ingredients of a cryptographic system



# 01. Crypto Intro - Hash

[https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

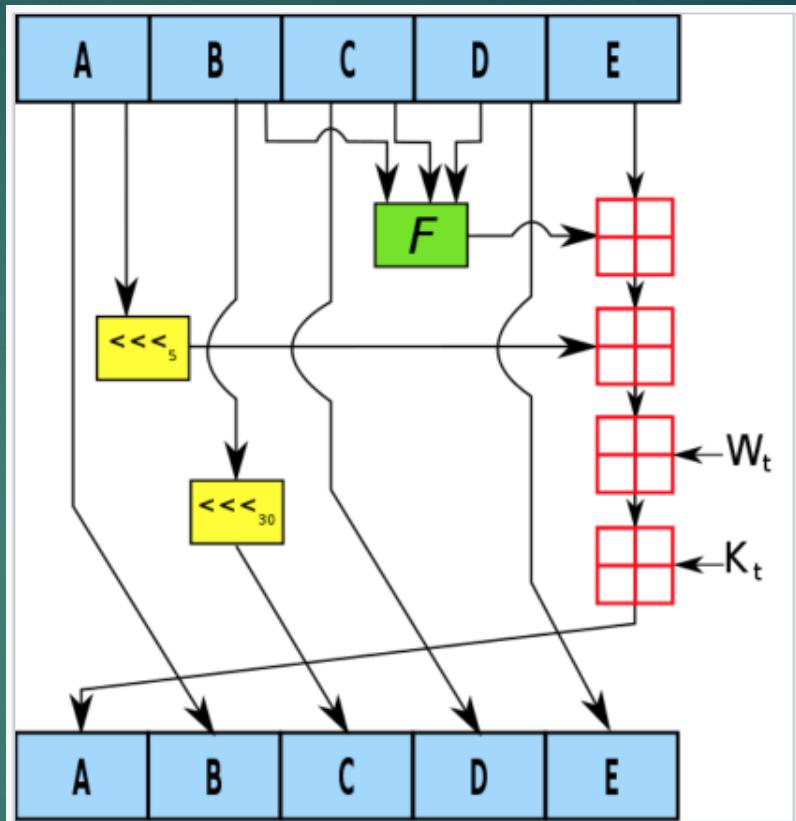
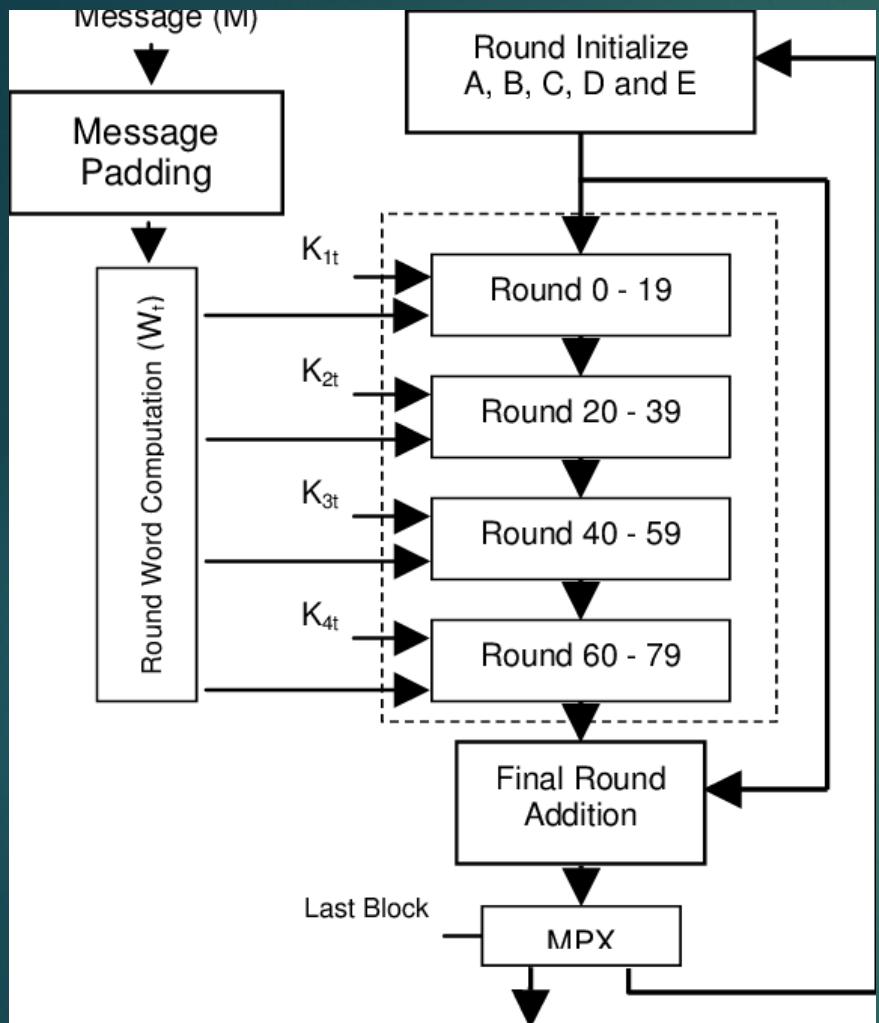
9



# 01. Crypto Intro - Hash

10

<https://en.wikipedia.org/wiki/SHA-1>



One iteration within the SHA-1 compression function:

A, B, C, D and E are 32-bit **words** of the state;

F is a nonlinear function that varies;

<<<<sub>n</sub> denotes a left bit rotation by n places;

n varies for each operation;

W<sub>t</sub> is the expanded message word of round t;

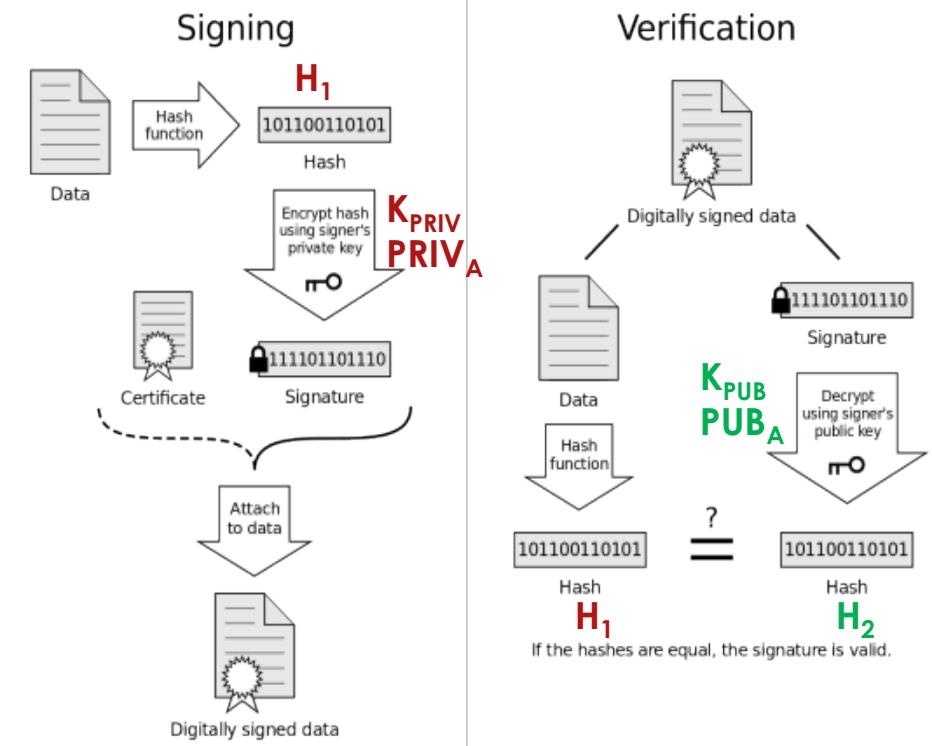
K<sub>t</sub> is the round constant of round t;

⊕ denotes addition modulo 2<sup>32</sup>.

<https://www.slideshare.net/ruchimaurya487/sha-secure-hashing-algorithm>

# 01. Crypto Intro – Asymmetric Keys

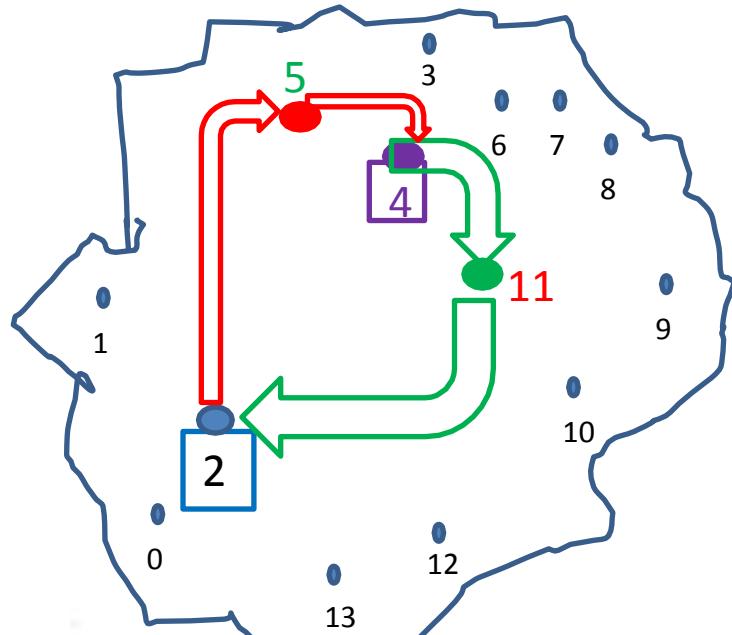
11



# Cryptography Elements used in Blockchains

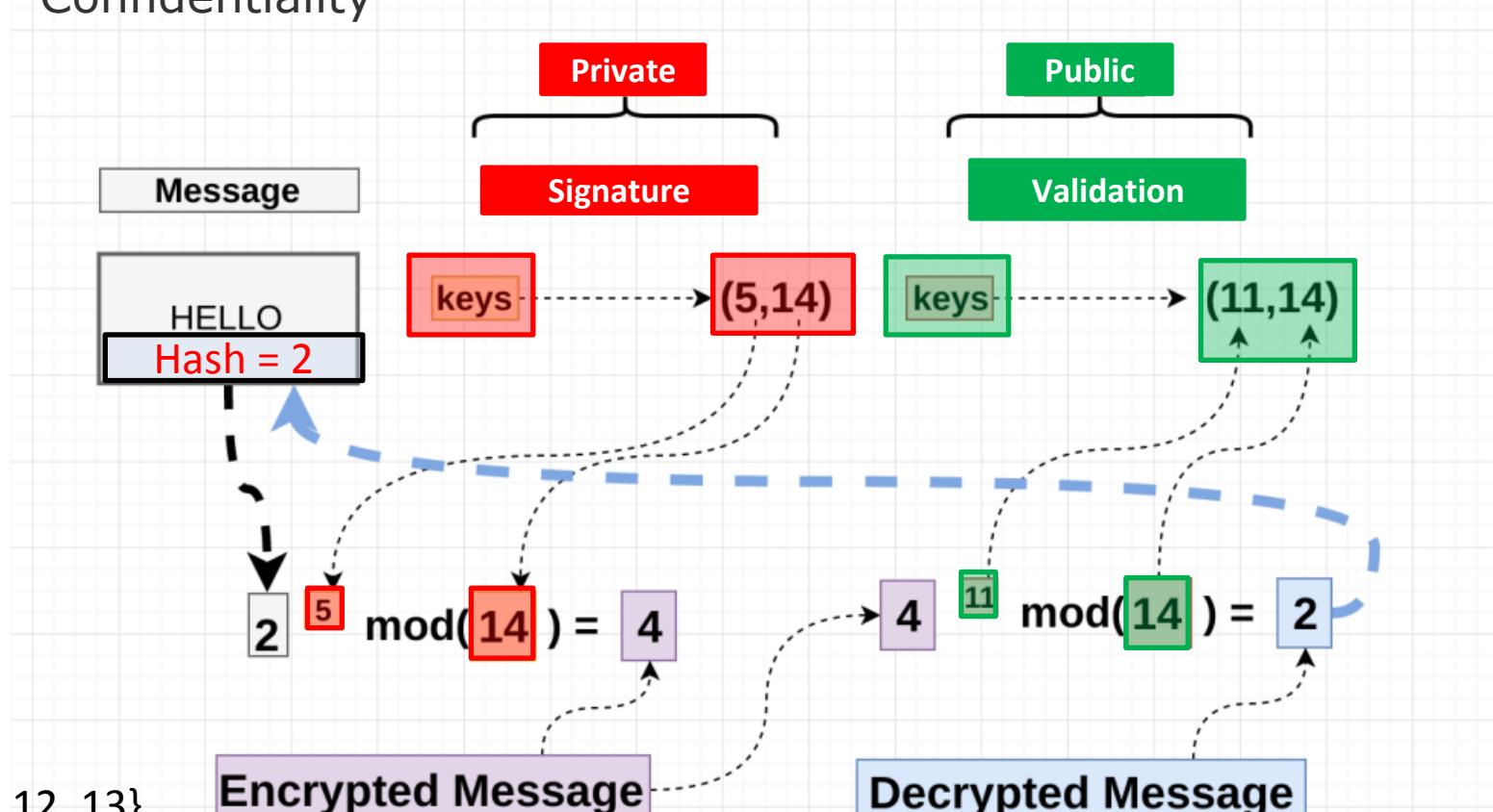
## Asymmetric-Key Cryptography - Rivest-Shamir-Adleman (RSA)

- RSA works for:
  - **Signature – Non-Repudiation**
  - Encryption/Decryption - Confidentiality



$$Z_{14} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$$

Source: <https://hackernoon.com/how-does-rsa-work-f44918df914b>



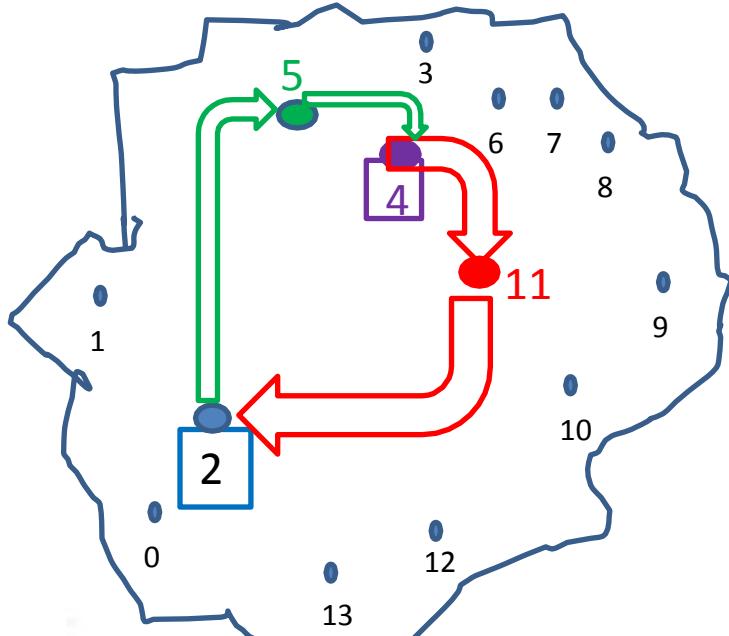
<https://www.mtholyoke.edu/courses/quenell/s2003/ma139/js/powermod.html>

<https://miniwebtool.com/modulo-calculator/>

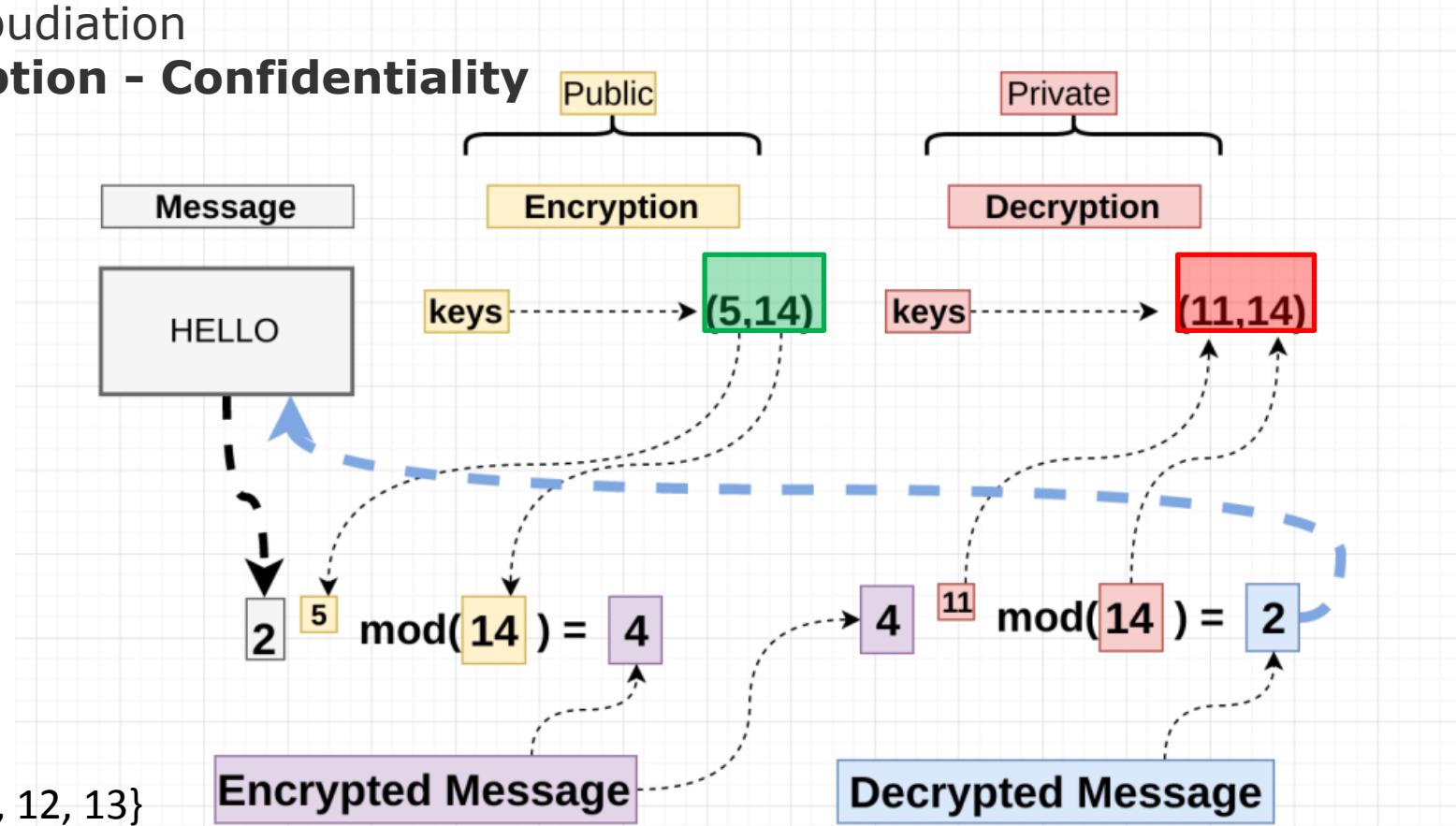
# Cryptography Elements used in Blockchains

## Asymmetric-Key Cryptography - Rivest-Shamir-Adleman (RSA)

- RSA works for:
  - Signature – Non-Repudiation
  - **Encryption/Decryption - Confidentiality**



$$Z_{14} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$$



Search in Presentation Share

Home Insert Draw Design Transitions Anim Mute Start Video Security Participants New Share Pause Share Annotate Remote Control More

Calibri (Body) 12 A A A You are screen sharing Stop Share

Cut Copy Format Paste New Slide Reset Section

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

4 01. Crypto Intro Cryptology vs Cryptography

5 01. Crypto Intro - XOR with asymmetric key

6 01. Crypto Intro - Symmetric Keys Key Ingredients of a cryptographic system

7 01. Crypto Intro - Hash Hashes are used to compare data integrity

8 01. Crypto Intro - Hash Hashes are used to compare data integrity

9 01. Crypto Intro - Asymmetric Keys

10 01. Crypto Intro - RSA simple numeric sample

**01. Crypto Intro – Asymmetric Keys**

**9**

**Document**  $M$  → Hash (DIGEST) → RSA → **S** → **S** (Signature)

**Hash (MD5, SHA-1)**

**USER A** **USER B**

**PUBA** → RSA → **H2**

**Hash (DIGEST)**

**H1** ← Hash ← **M**

**AUTHENTICATION**  
if  $(H1 == H2)$  the authentication is correct

**Signing**  
Data → Hash → Encrypt hash using signer's private key → Signature → Certificate → Attach to data → Digitally signed data

**Verification**  
Digitally signed data → Hash function → Signature → Decrypt using signer's public key → Hash → If the hashes are equal, the signature is valid.

$(2^6) \bmod 7 = (2^3) \bmod 7$

0, 1, 2, 3, 4, 5, 6

$M = 2$   
 $K_{PRIV} = 3$   
 $K_{PUB} = 6$

$(M^K_{PRIV}) \bmod 7 = 2^{3 \bmod 6} \bmod 7 = 8 \bmod 7 = 1$   
 $2^6 \bmod 7 = 64 \bmod 7 = [(2^3) * (2^3)] \bmod 7 = [(2^3) \bmod 7] * [(2^3) \bmod 7] = 1 * 1 = 1$

Notes Comments

Slide 9 of 134 English 130% 16 Feb 10:41

# 01. Crypto Intro – RSA simple numeric sample

15

The numeric example is just for the digital signature, is depict also in figure for the electronic signing process:

- given  $p=53$  and  $q=61$ , 2 secret prime numbers of A.
- given  $n_A = 53 \cdot 61 = 3233$  the product of these two numbers (if n very big the possibility to find p and q – **meaning to factorize on n** – is very low).
- The indicator of Euler calculated is:
  - $\Phi(n) = (p-1)*(q-1) = 52 \cdot 60 = 3120$ .
- Is chosen a secret private key **PRIV<sub>A</sub> = 71**; (meaning a number relatively prime with 3120 and in the period  $[\max(p,q)+1, n-1]$ )
- Is calculated the public key **PUB<sub>A</sub> = 791** as multiplied reverse mod 3120 of PRIV<sub>A</sub>:
  - $(71 \cdot 791) \bmod 3120 = 1$  i.e **56161 mod 3120 = 1;**

Online modulo calculators:

<https://www.mtholyoke.edu/courses/quenell/s2003/ma139/js/powermod.html>

<https://miniwebtool.com/modulo-calculator/>

# 01. Crypto Intro – RSA simple numeric sample

16

- Is considered a document whose digest  $H(M)=h$  is 13021426. As the value overpasses the length of the module  $n_A = 3233$ , this digest will be broken in two blocks (1302 and 1426), that A will sign separately, using the **private key of A -  $\text{PRIV}_A = (71, 3233) = (\text{EXP\_PRIV}_A, \text{MOD}_A)$** :
  - $(1302^{71}) \bmod 3233 = 1984;$
  - $(1426^{71}) \bmod 3233 = 2927;$
- the electronic signature obtained is  **$S = 1984\ 2927$** ;
- $S$  obtained is transmitted to the previous step and  $M$  (in clear – for this example, there is no confidentiality).
- When receiving, B (B receives the package  $S+M$ ) will calculate  $H(M)$  in two modes and will compare them:
- he will calculate  $H(M)$  with the **public key of A -  $\text{PUB}_A = (791, 3233) = (\text{EXP\_PUB}_A, \text{MOD}_A)$** :
  - $(1984^{791}) \bmod 3233 = 1302$
  - $(2927^{791}) \bmod 3233 = 1426;$
  - **So  $H_1 = 1302\ 1426$ ;**
- and will calculate  $H(M)$  on the received message:  
 **$H_2 = H(M) = 1302\ 1426;$**

Online modulo calculators:

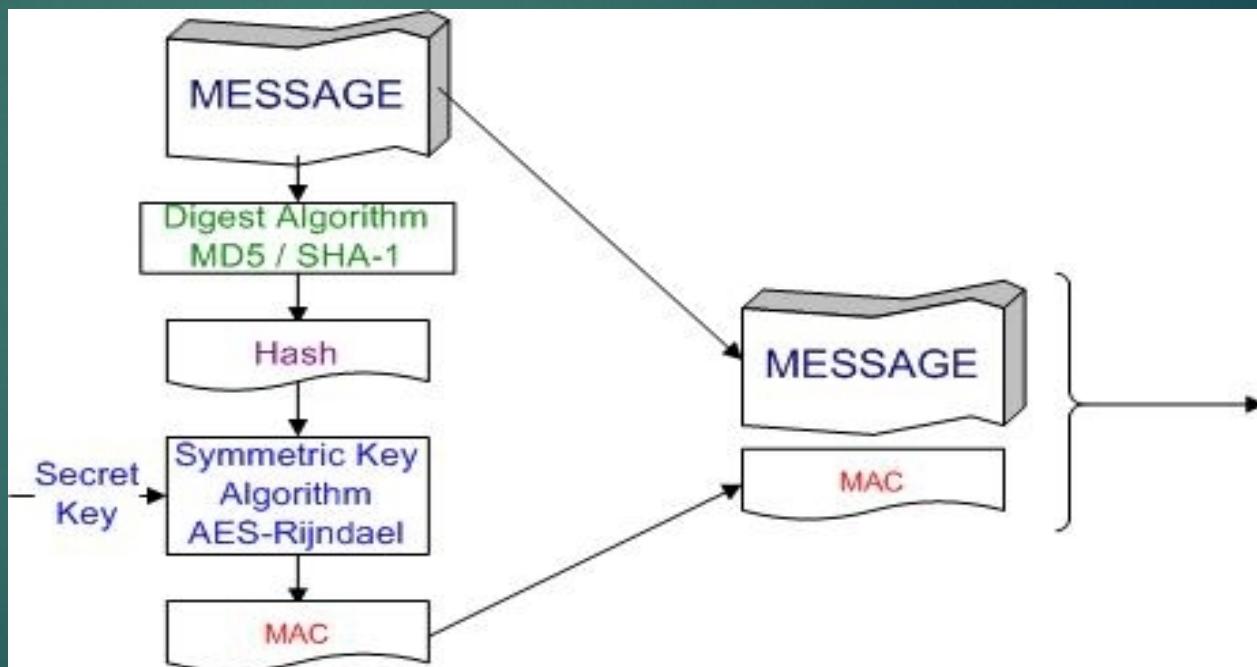
<https://www.mtholyoke.edu/courses/quenell/s2003/ma139/js/powermod.html>

<https://miniwebtool.com/modulo-calculator/>

# 01. Crypto Intro – HMAC

Wiki: “In [cryptography](#), an **HMAC** (sometimes expanded as either **keyed-hash message authentication code** or **hash-based message authentication code**) is a specific type of [message authentication code](#) (MAC) involving a cryptographic hash function and a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authenticity of a message.”

“Any cryptographic hash function, such as [SHA-2](#) or [SHA-3](#), may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-X, where X is the hash function used (e.g. HMAC-SHA256 or HMAC-SHA3-256). The cryptographic strength of the HMAC depends upon the [cryptographic strength](#) of the underlying hash function, the size of its hash output, and the size and quality of the key.”



Message Authentication Code Steps

# 01. Crypto Intro – HMAC

18

This definition is taken from [RFC 2104](#):

$$\text{HMAC}(K, m) = \text{H} \left( (K' \oplus opad) \parallel \text{H} \left( (K' \oplus ipad) \parallel m \right) \right)$$

$$K' = \begin{cases} \text{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

where

H is a cryptographic hash function

m is the message to be authenticated

K is the secret key

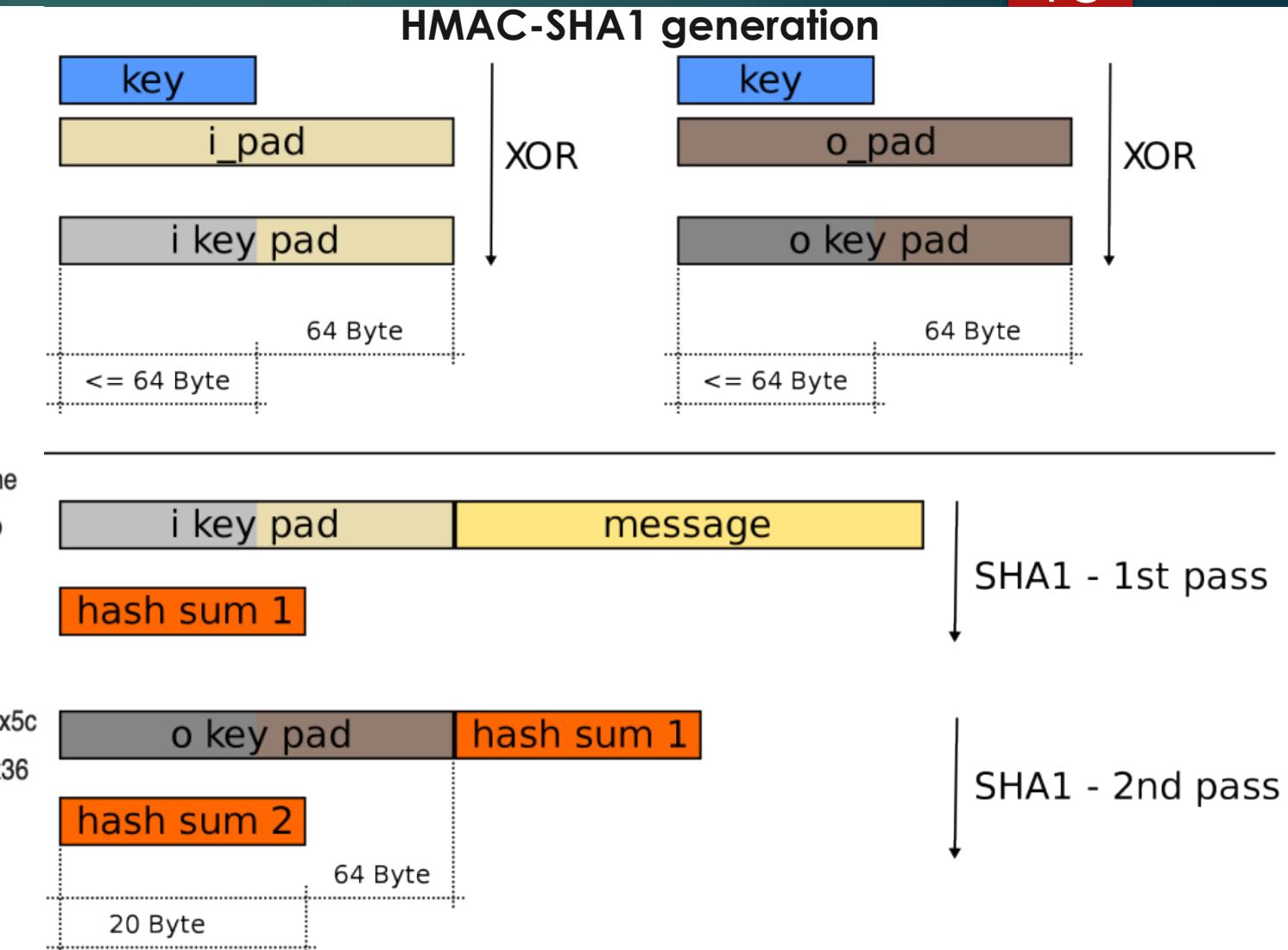
K' is a block-sized key derived from the secret key, K; either by padding to the right with 0s up to the block size, or by hashing down to less than or equal to the block size first and then padding to the right with zeros

|| denotes **concatenation**

$\oplus$  denotes bitwise **exclusive or (XOR)**

opad is the block-sized outer padding, consisting of repeated bytes valued 0x5c

ipad is the block-sized inner padding, consisting of repeated bytes valued 0x36



# 01. Crypto Intro a little bit more details

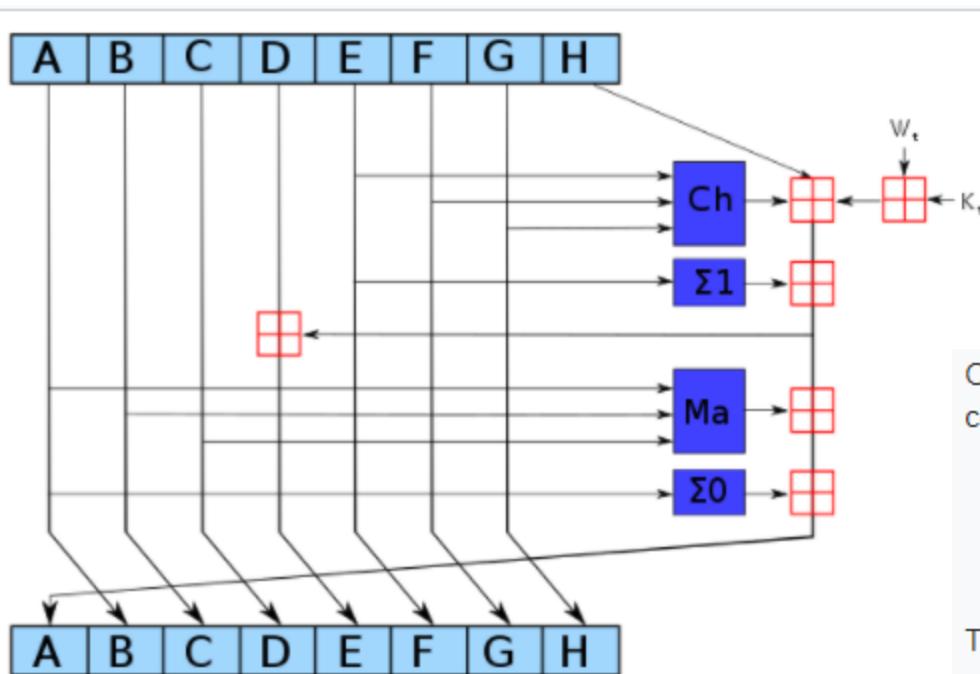
## Hash Functions

- A hash function maps a large input set to a smaller output set. The values returned by a hash function are called **hash values**, **hash codes**, **digests**, or simply **hashes**.
- Hash functions are related to (and often confused with) checksums, check digits, fingerprints, lossy compression, randomization functions, error-correcting codes, and ciphers.
- Cryptographic hash algorithms: MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160.

# Cryptography Elements used in Blockchains

## SHA-2

- SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA).



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\text{Ma}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.  
The red  $\square$  is addition modulo  $2^{32}$  for SHA-256, or  $2^{64}$  for SHA-512.

# Cryptography Elements used in Blockchain

## DSS/DSA without EC

Digital Signature Algorithm is the algorithm for digital signature of the DSS standard, issued by NIST in August 1991.

The parameters of the system are:

- **Global parameters** (the same for everybody):
  - $p$  is a prime number,  $p$  in  $(2^{511}, 2^{512})$  has 512 bits;
  - $q$  a prime divisor of  $(p-1)$  has 160 bits,  $q$  in  $(2^{159}, 2^{160})$ ;
  - $g$  an integer with the property:  $g = h^{(p-1)/q} \text{ mod } p$ , where  $h$  is a integer relatively prime with  $p$ ,  $h$  in  $(0..p)$  so that:  $h^{(p-1)/q} \text{ mod } p > 1$ .
  - $H = H(M)$  the hash function to calculate the digest of a message.
- The user's parameters (different from one user to another):
  - secret key:  $\text{PRIV}$ , an integer in  $(0..q)$ ;
  - public key:  $\text{PUB}$ , an integer, calculated so that:  
$$\text{PUB} = g^{\text{PRIV}} \text{ mod } p$$
- Signature parameters (different from one signature to another):
  - $M$  the message that will be signed;
  - $k$  a random integer,  $k$  in  $(0,q)$ , chosen different for each signature.

**The digital signature  $S$  with DSA** of a message  $m$  is the pair  $S=(r, s)$  and is made using the secret key of the user-sender, for instance A:

- it is chosen  $k$  in  $(0, q)$ , prime with  $q$ ;
- it is calculated:
  - $r = (g^k \text{ mod } p) \text{ mod } q$  and
  - $s = ((k^{-1}) * (H(M) + \text{PRIV}_A * r)) \text{ mod } q$

### Digital Signature Verification with DSA:

At reception B receives  $M$ ,  $S=(r, s)$ , that may eventually differ from  $M$  and  $S$ , transmitted from the origin, because of some tentative of fraud.

B will calculate:  $w \text{ or } s1 = s^{-1} \text{ mod } q$  ( $s$  must be reversible).

The signature is valid if is verified the equation  $r=r'$ , where  $r'$  is:  
$$r' = (g^{H(M)*s1} * (\text{PUB}_A)^{r*s1} \text{ mod } p) \text{ mod } q$$

# Cryptography Elements used in Blockchain

## DSS/DSA without EC

It is considered the following numeric example:

Given  $q=101$  and  $p=78*q+1 = 7879$ . Since 3 is the primitive root of the ring of rests classes  $Z_{7879}$ , it may be considered:  $g=3^{78} \text{ mod } 7879 = 170$ .

Given the secret key of the user A,  $\text{PRIV}_A=75$ . It results the pair public key:  $\text{PUB}_A = g^{\text{PRIV}_A} \text{ mod } 7879 = 4567$ . It is presumed that sender A, who has the secret key  $\text{PRIV}_A$ , will transmit a message M whose digest is 1234 to receiver B. User A chooses the constant  $k=50$  as parameter of the signature and he calculates:  $k^{-1} \text{ mod } 101 = 99 <= \text{verification: } 99*50 \text{ mod } 101 = 1$ .

The two components  $(r, s)$  of signature  $S$  are calculated:  
 $r=(g^k \text{ mod } p) \text{ mod } q = (170^{50} \text{ mod } 7879) \text{ mod } 101 = 2518 \text{ mod } 101 = 94$ .

$s=((k^{-1})*(H(M)+\text{PRIV}_A*r)) \text{ mod } q = 99 * (1234 + 75*94) \text{ mod } 101 = 97$ .

The signature of message M, the pair  $S=(94, 97)$ , is received by B who will check it with the help of A's public key:

$$w \text{ or } s1 = s^{-1} \text{ mod } q = 97^{-1} \text{ mod } 101 = 25; \\ r' = (g^{H(M)*s1} \text{ mod } q * (\text{PUB}_A)^{r*s1 \text{ mod } q} \text{ mod } p) \text{ mod } q.$$

It is calculated:

- $H(M) * s1 \text{ mod } q = 1234*25 \text{ mod } 101 = 45$ ;
- $r*s1 \text{ mod } q = 94*25 \text{ mod } 101 = 27$

and the verification:

$$r' = (170^{45} * 4567^{27} \text{ mod } 7879) \text{ mod } 101 = 2518 \text{ mod } 101 = 94.$$

Because  $r=r'$ , the signature is valid.

Online modulo calculators:

<https://www.mtholyoke.edu/courses/quenell/s2003/ma139/js/powermod.html>

<https://miniwebtool.com/modulo-calculator/>

<https://planetcalc.com/3311/>

# Cryptography Elements used in Blockchains

## ECDSA Signature for Bitcoin / Ethereum

The ECDSA signing algorithm ([RFC 6979](#)) takes as input a message (M): (*msg*) applying a private key [*privKey*] or [*d*] and produces as output a **signature (S)**, which consists of pair of integers {*r*, *s*}. =>  $S = \{r, s\} = M \rightarrow \text{ECDSA}[\text{privKey}]$  using the **elliptic curve secp256k1 from SEC-2**.

The **ECDSA signing** algorithm is based on the [ElGamal signature scheme](#) and works as follows (with minor simplifications):

1. Calculate the message **hash**, using a cryptographic hash function like SHA-256:  $h = \text{hash}(msg)$
2. Generate securely a **random** number *k* in the range [1..*n*-1]
  - o In case of **deterministic-ECDSA**, the value *k* is HMAC-derived from  $h + \text{privKey}$  (see [RFC 6979](#))
3. Calculate the random point  $R = k * G$  and take its x-coordinate:  $r = R.x$
4. Calculate the signature proof:  $s = k^{-1} * (h + r * \text{privKey}) \pmod{n}$ 
  - o The modular inverse  $k^{-1} \pmod{n}$  is an integer, such that  $k * k^{-1} \equiv 1 \pmod{n}$
5. Return the **signature** {*r*, *s*}.

The calculated **signature** {*r*, *s*} is a pair of integers, each in the range [1...*n*-1]. It encodes the random point  $R = k * G$ , along with a proof *s*, confirming that the signer knows the message *h* and the private key *privKey*. The proof *s* is by idea verifiable using the corresponding *pubKey*.

**ECDSA signatures** are **2 times longer** than the signer's **private key** for the curve used during the signing process. For example, for 256-bit elliptic curves (like secp256k1) the ECDSA signature is 512 bits (64 bytes) and for 521-bit curves (like secp521r1) the signature is 1042 bits.

## Verify ECDSA Signature for Bitcoin / Ethereum

The algorithm to **verify a ECDSA signature** takes as input:

- the message  $msg$  that has been signed
- the **signature  $\{r, s\}$**  produced from the signing algorithm
- the public key  $pubKey$ , corresponding to the signer's private key ( $pubKey = privKey * G$ ).

The output is boolean value: **valid** or **invalid** signature. The **ECDSA signature verify** algorithm works as follows (with minor simplifications):

1. Calculate the message **hash**, with the same cryptographic hash function used during the signing:  $h = \text{hash}(msg)$
2. Calculate the modular inverse of the signature proof:  $s1 = s^{-1} \pmod{n}$
3. Recover the random point used during the signing:  $R' = (h * s1) * G + (r * s1) * pubKey$
4. Take from  $R'$  its x-coordinate:  $r' = R'.x$
5. Calculate the signature validation **result** by comparing whether  $r' == r$

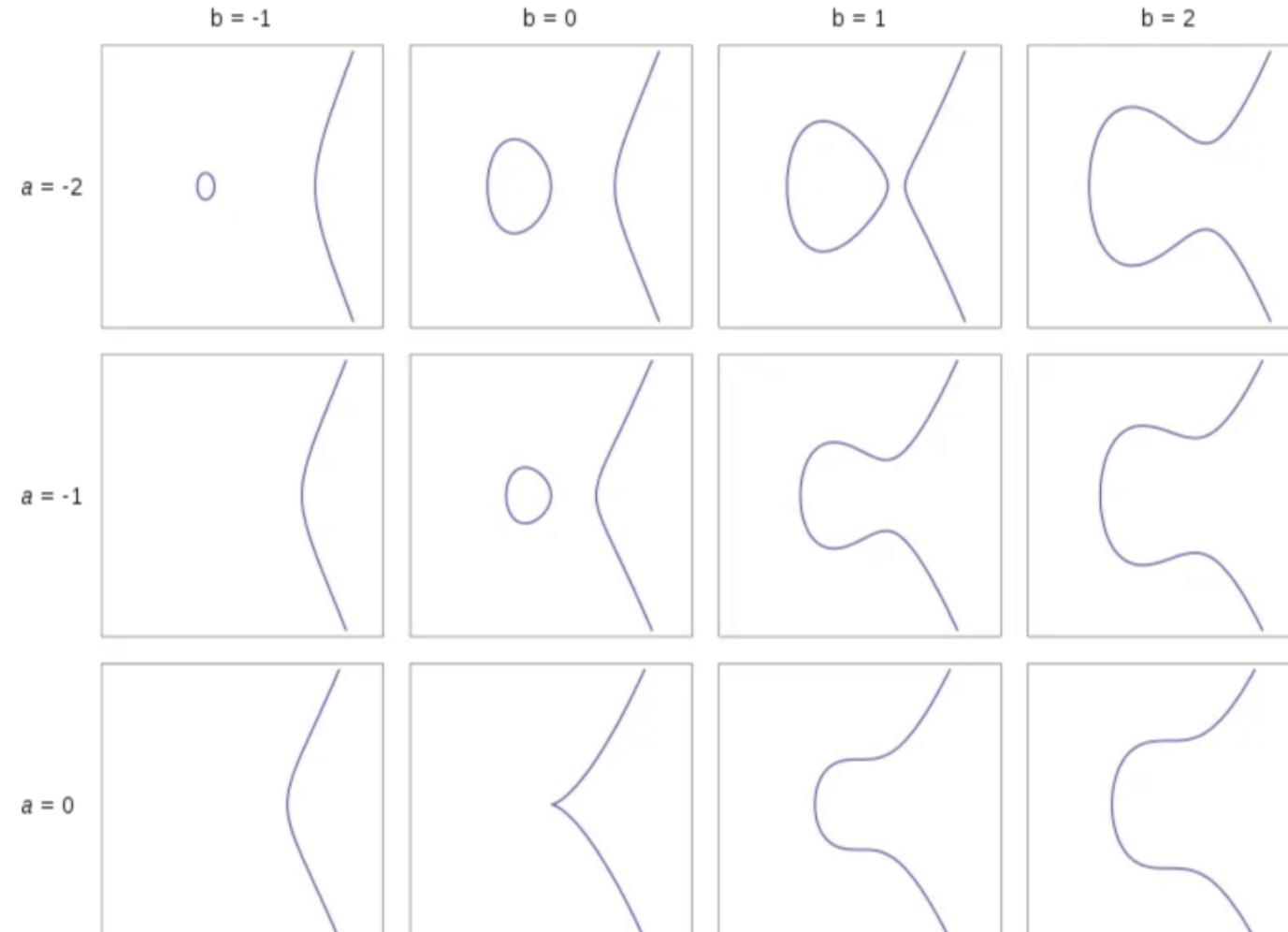
The general idea of the signature verification is to **recover the point  $R'$**  using the public key and check whether it is same point  $R$ , generated randomly during the signing process.

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

Elliptic curves have a general form of:

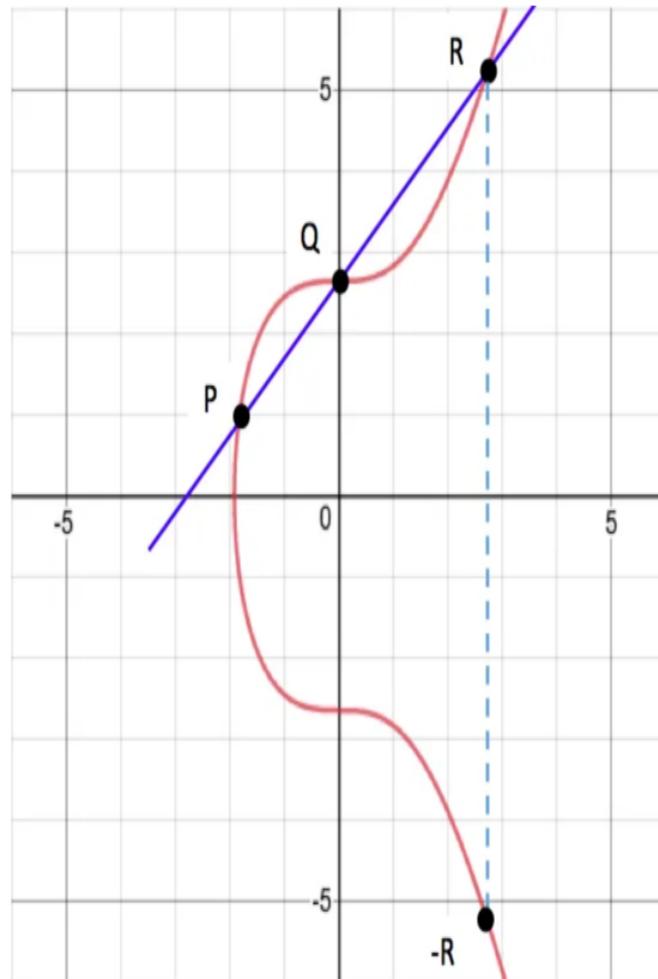
$$y^2 = x^3 + ax + b$$

and this is what they can look like:

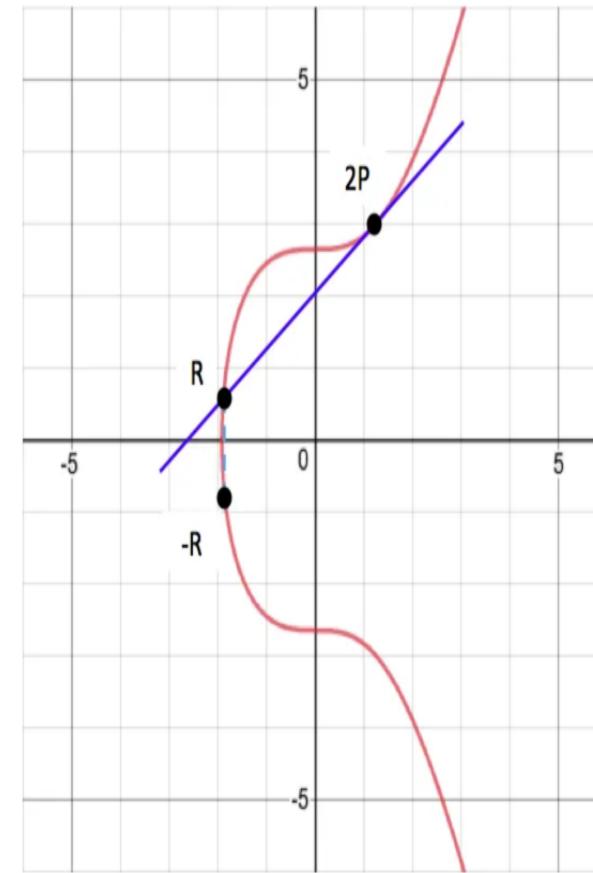


# Cryptography Elements used in Blockchains – ECDSA over Finite Field

However, this just gets us  $P + Q + R = 0$ . We want  $P + Q = \text{'something'}$  (which is  $P + Q = -R$ ). The 'negative' of a point is the mirror of the point, and elliptic curves just happen to be symmetric about the x-axis:



Point multiplication is a special case of point addition. If a line is tangent to the curve, then it will only intersect in 2 locations. This is the same characteristic as if  $P = Q$ , or  $2P = -R$ . This is also known as 'point doubling':



# Cryptography Elements used in Blockchains – ECDSA over Finite Field

## Generating Public-Private Keys

In ECDSA, a private key is simply a random number. The public key is found by multiplying the ‘base point’ by the private key. The base point is defined by whichever kind of ECDSA you are using (Bitcoin uses [secp256k1 \[Section 2.4.1\]](#)).

To see how this works, let's take a look at an example. Let's use 9 as a private key:

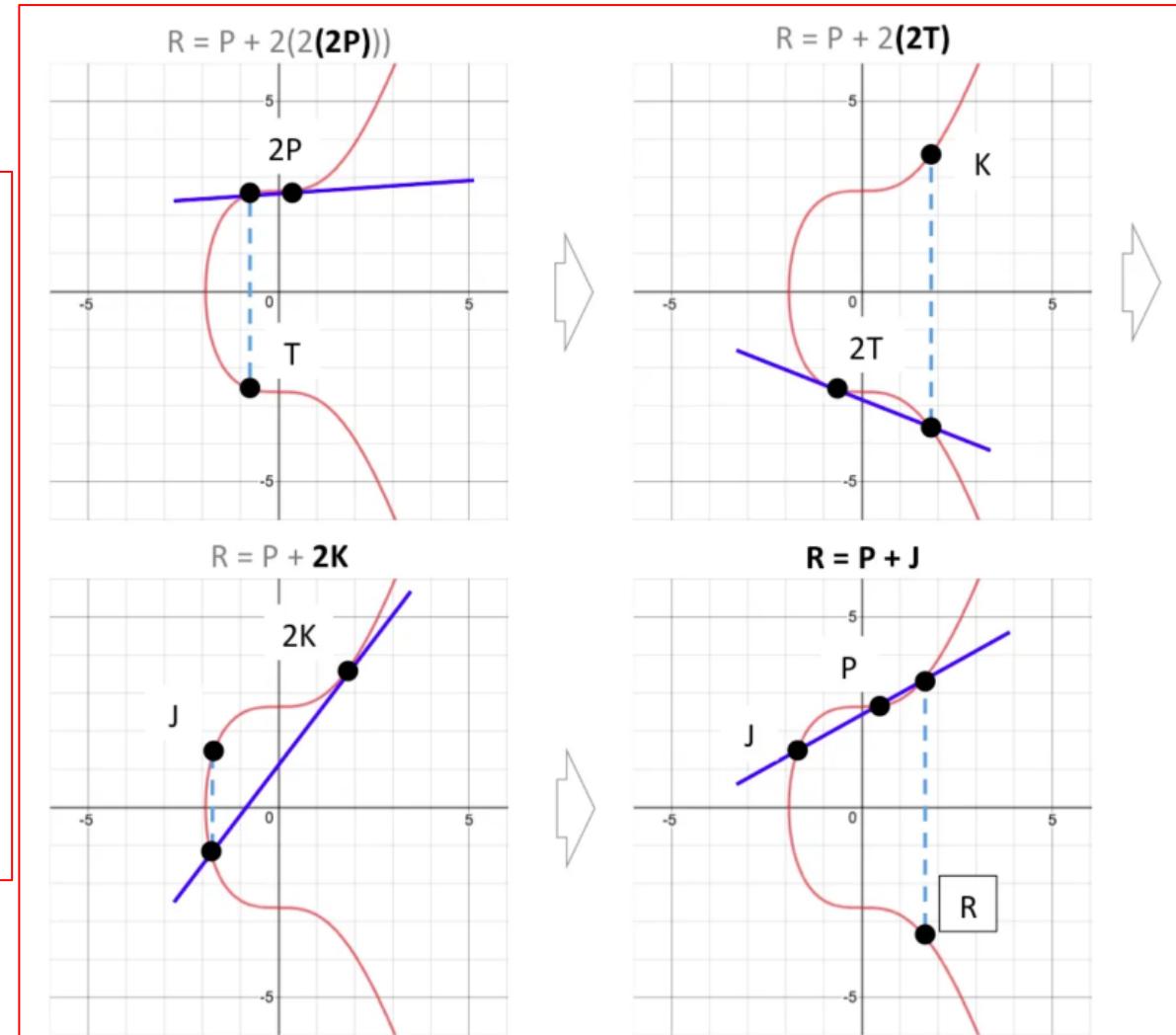
$$\text{Public} = \text{Private} * \text{Base} \rightarrow R = 9P$$

Again, as there is no point division, this is a one-way function. It is relatively easy to get the public key, but impossible to calculate the private key from the public key.

However, in elliptic math, we cannot straight up multiply a point by 9. We need to break down 9 into a series of point doublings and point additions:

$$9P = P + 8P = P + 2(4P) = P + 2(2(2P))$$

To multiply by 9 is really to do 3 sets of point doubling and a point addition.

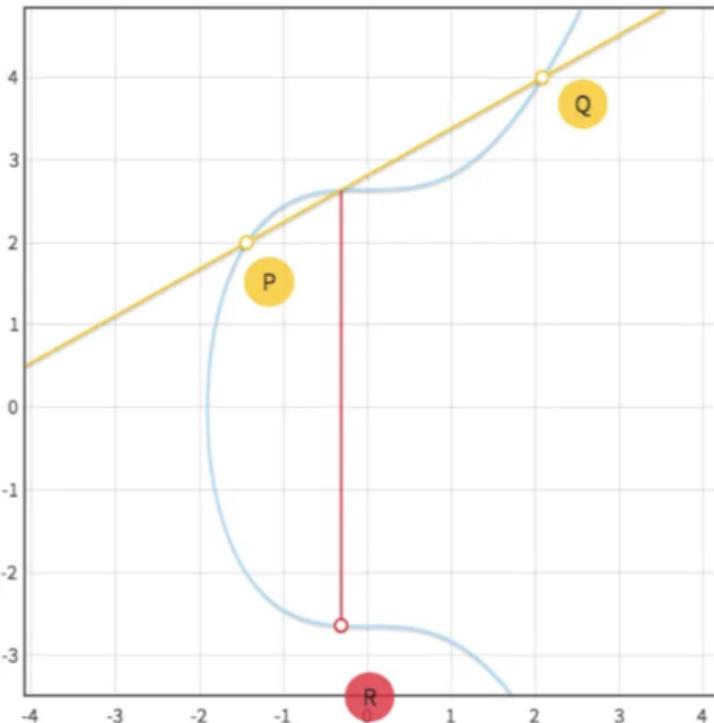


# Cryptography Elements used in Blockchains – ECDSA over Finite Field

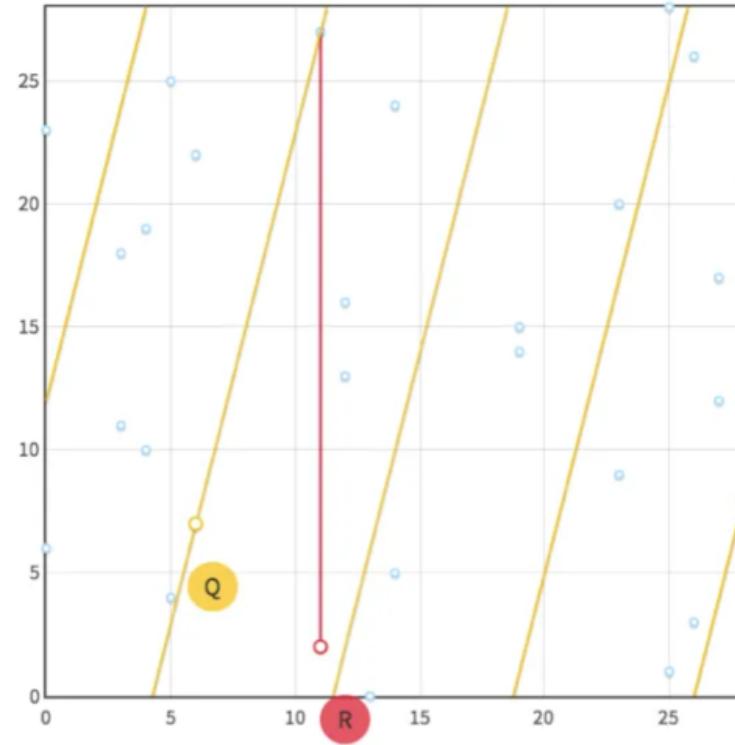
## A Short Look at Finite Fields

Computers like to work with whole numbers, and doing point addition and multiplication leads to not whole numbers rather quickly. To solve this, ECDSA puts elliptic curves onto finite fields. This basically transforms the curve into a series of points that are made of whole numbers. The mechanics of point addition and multiplication stay the same.

'Normal'  $P + Q = R$



'Finite Field'  $P + Q = R$



we need to add "mod p" at the end of every expression. Therefore, given  $P = (x_P, y_P)$ ,  $Q = (x_Q, y_Q)$  and  $R = (x_R, y_R)$ , we can calculate  $P + Q = -R$  as follows:

$$\begin{aligned}x_R &= (m^2 - x_P - x_Q) \bmod p \\y_R &= [y_P + m(x_R - x_P)] \bmod p \\&= [y_Q + m(x_R - x_Q)] \bmod p\end{aligned}$$

If  $P \neq Q$ , the slope  $m$  assumes the form:

$$m = (y_P - y_Q)(x_P - x_Q)^{-1} \bmod p$$

Else, if  $P = Q$ , we have:

$$m = (3x_P^2 + a)(2y_P)^{-1} \bmod p$$

For  $P=(5|36)$ ,  $a=0$ ,  $b=7$ , calculate  $2*P = P + P$

1. Online Visual:

<https://andrea.corbellini.name/ecc/interactive/modk-add.html>  
<https://andrea.corbellini.name/ecc/interactive/modk-mul.html>

2. Another Online Visualization:

<https://graui.de/code/elliptic2/> ( $a=0$ ,  $b=7$ ,  $r=97$ )

3. JCrypTool (Visuals-> Elliptic Curves Calculation)

$a=0$ ,  $b=7$ ,  $p=97 \Rightarrow 79$  points

<https://www.cryptool.org/en/jct/>

JCrypTool Local WebServer:

<http://127.0.0.1:59478/help/index.jsp?topic=%2Forg.jcryptool.visual.ecc%2Fhelp%2Fcontent%2Findex.html>

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

<https://andrea.corbellini.name/ecc/interactive/modk-add.html>

Elliptic Curve point addition ( $\mathbb{F}_p$ )

R ADDITION MULTIPLICATION  $\mathbb{F}_p$  ADDITION MULTIPLICATION

Curve:  $a$  0  $b$  7  
Field:  $p$  97  
 $P$ :  $x$  5  $y$  36  
 $Q$ :  $x$  5  $y$  36  
 $R = P + Q$ :  $x$  96  $y$  43

Point addition over the elliptic curve  $y^2 = x^3 + 7$  in  $\mathbb{F}_{97}$ .

The curve has 79 points (including the point at infinity).

R

$$\begin{aligned}x_R &= (m^2 - 2*x_p) \bmod 97 \\&= (94^2 - 10) \bmod 97 \\&= 8826 \bmod 97 = 96\end{aligned}$$

$$- y_R = y_p + m*(x_R - x_p) \bmod 97 = 36 + 94 * (96 - 5) \bmod 97 = 8590 \bmod 97 = 54$$

$$y_R = 97 - 54 = 43$$

$$(y_R = -y_p - m*(x_R - x_p) \bmod 97 = -36 - 94*(96-5) \bmod 97 = -(36 + 94*91) \bmod 97 = -54 \bmod 97 = 43)$$

we need to add "mod p" at the end of every expression. Therefore, given  $P = (x_P, y_P)$ ,  $Q = (x_Q, y_Q)$  and  $R = (x_R, y_R)$ , we can calculate  $P + Q = -R$  as follows:

$$\begin{aligned}x_R &= (m^2 - x_P - x_Q) \bmod p \\y_R &= [y_P + m(x_R - x_P)] \bmod p \\&= [y_Q + m(x_R - x_Q)] \bmod p\end{aligned}$$

If  $P \neq Q$ , the slope  $m$  assumes the form:

$$m = (y_P - y_Q)(x_P - x_Q)^{-1} \bmod p$$

Else, if  $P = Q$ , we have:

$$m = (3x_P^2 + a)(2y_P)^{-1} \bmod p$$

$$\begin{aligned}m &= (3*5^2 + 0)*(2*36)^{-1} \bmod 97 \\&= 75 * (72)^{-1} \bmod 97 \\&= 75 * 31 \bmod 97 \\&= 2325 \bmod 97 = 94\end{aligned}$$

For  $(72)^{-1} \bmod 97$  - Modular Multiplicative Inverse:

<https://planetcalc.com/3311/>

For modulo:

<https://miniwebtool.com/modulo-calculator/>

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

## The Process of Signing Data

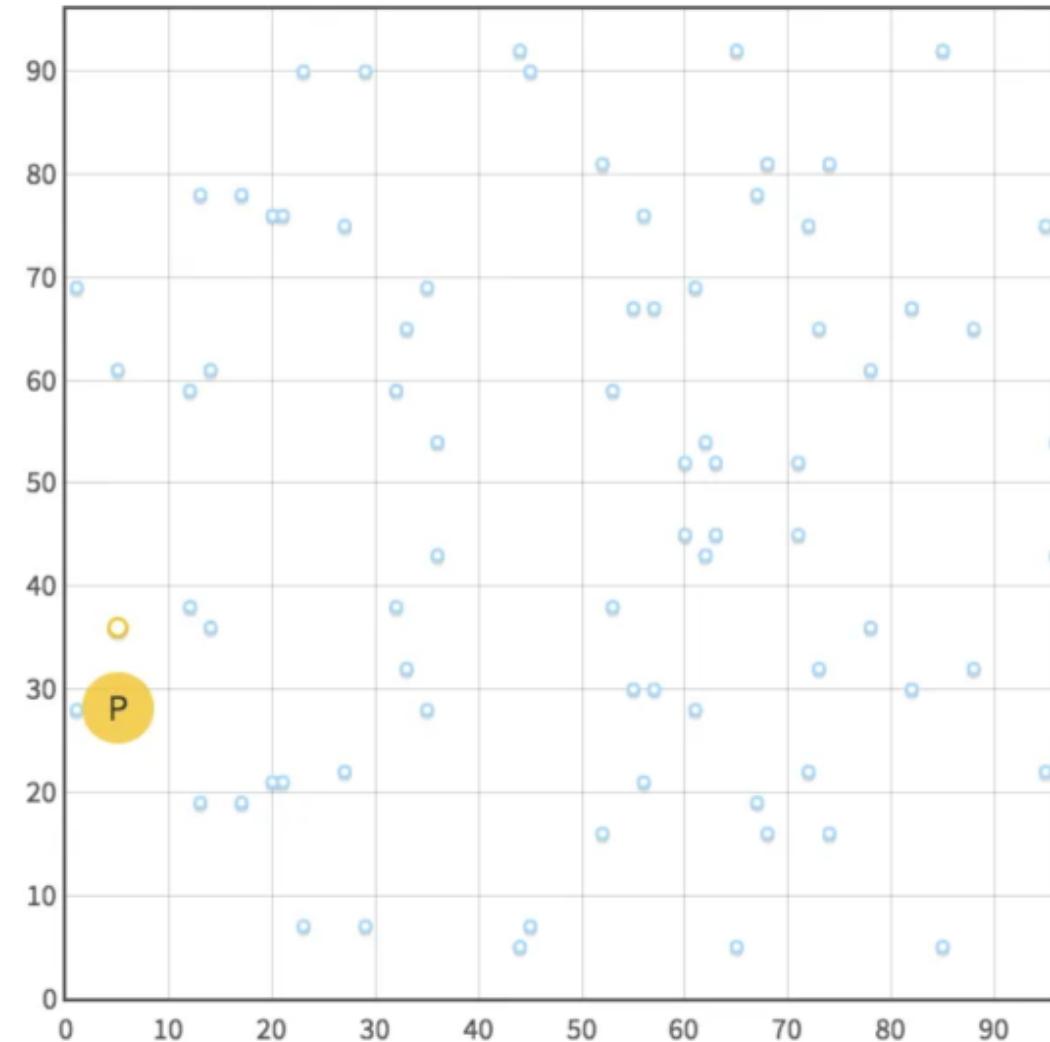
The basic process of signing data goes like this:

- Choose a private key
- Create a public key
- Create the data you want to sign
- ‘Sign’ the data by combining the data with your private key to find a point on an elliptic curve

Signature verification then compares your data and signature to your public key with a clever math trick. If they match, then consider yourself verified!

Actual Bitcoin ECDSA parameters are large (like size of the universe large), so we will use some smaller numbers:

- $a = 0$  &  $b = 7$  – the defining characteristic of the elliptic curve (These are actually the same as Bitcoin’s).
  - $P = 97$  – The Prime Modulus. This goes into deciding how many points are on the finite field
  - $N = 79$  – the number of points on the finite field
  - Base point ( $P$ ) =  $(5, 36)$  – The starting location for most of the math.
- For reference, the finite field for all of these calculations looks like this (with base point  $P$ ):



# Cryptography Elements used in Blockchains – ECDSA over Finite Field

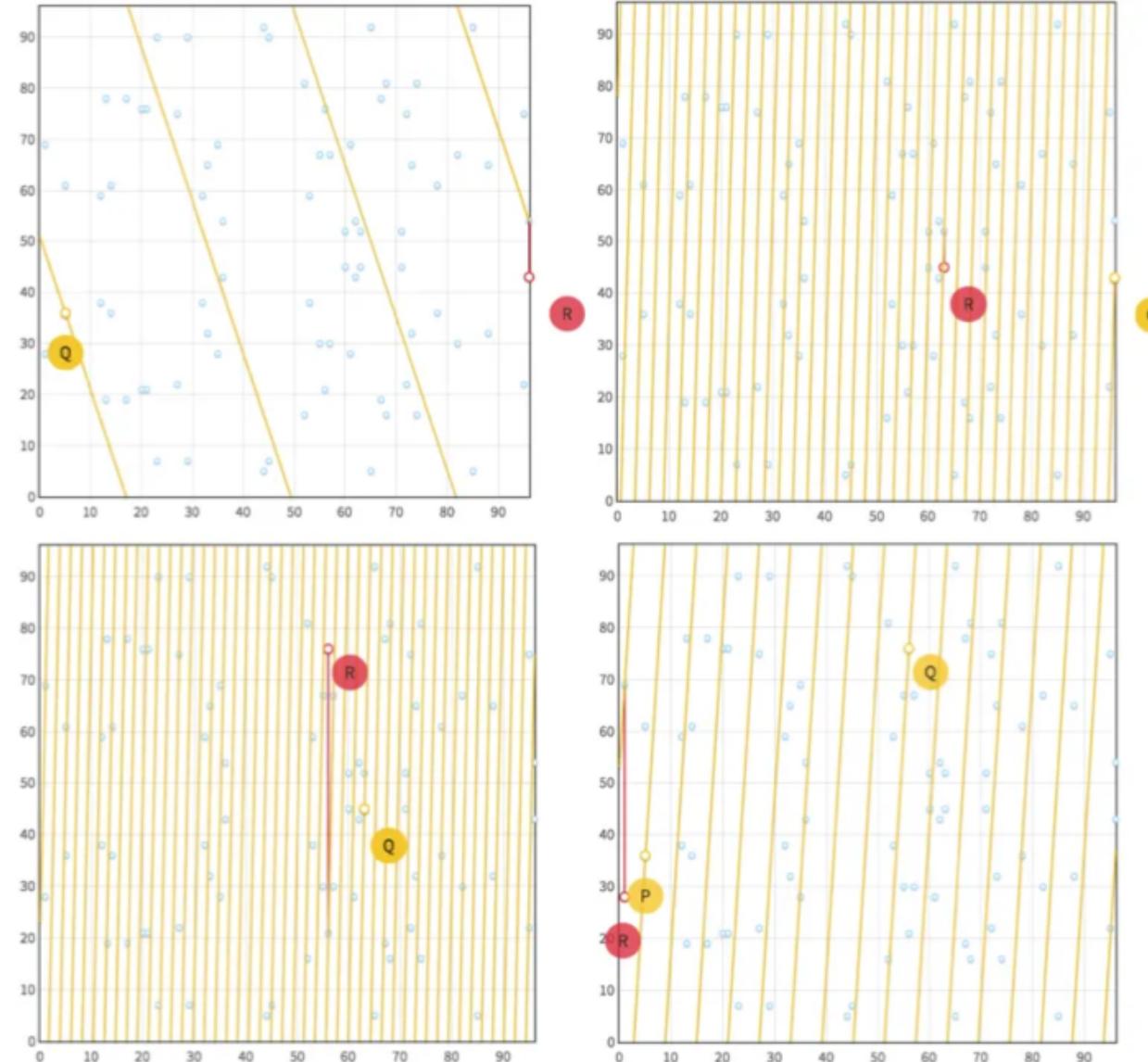
## Finding the Public Key

Let's use 9 as our private key since it worked well last time. The public key is found by multiplying the base point by the private key:

$$\text{Public} = 9 * \text{Base} \rightarrow$$

$$\text{Public} = \text{Base} + 2(2(2 * \text{Base}))$$

This is what the 3 point doublings and the single addition look like on a finite field:



So, doing the math:

$$\text{Public} = (5, 36) + 2(2(2 * (5, 36))) \rightarrow$$

$$\text{Public} = (5, 36) + 2(2 * (96, 43)) \rightarrow$$

$$\text{Public} = (5, 36) + 2 * (63, 45) \rightarrow$$

$$\text{Public} = (5, 36) + (56, 76) \rightarrow$$

$$\text{Public} = (1, 28)$$

Our public key is the point (1,28)

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

## Let the Signatures Begin!

Normally, you sign the hash of your data. As Bitcoin's hashes are large numbers (see [SHA-256](#)), we're just going to sign the **number 2 (hash value of the message)** instead.

Like the public key, the signed data is represented by another point,  $(r, s)$ . The first step in signing data, believe it or not, is to pick a random number - Lets use  **$k = 4$** .

**$k$**  is treated just like a private key and a point is found by multiplying the base point by  **$k$** :

$$(x, y) = k * \text{Base} \rightarrow$$

$$(x, y) = 4 * (5, 36) \rightarrow$$

$$(x, y) = (63, 45)$$

Now, using  $(x, y)$ ,  **$n$  (number of points)**,  **$k$** , **our data (hash value 2)**, and **the private key** we can find  $(r, s)$ , the signature.

**$r$**  is easy:

$$r = x \bmod n \rightarrow$$

$$r = 63 \bmod 79 \rightarrow$$

$$r = 63$$

**$s$**  requires a bit more math:

$$s = k^{-1} * (\text{data} + r * \text{Private}) \bmod n \rightarrow$$

$$s = 4^{-1} * (2 + 63 * 9) \bmod 79 \rightarrow$$

Inverses are a little weird in modular math. A number's inverse needs to follow this:

$$\text{Number's Inverse} * \text{Number} \bmod n = 1$$

So the inverse of 4 would be 20, as  $4 * 20 = 80$  and  $80 \bmod 79 = 1$ .

$$s = 20 * (569) \bmod 79 \rightarrow$$

$$s = 11380 \bmod 79 \rightarrow$$

$$s = 4$$

(If either  $r$  or  $s$  were to equal 0, then we'd need to pick another  $k$ )

The signature for our data, the **hash value 2**, is  **$(63, 4)$** . Now you send it to someone and they want to verify that it came from you.

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

## Signature Verification Without the Private Key

To verify that the signature, a new point is calculated  $(x_2, y_2)$  using the **Base Point**, **your Public Key**, **the Data**, and **the Signature**. **The signature is confirmed if r is equal to  $x_2$ . But why exactly does this work?**

Because  $x_2$  and  $r$  are the same if the public key used to find  $x_2$  belongs to the private key used to find  $r$ . Here's how:

This is the equation used to find  $(x_2, y_2)$ :

$$(x_2, y_2) = u_1 * \text{Base} + u_2 * \text{Public}$$

$u_1$  and  $u_2$  are found with these:

$$u_1 = \text{data} * s^{-1} \bmod n$$

$$u_2 = r * s^{-1} \bmod n$$

We know that the public key is just the base point times the private key...

$$(x_2, y_2) = u_1 * \text{Base} + u_2 * (\text{Private} * \text{Base})$$

and we can expand  $u_1$  and  $u_2$ ....

$$(x_2, y_2) = (\text{data} * s^{-1}) * \text{Base} + (r * s^{-1}) * \text{Private} * \text{Base} \bmod n$$

collecting like terms....

$$(x_2, y_2) = (\text{data} + r * \text{Private}) * s^{-1} * \text{Base} \bmod n$$

and replacing  $s$  with its definition above.

$$(x_2, y_2) = (\text{data} + r * \text{Private}) * (k^{-1})^{-1} * (\text{data} + r * \text{Private})^{-1} * \text{Base} \bmod n$$

The inverses cancel out and we are left this:

$$(x_2, y_2) = k * \text{Base} \bmod n$$

Which is the same equation used to find  $r$  in the first place:

$$(x, y) = k * \text{Base} \bmod n \rightarrow r = x$$

Which is a pretty neat and a clever trick.

# Cryptography Elements used in Blockchains – ECDSA over Finite Field

## Signature Verification Without the Private Key

To verify that the signature, a new point is calculated ( $x_2, y_2$ ) using the **Base Point**, **your Public Key**, **the Data**, and **the Signature**. **The signature is confirmed if r is equal to  $x_2$ . But why exactly does this work?**

Because  $x_2$  and  $r$  are the same if the public key used to find  $x_2$  belongs to the private key used to find  $r$ . Here's how:

This is the equation used to find ( $x_2, y_2$ ):

$$(x_2, y_2) = u_1 * \text{Base} + u_2 * \text{Public}$$

$u_1$  and  $u_2$  are found with these:

$$u_1 = \text{data} * s^{-1} \bmod n$$

$$u_2 = r * s^{-1} \bmod n$$

We know that the public key is just the base point times the private key...

$$(x_2, y_2) = u_1 * \text{Base} + u_2 * (\text{Public})$$

$$(x_2, y_2) = (\text{data} * s^{-1}) * \text{Base} + (r * s^{-1}) * (\text{Public}) \bmod n$$

Proceeding with calculating the example:

$$u_1 = \text{data} * s^{-1} \bmod n \rightarrow$$

$$u_1 = 2 * 4^{-1} \bmod 79 \rightarrow$$

$$u_1 = 40$$

$$u_2 = r * s^{-1} \bmod n \rightarrow$$

$$u_2 = 63 * 4^{-1} \bmod 79 \rightarrow$$

$$u_2 = 75$$

$$(x_2, y_2) = u_1 * \text{Base} + u_2 * \text{Public} \rightarrow$$

$$(x_2, y_2) = 40 * (5,36) + 75 * (1,28) \rightarrow$$

$$(x_2, y_2) = (82,30) + (67,78) \rightarrow$$

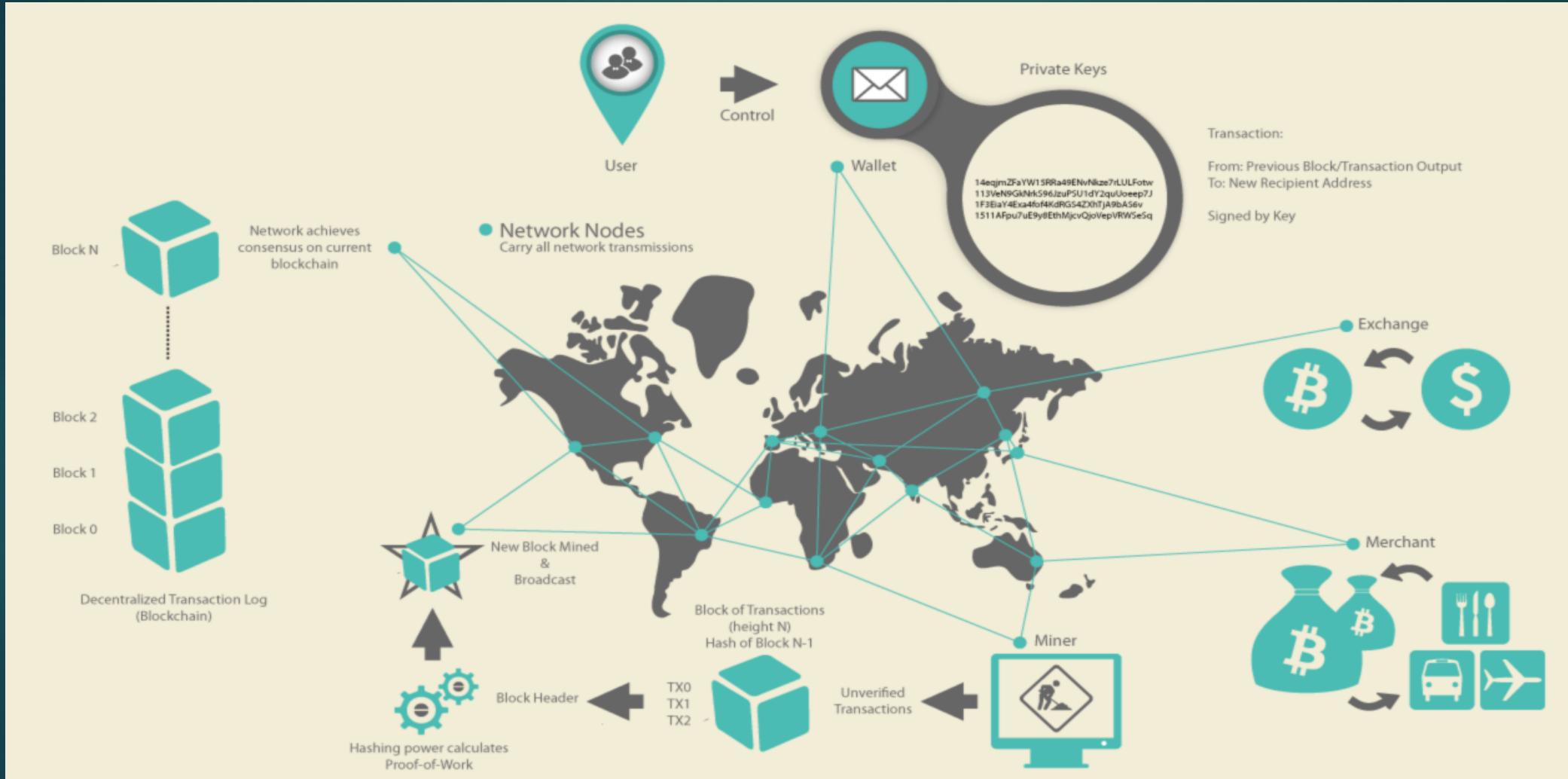
$$(x_2, y_2) = (63,45)$$

$x_2 = 63$ , and  $r = 63$ .

**Consider our signature verified.**

# 02. Bitcoin | Ethereum Blockchain Technology

36



Confidential – [www.ism.ase.ro](http://www.ism.ase.ro) | [www.acs.ase.ro](http://www.acs.ase.ro)

# 02. Bitcoin | Ethereum Blockchain Technology

37

## Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)



1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA  
- (Unspent) 0.015 BTC  
1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK -  
(Unspent) 0.0845 BTC

97 Confirmations

0.0995 BTC

### Summary

Size 258 (bytes)

Received Time 2013-12-27 23:03:05

Included In 277316 (2013-12-27 23:11:54 +9  
Blocks minutes)

### Inputs and Outputs

Total Input 0.1 BTC

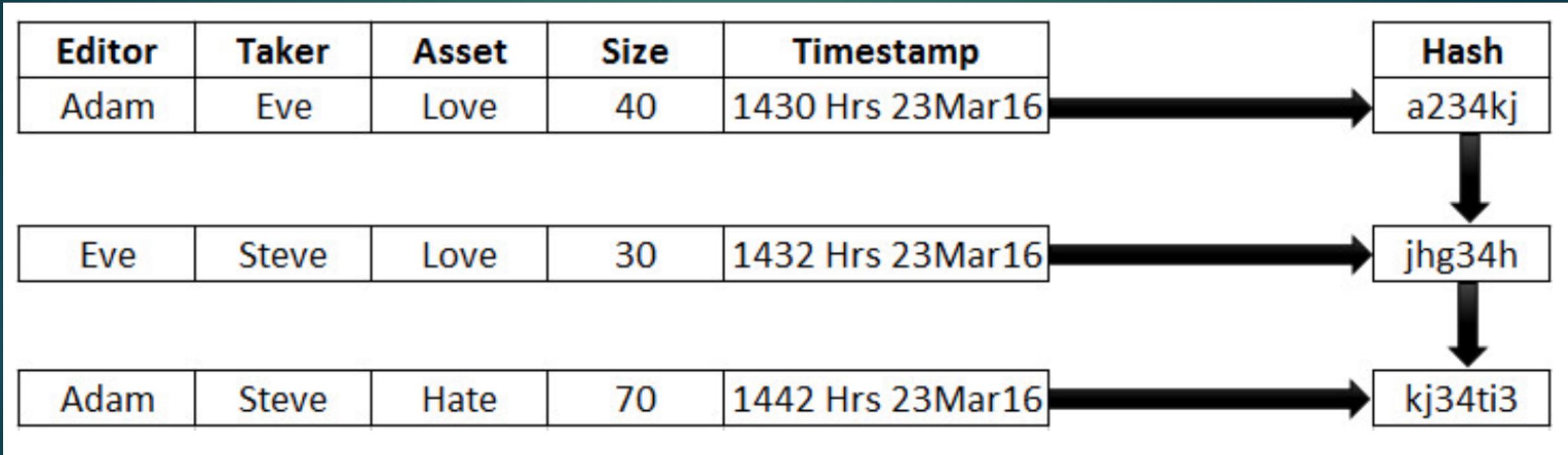
Total Output 0.0995 BTC

Fees 0.0005 BTC

Estimated BTC Transacted 0.015 BTC

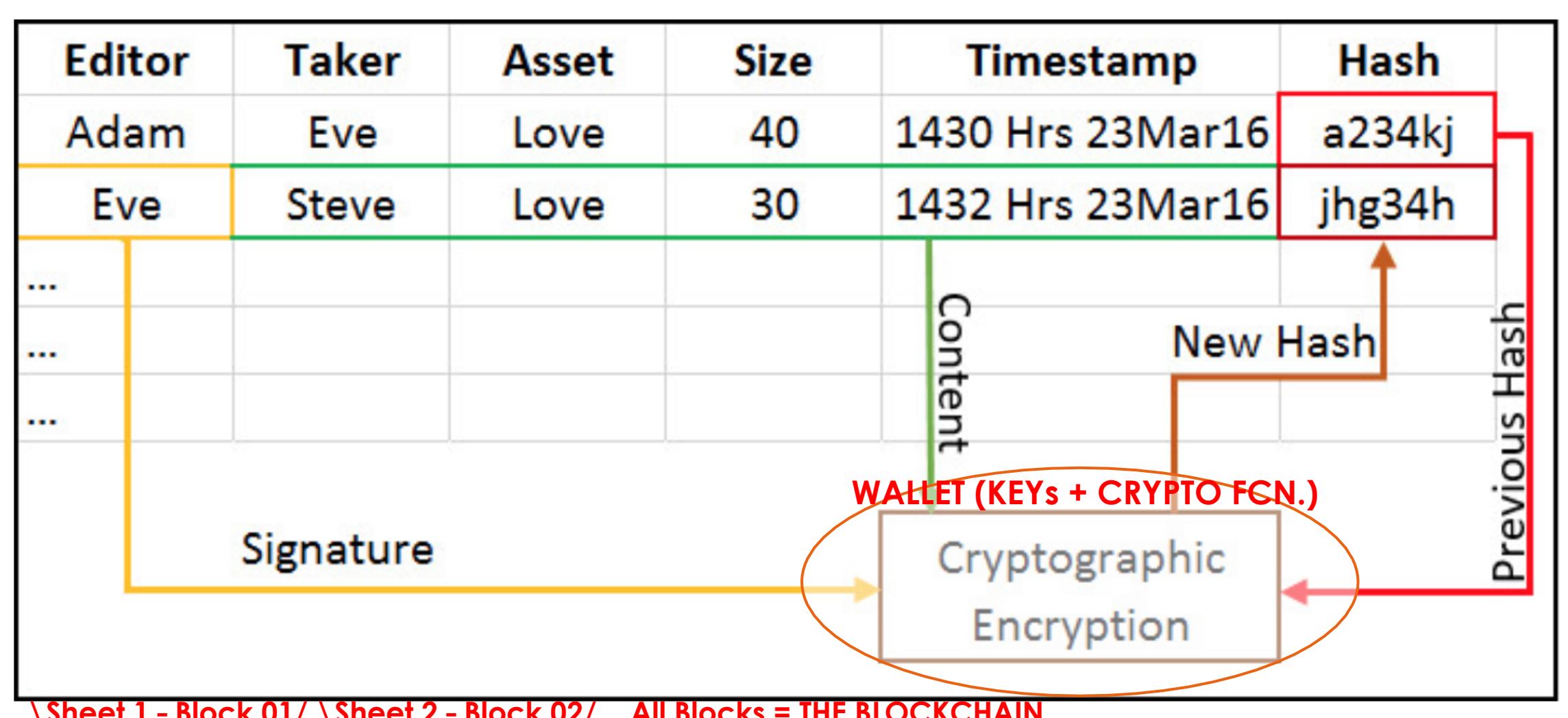
## 02. Bitcoin / Ethereum Blockchain Technology

Bitcoin / Ethereum != Blockchain; // They use blockchain technology



## 02. Bitcoin / Ethereum Blockchain Technology

((Bitcoin / Ethereum / crypto currencies) != Blockchain); // They use blockchain technology



\Sheet 1 - Block 01/ \Sheet 2 - Block 02/ ... All Blocks = THE BLOCKCHAIN

<https://www.linkedin.com/pulse/introduction-blockchain-bankers-excel-sheet-analogy-jayaram-firm>

# 02. Google Sheet DEMO

// <https://docs.google.com/spreadsheets/>

// Create Blank Sheet and Tools->Script Editor

//=getSha1(CONCATENATE(A4,B4,C4,D4,E4))

```
function getSha1(input) {
  var rawHash = Utilities.computeDigest(Utilities.DigestAlgorithm.SHA_1, input);
  var txtHash = "";

  for (var j = 0; j < rawHash.length; j++) {
    var hashVal = rawHash[j];

    if (hashVal < 0)
      hashVal += 256;

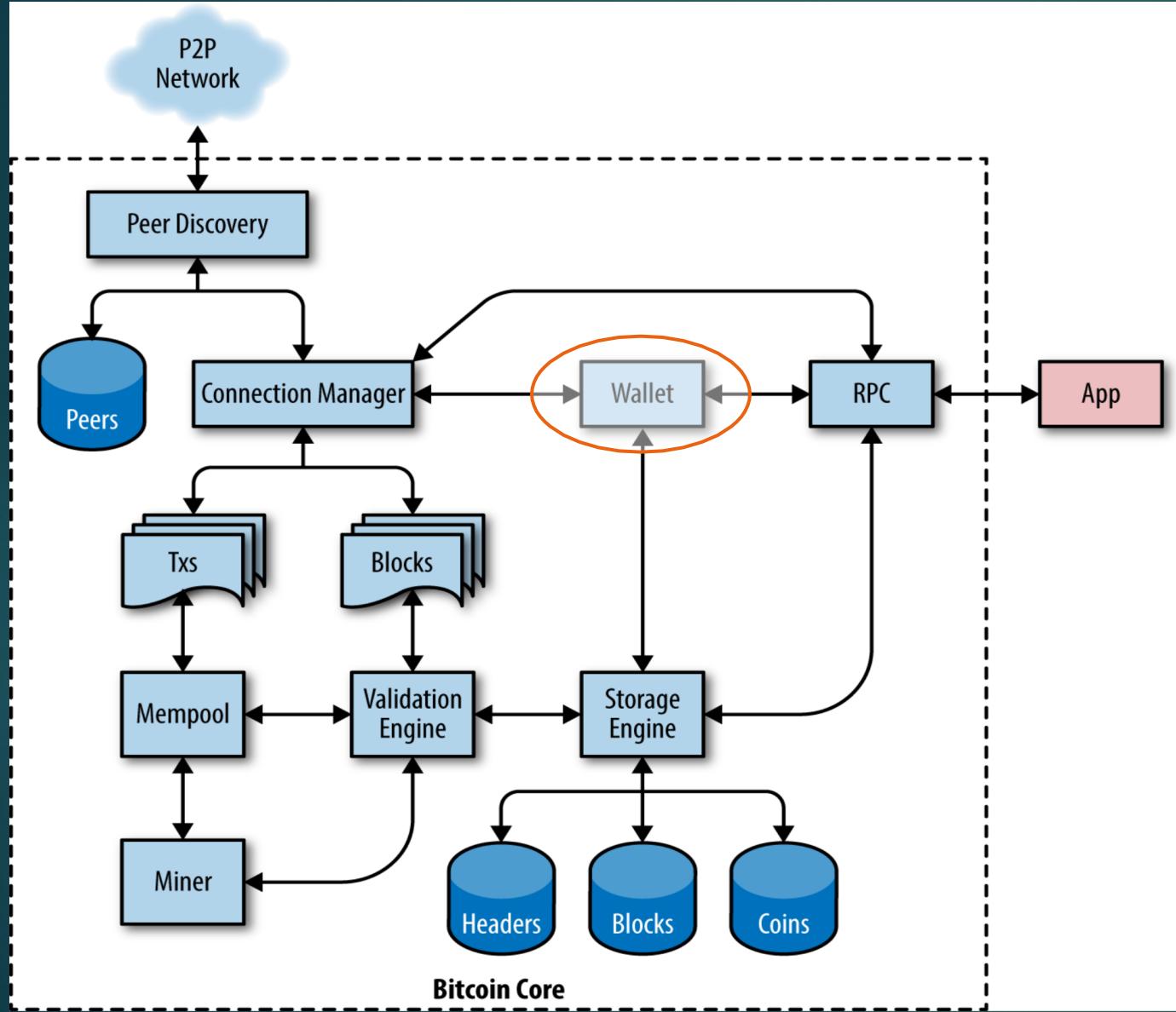
    if (hashVal.toString(16).length == 1)
      txtHash += "0";

    txtHash += hashVal.toString(16);
  }
  return txtHash;
}
```

<https://docs.google.com/spreadsheets/d/1jSLoPvsW6-L2BcTsOW9tULBop0thN1J05Fdt28QFWcA/edit?usp=sharing>

No	WalletSRC	WalletDST	DateTime	Amount	SHA-1 Row	SHA-1 Final
1	W01	W02	20210216110607	2	a6c168d7f2af54e611Bea10fcbe5985da6db95c8	
2	W02	W04	20210216110607	3	3b153c1724f1535deca2aa917613bf16240bed	febdd3e72d82f7ac0bad9bb02e776228eabff1s9
3	W02	W05	20210216110908	1.5	44741692955a56f455620f11042e05b4bcab0100	f11204dd7aaef93717976409b9c7e2c04ca83d

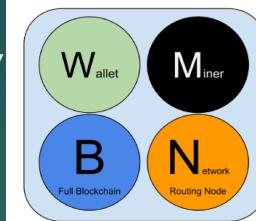
# 02 Bitcoin | Ethereum Blockchain Technology



Copyright: Andreas M. Antonopoulos. "Mastering Bitcoin, 2nd Edition", Pub

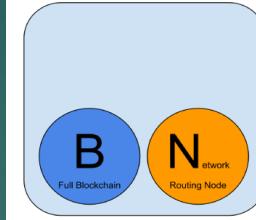
## Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



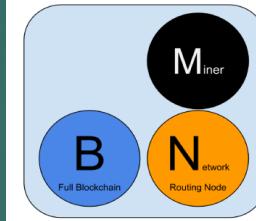
## Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



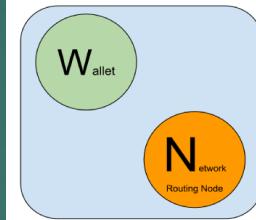
## Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



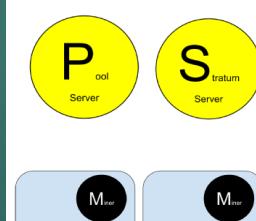
## Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



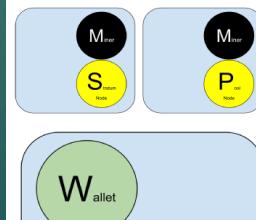
## Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



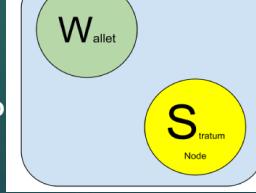
## Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



## Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.



## 02. Blockchain P2P Network – Consensus Algorithm

42

### Proof of Work

VS

### Proof of Stake



Mining capacity depends on computational power



Validating capacity depends on the stake in the network



Miners receive block rewards to solve a cryptographic puzzle



Validators do not receive a block reward, instead, they collect transaction fees as reward



Hackers would need to have a computer powerful than 51% of the network to add a malicious block, leading to 51% attack

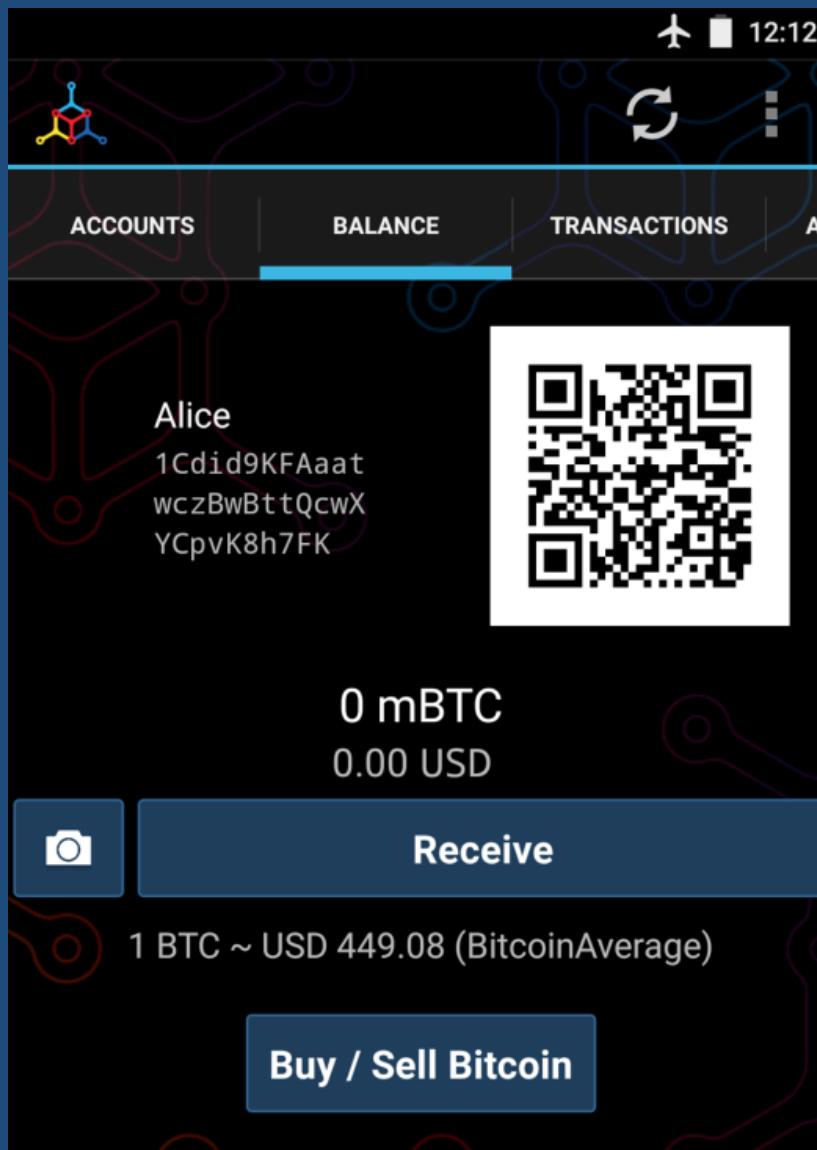


Hacker would need to own 51% of all the cryptocurrency on the network, which is practically impossible and therefore, making 51% attacks impossible.

The **proof of work (POW)** is a **common consensus algorithm** used by the most popular cryptocurrency networks like bitcoin and lite-coin. It requires a participant node to prove that the work done and submitted by them qualifies them to receive the right to add new transactions to the blockchain.

# 02 Bitcoin | Ethereum Blockchain Technology – DEMO

## with iOS and Android Mycellium



## Transaction

View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

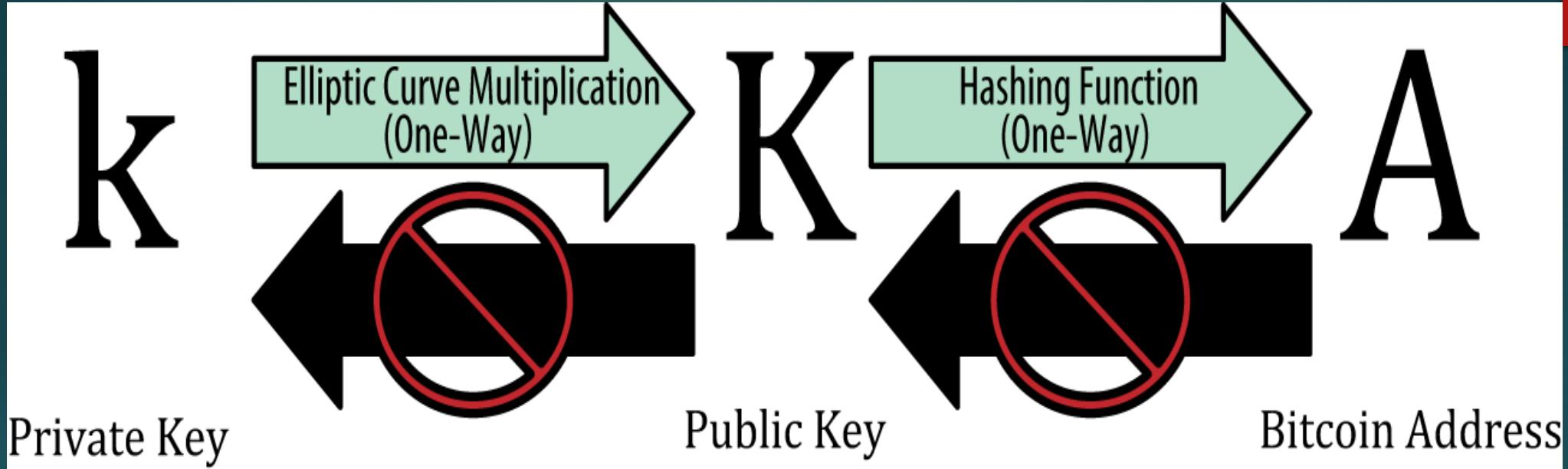
→ 1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA  
- (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK -  
(Unspent) 0.0845 BTC

97 Confirmations 0.0995 BTC

Summary		Inputs and Outputs	
Size	258 (bytes)	Total Input	0.1 BTC
Received Time	2013-12-27 23:03:05	Total Output	0.0995 BTC
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)	Fees	0.0005 BTC
		Estimated BTC Transacted	0.015 BTC

## 02 Bitcoin / Ethereum Blockchain Technology - Keys



**k = private key is simply a number, picked at random. (e.g. `bitcoin-cli getnewaddress`)**

```
$ bitcoin-cli getnewaddress  
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

```
$ bitcoin-cli dumpprivkey  
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy  
KxFc1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
```

## 02 Bitcoin / Ethereum Blockchain Technology - Keys

K = public key is calculated from the private key using elliptic curve multiplication, which is irreversible:  $K = k * G$ , where k is the private key, G is a constant point called the generator point, and K is the resulting public key.

Bitcoin/Ethereum uses a specific elliptic curve and set of mathematical constants, as defined in a standard called **secp256k1**, established by the National Institute of Standards and Technology (NIST). The **secp256k1** curve is defined by the following function, which produces an elliptic curve:

$$Y^2 \bmod P = (X^3 + 7) \bmod P$$

$$Y = \text{SQRT}(8) \Rightarrow Y = \pm 2.82$$

$$(X=1, Y = \pm 2.82) \text{ or } (X=3, Y = \pm 5.83)$$

$$y^2 = (x^3 + 7) \text{ over } (\mathbb{F}_p)$$

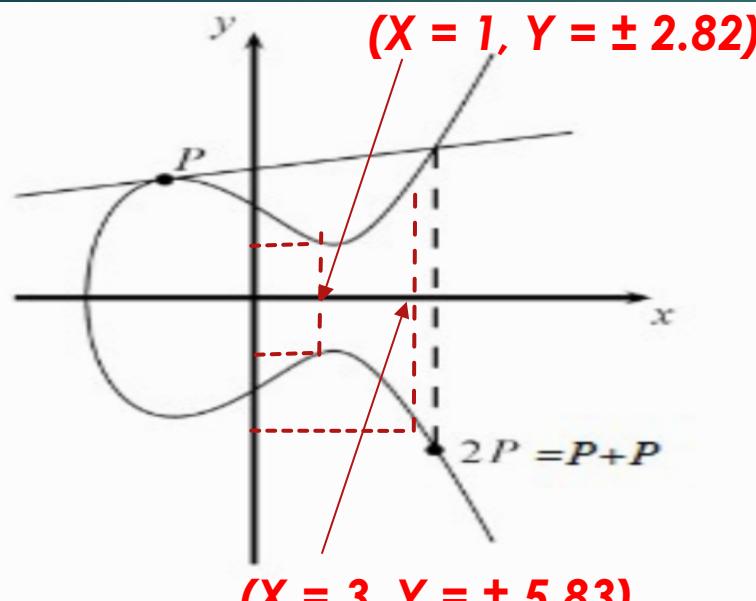
or

$$y^2 \bmod p = (x^3 + 7) \bmod p$$

The  $\bmod p$  (modulo prime number p) indicates that this curve is over a finite field of prime order  $p$ , also written as  $\mathbb{F}_P$ , where  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ , a very large prime number.

## 02 Bitcoin / Ethereum Blockchain Technology - Addresses & Keys

Most bitcoin/ETH implementations use the OpenSSL cryptographic library to do the elliptic curve math. For example, to derive the public key, the function EC\_POINT\_mul() is used.



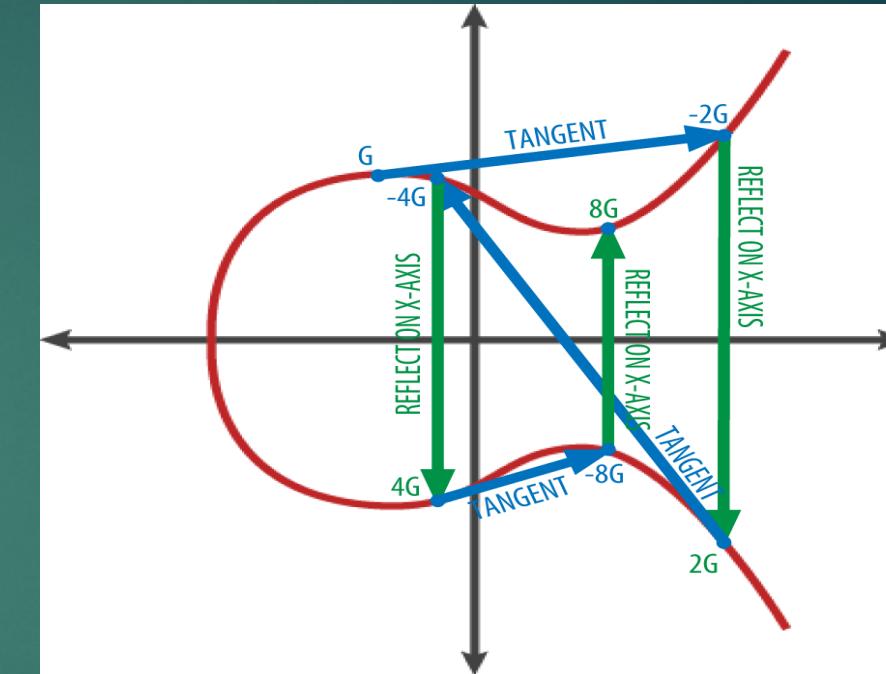
Graph representing point doubling over real numbers

In the case of point doubling, the equation becomes:

$$x_3 = s^2 - x_1 - x_2 \bmod p$$

$$y_3 = s(x_1 - x_3) - y_1 \bmod p$$

$$S = \frac{3x_1^2 + a}{2y_1}$$



The private key in this picture is 8, e.g. privKey = 8

G (compressed) = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9  
59F2815B 16F81798

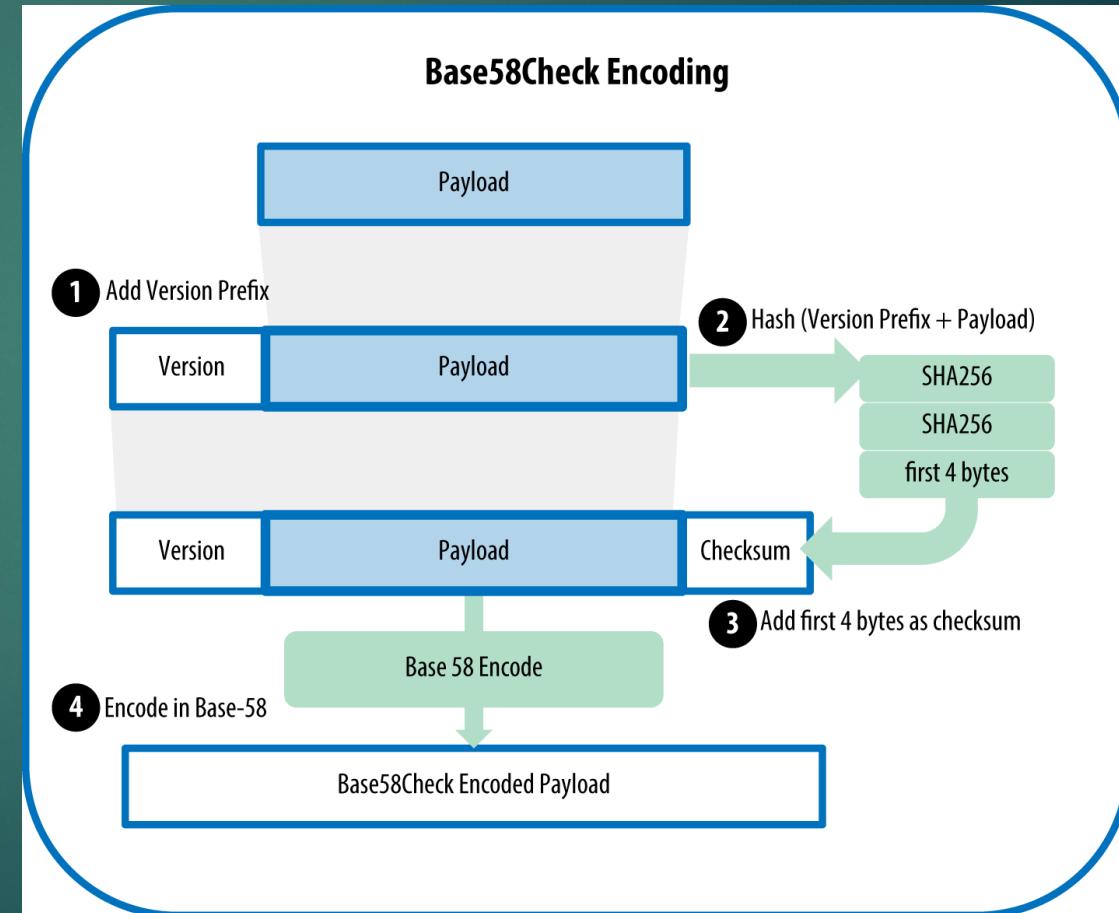
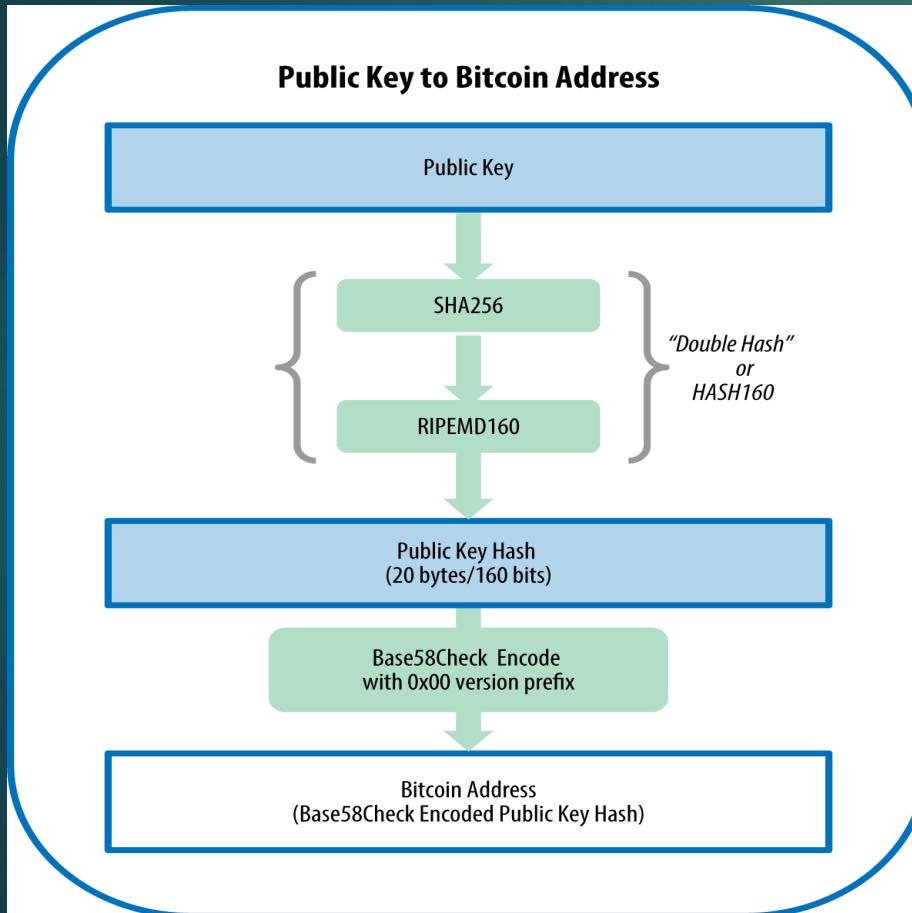
G (uncompressed) = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9  
59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419  
9C47D08F FB10D4B8

## 02 Bitcoin / Ethereum Blockchain Technology - Addresses & Keys

Starting with the public key  $K$ , we compute the SHA256 hash and then compute the RIPEMD160 hash of the result, producing a 160-bit (20-byte) number:

$$A = \text{RIPEMD160}(\text{SHA256}(K))$$

where  $K$  is the public key and  $A$  is the resulting bitcoin address.



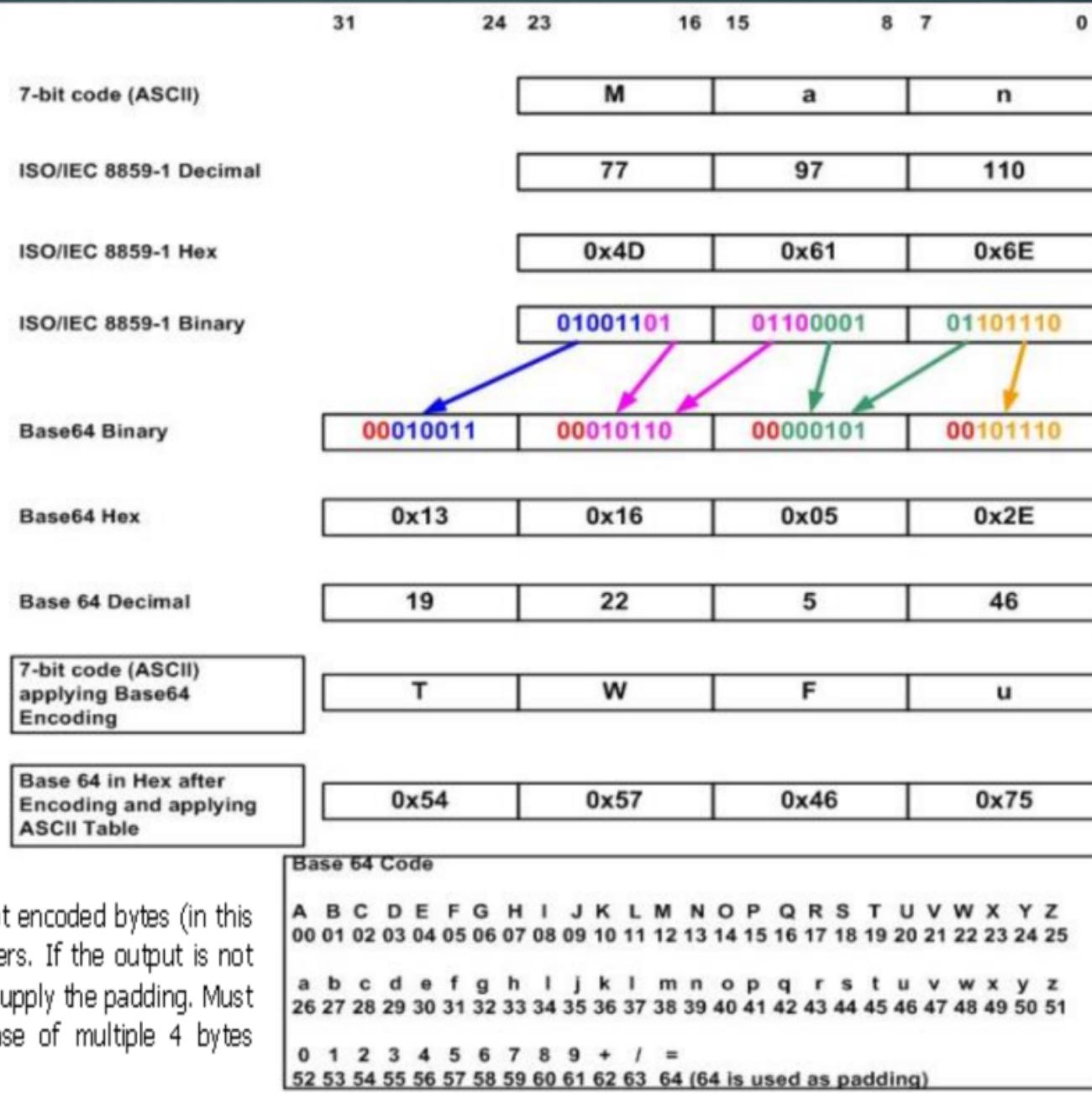
Public key to bitcoin address: conversion of a public key into a bitcoin address.

## 02 Bitcoin / Ethereum Blockchain Technology - Addresses & Keys

Base64 encoding is used in practice usually for transport over the network and heterogeneous environments binary code such as pictures or executable code. The techniques is very simple: to transform each 3 bytes values into 4 bytes value in order to avoid to obtain values greater then 127 per byte.

For instance, if the scope is to encode the word "Man" into Base64 encoding then it is encoded as "TWFu". Encoded in ASCII (in ISO 8859-1, one value per byte), M, a, n are stored as the bytes 77 (0x4D), 97 (0x61), 110 (0x6E), which are 01001101, 01100001, 01101110 in base 2.

As this example illustrates, the encoding converts 3 not encoded bytes (in this case, ASCII characters) into 4 encoded ASCII characters. If the output is not multiple of 4 bytes then the '=' sign is put in order to supply the padding. Must be considered the padding with = (64 value) in case of multiple 4 bytes number.



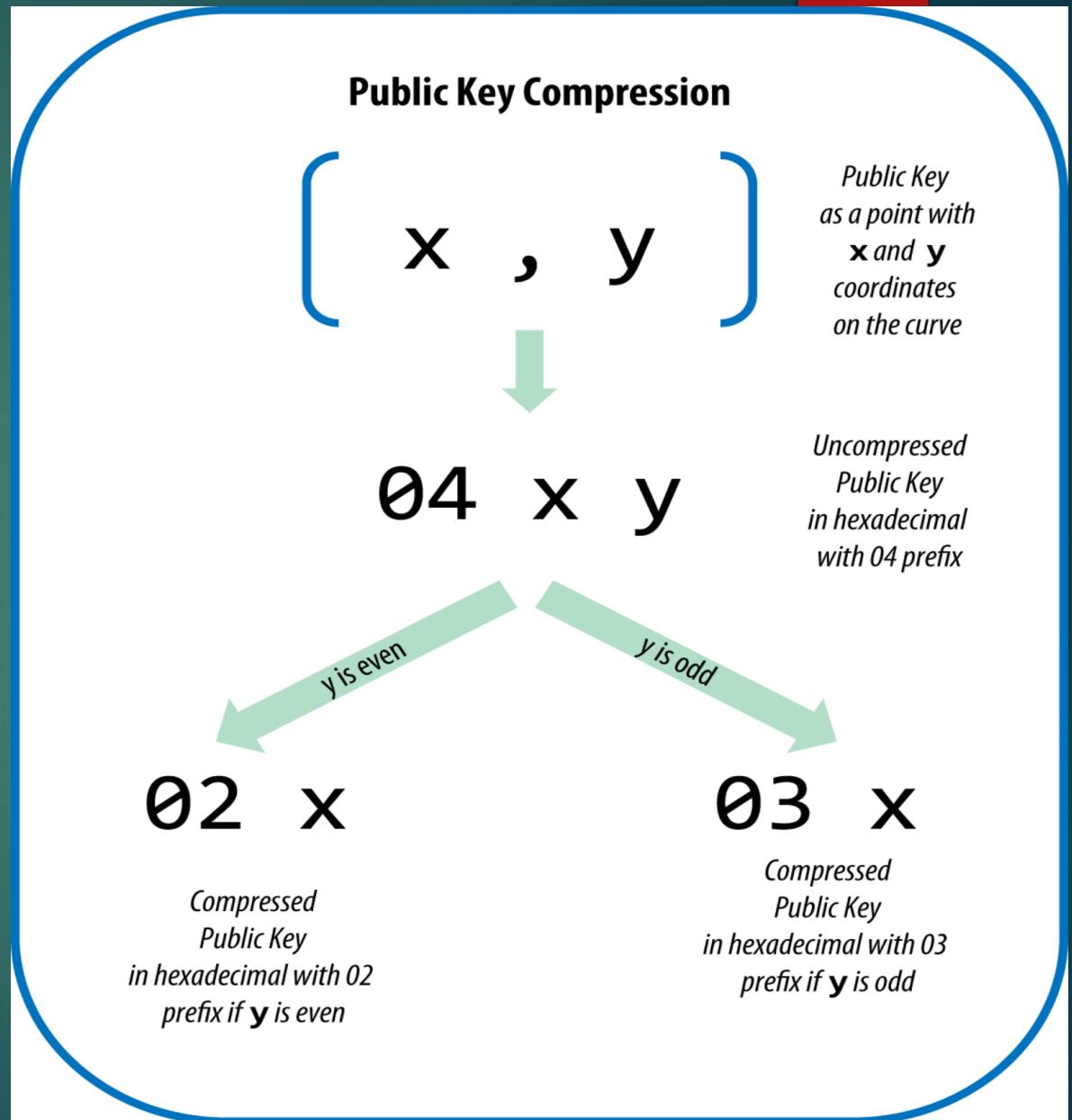
Base64  
to  
Base58  
Analogy

## 02 Bitcoin / Ethereum Blockchain Technology - Addresses & Keys

A public key is a point **(x,y)** on an elliptic curve. Because the curve expresses a mathematical function, a point on the curve represents a solution to the equation and, therefore, if we know the x coordinate we can calculate the y coordinate by solving the equation:

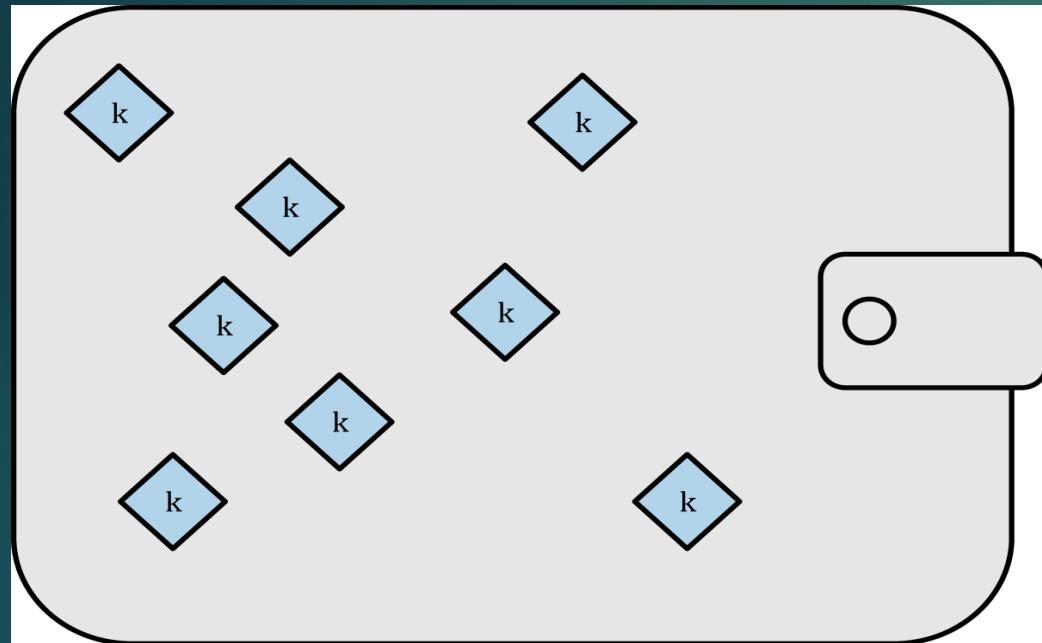
$$y^2 \bmod p = (x^3 + 7) \bmod p.$$

That allows us to store only the x coordinate of the public key point, omitting the y coordinate and reducing the size of the key and the space required to store it by 256 bits. An almost 50% reduction in size in every transaction adds up to a lot of data saved over time!



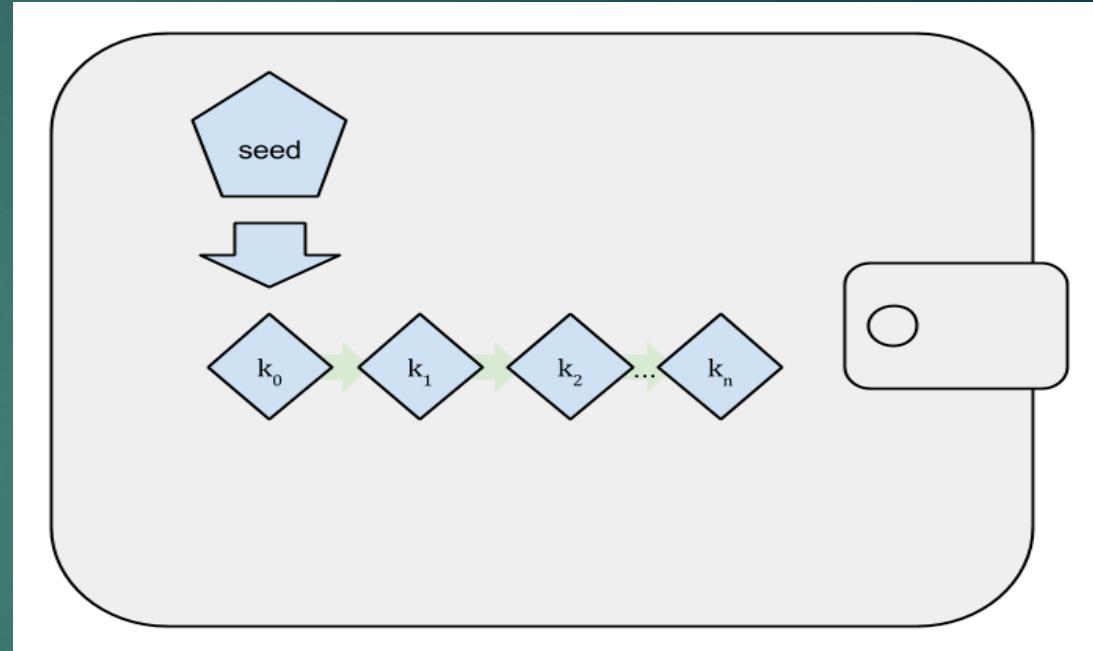
## 02 Bitcoin / Ethereum Blockchain Technology - Wallets

A common misconception about bitcoin is that bitcoin wallets contain bitcoin. In fact, the wallet contains only keys. The “coins” are recorded in the blockchain on the bitcoin network. Users control the coins on the network by signing transactions with the keys in their wallets. In a sense, a bitcoin wallet is a keychain.



The first type is a nondeterministic wallet, where each key is independently generated from a random number. The keys are not related to each other. This type of wallet is also known as a JBOK wallet from the phrase “Just a Bunch Of Keys.”

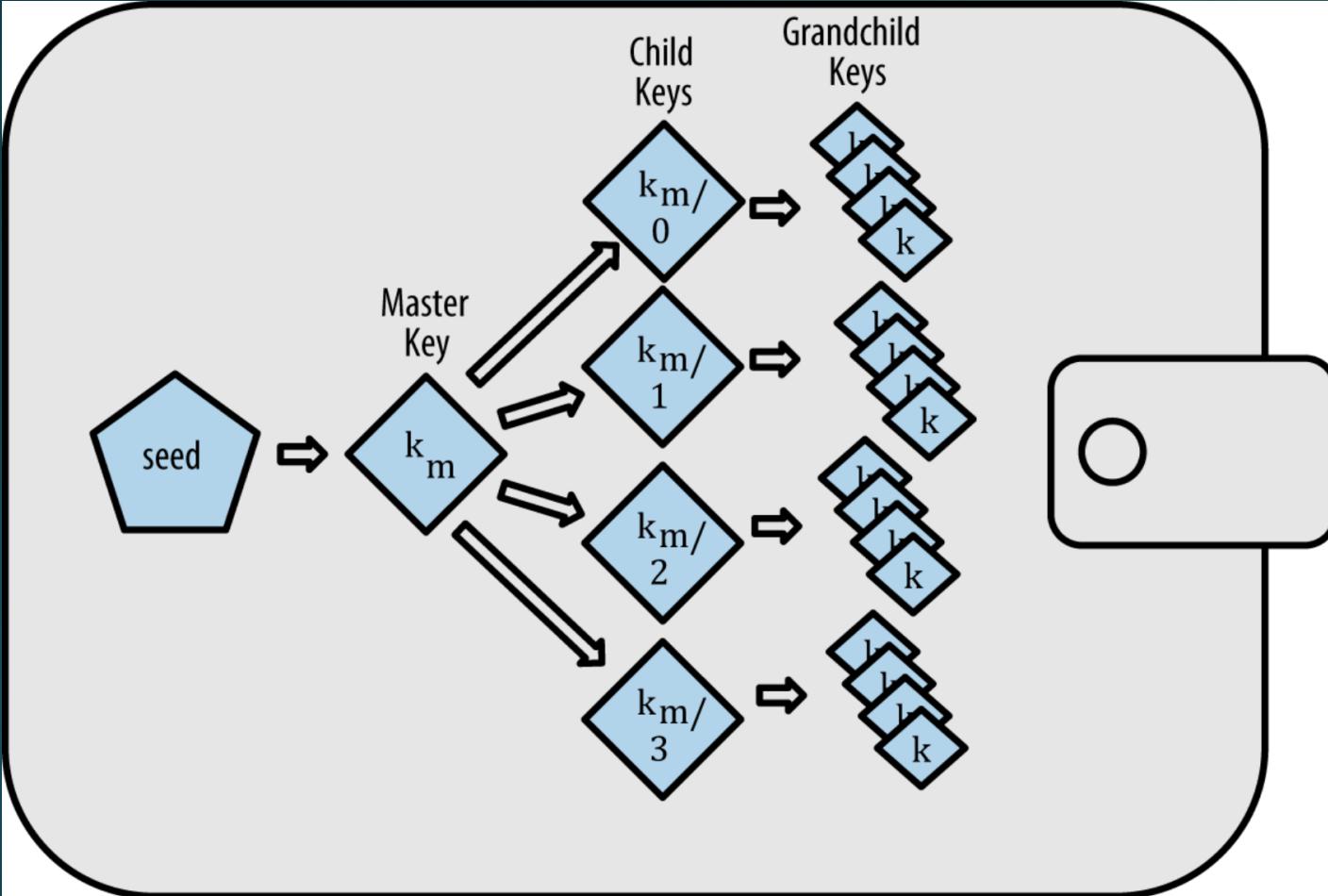
**Type-0 nondeterministic (random) wallet:** a collection of randomly generated keys



The second type of wallet is a deterministic wallet, where all the keys are derived from a single master key, known as the seed. All the keys in this type of wallet are related to each other and can be generated again if one has the original seed.

**Type-1 deterministic (seeded) wallet:** a deterministic sequence of keys derived from a seed

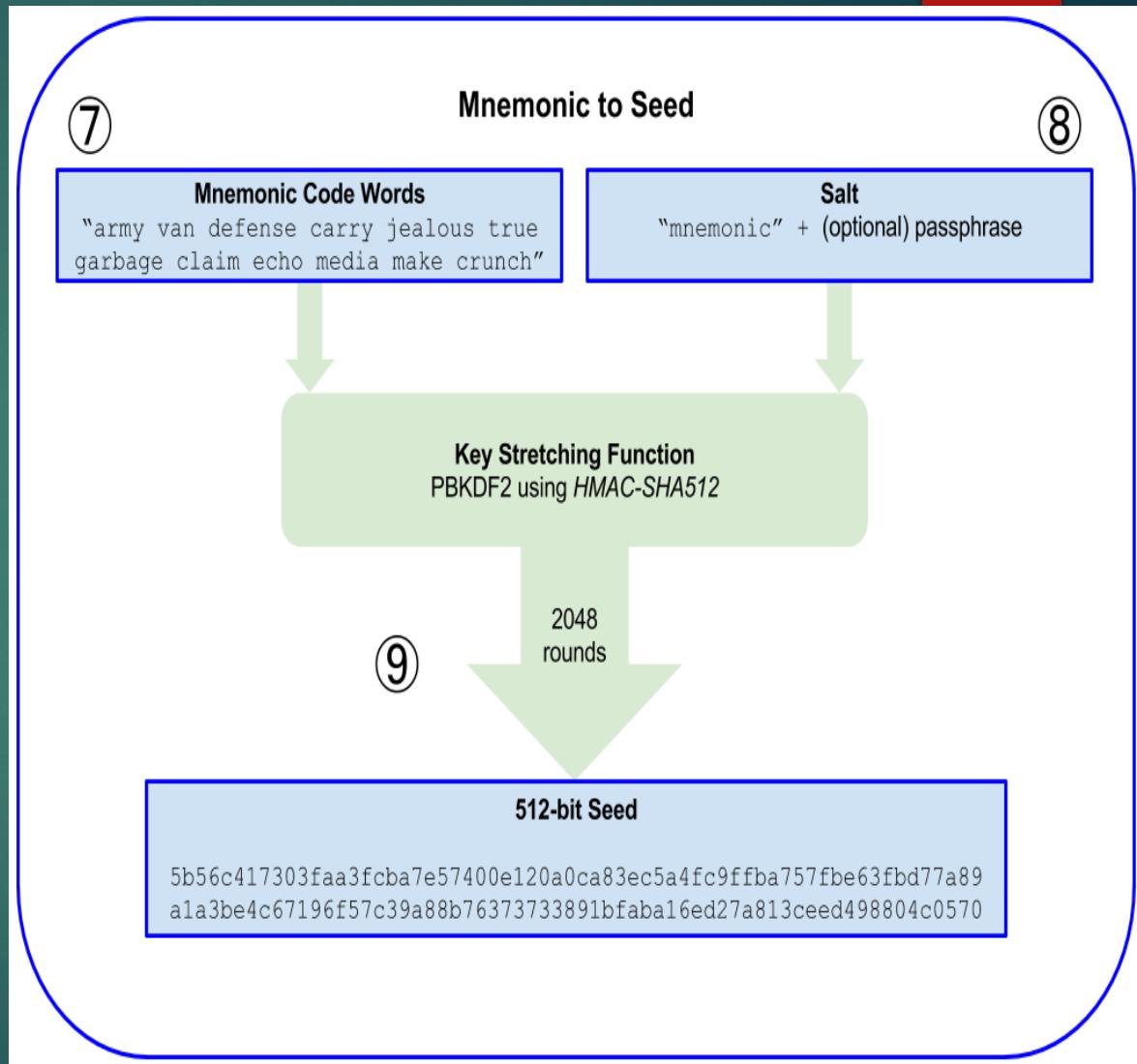
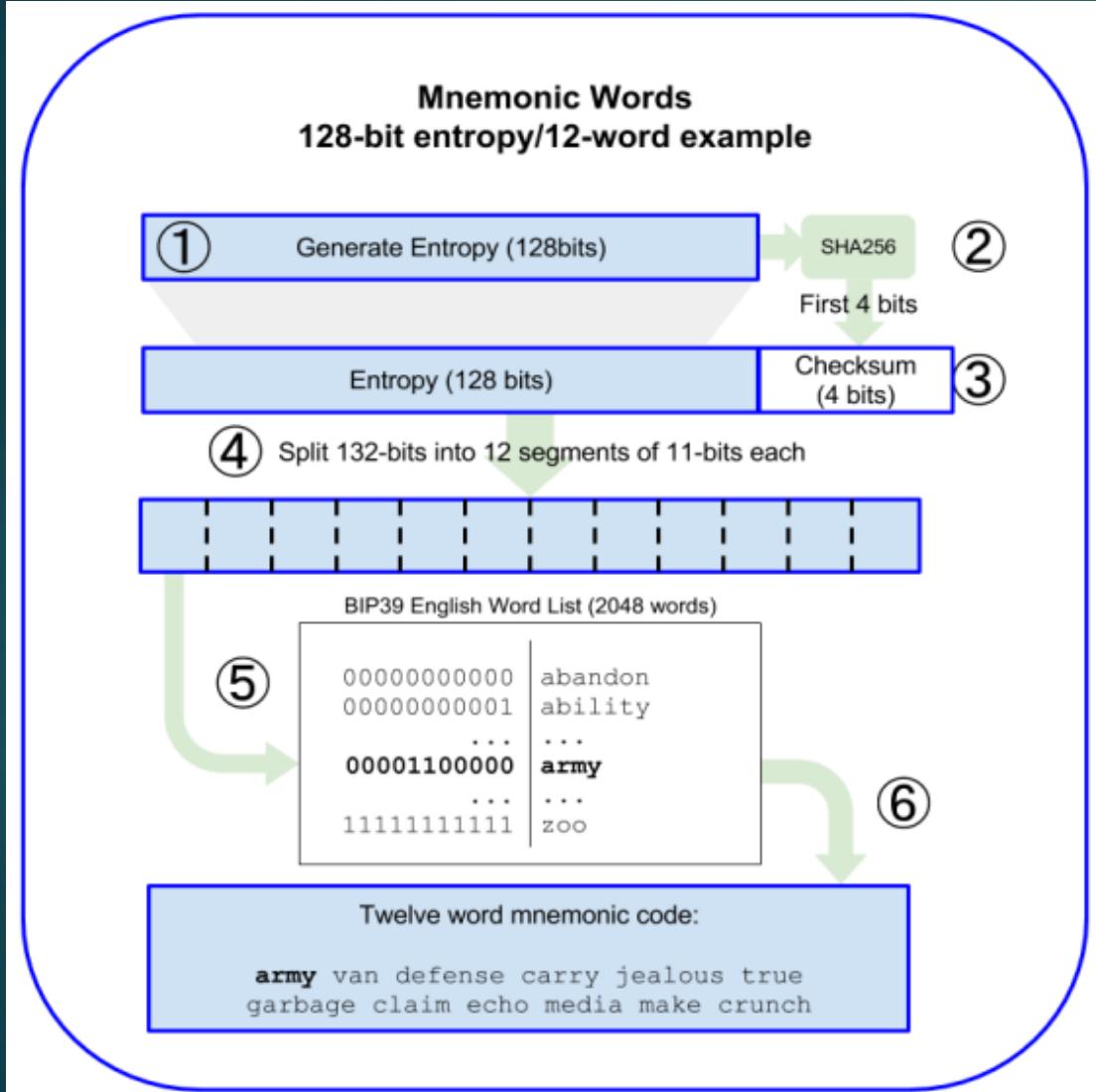
## 02 Bitcoin / Ethereum Blockchain Technology - Wallets



Type-2 HD (*Hierarchical Deterministic*) Wallet: a tree of keys generated from a single seed  
HD Wallets (Bitcoin Improvement Proposal: BIP-32/BIP-44)

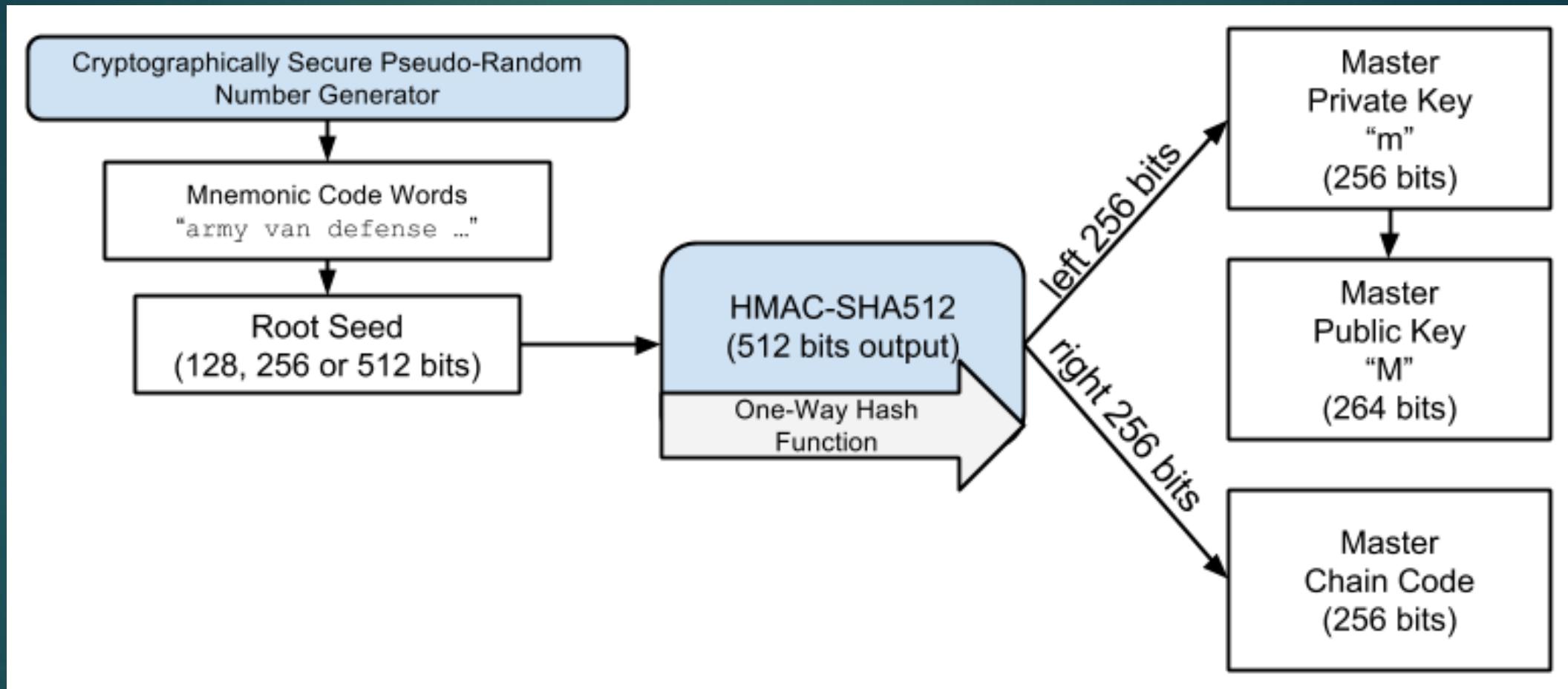
# 02 Bitcoin / Ethereum Blockchain Technology - HD Wallets

Generating entropy and encoding as mnemonic words



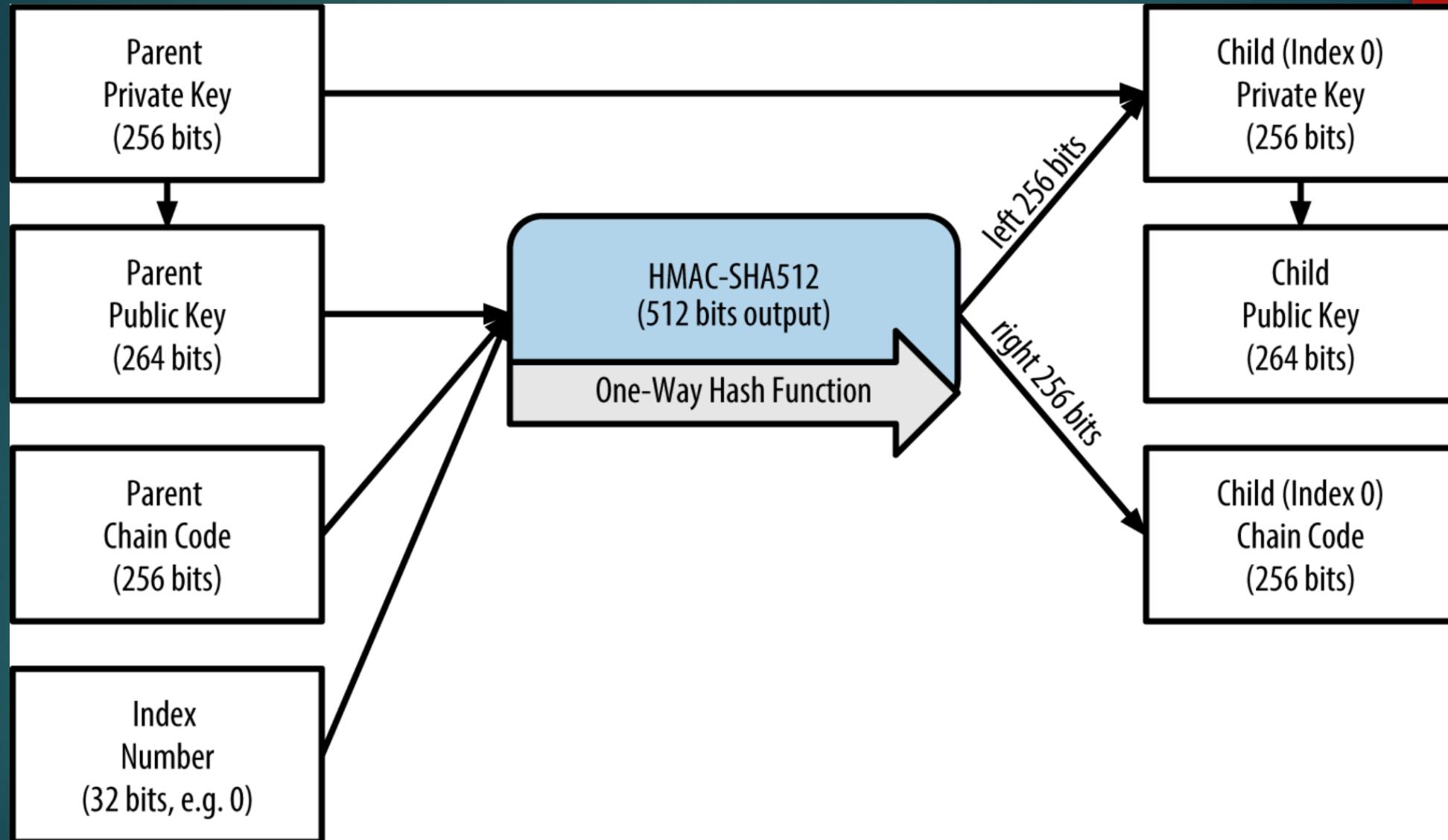
## 02. Bitcoin / Ethereum Blockchain Technology - HD Wallets

Creating an HD Wallet from the Seed HD wallets are created from a single root seed, which is a 128-, 256-, or 512-bit random number. Most commonly, this seed is generated from a mnemonic:



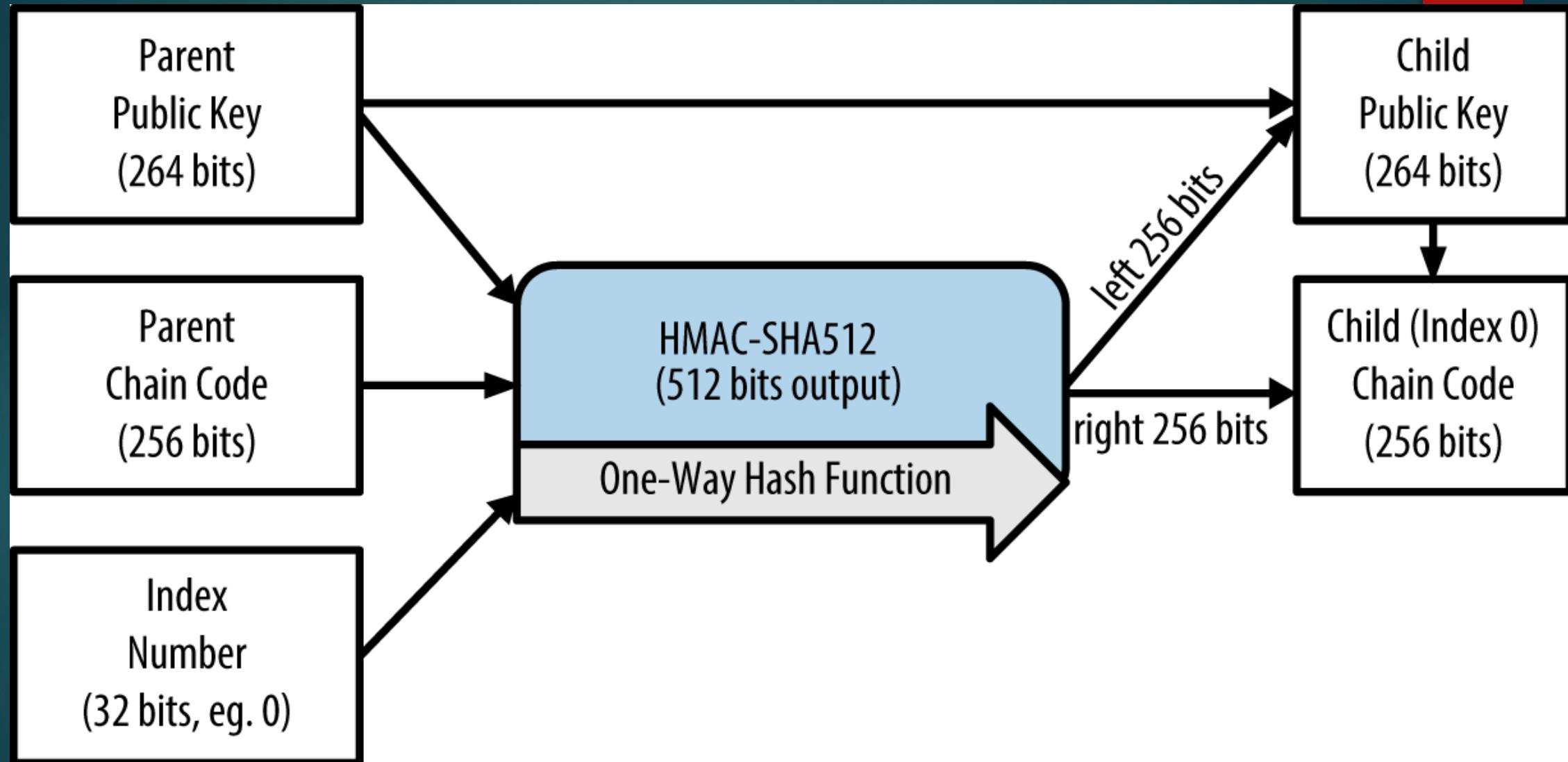
Creating master keys and chain code from a root seed

## 02. Bitcoin / Ethereum Blockchain Technology - HD Wallets



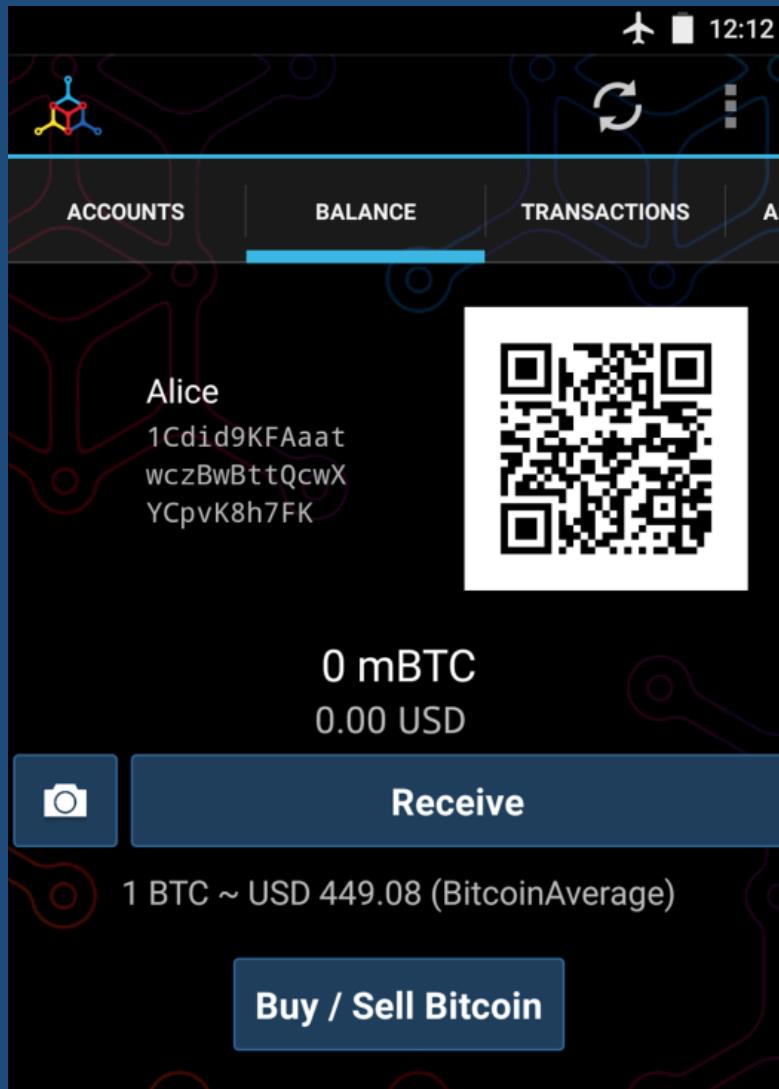
Extending a parent private key to create a child private key

## 02. Bitcoin / Ethereum Blockchain Technology - HD Wallets



Extending a parent public key to create a child public key

## 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions



**Transaction** View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2

1CdId9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA  
- (Unspent) 0.015 BTC

1CdId9KFAaatwczBwBttQcwXYCpvK8h7FK -  
(Unspent) 0.0845 BTC

97 Confirmations 0.0995 BTC

Summary	
Size	258 (bytes)
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)

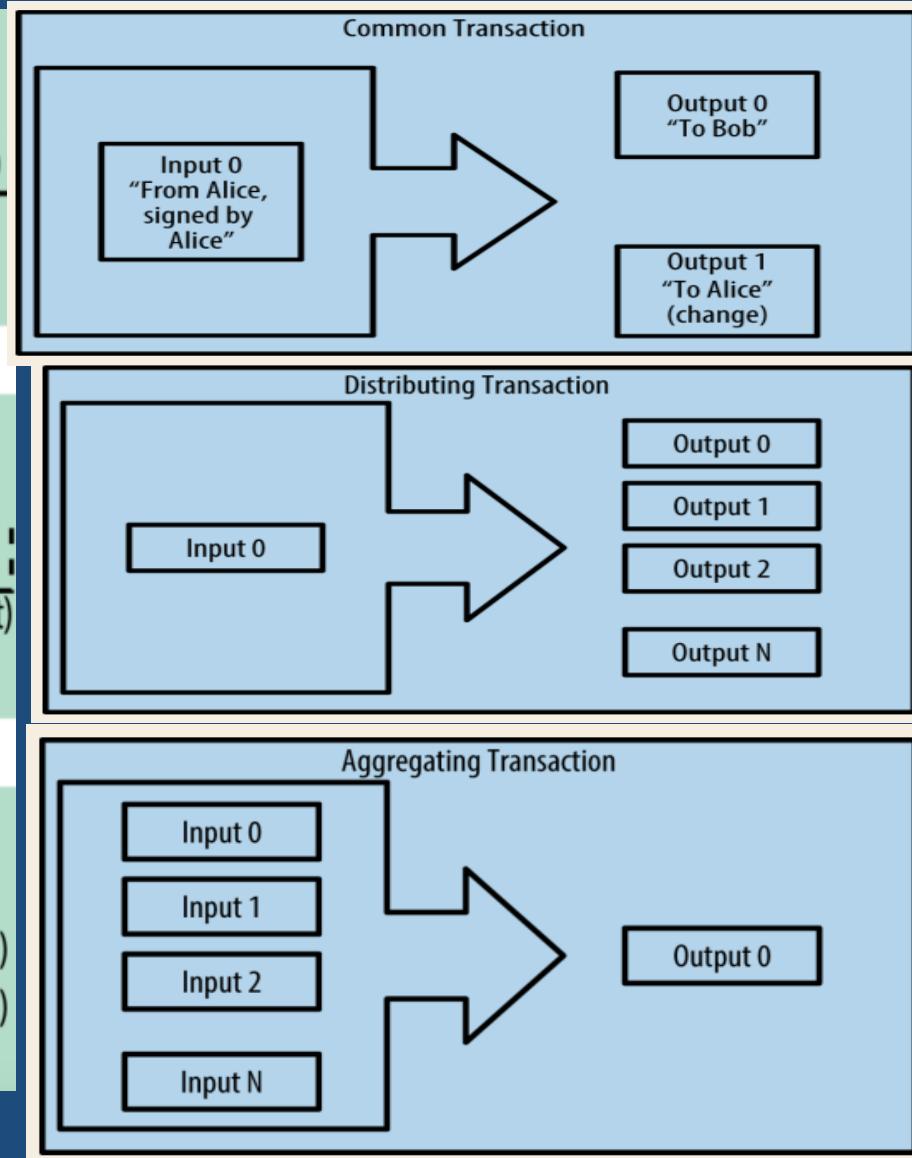
Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

## 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

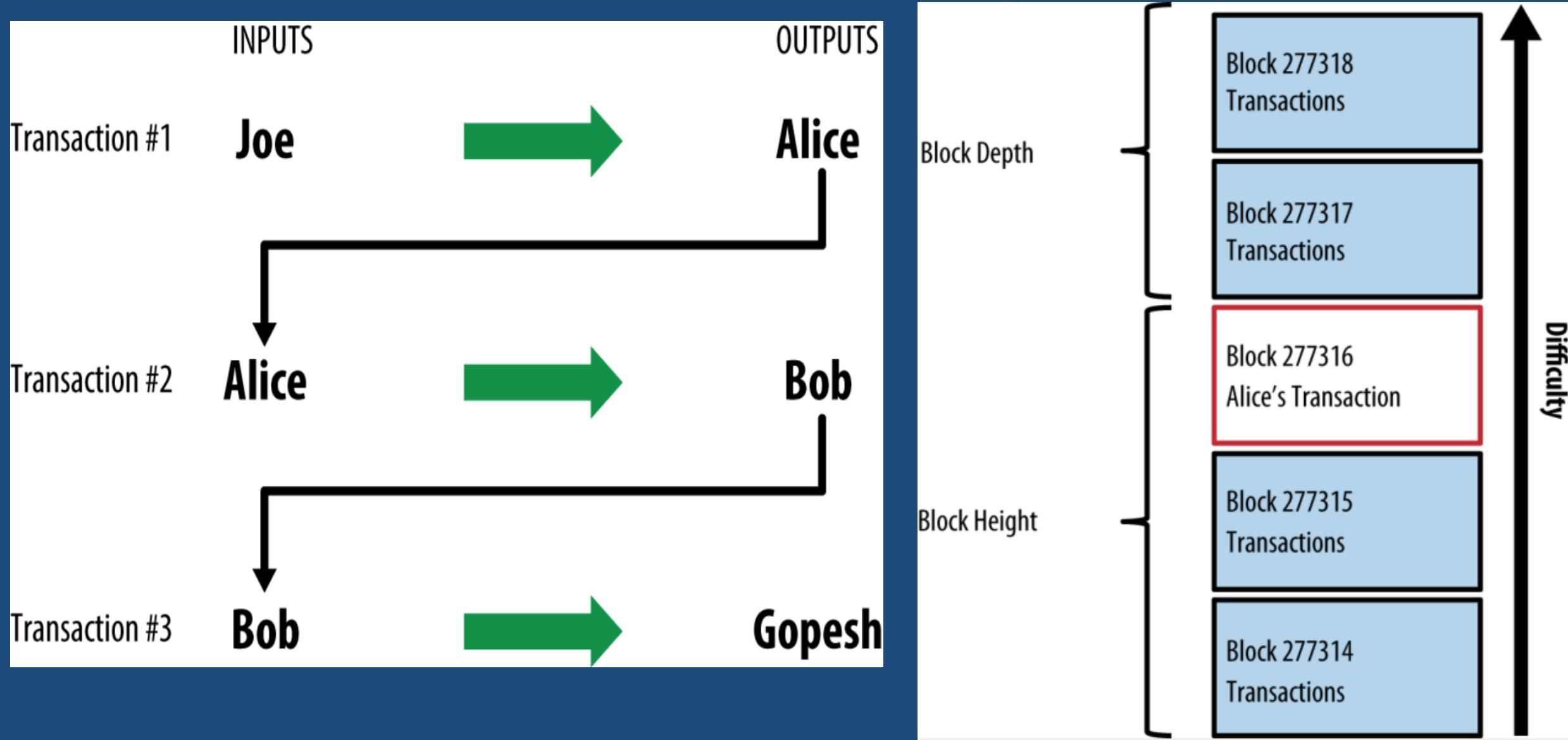
Transaction 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18			
INPUTS From		OUTPUTS To	
From (previous transactions Joe has received):	Joe	Output #0 Alice's Address	0.1000 BTC (spent)
		Transaction Fees:	0.0000 BTC

Transaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fb8a57286c345c2f2			
INPUTS From		OUTPUTS To	
From (previous transactions Alice has received):	Alice	Output #0 Bob's Address	0.0150 BTC (spent)
		Output #1 Alice's Address (change)	0.0845 BTC (unspent)
		Transaction Fees:	0.0005 BTC

Transaction 2bbac8bb3a57a2363407ac8c16a67015ed2e88a4388af58cf90299e0744d3de4			
INPUTS From		OUTPUTS To	
From (previous transactions Bob has received):	Bob	Output #0 Gopesh's Address	0.0100 BTC (unspent)
		Output #1 Bob's Address (change)	0.0045 BTC (unspent)
		Transaction Fees:	0.0005 BTC

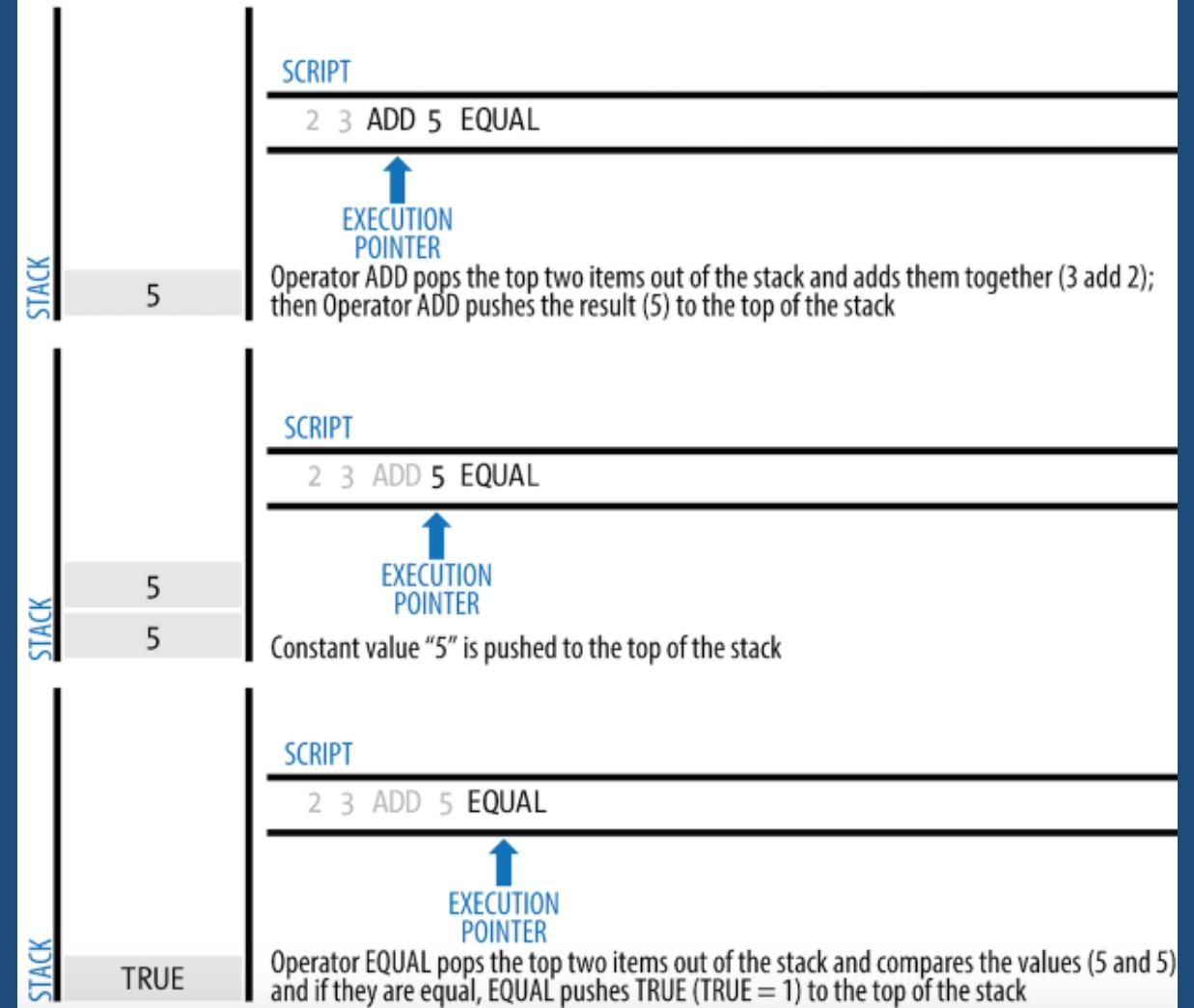
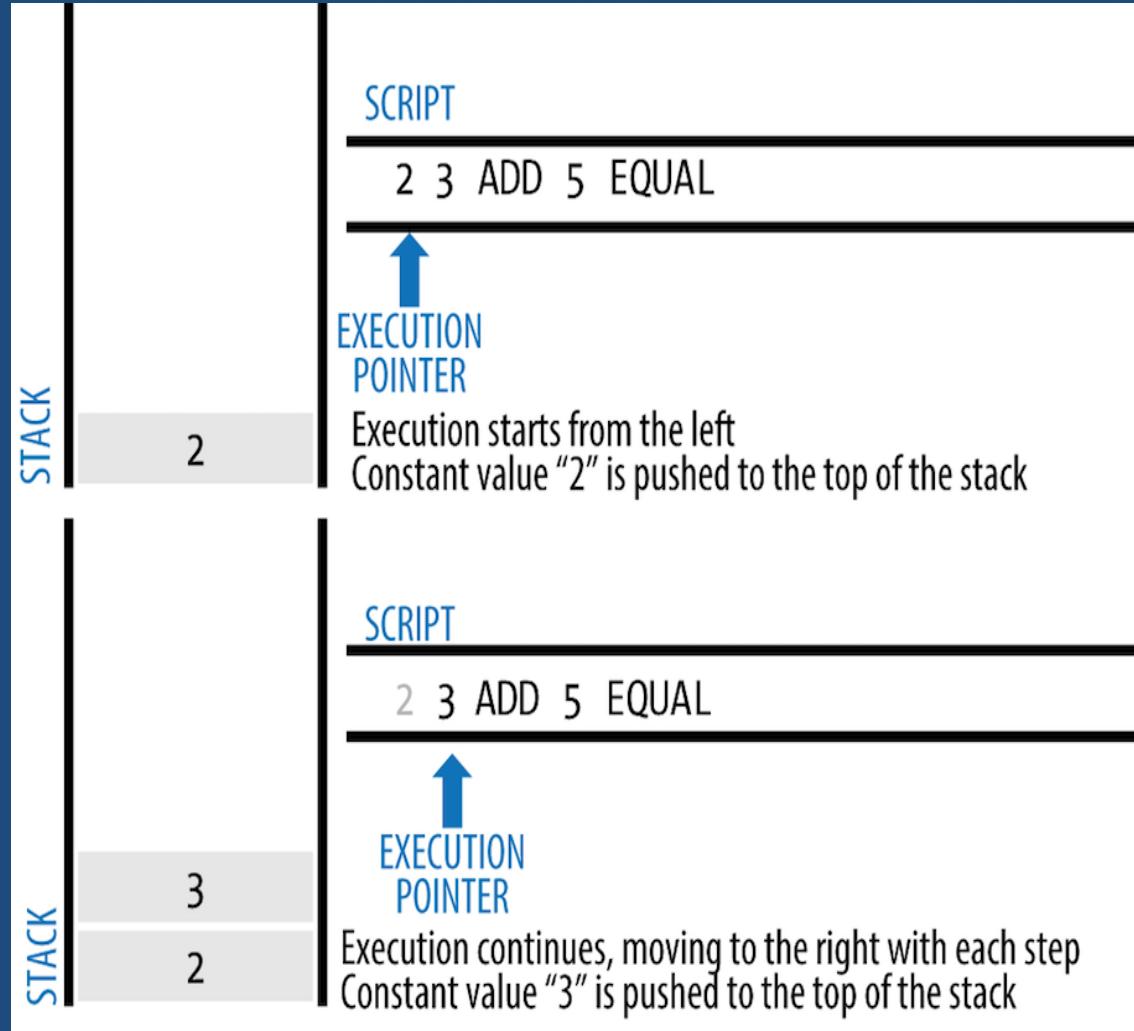


## 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions



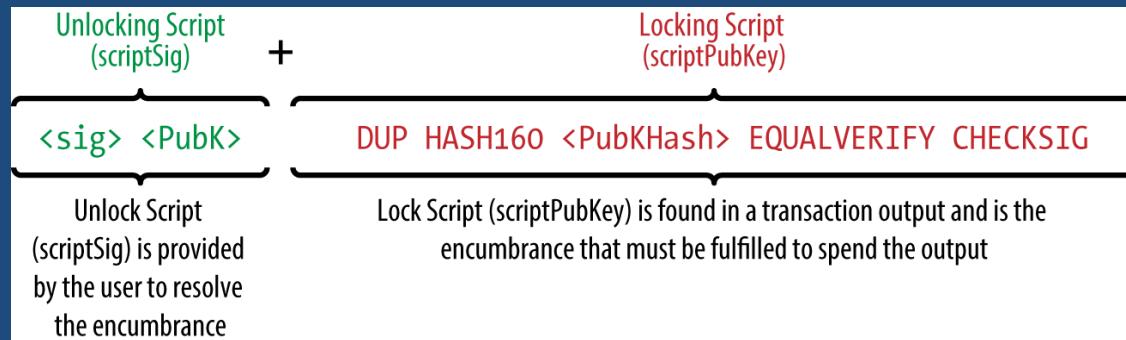
## 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

### Bitcoin's script validation doing simple math

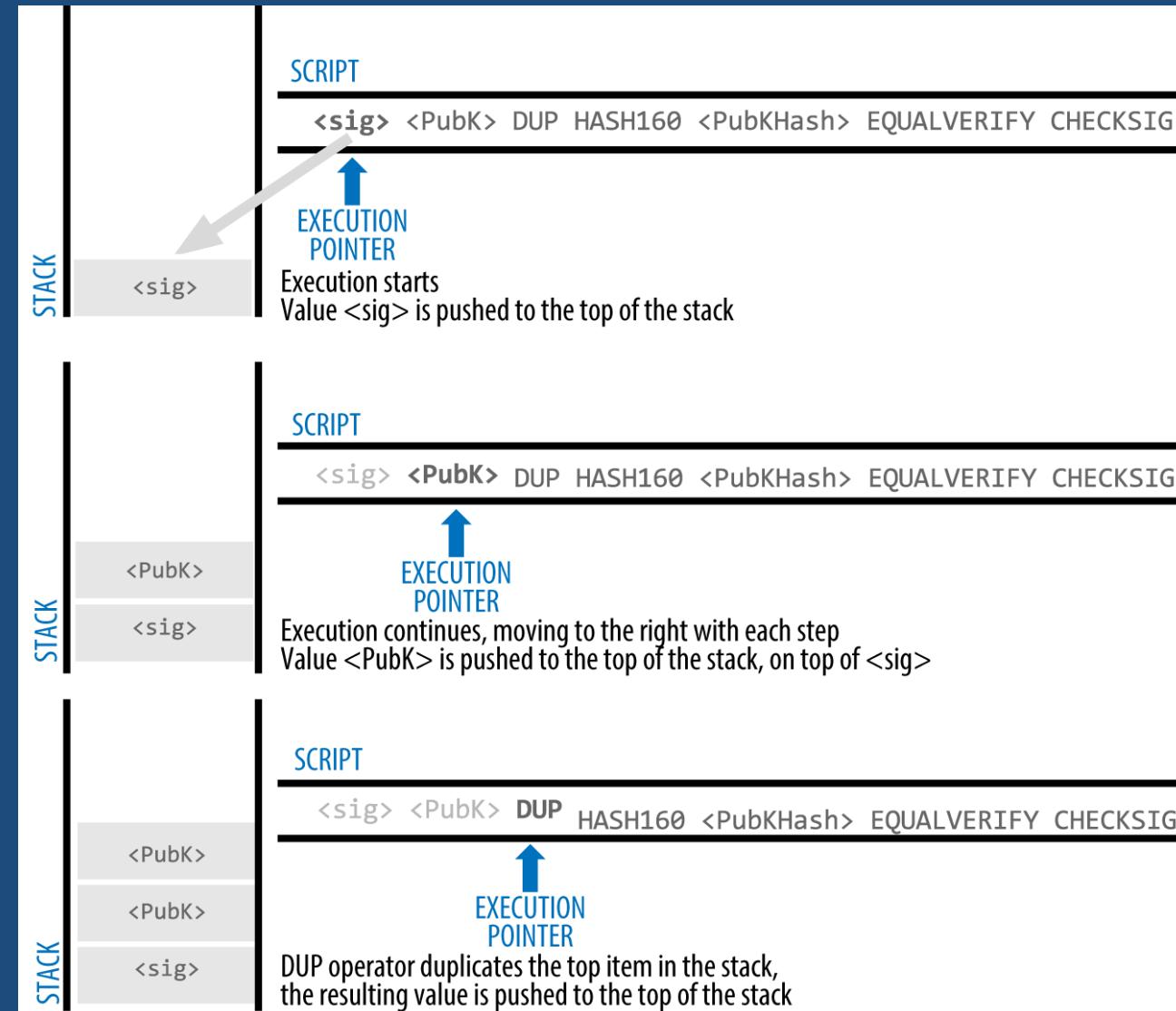


## 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

### Evaluating a script for a P2PKH (Pay-to-Public-Key-Hash) transaction



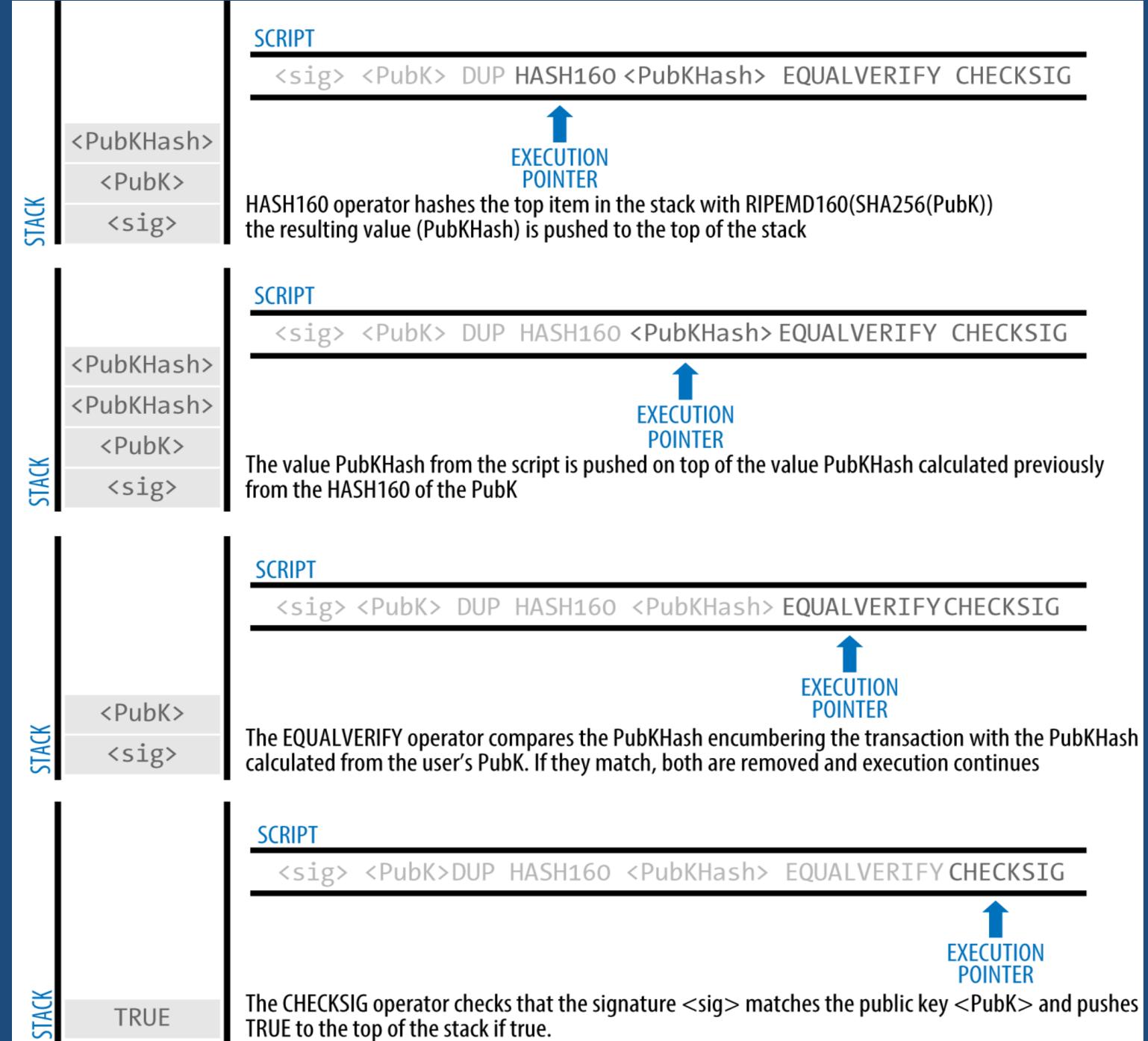
0100000001186f9f998a5aa6f048e51dd8419a14d8a  
0f1a8a2836dd734d2804fe65fa35779000000008  
b483045022100884d142d86652a3f47ba4746ec  
719bbfb040a570b1deccbb6498c75c4ae24cb02  
204b9f039ff08df09cbe9f6addac960298cad530a  
863ea8f53982c09db8f6e381301410484ecc0d46  
f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8  
416ab9fe423cc5412336376789d172787ec3457e  
ee41c04f4938de5cc17b4a10fa336a8d752adffff  
ffff0260e31600000000001976a914ab68025513  
c3dbd2f7b92a94e0581f5d50f654e788acd0ef80000  
00000001976a9147f9b1a7fb68d60c536c2fd8aea5  
3a8f3cc025a888ac00000 000



# 2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

## Pay-to-Public-Key-Hash (P2PKH)

The vast majority of transactions processed on the bitcoin network spend outputs locked with a Pay-to-Public-Key-Hash or “P2PKH” script. These outputs contain a locking script that locks the output to a public key hash, more commonly known as a bitcoin address. An output locked by a P2PKH script can be unlocked (spent) by presenting a public key and a digital signature created by the corresponding private key



## 2.1 Bitcoin / Ethereum Blockchain Technology – Transactions/Signature

*Signature = implementation of the ECDSA algorithm; the “message” being signed is the transaction, or more accurately a hash of a specific subset of the data in the transaction.* The signing key is the user’s private key. The result is the Bitcoin/Ethereum signature:

$$Sig = F_{sig}(F_{hash}(m), dA)$$

where:

- $dA$  is the signing private key
- $m$  is the transaction (or parts of it)
- $F_{hash}$  is the hashing function
- $F_{sig}$  is the signing algorithm
- $Sig$  is the resulting signature

In Ethereum’s implementation of ECDSA, the “message” being signed is the transaction, or more accurately, the Keccak256 hash of the RLP-encoded data from the transaction. The signing key is the EOA’s private key. The result is the signature:

$$Sig = F_{sig}(F_{keccak256}(m), k)$$

where:

- $k$  is the signing private key
- $m$  is the RLP-encoded transaction
- $F_{keccak256}$  is the Keccak256 hash function
- $F_{sig}$  is the signing algorithm
- $Sig$  is the resulting signature

## 2.1 Bitcoin / Ethereum Blockchain Technology – Transactions/Signature

The function  $F_{sig}$  produces a signature  $Sig$  that is composed of two values, commonly referred to as  $R$  and  $S$ :

$$Sig = (R, S)$$

Now that the two values  $R$  and  $S$  have been calculated, they are serialized into a byte-stream using an international standard encoding scheme called the *Distinguished Encoding Rules*, or *DER*.

- `0x30`—indicating the start of a DER sequence
- `0x45`—the length of the sequence (69 bytes)
- `0x02`—an integer value follows
- `0x21`—the length of the integer (33 bytes)
- $R$ —  
`00884d142d86652a3f47ba4746ec719bbfb040a570b1  
deccbb6498c75c4ae24cb`
- `0x02`—another integer follows
- `0x20`—the length of the integer (32 bytes)
- $S$ —  
`4b9f039ff08df09fbe9f6addac960298cad530a863ea8  
f53982c09db8f6e3813`
- A suffix (`0x01`) indicating the type of hash used (`SIGHASH_ALL`)

## **2.1 Bitcoin / Ethereum Blockchain Technology – Transactions/Block**

- The blockchain data structure is an ordered, back-linked list of blocks of transactions.
- The blockchain can be stored as a flat file, or in a simple database.
- A block is a container data structure that aggregates transactions for inclusion in the public ledger, the blockchain.
- The block header is 80 bytes, whereas the average transaction is at least 400 bytes and the average block contains more than 1900 transactions.
- A complete block, with all transactions, is therefore 10,000 times larger than the block header.

## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain

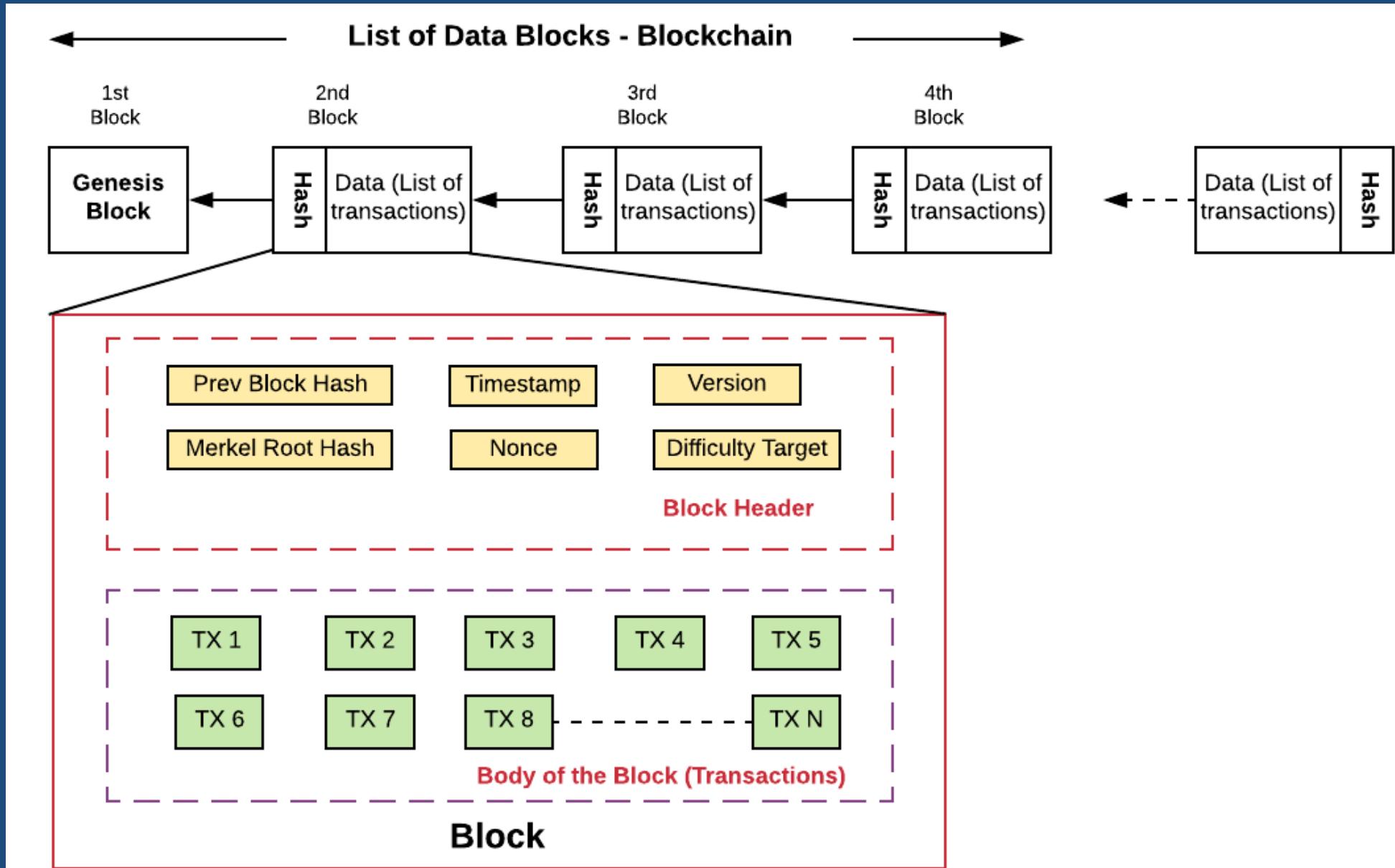
### A. The block structure

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

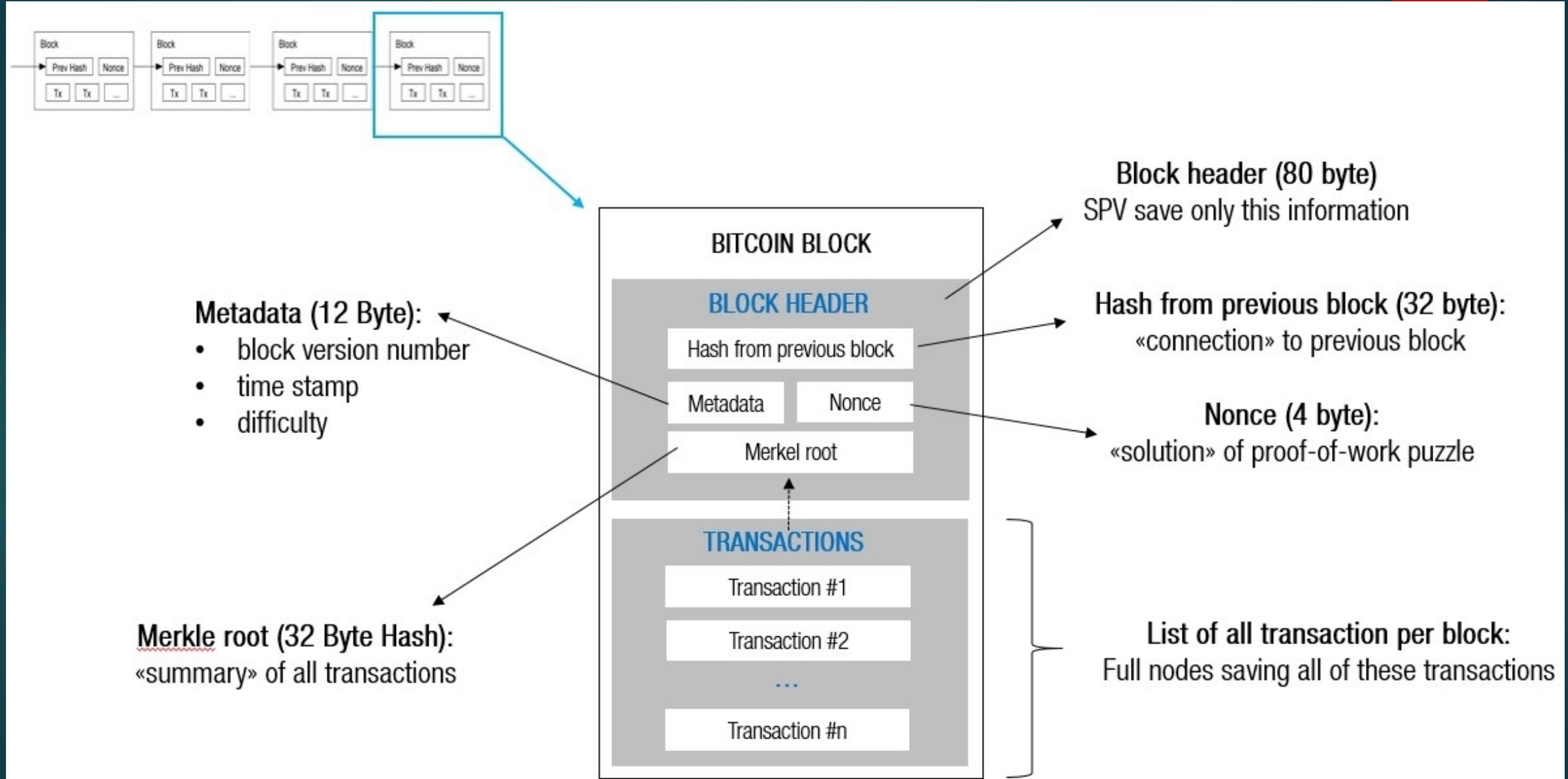
### B. The Block Header structure

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

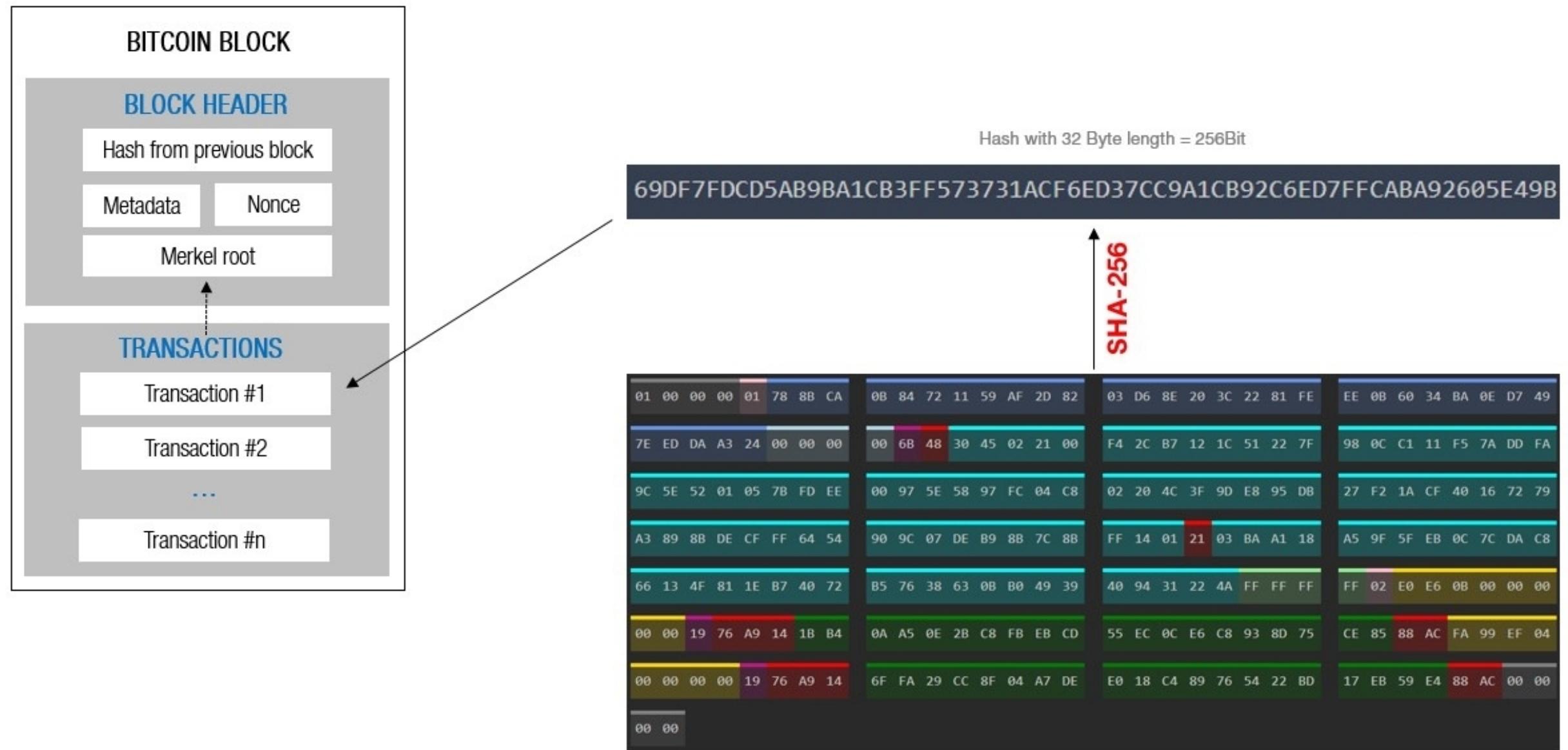
## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain



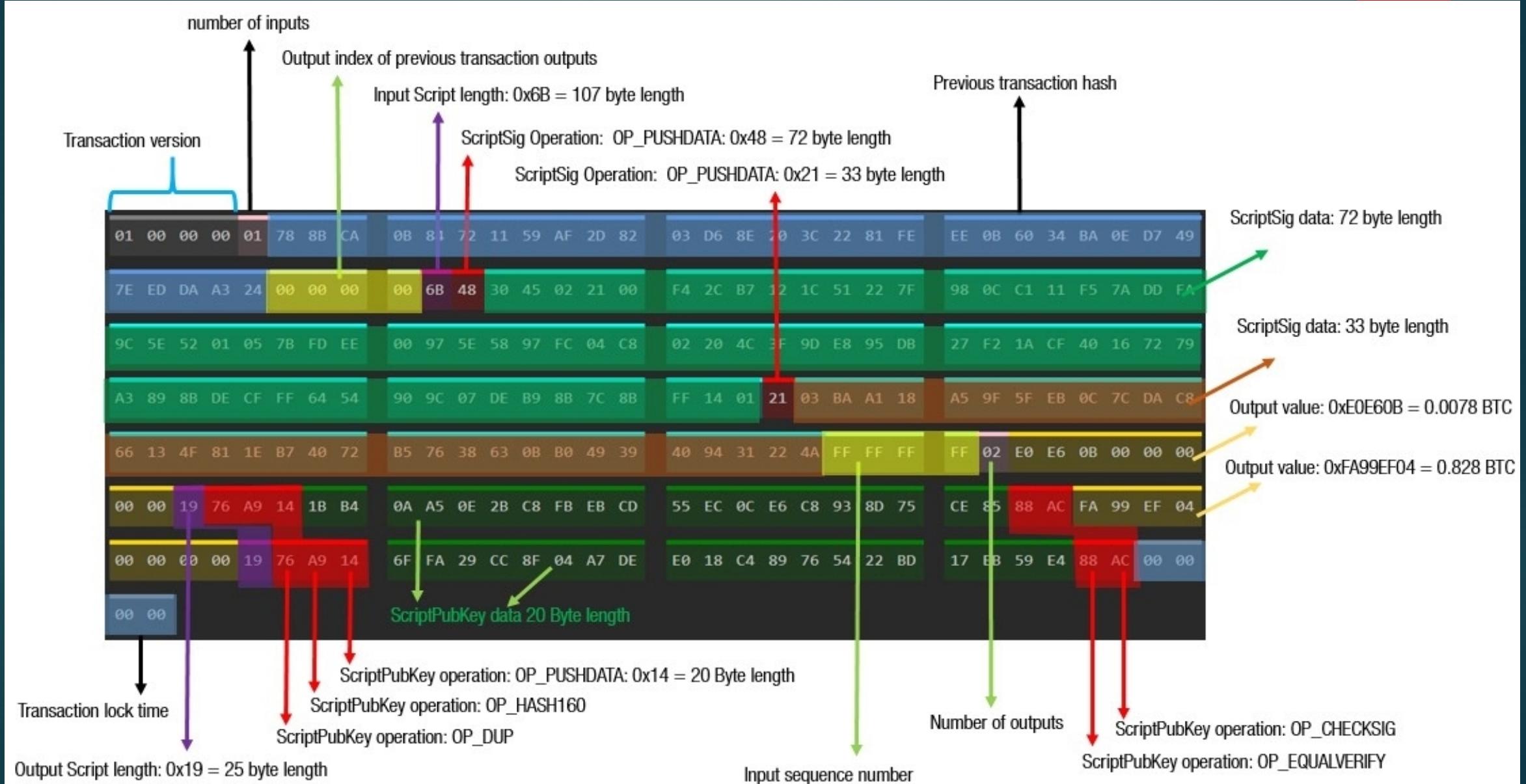
## 2.1 Bitcoin Block from the chain



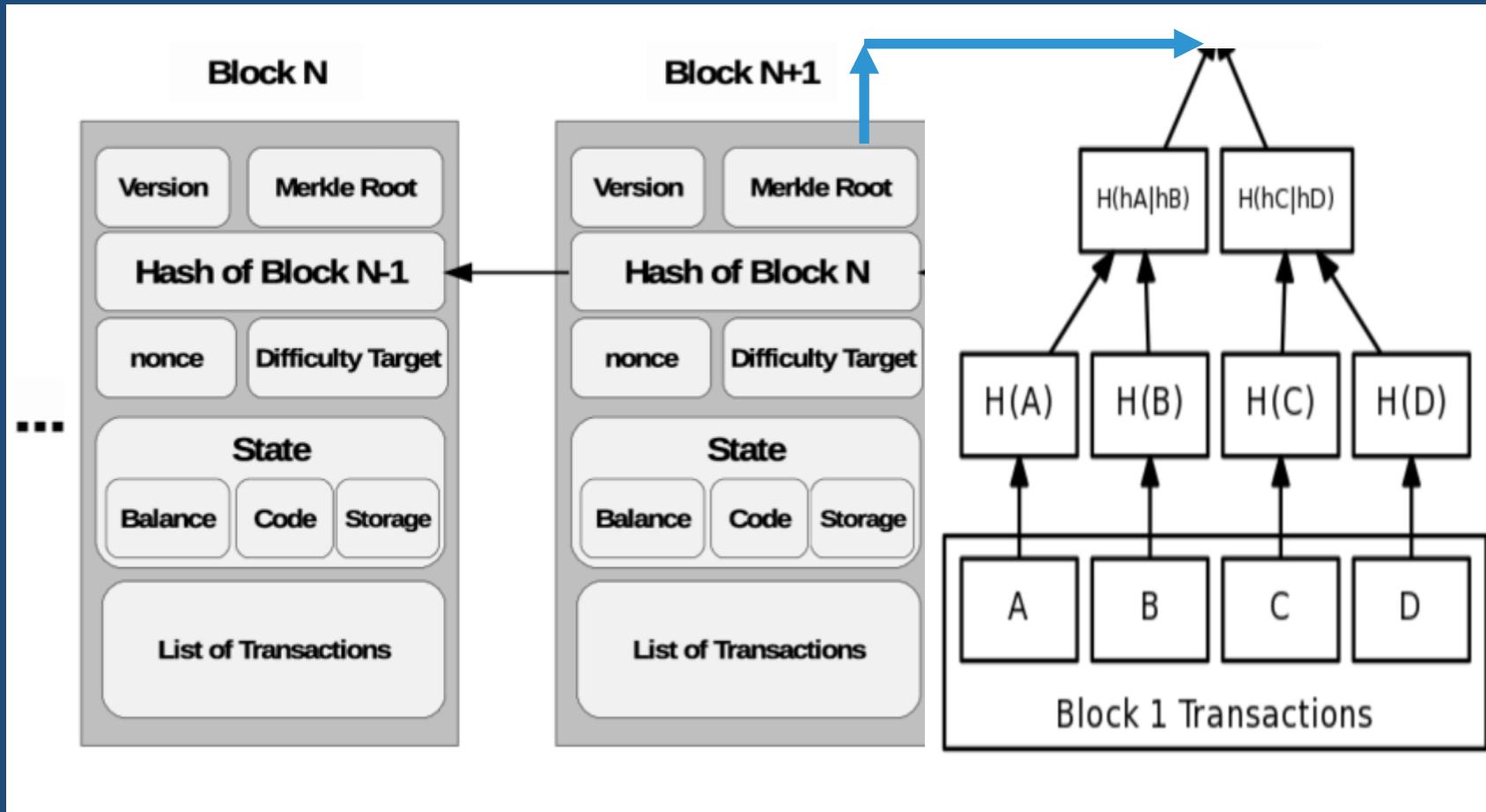
## 2.1 Bitcoin - Method from a raw Bitcoin HEX transaction into a block



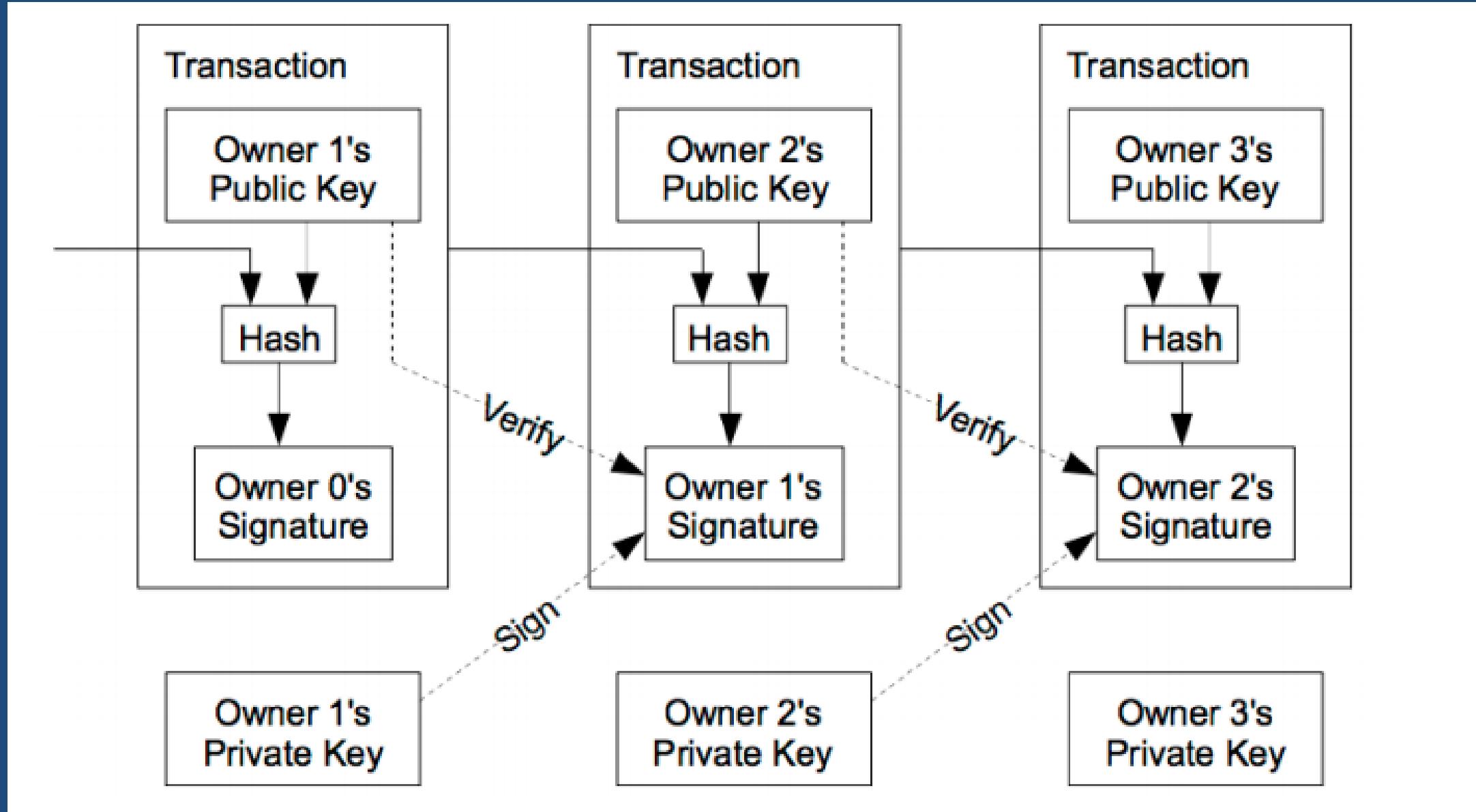
## 2.1 Detailed analysis of a raw Bitcoin transaction



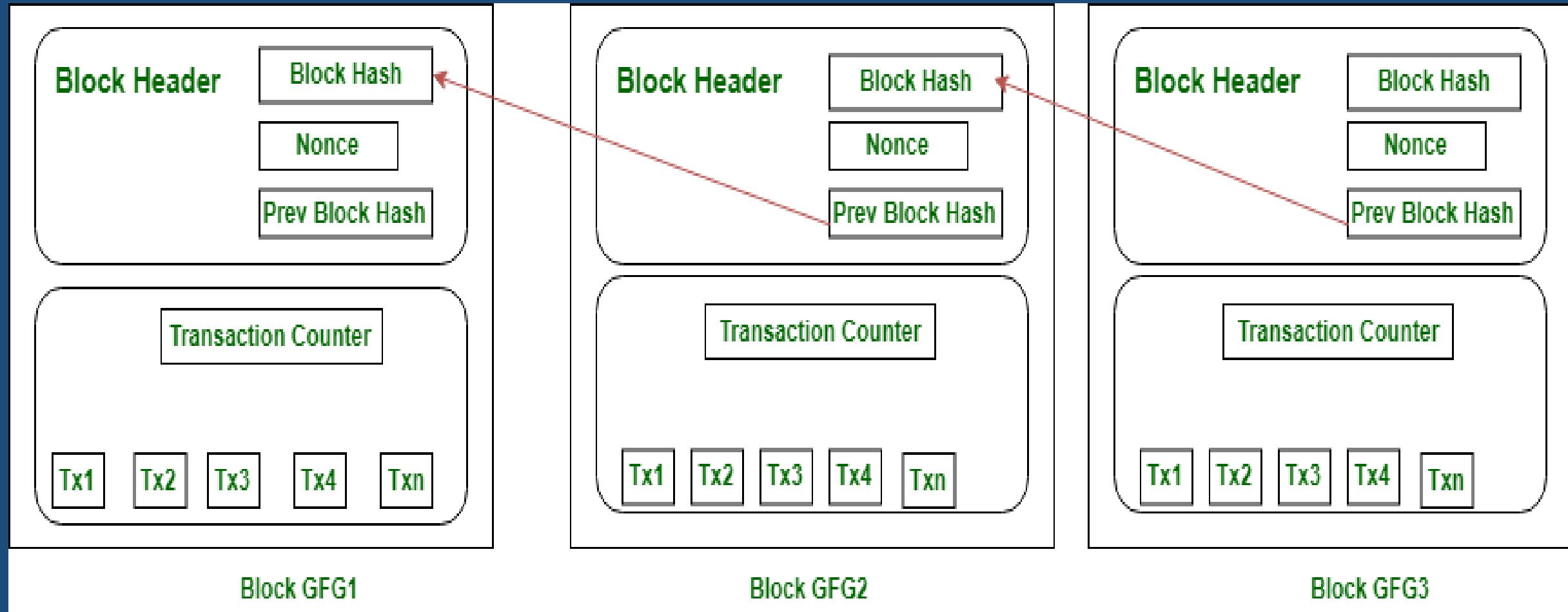
## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain



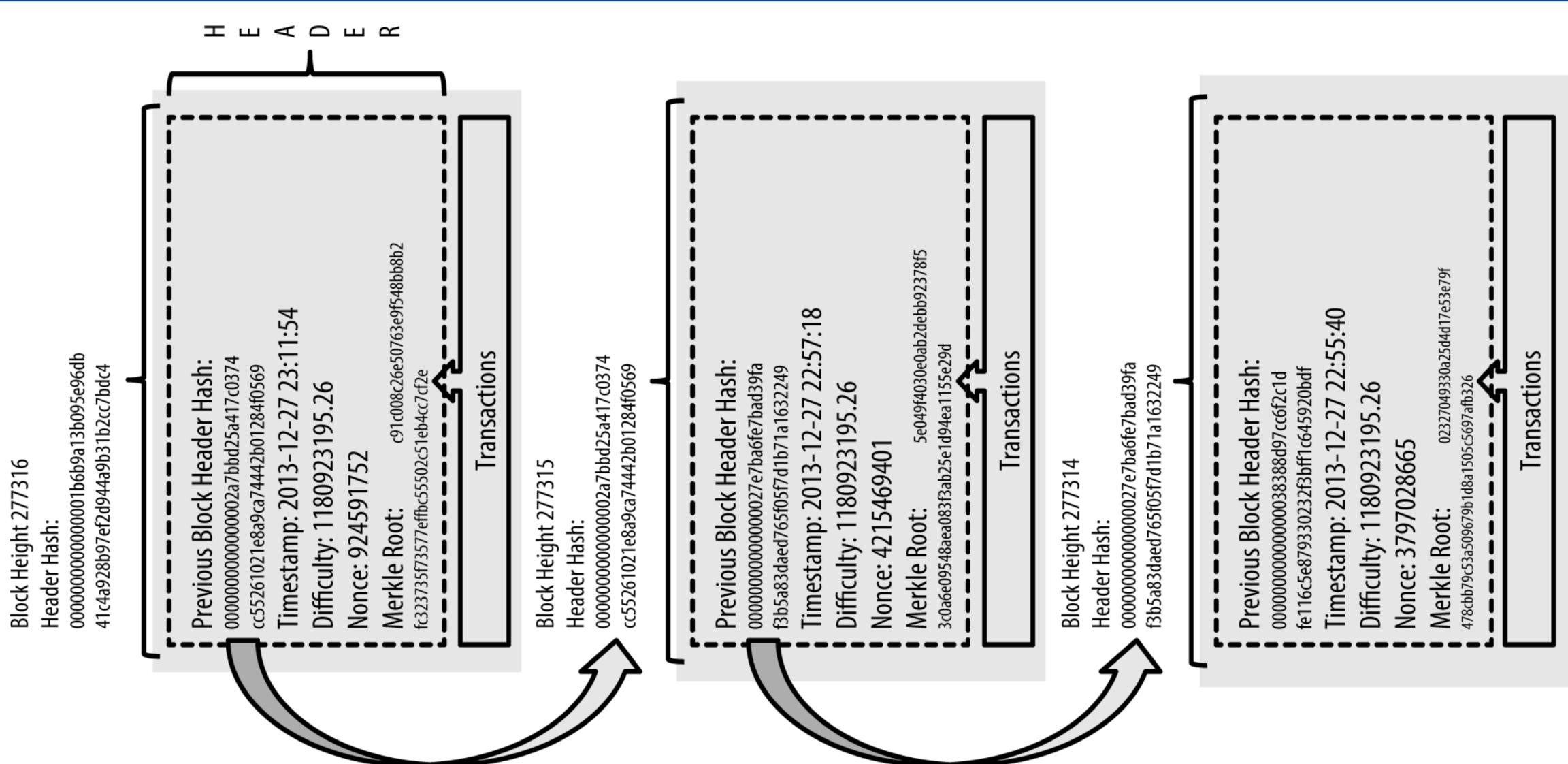
## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain



## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain

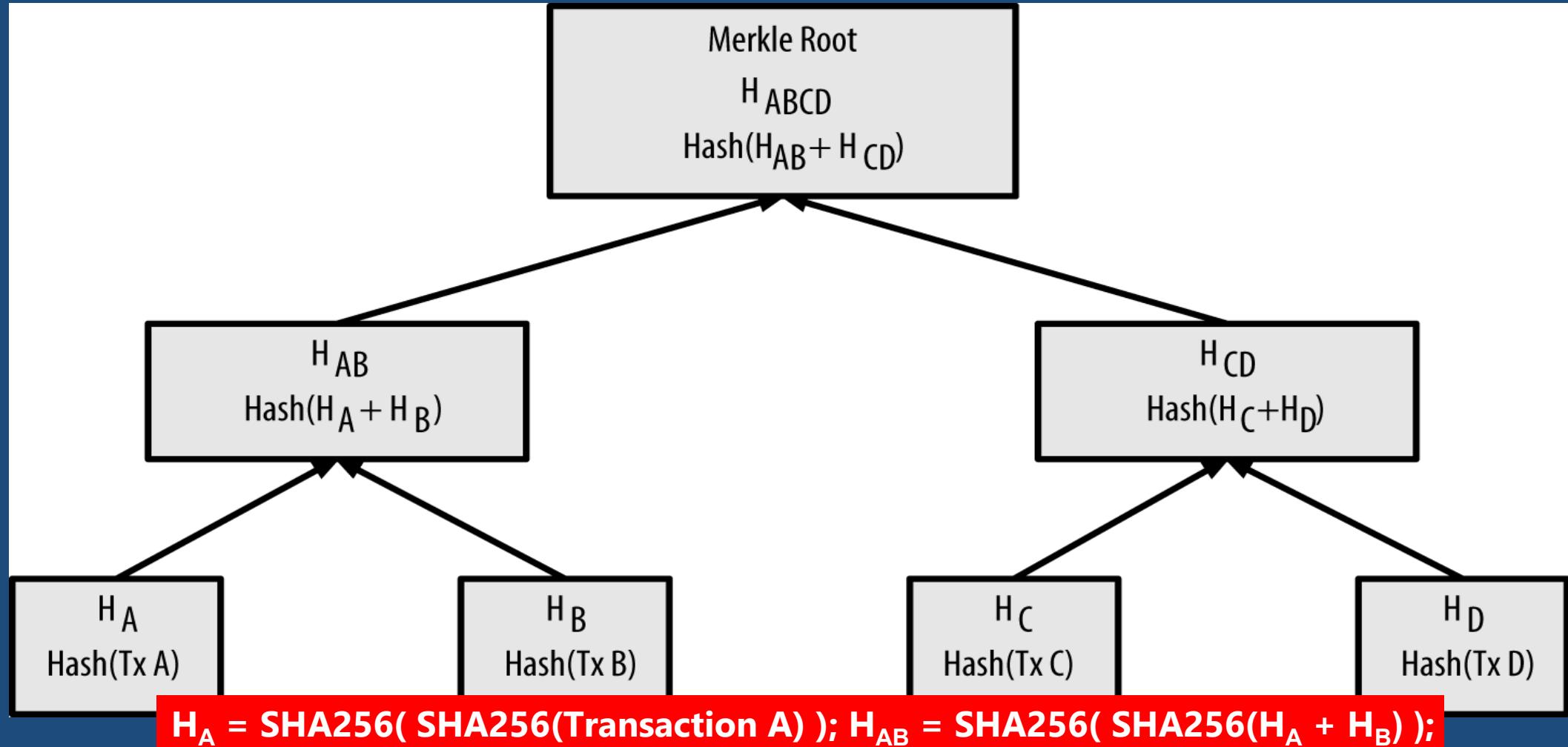


## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain

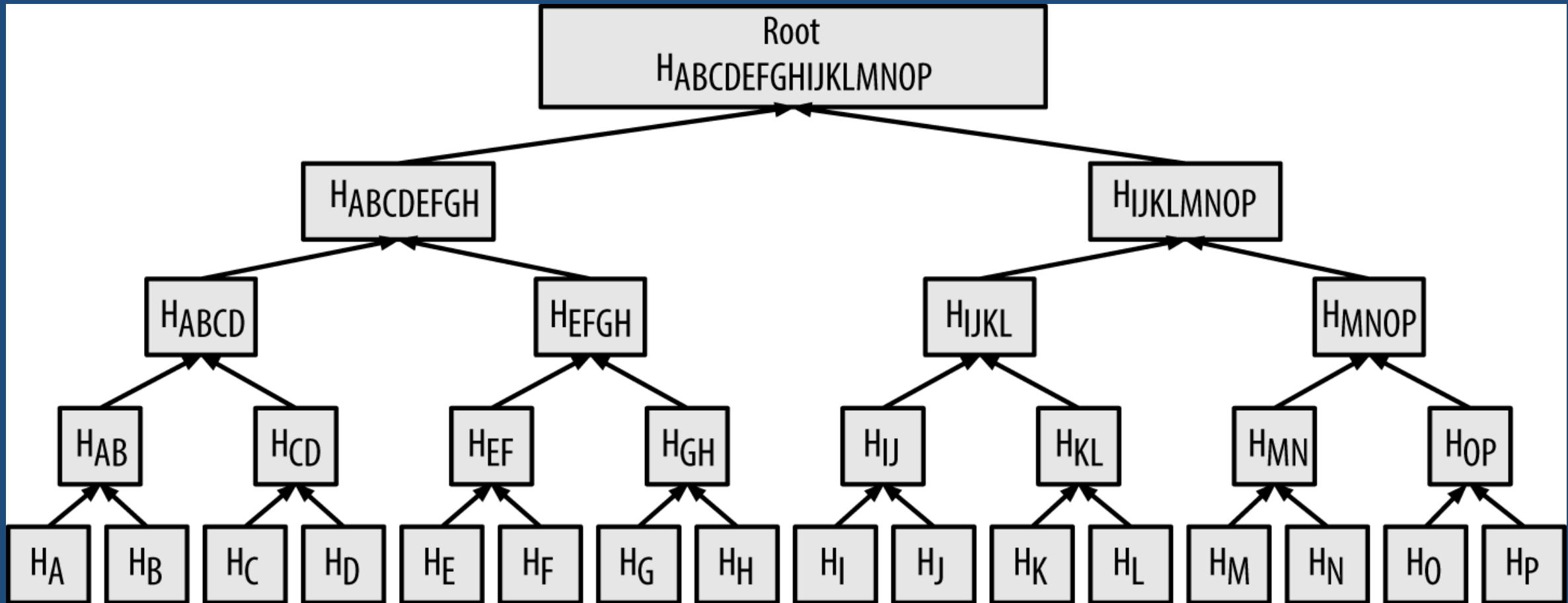


## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain – Transactions Hashes in Merkle Trees

*Merkle trees are used in bitcoin to summarize all the transactions in a block, producing an overall digital fingerprint of the entire set of transactions, providing a very efficient process to verify whether a transaction is included in a block. The Merkle Tree is constructed bottom-up. If we have 4 transactions, A, B, C, and D, which form the leaves of the Merkle Tree:*

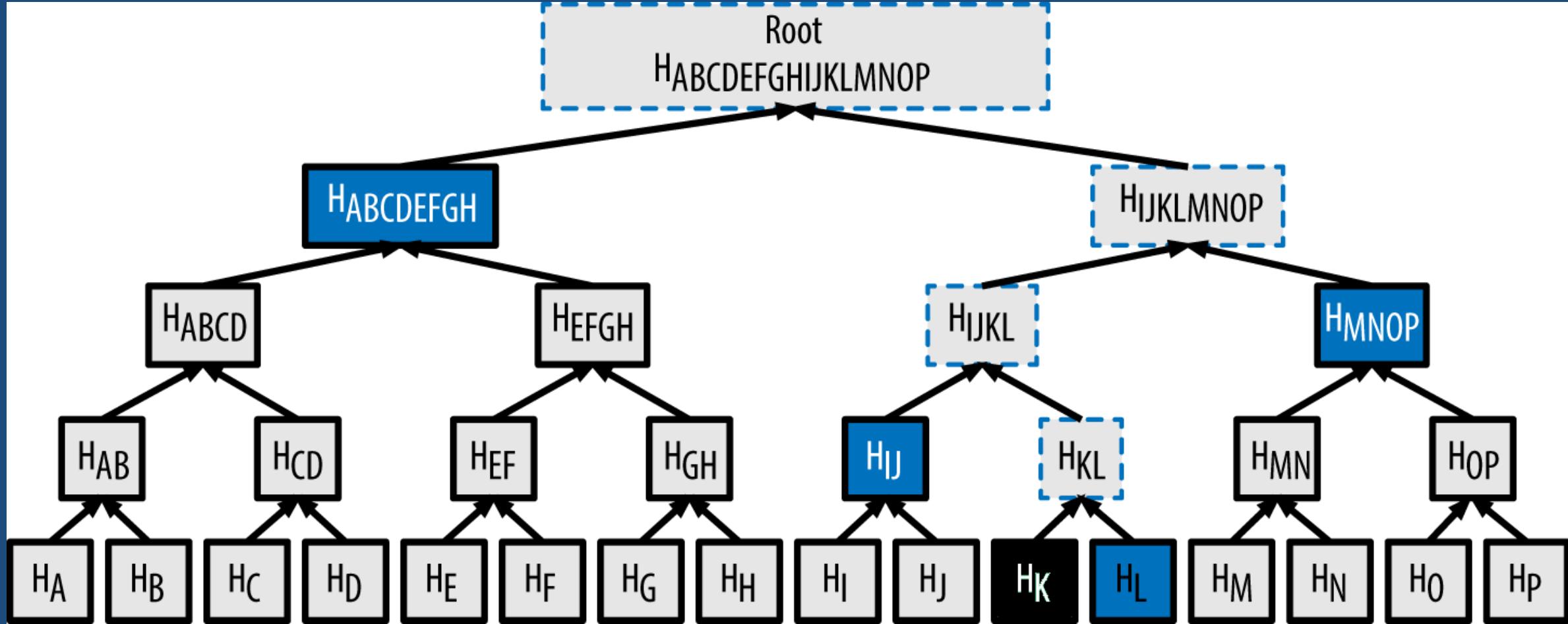


## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain – Transactions Hashes in Merkle Trees



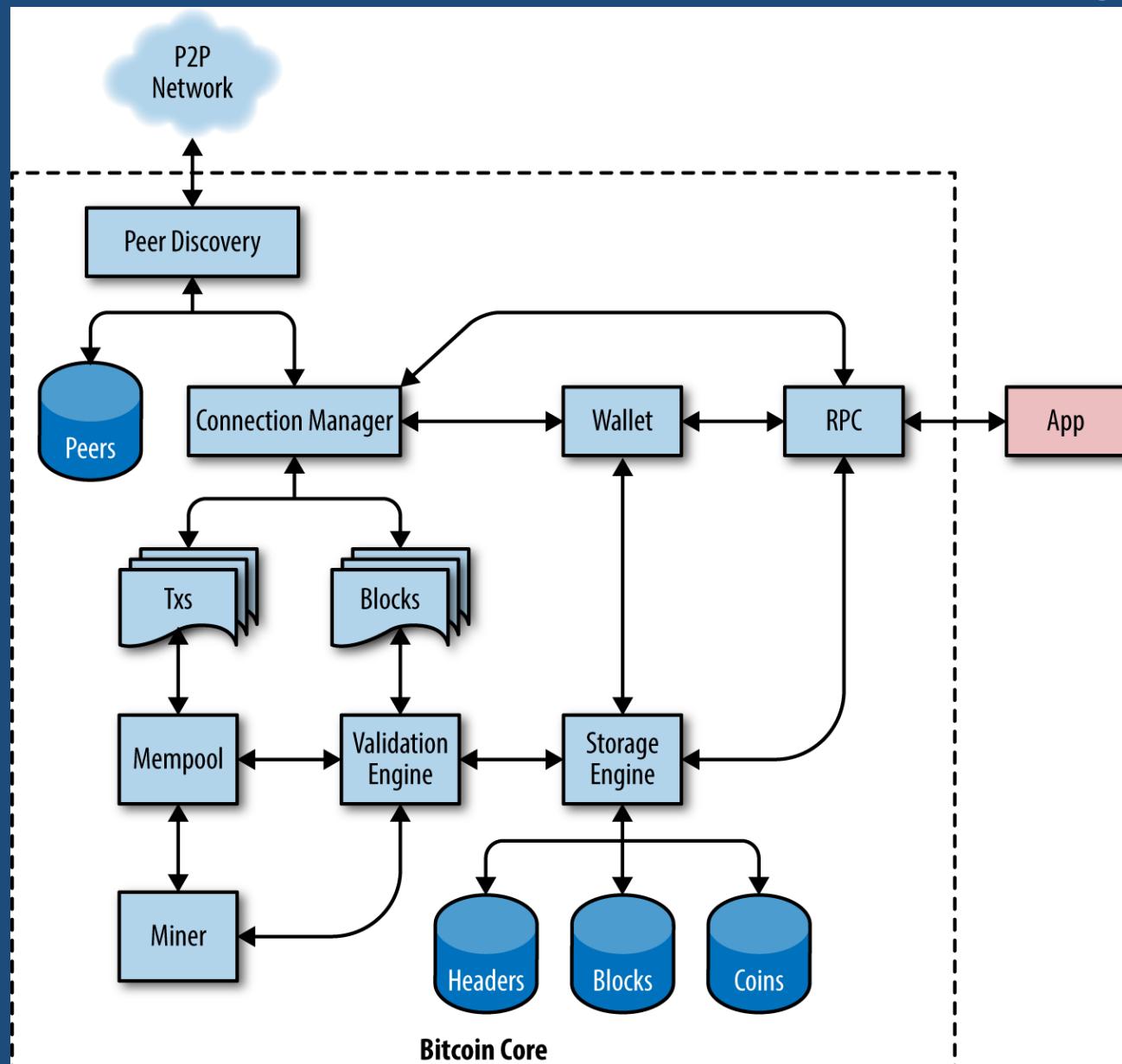
This tree is built from hashes for 16 transactions. Although the root looks bigger than the leaf nodes in the diagram, it is the exact same size, just 32 bytes.

## 2.1 Bitcoin / Ethereum Blockchain Technology – Blockchain – Transactions Hashes in Merkle Trees



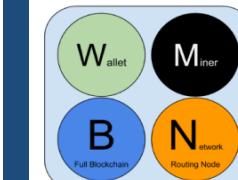
A node can prove that a transaction K is included in the block by producing a Merkle path that is only four 32-byte hashes long (128 bytes total). The path consists of the four hashes  $H_L$ ,  $H_{IJ}$ ,  $H_{MNOP}$ , and  $H_{ABCDEFGHI}$ . With those 4 hashes provided as an authentication path, any node can prove that  $H_K$  (with a black background at the bottom of the diagram) is included in the Merkle root by computing 4 additional pairwise hashes  $H_{KL}$ ,  $H_{IJKL}$ ,  $H_{IJKLMNOP}$ , and the Merkle tree root (outlined in a dashed line in the diagram).

# 2.1 Bitcoin / Ethereum Blockchain Technology – Reference Implementation



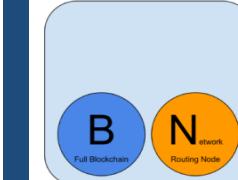
## Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



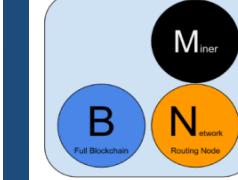
## Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



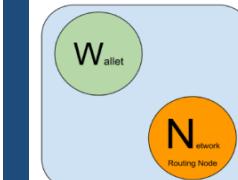
## Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



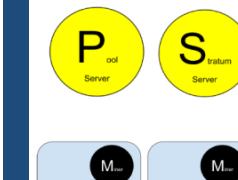
## Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



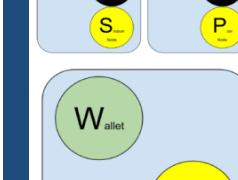
## Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



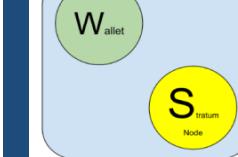
## Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



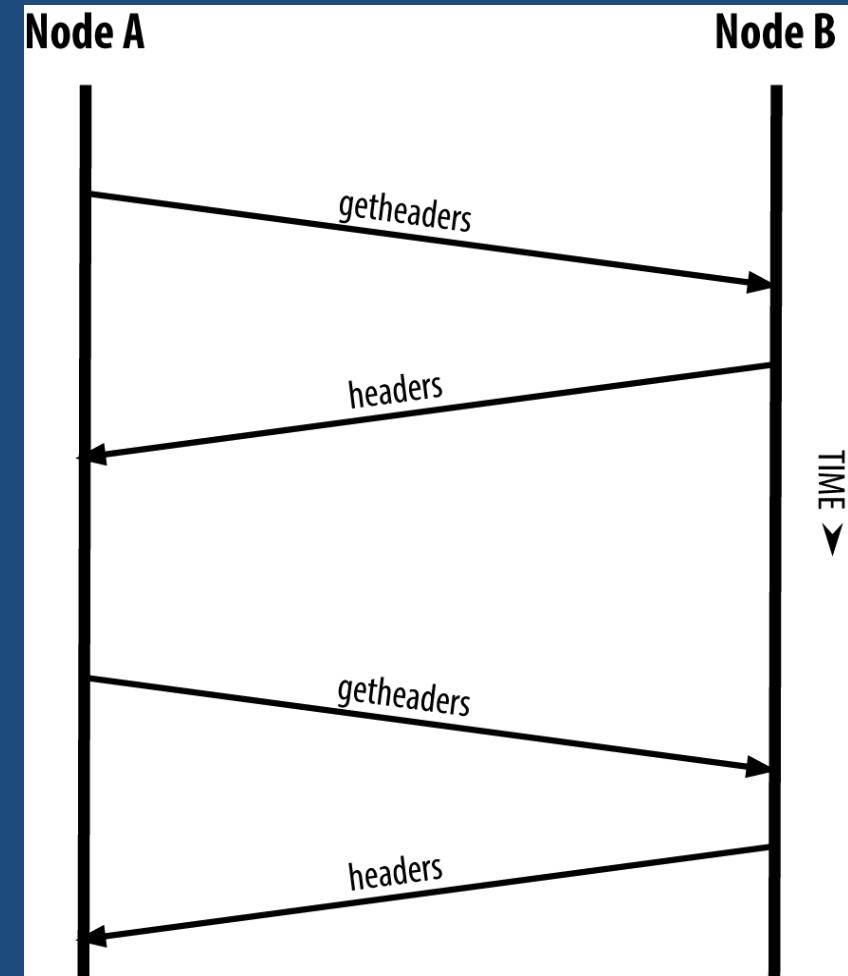
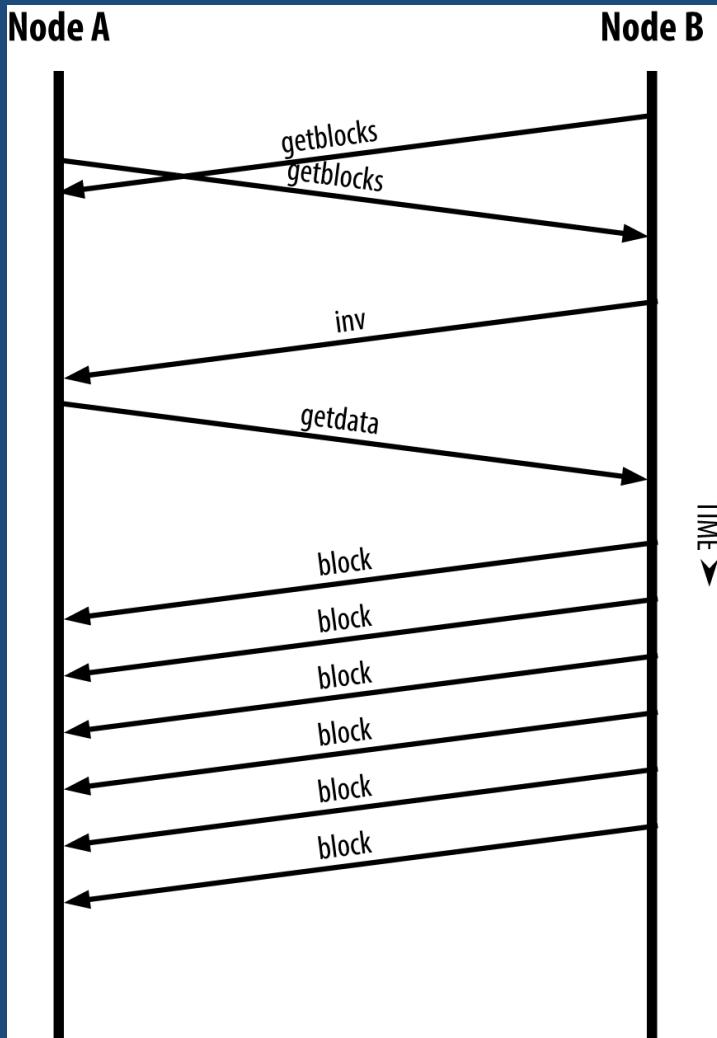
## Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.



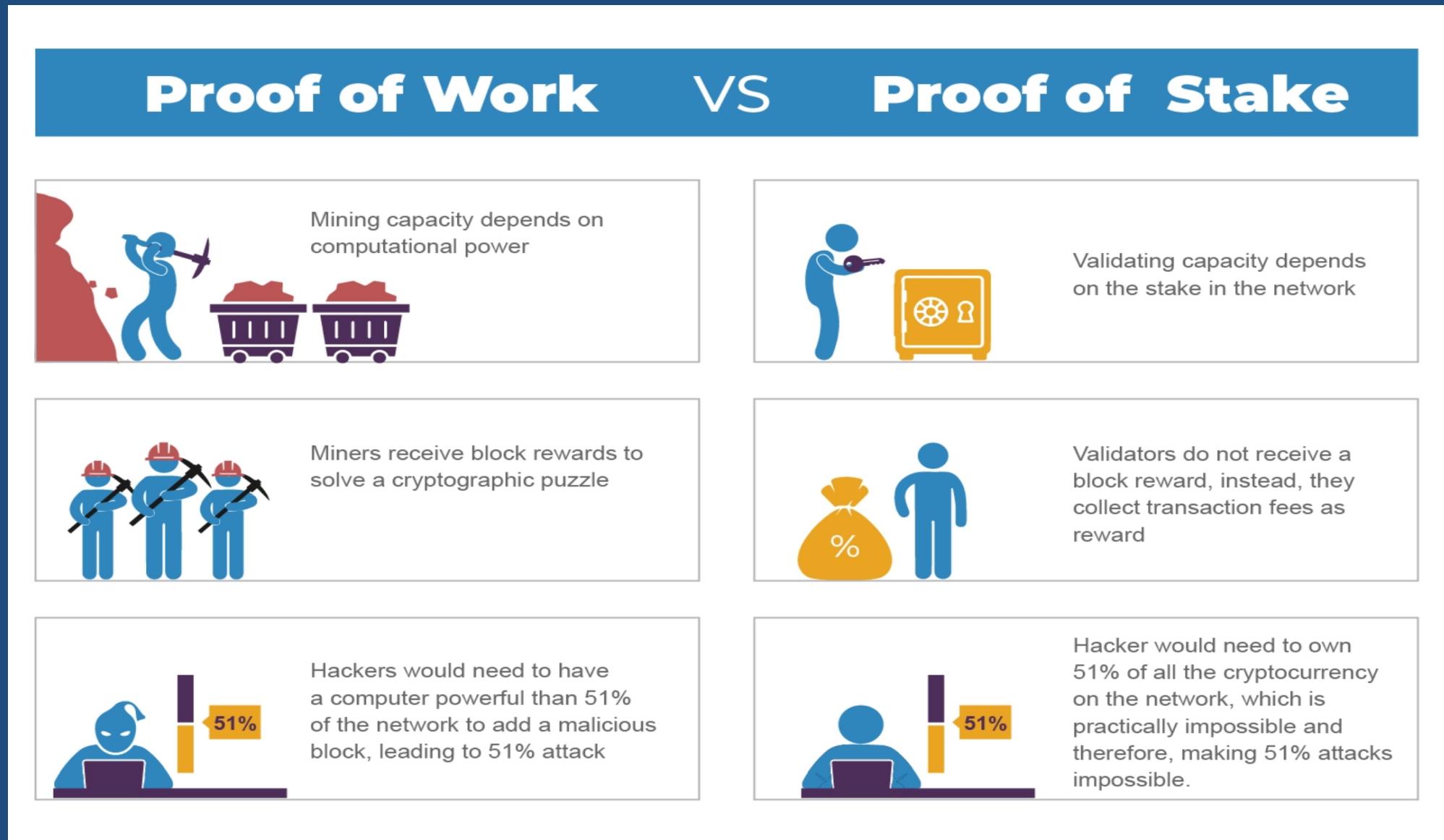
## 2.1 Bitcoin / Ethereum Blockchain Technology – P2P Network

Node synchronizing the blockchain by retrieving blocks from a peer



SPV node synchronizing the block headers

## 2.1 Consensus Algo - Bitcoin / Ethereum Blockchain Technology – P2P Network



The proof of work (POW) is a common **consensus algorithm** used by the most popular **cryptocurrency** networks like **bitcoin** and **litecoin**. It requires a participant node to prove that the work done and submitted by them qualifies them to receive the right to add new transactions to the **blockchain**.

## 2.1 Bitcoin Standards

luke-jr Merge pull request #740 from skddc/patch-1 ...		Latest commit 954df0d on 14 Dec 2018
bip-0001	Fix formatting	5 years ago
bip-0002	Add obsolete status to process image	2 years ago
bip-0008	Amend BIP8 by height	2 years ago
bip-0009	BIP 9: Misplaced table cells typo	11 months ago
bip-0016	fix bip-0016 link 404	4 months ago
bip-0032	Fix formatting	5 years ago
bip-0039	Fix two errors in the BIP 39 French wordlist	a year ago
bip-0042	Include image for BIP42	5 years ago
bip-0047	BIP-0047: Reusable payment codes	4 years ago
bip-0068	Improve title, add encoding diagram and small fixup	3 years ago
bip-0069	add EOF newlines per @luke-jr	3 years ago
bip-0070	Put BIP 75 in the right place in README, and clean up formatting a bit	3 years ago
bip-0073	Fix formatting	5 years ago
bip-0075	- Update identifier comment in paymentrequest.proto	2 years ago
bip-0098	BIP-0098: Fast Merkle Trees	a year ago
bip-0114	BIP114: MAST proposal v2	3 years ago
bip-0122	Assign BIP 122	3 years ago
bip-0135	Assign BIP 135: Generalized version bits voting	2 years ago
bip-0144	BIP141: commitment clarification. BIP144: new diagram	3 years ago
bip-0152	header message can also get replied by getdata (CMPCT_BLOCK)	3 years ago
bip-0156	renamed files and updated README.mediawiki index	5 months ago
bip-0158	BIP 158: Add more cases to test vectors.	5 months ago
bip-0174	BIP 174 workflow graphics	7 months ago
scripts	scripts/buildtable: Support License-Code header	a year ago



## Ethereum is often described as “the world computer.”

From a more practical perspective, Ethereum is:

- an open source,
- globally decentralized computing infrastructure that executes programs called ***smart contracts***.
- it uses a blockchain to synchronize and store the system's state changes, along with a cryptocurrency called ***Ether*** to meter and constrain execution resource costs.

The Ethereum platform enables developers to build powerful decentralized applications with built-in economic functions. While providing high availability, auditability, transparency, and neutrality, it also reduces or eliminates censorship and reduces certain counterparty risks.

# #Ethereum



- **Similar:**

Ethereum shares many common elements with other open blockchains – e.g. Bitcoin:

- a **(P2P)** peer-to-peer network connecting participants;
- a **Byzantine fault-tolerant consensus algorithm** for synchronization of state updates (a **Proof-of-Work** blockchain);
- the use of cryptographic primitives such as digital signatures and hashes, and a digital currency (**Ether**).

- **Not similar:**

Unlike Bitcoin, which has a very limited scripting language, Ethereum is designed to be:

- a general-purpose programmable blockchain that runs a virtual machine (EMV) capable of executing code of arbitrary and unbounded complexity.
- Where Bitcoin's Script language is, intentionally, constrained to simple true/false evaluation of spending conditions,
  - **Ethereum's language is Turing complete, meaning that Ethereum can straightforwardly function as a general-purpose computer.**

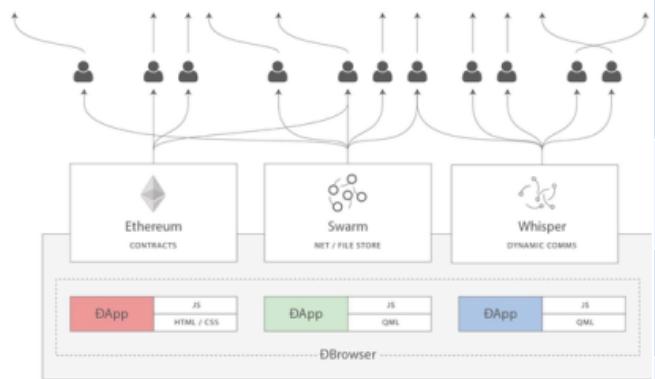
# #Ethereum

**Components of a Blockchain:** The components of an open, public blockchain are (usually):

- A **peer-to-peer (P2P)** network connecting participants and propagating transactions and blocks of verified transactions, based on a standardized “gossip” protocol
- **Messages**, in the form of transactions, representing state transitions
- A **set of consensus rules**, governing what constitutes a transaction and what makes for a valid state transition
- A **state machine** that processes transactions according to the consensus rules
- A **chain of cryptographically secured blocks** that acts as a journal of all the verified and accepted state transitions
- A **consensus algorithm** that decentralizes control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules
- A **game-theoretically sound incentivization scheme (e.g., proof-of-work costs plus block rewards)** to economically secure the state machine in an open environment
- One or more **open source software implementations** of the above (“clients”) - All or most of these components are usually combined in a single software client.

\* For example, in Bitcoin, the reference implementation is developed by the *Bitcoin Core* open source project and implemented as the *bitcoind* client. In Ethereum, rather than a reference implementation there is a *reference specification*, a mathematical description of the system in the Yellow Paper

# #Ethereum



Blockchain Components	Ethereum
<b>P2P Network</b>	Ethereum runs on the <i>Ethereum main network</i> , which is addressable on TCP port 30303, and runs a protocol called <i>Devp2p</i> . The dynamic communication between the peers is implemented with <i>Whisper</i> . <b>Whisper</b> is a part of the <b>Ethereum</b> P2P protocol suite that allows for messaging between users via the same network that the blockchain runs on.
<b>Consensus rules</b>	Ethereum's consensus rules are defined in the reference specification, the Yellow Paper - <a href="https://ethereum.github.io/yellowpaper/paper.pdf">https://ethereum.github.io/yellowpaper/paper.pdf</a>
<b>Transactions</b>	Ethereum transactions are network messages that include (among other things) a sender, recipient, value, and data payload.
<b>State machine</b>	Ethereum state transitions are processed by the <b>Ethereum Virtual Machine (EVM)</b> , a stack-based virtual machine that executes <i>bytecode</i> (machine-language instructions). <b>EVM programs</b> , called " <b>smart contracts</b> ," are written in high-level languages (e.g., Solidity or Vyper) and compiled to bytecode for execution on the EVM.
<b>Data structures</b>	Ethereum's state is stored locally on each node as a <i>database</i> (usually Google's LevelDB), which contains the transactions and system state in a serialized hashed data structure called a <i>Merkle Patricia Tree</i> . <b>Swarm</b> or IPFS may be used to store the Solidity smart contracts files.
<b>Consensus algorithm</b>	Ethereum uses Bitcoin's consensus model, Nakamoto Consensus, which uses sequential single-signature blocks, weighted in importance by PoW to determine the longest chain and therefore the current state. However, there are plans to move to a PoS weighted voting system, codenamed <b>Casper</b> , in the near future.
<b>Economic security</b>	Ethereum currently uses a PoW algorithm called Ethash, but this will eventually be dropped with the move to PoS at some point in the future.
<b>Clients</b>	Ethereum has several interoperable implementations of the client software, the most prominent of which are <b>Go-Ethereum (Geth)</b> and <b>Parity</b> .

# #Ethereum

**Further Reading-** The following references provide additional information on the technologies mentioned here:

- **Framework and consensus rules:**
  - The Ethereum Yellow Paper: <https://ethereum.github.io/yellowpaper/paper.pdf>
  - The Beige Paper, a rewrite of the Yellow Paper for a broader audience in less formal language: <https://github.com/chronaeon/beigepaper>
- **P2P Network:**
  - DΞVp2p network protocol: <http://bit.ly/2quAlTE>
- **State Machine:**
  - Ethereum Virtual Machine list of resources: <http://bit.ly/2PmtjiS>
- **Data structures:**
  - LevelDB database (used most often to store the local copy of the blockchain): <http://leveldb.org>
  - Merkle Patricia trees: <https://github.com/ethereum/wiki/wiki/Patricia-Tree>
- **Consensus Algorithms:**
  - Ethash PoW algorithm: <https://github.com/ethereum/wiki/wiki/Ethash>
  - Casper PoS v1 Implementation Guide: <http://bit.ly/2DyPr3l>
- **Clients:**
  - Go-Ethereum (Geth) client: <https://geth.ethereum.org/>
  - Parity Ethereum client: <https://parity.io/>

# #Ethereum

## From General-Purpose Blockchains to Decentralized Applications (DApps / DApps)

Ethereum started as a way to make a general-purpose blockchain that could be programmed for a variety of uses. But very quickly, Ethereum's vision expanded to become a platform for programming DApps. DApps represent a broader perspective than smart contracts. A DApp is, at the very least, a smart contract and a web user interface.

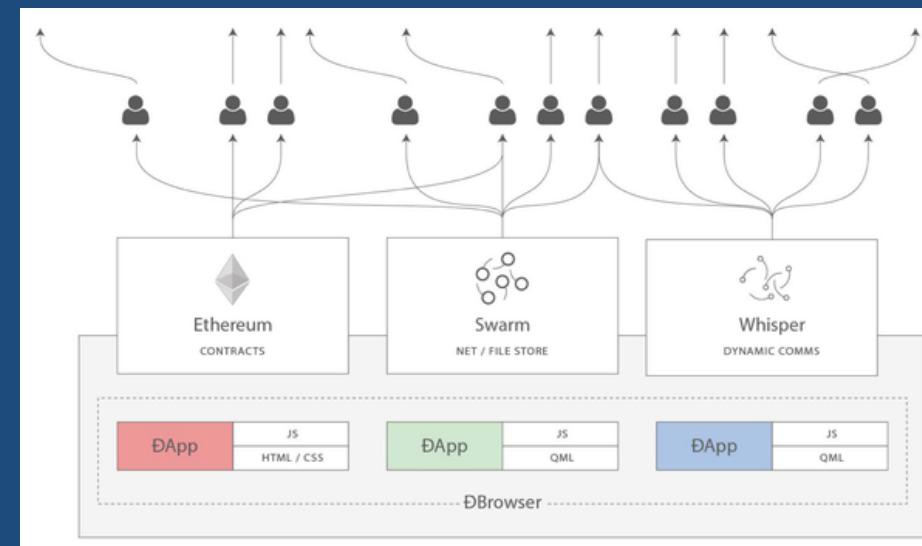
More broadly, a DApp is a web application that is built on top of open, decentralized, peer-to-peer infrastructure services.

A DApp is composed of at least:

- Smart contracts on a blockchain
- A web frontend user interface

In addition, many DApps include other decentralized components, such as:

- A decentralized (P2P) storage protocol and platform – Swarm
- A decentralized (P2P) messaging protocol and platform – Whisper



# #Ethereum – P2P Networks

P2P Name	Description
<b>Main Ethereum Network</b>	The main public Ethereum blockchain. Real ETH, real value, and real consequences.
<b>Ropsten Test Network</b>	Ethereum public test blockchain and network. ETH on this network has no value.
<b>Kovan Test Network</b>	Ethereum public test blockchain and network using the Aura consensus protocol with proof of authority (federated signing). ETH on this network has no value. The Kovan test network is supported by Parity only. Other Ethereum clients use the Clique consensus protocol, which was proposed later, for proof of authority-based verification.
<b>Rinkeby Test Network</b>	Ethereum public test blockchain and network, using the Clique consensus protocol with proof of authority (federated signing). ETH on this network has no value.
<b>localhost 8545</b>	Connects to a node running on the same computer as the browser. The node can be part of any public blockchain (main or testnet), or a private testnet.
<b>Custom RPC</b>	Allows you to connect MetaMask to any node with a Geth-compatible Remote Procedure Call (RPC) interface. The node can be part of any public or private blockchain.

# #Ethereum

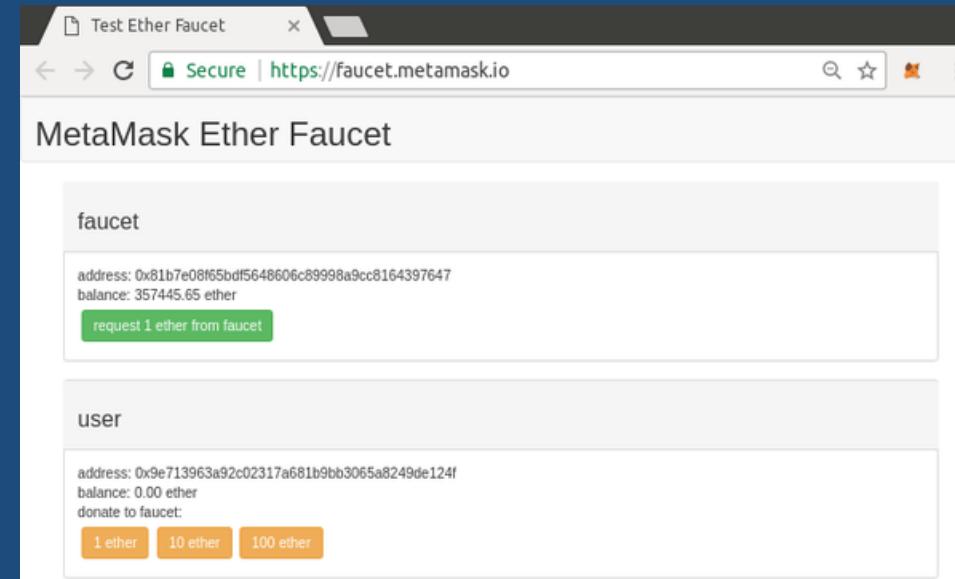
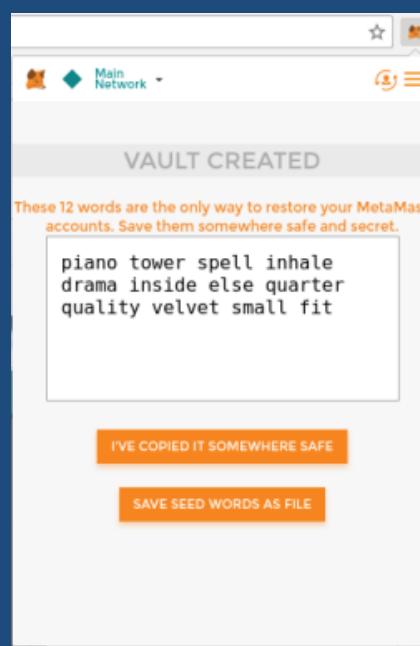
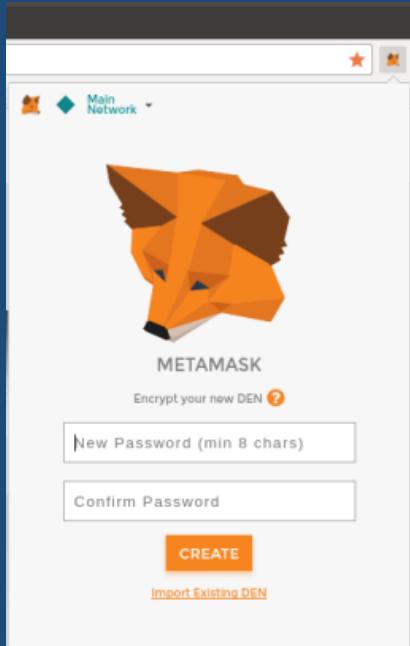
## *Ether denominations and unit names*

	<b>Ethereum Wallets</b>
<b>MetaMask</b> <a href="https://metamask.io/">https://metamask.io/</a>	<p>It is a browser extension wallet that runs in your browser (Chrome, Firefox, Opera, or Brave Browser). It is easy to use and convenient for testing, as it is able to connect to a variety of Ethereum nodes and test blockchains.</p> <p>MetaMask is a web-based wallet.</p>
<b>Jaxx</b>	<p>It is a multiplatform and multicurrency wallet that runs on a variety of operating systems, including Android, iOS, Windows, macOS, and Linux. It is often a good choice for new users as it is designed for simplicity and ease of use. Jaxx is either a mobile or a desktop wallet, depending on where you install it.</p>

Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	$10^3$	Babbage	Kilowei or femtoether
1,000,000	$10^6$	Lovelace	Megawei or picoether
1,000,000,000	$10^9$	Shannon	Gigawei or nanoether
1,000,000,000,000	$10^{12}$	Szabo	Microether or micro
1,000,000,000,000,000	$10^{15}$	Finney	Milliether or milli
1,000,000,000,000,000,000	$10^{18}$	Ether	Ether
1,000,000,000,000,000,000,000	$10^{21}$	Grand	Kiloether
1,000,000,000,000,000,000,000,000	$10^{24}$		Megaether

# #Ethereum – DEMO with MetaMask and Remix IDE in Mozilla Firefox

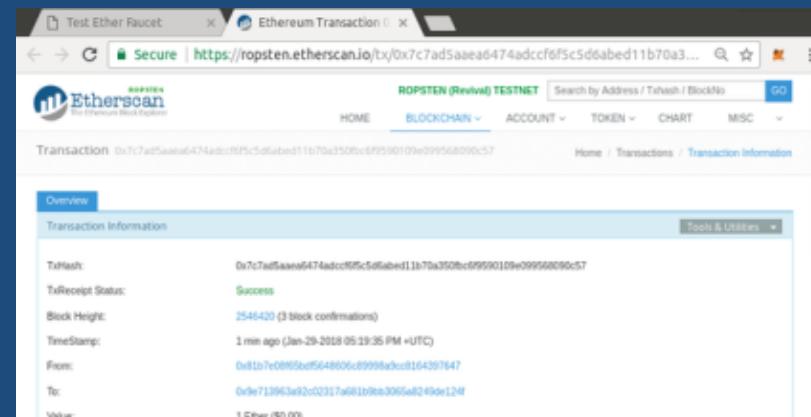
## D1. MetaMask Password and Mnemonic setup D2. MetaMask buy Ether from Ropsten Test Network's Faucet



There are two types of accounts:

- **Externally owned account (EOAs):** an account controlled by a private key, and if you own the private key associated with the EOA you have the ability to send ether and messages from it.
- **Contract:** an account that has its own code, and is controlled by code.

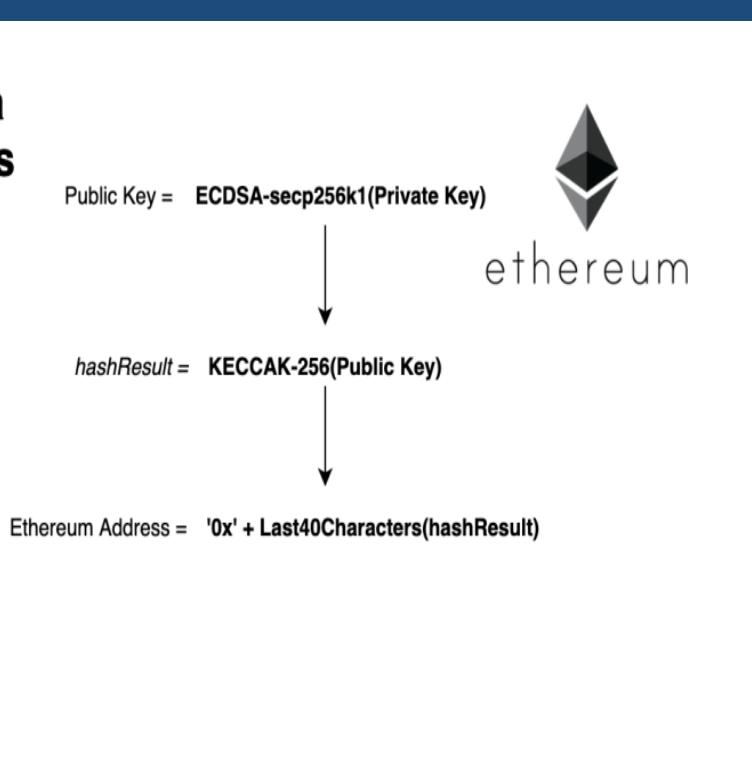
## D3. Transaction in RopstenEtherscan.io



# #Ethereum – Wallet Address Generation is (a little bit) different than Bitcoin

## How to generate a Ethereum Address

Ethereum addresses can be created using open-source libraries based on algorithmic hash functions. More information about Ethereum address generation at <http://mycrypto.tools>



1 - Generate Private Key using open source libraries such as Ethereumj or SHA256 hash function with a randomly generated number.

**0: Private Key:**

**09e910621c2e988e9f7f6ffcd7024f54ec1461fa6e86a4b545e9e1fe21c28866**

2. - Generate Public key using ECDSA – secp256k1 – algorithm applied to Private Key. The public key is a point on the Elliptic Curve Algorithm. It has x and y coordinates which are used to create Ethereum address.

**2.1: Public Key:**

**048e66b3e549818ea2cb354fb70749f6c8de8fa484f7530fc447d5fe80a1c424e4  
f5ae648d648c980ae7095d1efad87161d83886ca4b6c498ac22a93da5099014a**

-Apply keccak256 – Ethereum hash function – using x and y coordinates to create Ethereum address.

**2.2: Ethereum Address: 0x00B54E93EE2EBA3086A55F4249873E291D1AB06C**  
See address on Ethereum

Blockchain **0x00B54E93EE2EBA3086A55F4249873E291D1AB06C**

Dev Overview – it is modified as implementation but it is of the ETH creator:  
<https://eth.wiki/en/howto/ethereum-development-tutorial>

Dev Resources:

<https://ethereum.org/en/developers/docs/programming-languages/>

JAVA:

<https://ethereum.org/en/developers/docs/programming-languages/java/>

<https://github.com/web3j/web3j>

<https://kauri.io/#communities/Java%20Ethereum/connecting-to-an-ethereum-client-with-java-eclips/>

Node.js – JavaScript:

<https://codeforgeek.com/how-to-generate-ethereum-private-key-and-address/>

## 2. Ethereum Blockchain Block – ETH Transaction Schematic

From web3 console, execute the following (depending on the client, might need to use 'eth.getRawTransaction' instead)

```
eth.getTransaction('0x14a298c1eea89f42285948b7d51eeac2876ca7406c9784b9b90dd3591d156d64').raw
```

This should give

```
"0xf86b80850ba43b7400825208947917bc33eea648809c285607579c9919fb864f8f8703baf82d03a0008025a0067940651530790861714b2e8fd8b080361d1ada048189000c07a66848afde46a069b041db7c29dbcc6becf42017ca7ac086b12bd53ec8ee494596f790fb6a0a69"
```

which is 109 bytes. If we parse the data:

f86b length

80 nonce (0: this is the minimum an account can have)

85 0ba43b7400 gas price

82 5208 gas limit (this is fixed for simple payments)

94 7917bc33eea648809c285607579c9919fb864f8f (address, always 20 bytes)

87 03baf82d03a000 (value, in theory this can be shrunken to zero)

80 (data, already zero length)

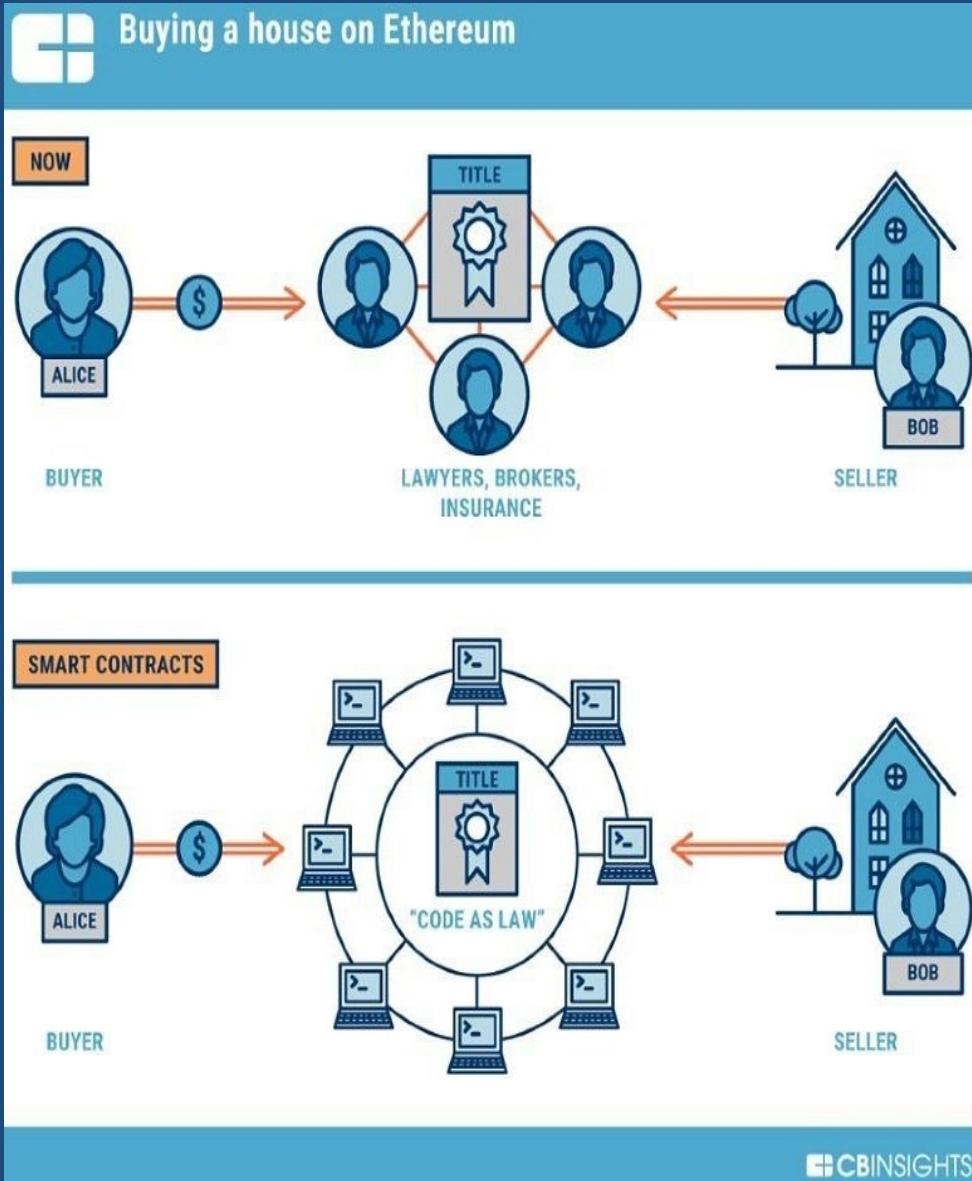
25 (V, one byte)

a0 067940651530790861714b2e8fd8b080361d1ada048189000c07a66848afde46 (R)

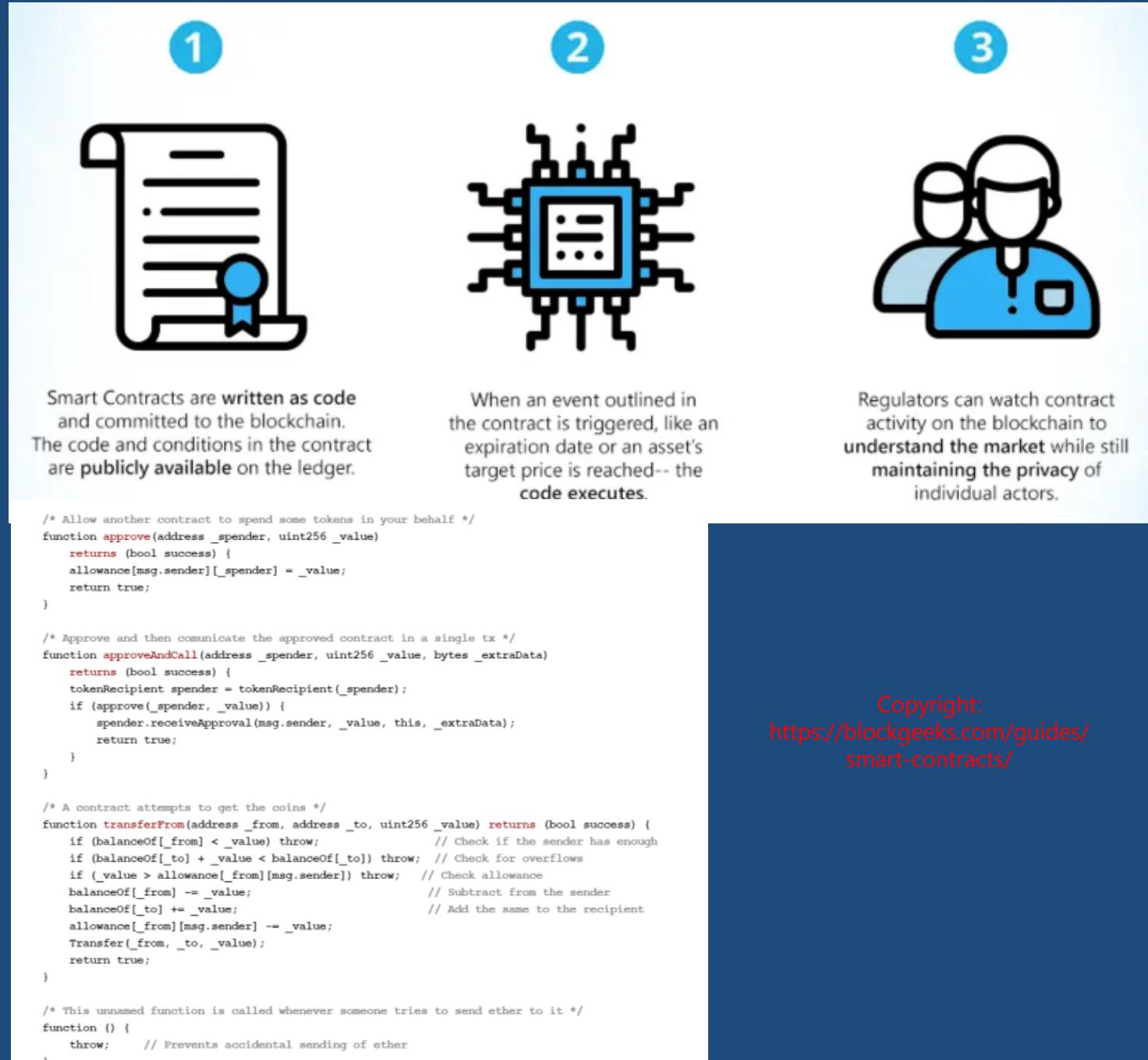
a0 69b041db7c29dbcc6becf42017ca7ac086b12bd53ec8ee494596f790fb6a0a69 (S)

From this sample, there isn't much more to be shortened. A smaller value can reduce the value field to one byte. Lower gas price may shrink one or two bytes further. Thus it seems a realistic "minimum" transaction size is should be greater than 100 bytes (109 - 7 -2).

# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?



Copyright: <https://www.investopedia.com/news/ethereum-smart-contracts-vulnerable-hacks-4-million-ether-risk/>



# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

## Autonomy

You're the one making the agreement; there's no need to rely on a broker or lawyer

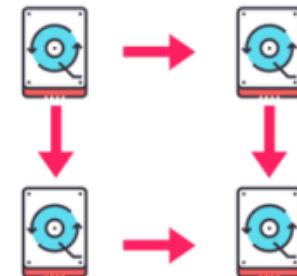
1



## Backup

On the blockchain, your documents are duplicated many times over

2



## Trust

Your documents are encrypted on a shared ledger

3

## Accuracy

Smart contracts are not only faster and cheaper but also avoid the errors that come from manually filling out heaps of forms.

4



## Savings

Smart contracts save you money since they knock out the presence of an intermediary



5

[www.Blockgeeks.com](http://www.Blockgeeks.com)

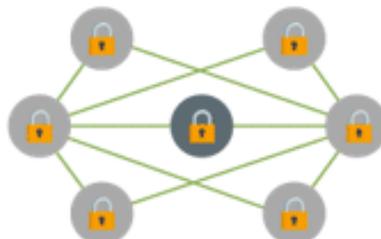


# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

## Physical Contracts



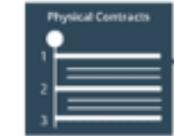
Alice +  
Bob  
Blockchain/permissioned ledger,  
programming & encryption



Transacting parties  
Individuals or Institutions

## Smart Contracts

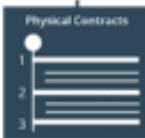
Lower operational  
Overheads & costs leading  
To economical financial  
products



Smart Contracts  
A Software program  
on the distributed  
Ledger, allowing an  
immutable & Verifiable  
records of all Contracts &  
Transactions

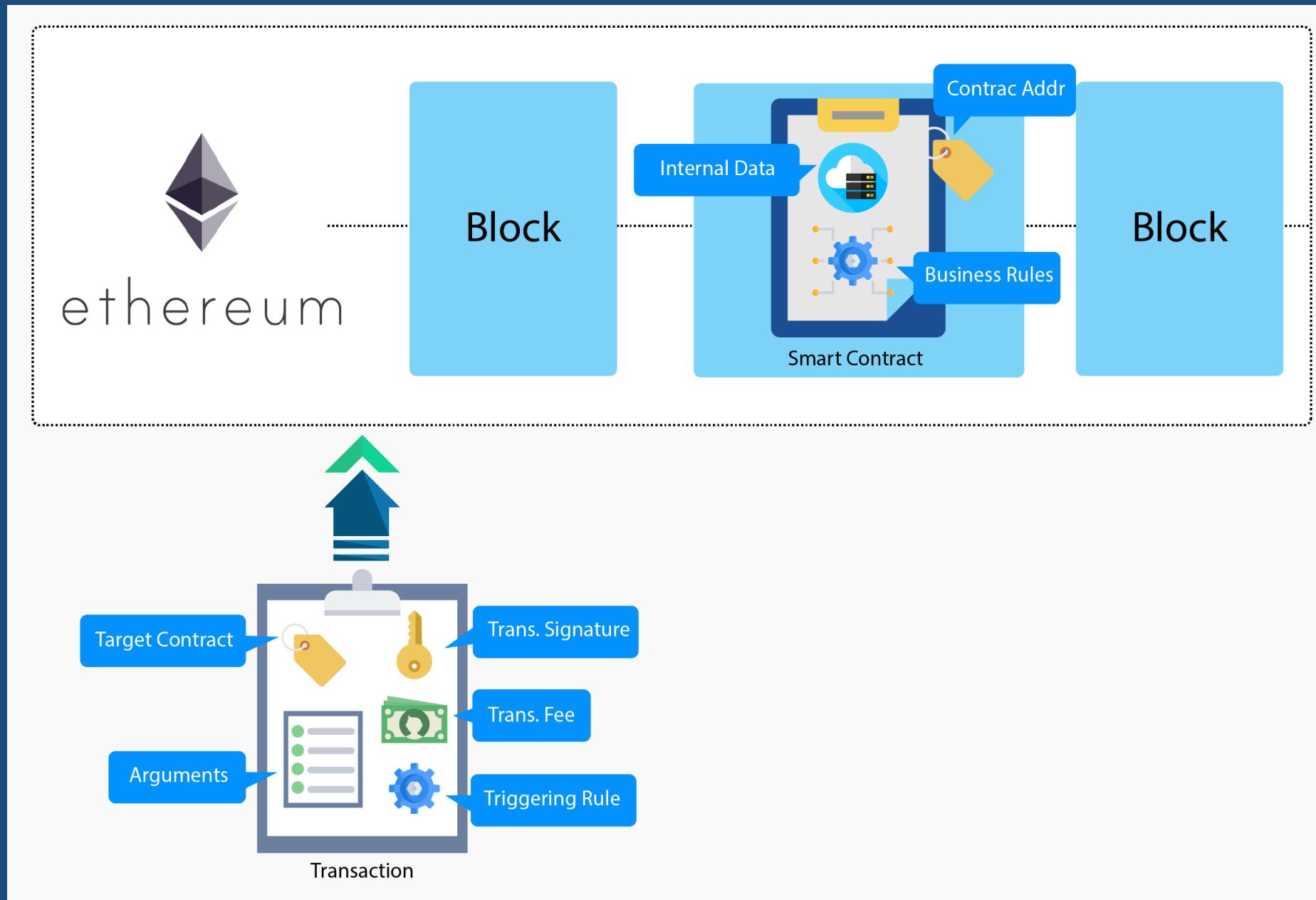
Banks, Insurers,  
Capital Markets  
Act as custodians of assets,  
validators & authorities of all  
contracts & transactions

Reduced administration  
& service costs Owing  
to automation & ease  
of compliance & reporting



Central authorities that keep a tab on the system with a  
wide ranging read-access to blockchain

# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?



# #Ethereum – DEMO with MetaMask and Remix IDE in Mozilla Firefox

## D4. Develop, Deploy and Run Smart Contract in Remix IDE @ same address for all MetaMask Wallets

The screenshot shows the Mozilla Firefox browser interface with the Remix IDE extension open. The address bar displays the URL <https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=0.8.0>. The Remix IDE window contains a code editor with Solidity code for a 'Faucet' contract, which handles withdrawals. Below the code editor is a hex dump of the generated EVM bytecode.

**FILE EXPLORERS**

- browser
- contracts
  - artifacts
  - 1\_Storage.sol
  - 2\_Owner.sol
  - 3\_Ballot.sol
  - Faucet.sol
- scripts
  - deploy\_web3.js
  - deploy\_ether.js
  - withdraw\_faucet.js
- tests
- README.txt

**Faucet.sol**

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.8.0;

// Our first contract is a faucet!
contract Faucet {
    // Accept any incoming amount
    receive () external payable {}

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {
        // Limit withdrawal amount
        require(withdraw_amount <= 10000000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }
}
```

**Activity**

Event	Value
Give Right To Vote	-0 ETH
Feb 19 · remix.ethereum.org	-0 ETH
Receive	1 ETH
Feb 19 · From: 0x81b7...7647	1 ETH

# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

There are two types of accounts:

- **Externally owned account (EOAs)**: an account controlled by a private key, and if you own the private key associated with the EOA you have the ability to send ether and messages from it.
- **Contract**: an account that has its own code, and is controlled by code.

\*\*\* By default, the Ethereum execution environment is lifeless; nothing happens and the state of every account remains the same. However, ***any user can trigger an action by sending a transaction from an Externally Owned Account, setting Ethereum's wheels in motion.*** If the destination of the transaction is another EOA, then the transaction may transfer some Ether but otherwise does nothing. ***However, if the destination is a contract, then the contract in turn activates, and automatically runs its code.***

***The (smart contract) code (running in EVM) has the ability to read/write to its own internal storage (a database mapping 32-byte keys to 32-byte values), read the storage of the received message, and send messages to other contracts, triggering their execution in turn.*** Once execution stops, and all sub-executions triggered by a message sent by a contract stop (this all happens in a deterministic and synchronous order, ie. a sub-call completes fully before the parent call goes any further), the execution environment halts once again, until woken by the next transaction.

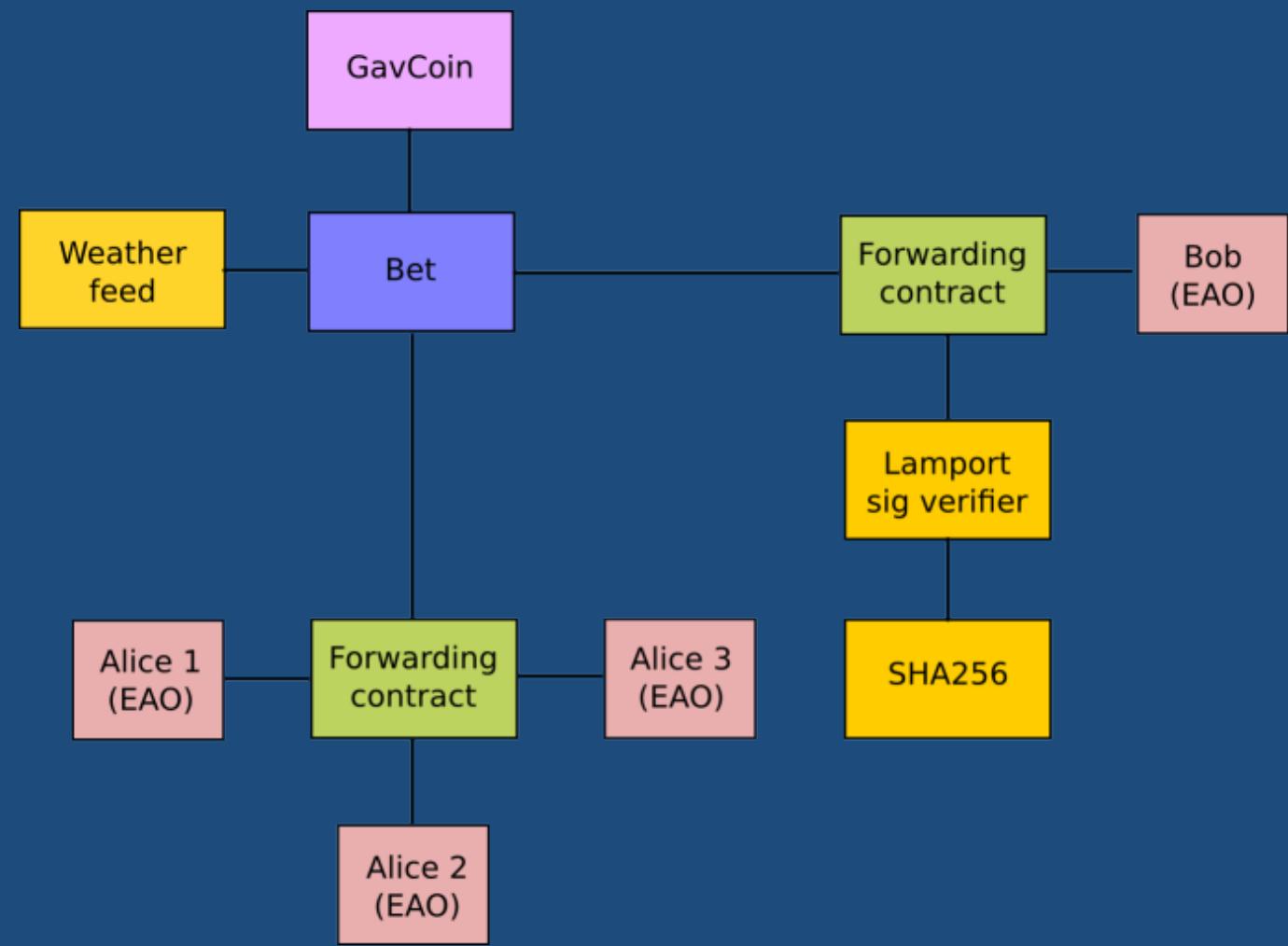
# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

**As an example, consider a situation where Alice and Bob are betting 100 GavCoin that the temperature in San Francisco will not exceed 35°C at any point in the next year.**

However, Alice is very security-conscious, and as her primary account uses a forwarding contract which only sends messages with the approval of two out of three private keys.

Bob is paranoid about quantum cryptography, so he uses a forwarding contract which passes along only messages that have been signed with Lamport signatures alongside traditional ECDSA (but because he's old fashioned, he prefers to use a version of Lamport sigs based on SHA256, which is not supported in Ethereum directly).

The betting contract itself needs to fetch data about the San Francisco weather from some contract, and it also needs to talk to the GavCoin contract when it wants to actually send the GavCoin to either Alice or Bob (or, more precisely, Alice or Bob's forwarding contract).

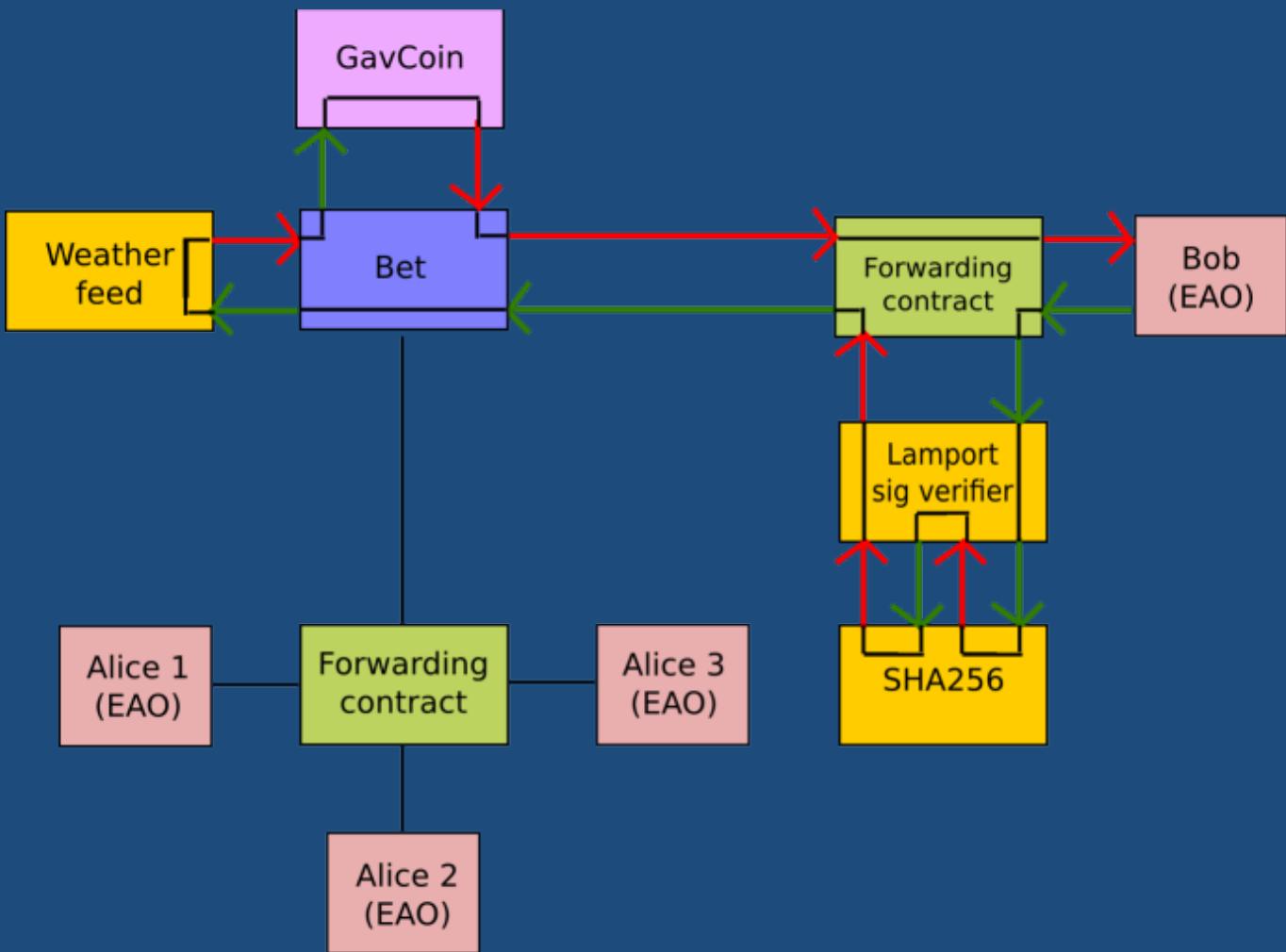


# #Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

When Bob wants to finalize the bet, the following steps happen:

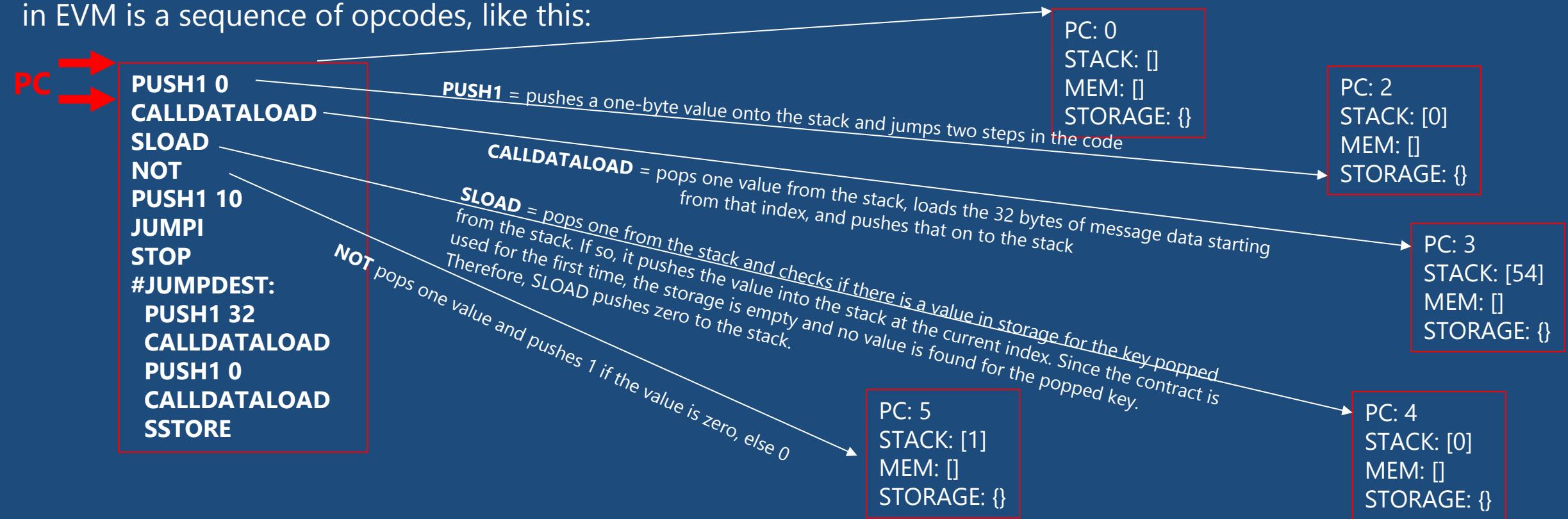
1. A transaction is sent, triggering a message from Bob's EOA to Bob's forwarding contract.
2. Bob's forwarding contract sends the hash of the message and the Lamport signature to a contract which functions as a Lamport signature verification library.
3. The Lamport signature verification library sees that Bob wants a SHA256-based Lamport sig, so it calls the SHA256 library many times as needed to verify the signature.
4. Once the Lamport signature verification library returns 1, signifying that the signature has been verified, it sends a message to the contract representing the bet.
5. The bet contract checks the contract providing the San Francisco temperature to see what the temperature is.
6. The bet contract sees that the response to the messages shows that the temperature is above 35°C, so it sends a message to the GavCoin contract to move the GavCoin from its account to Bob's forwarding contract.

\* Note that the GavCoin is all “stored” as entries in the GavCoin contract’s database; the word “account” in the context of step 6 simply means that there is a data entry in the GavCoin contract storage with a key for the bet contract’s address and a value for its balance. After receiving this message, the GavCoin contract decreases this value by some amount and increases the value in the entry corresponding to Bob’s forwarding contract’s address.



# #Ethereum Blockchain Smart Contract running in EMV and the State Machine

Computation in the EVM is done using a stack-based bytecode language that is like a cross between Bitcoin Script, traditional assembly and Lisp (the Lisp part being due to the recursive message-sending functionality). A program in EVM is a sequence of opcodes, like this:



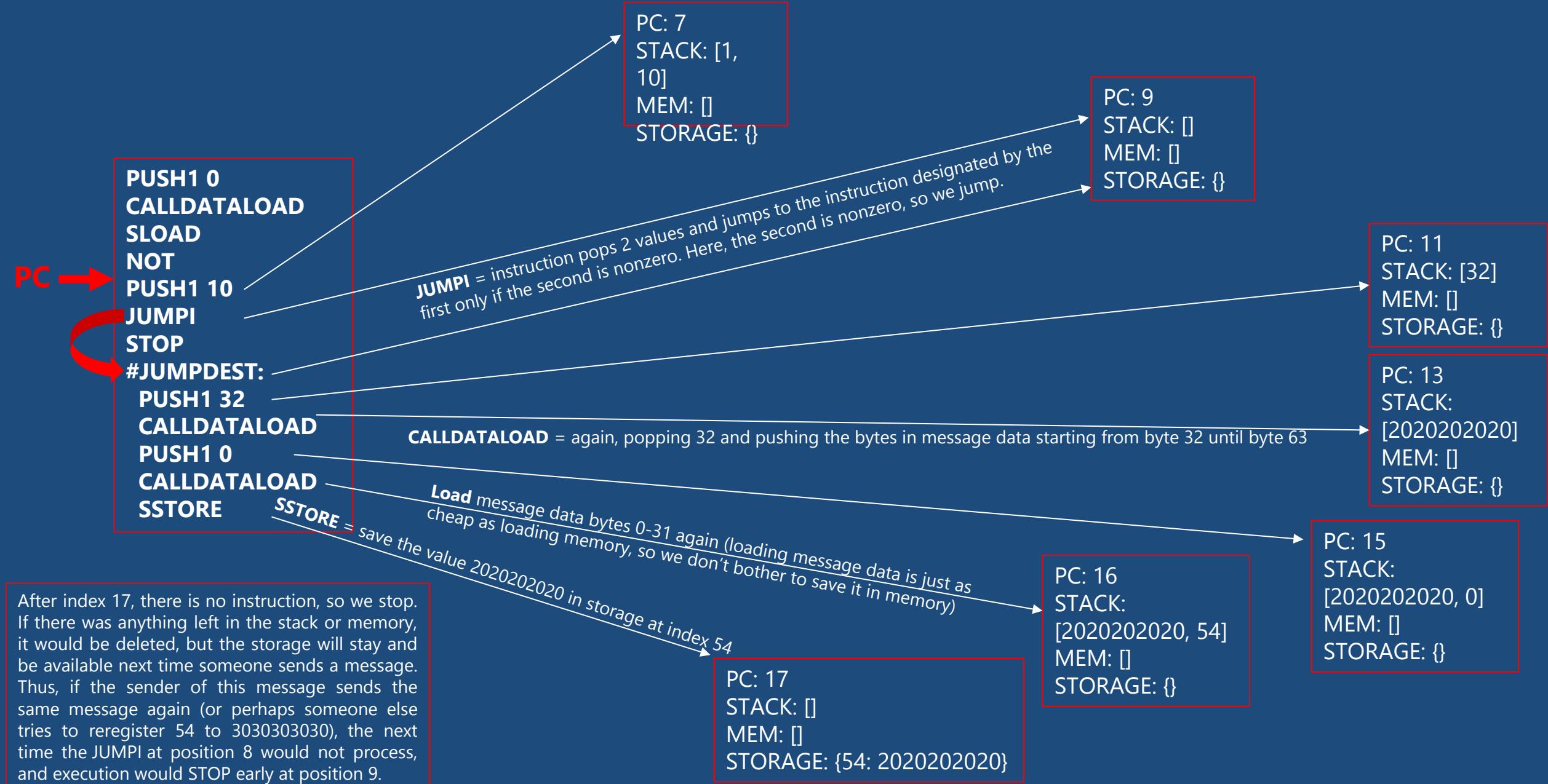
The purpose of this particular contract is to serve as a name registry; anyone can send a message containing 64 bytes of data, 32 for the key and 32 for the value. The contract checks if the key has already been registered in storage, and if it has not been then the contract registers the value at that key.

During execution, an infinitely expandable byte-array called "memory", the "program counter" pointing to the current instruction, and a stack of 32-byte values is maintained. At the start of execution, memory and stack are empty and the PC is zero. Now, let us suppose the contract with this code is being accessed for the first time, and a message is sent in with 123 wei ( $10^{18}$  wei = 1 ether) and 64 bytes of data where the first 32 bytes encode the number 54 and the second 32 bytes encode the number 2020202020.

{K = 54, V = 2020202020}

# #Ethereum Blockchain Smart Contract running in EMV and the State Machine

{**K** = 54, **V** = 2020202020}



# #Ethereum Blockchain Smart Contract running in EMV and the State Machine

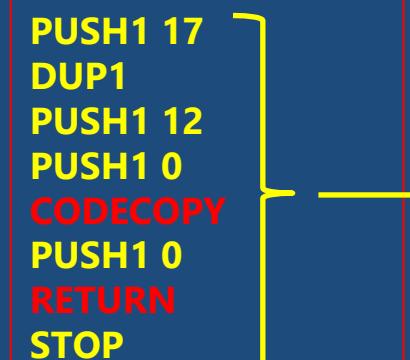
Fortunately, you do not have to program in low-level assembly; a high-level language especially designed for writing contracts, known as Solidity, exists to make it much easier for you to write contracts (there are several others, too, including LLL, Vyper and Mutan, which you may find easier to learn or use depending on your experience).

**Any code you write in these languages gets compiled into EVM, and to create the contracts you send the transaction containing the EVM bytecode.**

There are two types of transactions:

- a **sending transaction** and
- a **contract creating transaction**.

A **sending transaction** is a standard transaction, containing a receiving address, an ether amount, a data byte-array and some other parameters, and a signature from the private key associated with the sender account.

A **contract creating transaction** looks like a standard transaction, **except the receiving address is blank**. When a contract creating transaction makes its way into the blockchain, the data byte-array in the transaction is interpreted as EVM code, and the value returned by that EVM execution is taken to be the code of the new contract; hence, you can have a transaction do certain things during initialization. The address of the new contract is deterministically calculated based on the sending address and the number of times that the sending account has made a transaction before (this value, called the account nonce, is also kept for unrelated security reasons). Thus, **the full code that you need to put onto the blockchain to produce the above name registry** is as follows: 

```
PUSH1 17  
DUP1  
PUSH1 12  
PUSH1 0  
CODECOPY  
PUSH1 0  
RETURN  
STOP  
  
PUSH1 0  
CALLDATALOAD  
SLOAD  
NOT  
PUSH1 10  
JUMPI  
STOP  
#JUMPDEST:  
PUSH1 32  
CALLDATALOAD  
PUSH1 0  
CALLDATALOAD  
SSTORE
```

The key opcodes are **CODECOPY**, copying the 17 bytes of code starting from byte 12 into memory starting at index 0, and **RETURN**, returning memory bytes 0-16, ie. code bytes 12-28. Code bytes 12-28 are, of course, the actual code as we saw previously.

# #Ethereum Blockchain Smart Contract running in EMV and Gas

**One important aspect of the way the EVM works is that every single operation that is executed inside the EVM is actually simultaneously executed by every full node.**

This is a necessary component of the Ethereum 1.0 consensus model, and has the benefit that any contract on the EVM can call any other contract at almost zero cost, but also has the drawback that **computational steps on the EVM are very expensive**.

- Acceptable uses of the EVM include running business logic ("if this then that") and verifying signatures and other cryptographic objects; at the upper limit of this are applications that verify parts of other blockchains (eg. a decentralized ether-to-bitcoin exchange);
- Unacceptable uses include using the EVM as a file storage, email or text messaging system, anything to do with graphical interfaces, and applications best suited for cloud computing like genetic algorithms, graph analysis or machine learning.

In order to prevent deliberate attacks and abuse, the Ethereum protocol charges a fee per computational step. The fee is market-based, though mandatory in practice; a floating limit on the number of operations that can be contained in a block forces even miners who can afford to include transactions at close to no cost to charge a fee commensurate with the cost of the transaction to the entire network; see [the whitepaper section on fees](#) for more details on the economic underpinnings of our fee and block operation limit system.

The way the fee works is as follows. Every transaction must contain, alongside its other data, a **GASPRICE** and **STARTGAS** value. **STARTGAS** is the amount of "gas" that the transaction assigns itself, and **GASPRICE** is the fee that the transaction pays per unit of gas; thus, when a transaction is sent, the first thing that is done during evaluation is subtracting **STARTGAS \* GASPRICE** wei plus the transaction's value from the sending account balance. **GASPRICE** is set by the transaction sender, but miners will likely refuse to process transactions whose **GASPRICE** is too low.

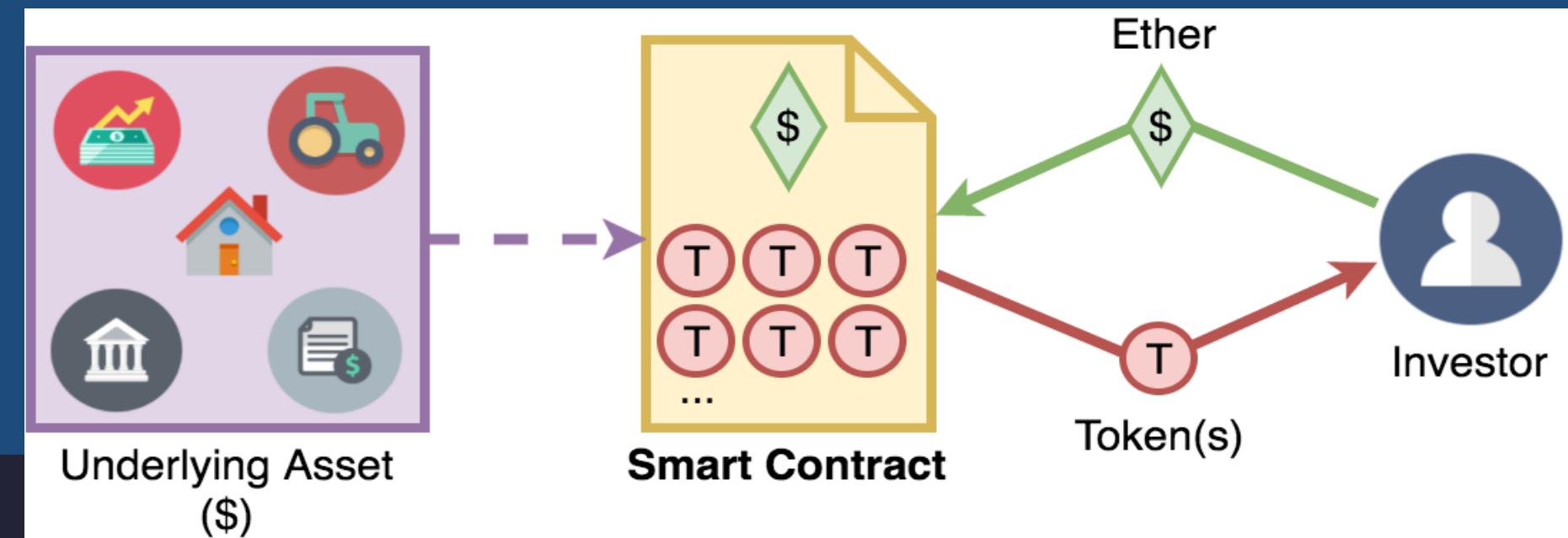
# #Ethereum How Smart Contract is working?

The basic architecture of the **EVM** ([Ethereum Virtual Machine](#)) that runs smart contracts is that all calls to the contract are executed as a transaction where the ether required for a contract method executed is transferred from the calling account address to the contract account address.

The contract code resides on the contract address on the blockchain and expects the calls to come in as transactions carrying the method parameter data along with the transaction as "input".

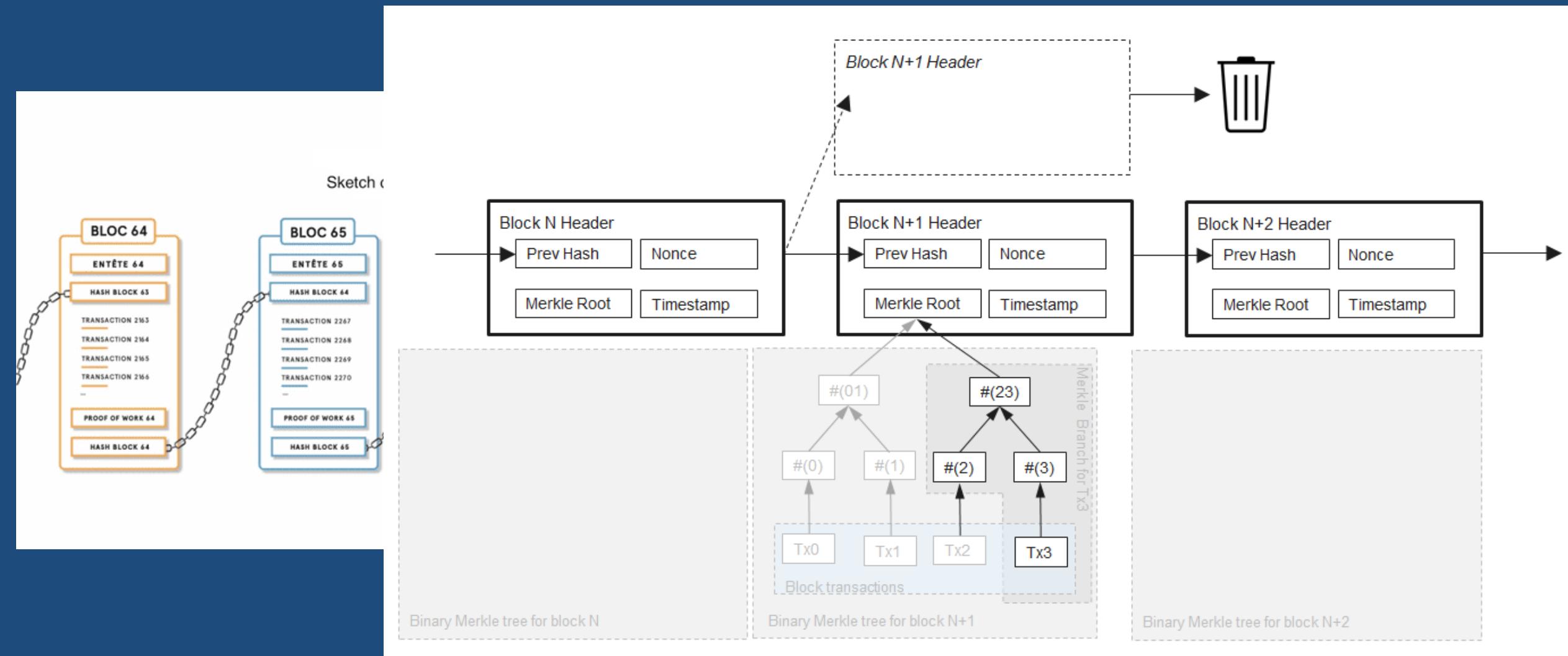
To enable a standard format for all clients, the method name, and parameters need to be marshaled in a recommended format.

<https://medium.com/programmers-blockchain/creating-your-first-blockchain-with-java-part-2-transactions-2cdac335e0ce>

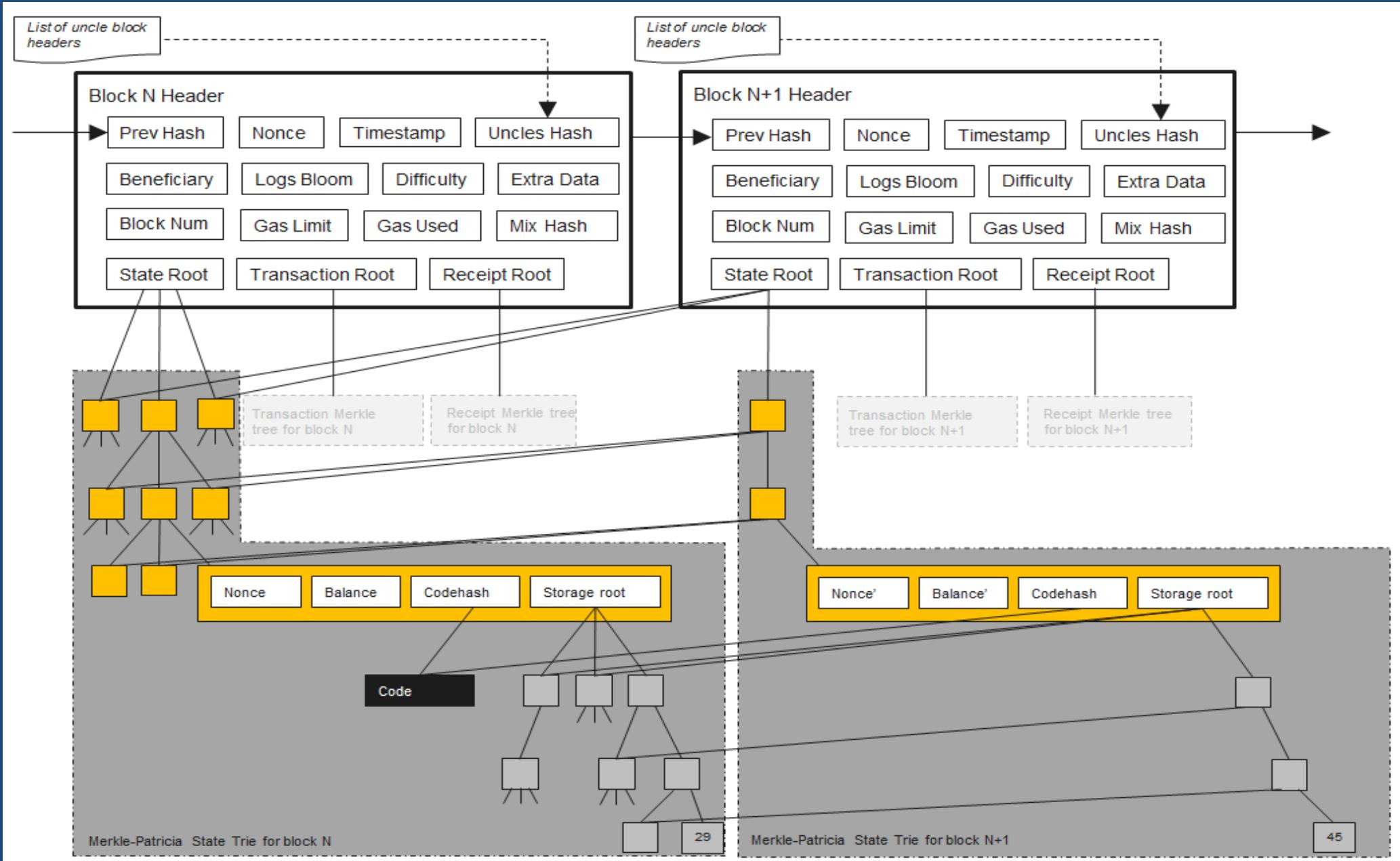


```
// in HTML page:  
function start(){  
    var Web3 = require('web3');  
    var web3 = new Web3();  
    web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'));  
    var abi = [...];  
    var corecontractContract = web3.eth.contract(abi);  
  
    var corecontractContractInst = corecontractContract.at('0x0e22a4f27c2fc3b47e66b70fada85e1c4ca33681');  
    console.log(corecontractContract.createCustomer(287187, "custName", 13243244, 1213334));  
}
```

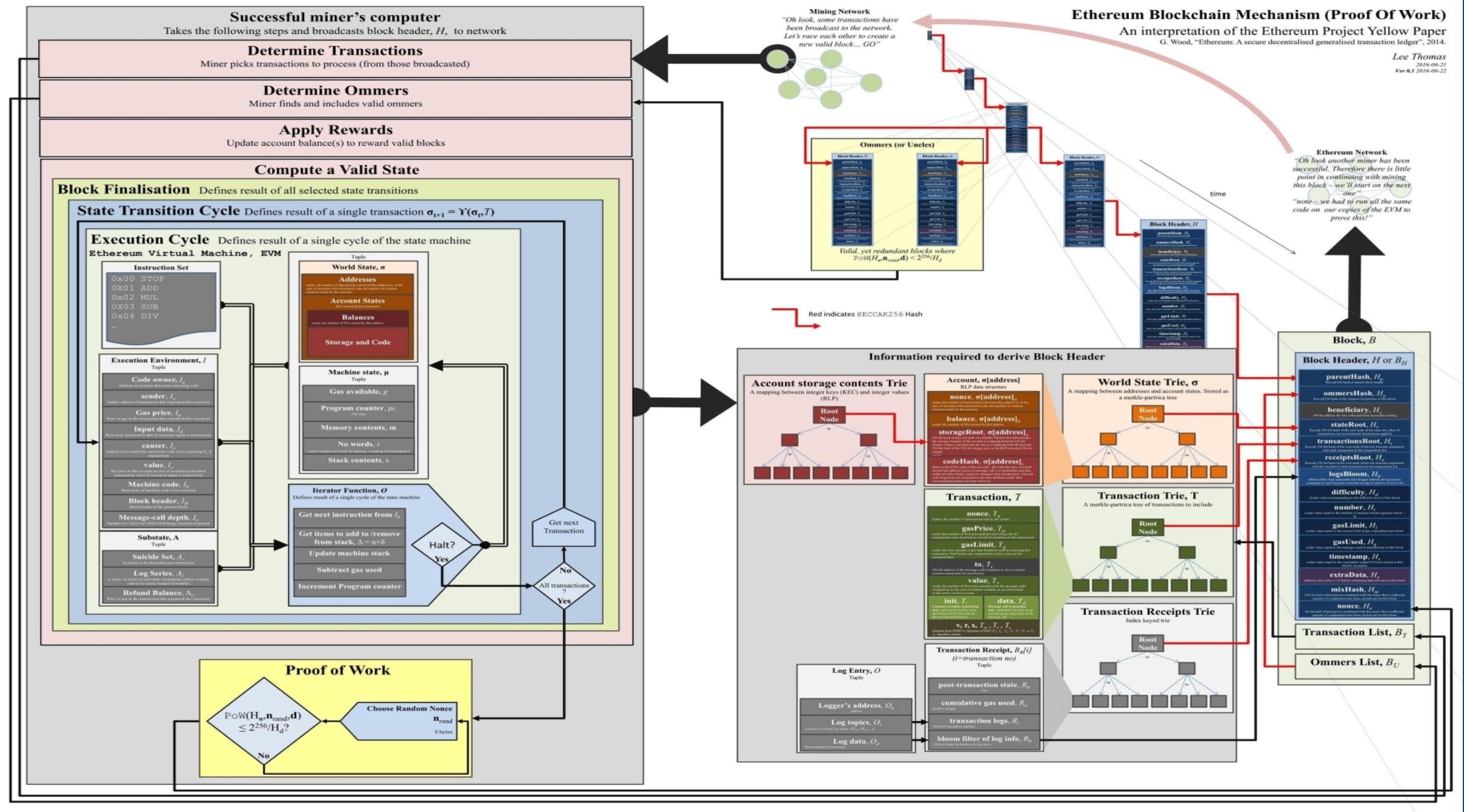
## 2. Ethereum Blockchain Block – Bitcoin Schematic



## 2. Ethereum Blockchain Block – ETH Schematic



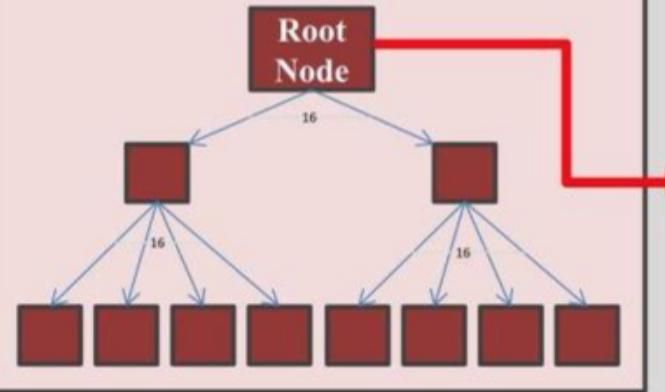
# 2. Ethereum Blockchain Block – ETH Schematic



## 2. Ethereum

### Account storage contents Trie

A mapping between integer keys (KEC) and integer values (RLP)



### Account, $\sigma[\text{address}]$

RLP data structure

#### nonce, $\sigma[\text{address}]_n$

scalar; the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

#### balance, $\sigma[\text{address}]_b$

scalar; the number of Wei owned by this address

#### storageRoot, $\sigma[\text{address}]_s$

256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values

#### codeHash, $\sigma[\text{address}]_c$

Hash of the EVM code of this account - the code that gets executed should this address receive a message call; it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their corresponding hashes for later retrieval.

### Transaction, $T$

#### nonce, $T_n$

Scalar; the number of transactions sent by the sender

#### gasPrice, $T_p$

scalar; the number of Wei to be paid per unit of gas for all computation costs incurred as a result of execution of this transaction

#### gasLimit, $T_g$

scalar; the max amount of gas that should be used in executing this transaction. Paid before any computation is done, may not be increased later.

#### to, $T_t$

160-bit address of the message call's recipient or, for a contract creation transaction,  $\emptyset$  (zero bytes).

#### value, $T_v$

scalar; the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account

#### init, $T_i$

Contract creation transaction only; unlimited size byte array specifying the EVM-code for the account initialisation

#### data, $T_d$

Message call transaction only; unlimited size byte array specifying the input data of the message call

#### v, r, s, $T_w, T_r, T_s$

Outputs from EDSCA signature of KEC( $T_n, T_p, T_g, T_t, T_i, T_d$ ,  $T_s$ ) or  $T_d$ ); identifies sender.

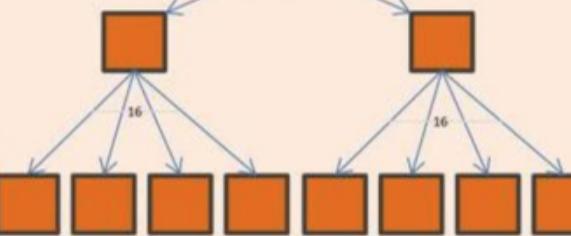
### Transaction Receipt, $B_R[i]$

( $i = \text{transaction no}$ )

### World State Trie, $\sigma$

A mapping between addresses and account states. Stored as a merkle-partricia tree

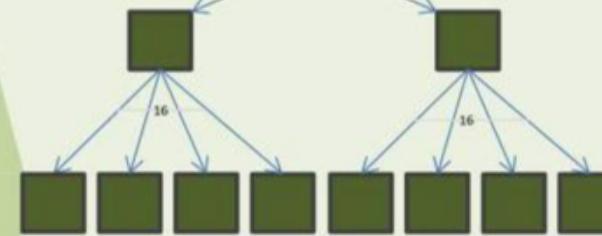
### Root Node



### Transaction Trie, $T$

A merkle-partricia tree of transactions to include

### Root Node

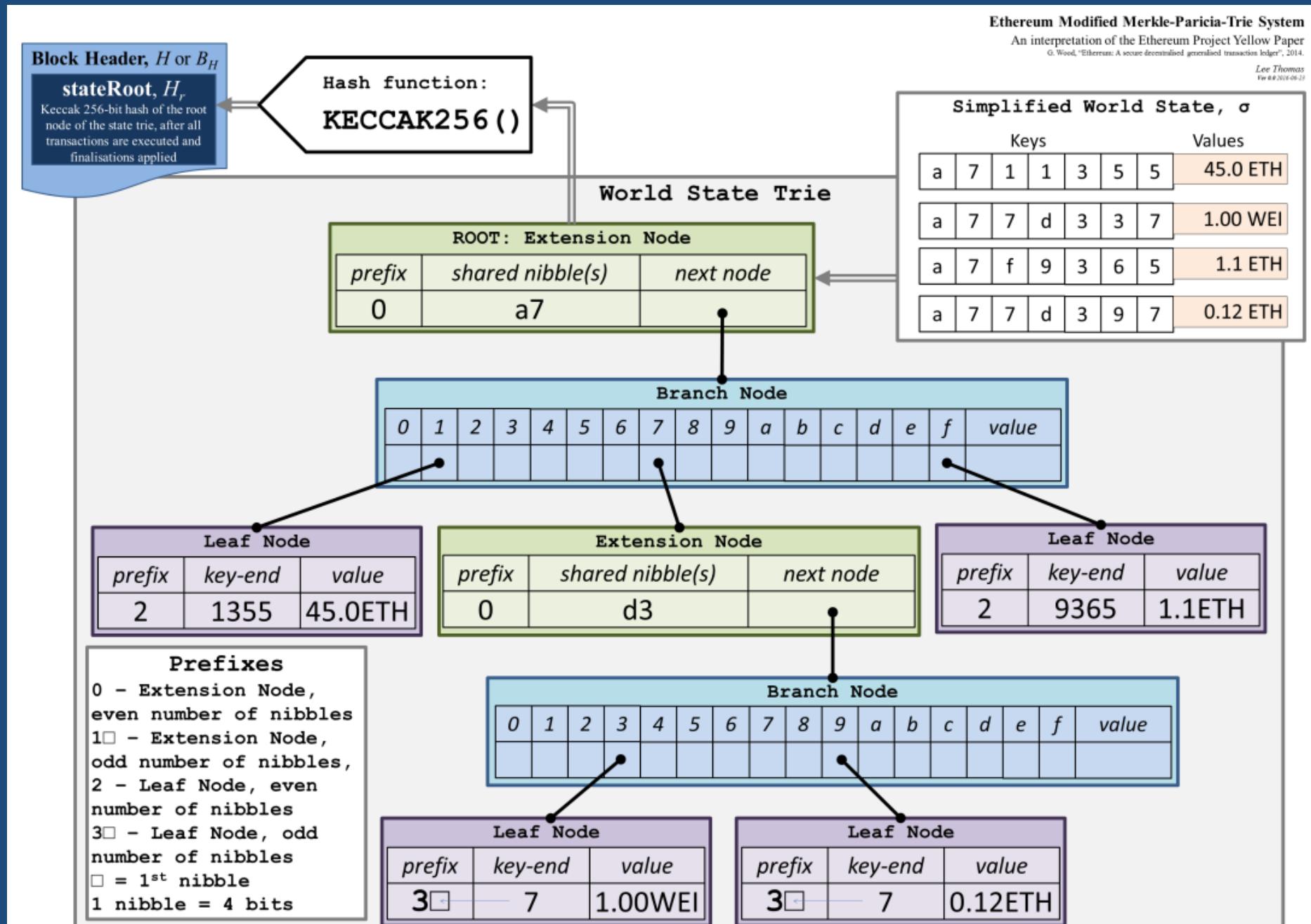


### Transaction Receipts Trie

Index keyed trie

### Root Node

## 2. Ethereum Blockchain Block – ETH Schematic



## 2. Ethereum Blockchain Block – ETH Schematic

```
type BlockChain struct {
    config *params.ChainConfig // chain & network configuration

    hc      *HeaderChain
    chainDb ethdb.Database
    rmLogsFeed event.Feed
    chainFeed event.Feed
    chainSideFeed event.Feed
    chainHeadFeed event.Feed
    logsFeed event.Feed
    scope    event.SubscriptionScope
    genesisBlock *types.Block

    mu      sync.RWMutex // global mutex for locking chain operations
    chainmu sync.RWMutex // blockchain insertion lock
    procmu sync.RWMutex // block processor lock

    checkpoint int        // checkpoint counts towards the new checkpoint
    currentBlock *types.Block // Current head of the block chain
    currentFastBlock *types.Block // Current head of the fast-sync chain (may be above the block chain!)

    stateCache state.Database // State database to reuse between imports (contains state cache)
    bodyCache  *lru.Cache    // Cache for the most recent block bodies
    bodyRLPCache *lru.Cache // Cache for the most recent block bodies in RLP encoded format
    blockCache *lru.Cache   // Cache for the most recent entire blocks
    futureBlocks *lru.Cache // future blocks are blocks added for later processing

    quit chan struct{} // blockchain quit channel
    running int32       // running must be called atomically
    // procInterrupt must be atomically called
    procInterrupt int32 // interrupt signaler for block processing
    wg      sync.WaitGroup // chain processing wait group for shutting down

    engine consensus.Engine
    processor Processor // block processor interface
    validator Validator // block and state validator interface
    vmConfig vm.Config

    badBlocks *lru.Cache // Bad block cache
}
```

```
// Block represents an entire block in the Ethereum blockchain.
type Block struct {
    header      *Header
    uncles     []*Header
    transactions Transactions

    // caches
    hash atomic.Value
    size atomic.Value

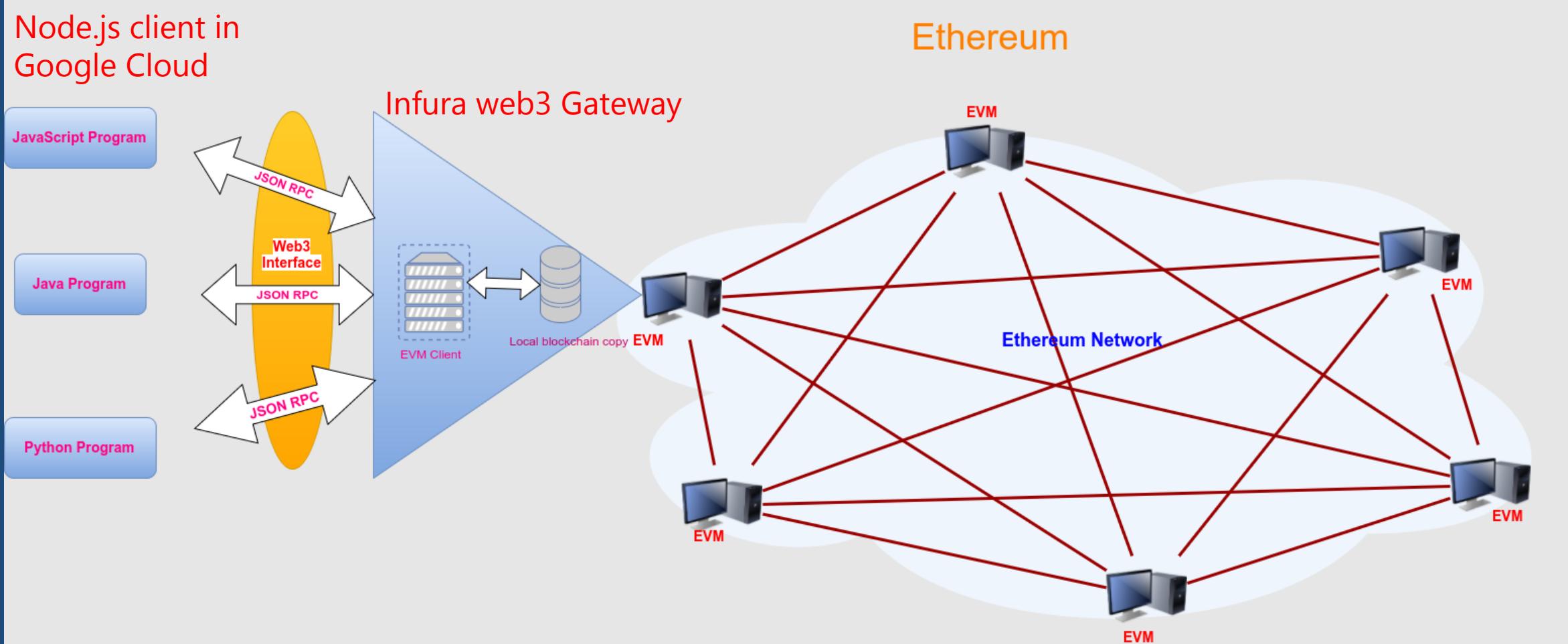
    // Td is used by package core to store the total difficulty
    // of the chain up to and including the block.
    td *big.Int

    // These fields are used by package eth to track
    // inter-peer block relay.
    ReceivedAt time.Time
    ReceivedFrom interface{}
}
```

```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
    ParentHash common.Hash `json:"parentHash" gencodec:"required"`
    UncleHash  common.Hash `json:"sha3Uncles" gencodec:"required"`
    Coinbase   common.Address `json:"miner" gencodec:"required"`
    Root       common.Hash `json:"stateRoot" gencodec:"required"`
    TxHash     common.Hash `json:"transactionsRoot" gencodec:"required"`
    ReceiptHash common.Hash `json:"receiptsRoot" gencodec:"required"`
    Bloom      Bloom `json:"logsBloom" gencodec:"required"`
    Difficulty *big.Int `json:"difficulty" gencodec:"required"`
    Number     *big.Int `json:"number" gencodec:"required"`
    GasLimit   *big.Int `json:"gasLimit" gencodec:"required"`
    GasUsed    *big.Int `json:"gasUsed" gencodec:"required"`
    Time       *big.Int `json:"timestamp" gencodec:"required"`
    Extra      []byte `json:"extraData" gencodec:"required"`
    MixDigest  common.Hash `json:"mixHash" gencodec:"required"`
    Nonce      BlockNonce `json:"nonce" gencodec:"required"`
}
```

## ETHEREUM BLOCK STRUCTURE

#Ethereum Blockchain Demo – node.js / web3: <https://github.com/critoma/blockchain>

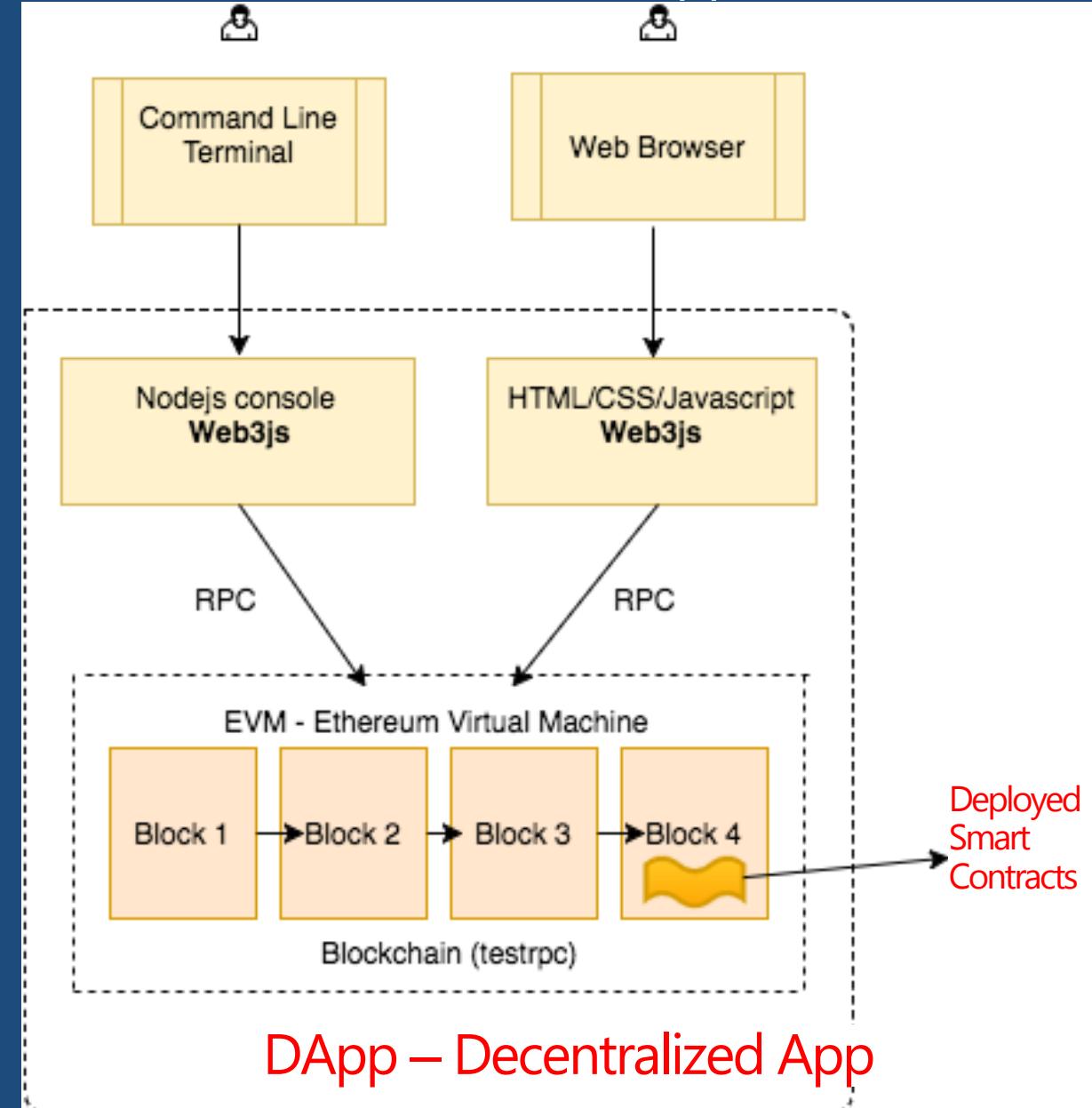
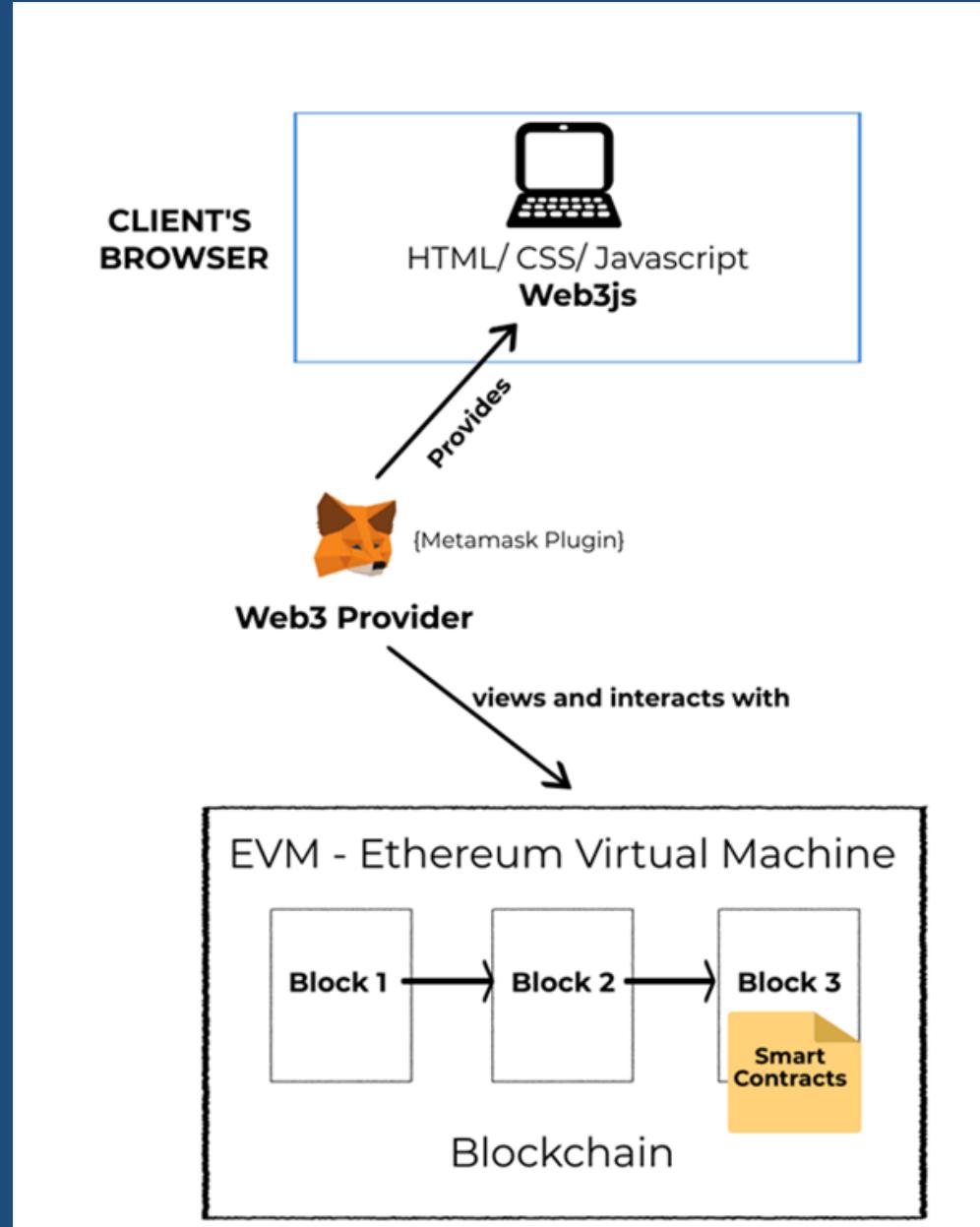


<https://cypherpunks-core.github.io/ethereumbook/10tokens.html> | <https://github.com/ethereumbook/ethereumbook>

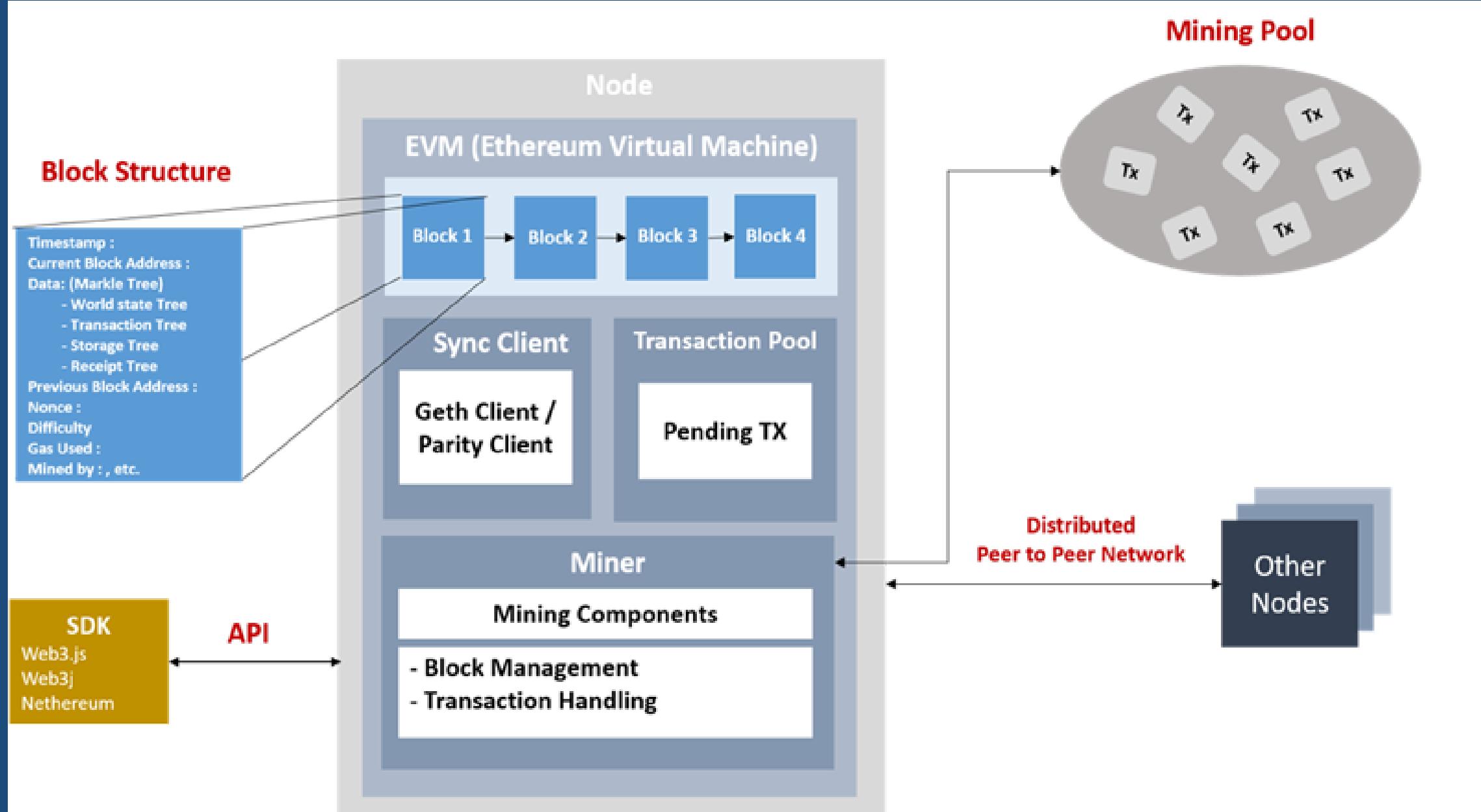
Google Shell Cloud: <https://github.com/critoma/blockchain/blob/main/testInfura.js> | Remix IDE: <https://docs.metamask.io/guide/sending-transactions.html>

<https://www.dappuniversity.com/articles/web3-js-intro>

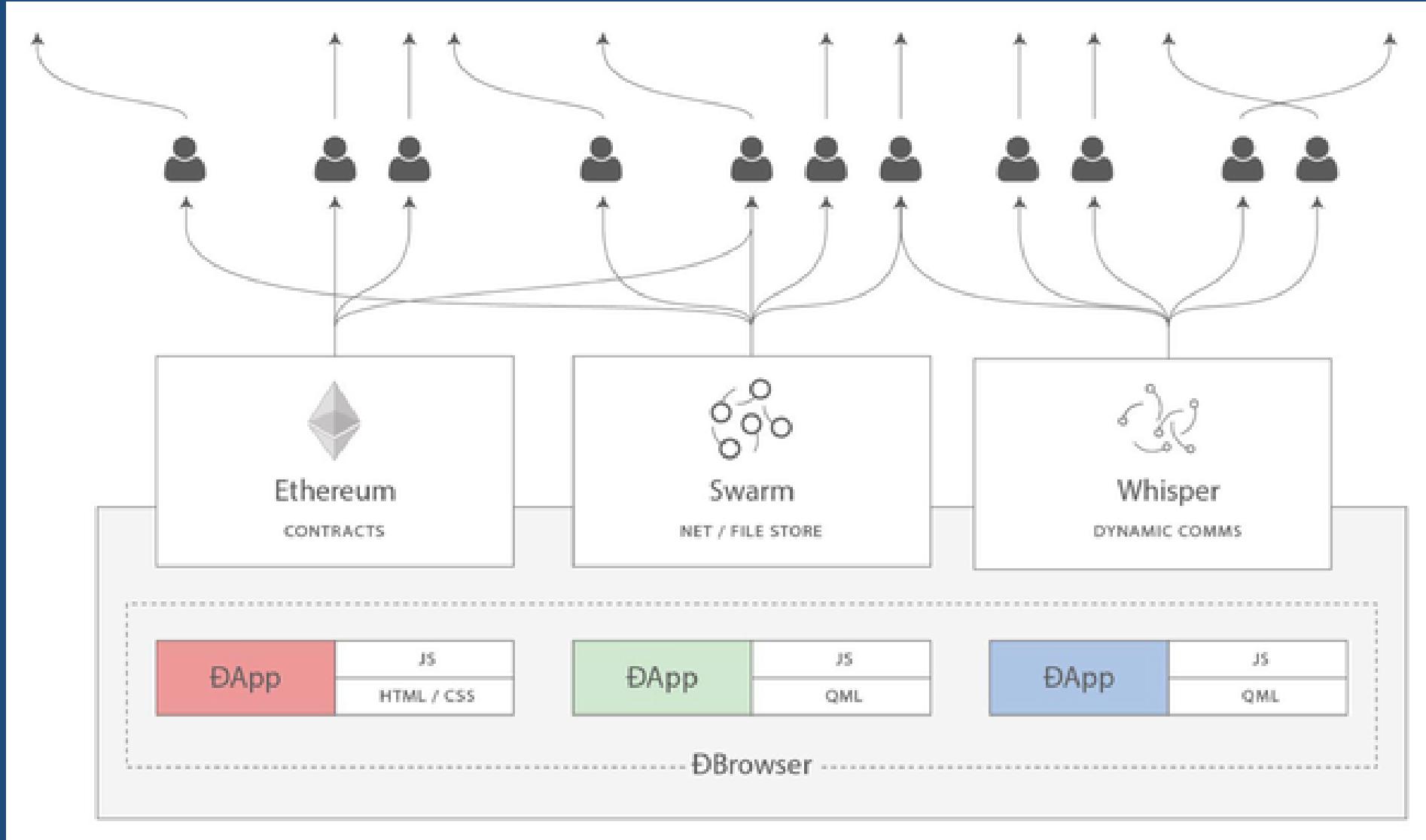
## 2. Ethereum Blockchain Peer2Peer Network Architecture & DApp



## 2. Ethereum Blockchain Peer2Peer Network Architecture & DApp



## 2. Ethereum Blockchain Peer2Peer Network Architecture & DApp

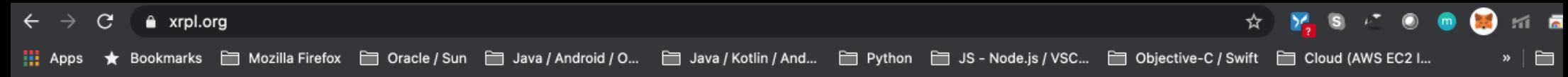


**Web3: A suite of Dapp - Decentralized APP-lication components for the next evolution of the web**

# #Ethereum:

	<b>Bitcoin (BTC)</b>	<b>Ether (ETH)</b>
<b>What is it?</b>	A currency	A token
<b>Inventor</b>	Satoshi Nakamoto	Vitalik Buterin, Joseph Lubin, Gavin Wood, etc.
<b>Went alive</b>	January 2009	July 2015
<b>Supply Style</b>	Deflationary (a finite # of bitcoin will be made)	Inflationary (much like fiat currency, where more tokens can be made over time)
<b>Supply Cap</b>	21 million in total	18 million every year
<b>Smallest Unit</b>	1 Satoshi = 0.00000001 BTC	1 Wei = 0.0000000000000001 ETH
<b>New token issuance time</b>	Every 10 minutes approximately	Every 10 to 20 seconds
<b>Amount of new token at issuance</b>	12.5 at the moment. Half at every 210,000 blocks	5 per every new block
<b>Utility</b>	Used for purchasing goods and services, as well as storing value (much like how we currently use gold).	Used for making dApps (decentralized apps) on the Ethereum blockchain.
<b>Price</b>	Around \$18,000 - \$55,000 (in 2020 - 2021)	Around \$610 - \$2,000 (in 2020 - 2021)
<b>Purpose</b>	Bitcoin is a new currency created to compete against the gold standard and fiat currencies	Ethereum is a token capable of facilitating <a href="#">Smart Contracts</a> (For example a lawyer's contract, an exchange of ownership of property, and voting)

**#XRP:** XRP is a digital asset built for payments. It is the native digital asset on the XRP Ledger - an open-source, permissionless and decentralized blockchain technology that can settle transactions in 3-5 seconds.



Learn

Explore

Build

Contribute

Edit



Get Started



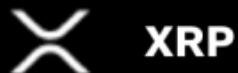
# XRP: The Best Digital Asset for Payments

Copyright: <https://xrpl.org>

# #XRP:

Benefits

## Benefits



**Fast**

3-5 seconds to settle

500 seconds to settle



**Low-Cost**

\$0.0002/transaction

\$0.50/transaction



**Scalable**

1,500 transactions per second

3 transactions per second



**Sustainable**

Environmentally sustainable  
and carbon neutral

0.3% of global energy  
consumption

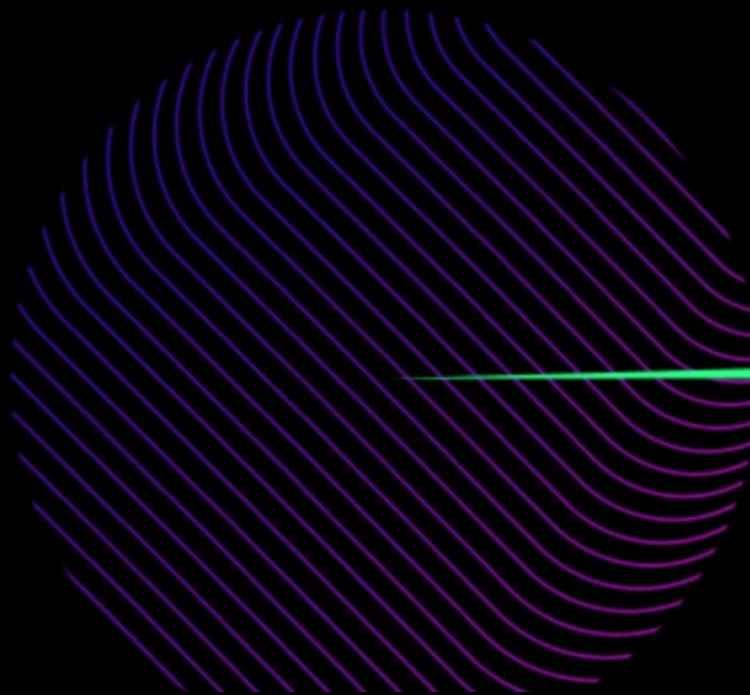
# #XRP:

## Finite Supply of XRP

XRP can be sent directly without needing a central intermediary, making it a convenient instrument in bridging two different currencies quickly and efficiently. It is freely exchanged on the open market and used in the real world for enabling cross-border payments and microtransactions.

Unlike Bitcoin, there is a finite amount of XRP. All XRP is already in existence today—100 billion in total.

**100 Billion XRP**



- Based on historic daily averages, 9.1 million XRP has been destroyed through the transaction execution process.

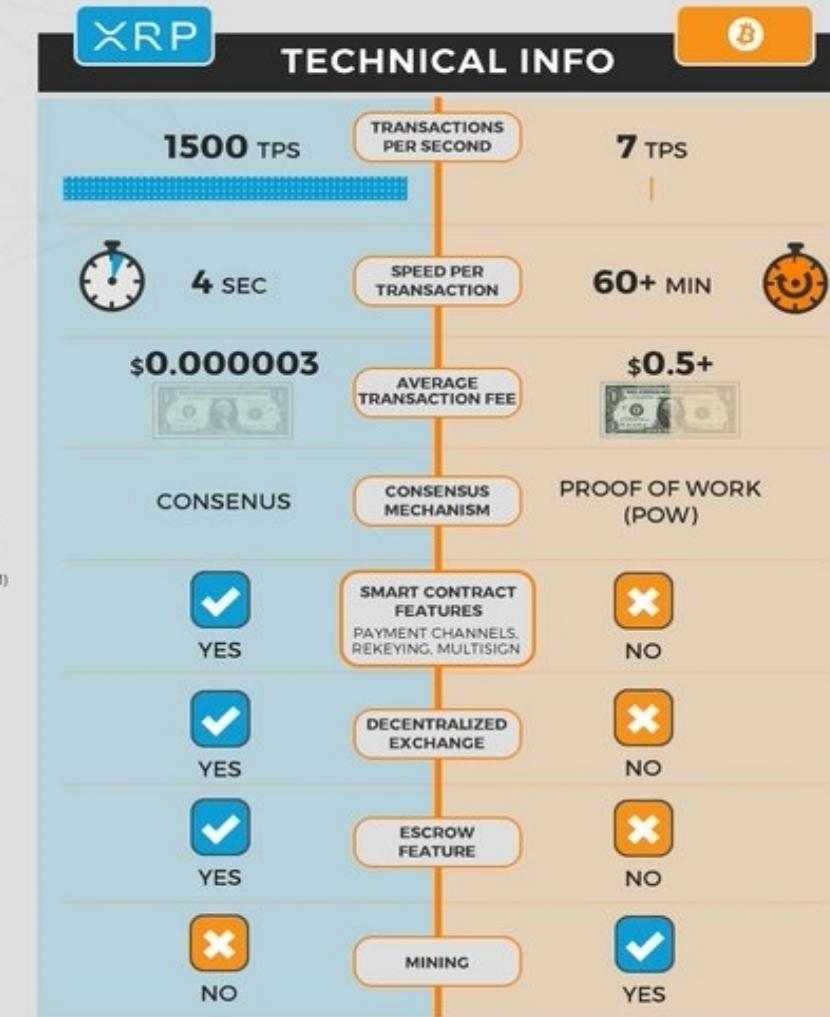
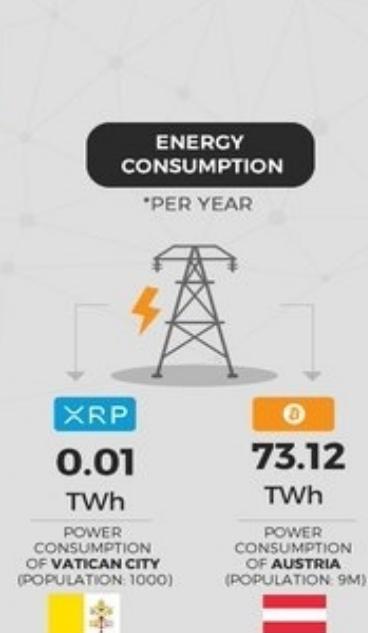
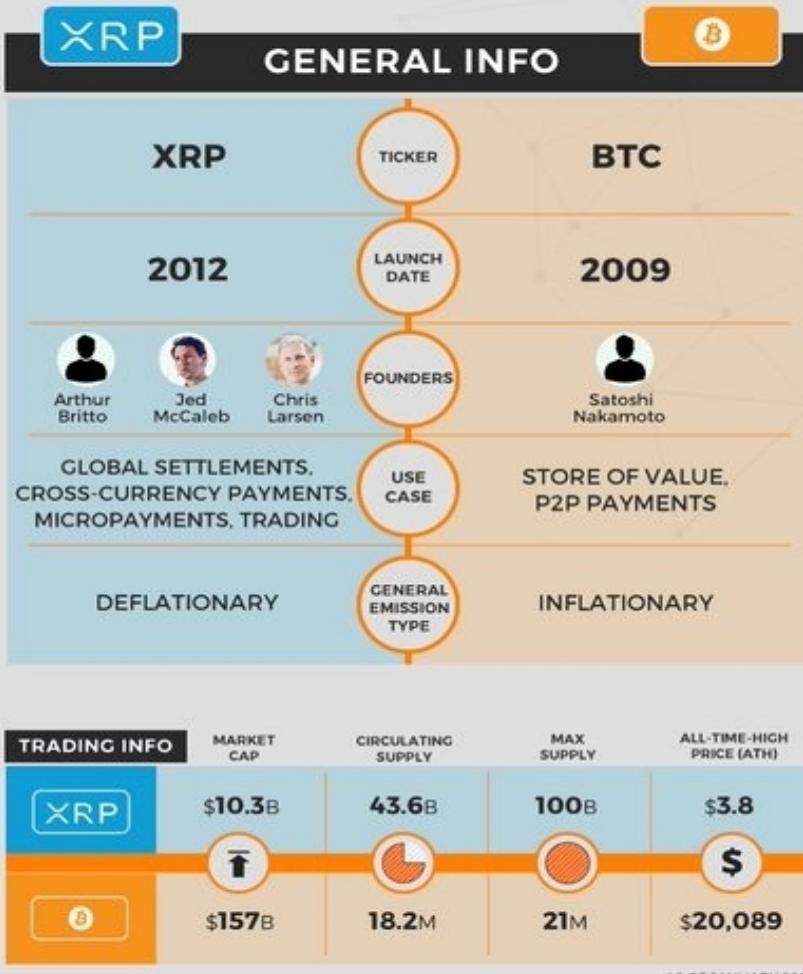


A DIGITAL ASSET FOR GLOBAL PAYMENTS



@STEDAS

FIRST DECENTRALIZED DIGITAL CURRENCY



WWW.STEDAS.HR  
@stedor

January 2020

SOURCES: RIPPLE, COINMARKETCAP, DIGICONOMIST, MESSARI

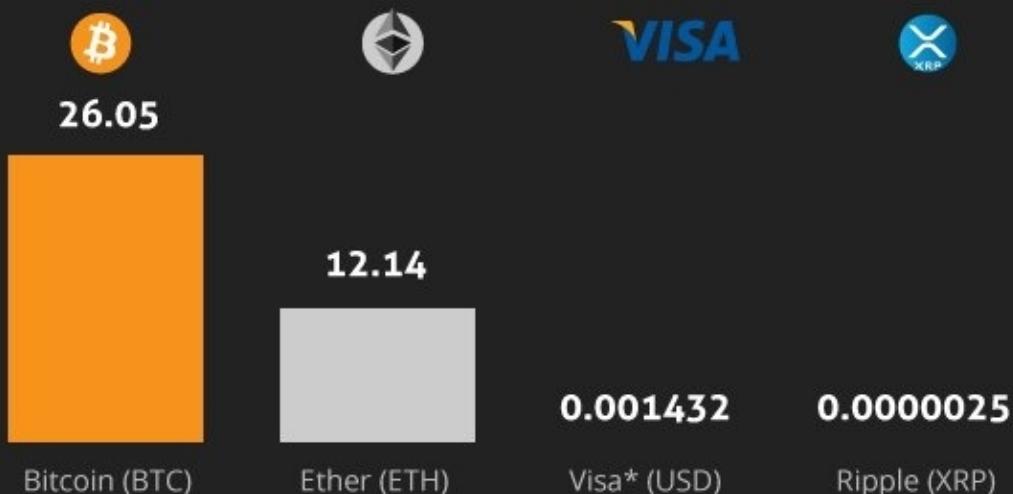
# ECO-FRIENDLY CURRENCIES

*Sid John Leopold and Niclas Englesson*

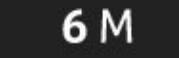
SLIDE 3

## Hypothetical annual electricity consumption with 220M transactions\*

\*Bitcoins hypothetical maximum per annum



Number of light bulbs, dishwashers and driven car miles that could be powered by the energy use of 220M transactions

	 149 M	 70 M	 115 M
 69 M	 32 M	 52 M	
 8,177	 3,826	 6 M	
 14	 6		 11,018



@stedas



suveljak

Copyright, 2018

# XRP consumes less energy than ETH & BTC

1/100 of a ☺

Annually, XRP consumes (no mining)

Ripple 0.01TWh

Equivalent to the ☺ of Oatman, Arizona  
(Population: 140)

Annually, ETH mining consumes

Ethereum 19.62TWh

Equivalent to the ☺ of Jordan  
(Population: 9.456 million)



Annually, BTC mining consumes

Bitcoin 68.81TWh

Equivalent to the ☺ of The Czech Republic  
(Population: 10.56 million)

1 ☺ = 1TWh

## Sources

- Bitcoin Energy Consumption (<https://digiconomist.net/bitcoin-energy-consumption>)
- Ethereum Energy Consumption (<https://digiconomist.net/ethereum-energy-consumption>)
- Global Energy Production (<http://world.bymap.org/ElectricityProduction.html>)
- XRP power consumption: calculated based on a hypothetical 1,000 validating servers running consecutively for an entire year.

[https://www.reddit.com/r/Ripple/comments/a1luvd/xrp\\_consumes\\_less\\_energy\\_than\\_eth\\_btc\\_xrp\\_is\\_eco/](https://www.reddit.com/r/Ripple/comments/a1luvd/xrp_consumes_less_energy_than_eth_btc_xrp_is_eco/)



## BITCOIN

Bitcoin Core (BTC) code was designed, written, and deployed by Satoshi Nakamoto.

It uses blockchain technology to jointly manage the database that records Bitcoin transactions. It operates on a user-to-user basis.

Bitcoin is used in the capacity of money and was developed as a digital currency with the aim of paying for services and goods.

Users can pay the miner to prioritize their transaction. Plus, when sending a Bitcoin transaction, its fee is proportionate to its size.



## ETHEREUM

Ethereum is developed by Vitalik Buterin, a programmer from Toronto.

It uses distributed computing platform and operating system featuring smart contract (scripting) functionality.

Beyond a tradable cryptocurrency, ether is also used by application developers to pay for transaction fees and services on the Ethereum network.



## RIPPLE

Ripple was founded by a single company, Ripple Labs.

Ripple is a payment protocol & it uses blockchain technology to help banks conduct fast global financial settlements.

Ripple serves as the middleman for all global FX transactions. The Ripple network is a totally decentralized currency exchange, while Bitcoin entails centralized ones.

Ripple requires a minimum transaction cost to avoid overloading the network. To pay the fee, the network destroys the XRP rather than paying it to anyone in particular, which in turn increases the value of the remaining XRP.

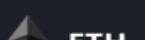
Each transaction takes on average 40 – 70 minutes.

Ethereum delivers a transaction speed of two minutes.

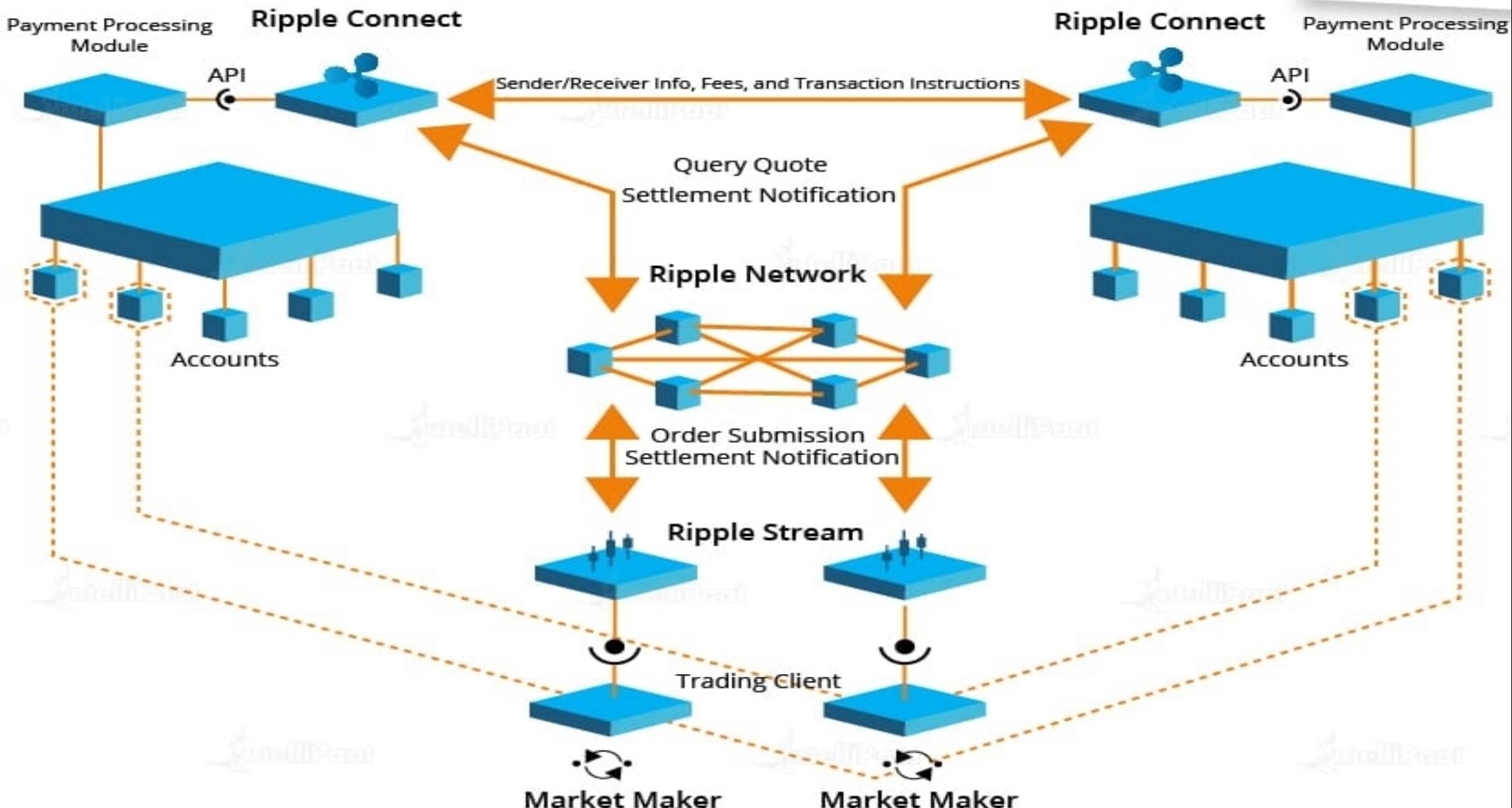
Ripple settles the transactions in as little as four seconds.

# XRP: Transactions at the Scale of Visa

XRP consistently handles 1,000 transactions per second and can scale to handle the same throughput as Visa.\*



#XRP:



# #XRP: XRP Signatures for Transactions and Demos

The elliptic curve: Curve25519

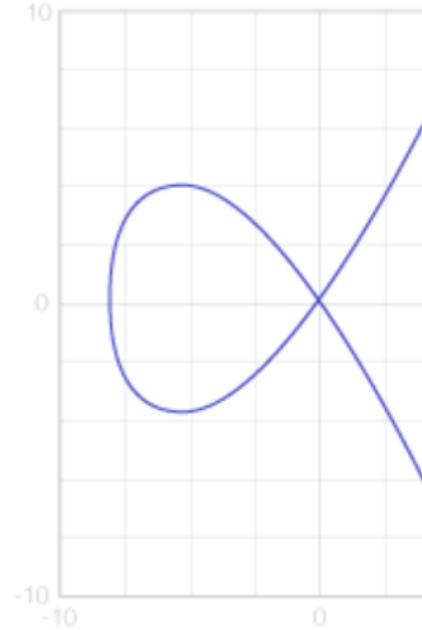
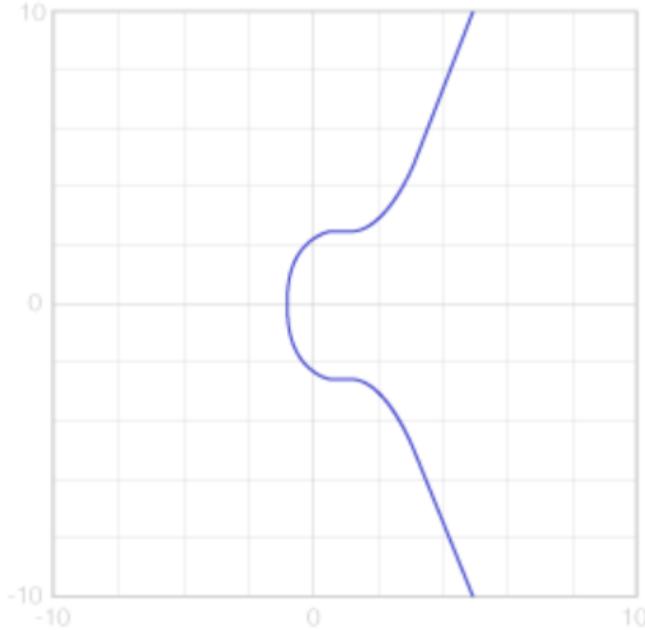
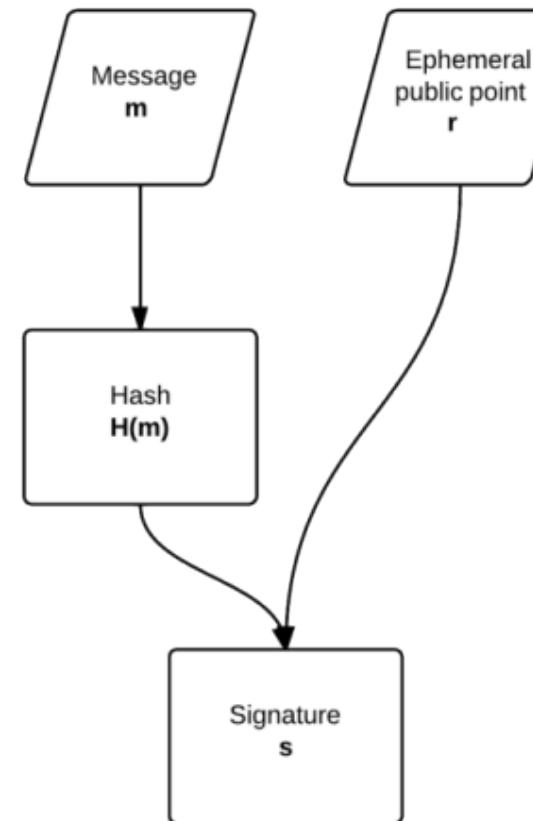
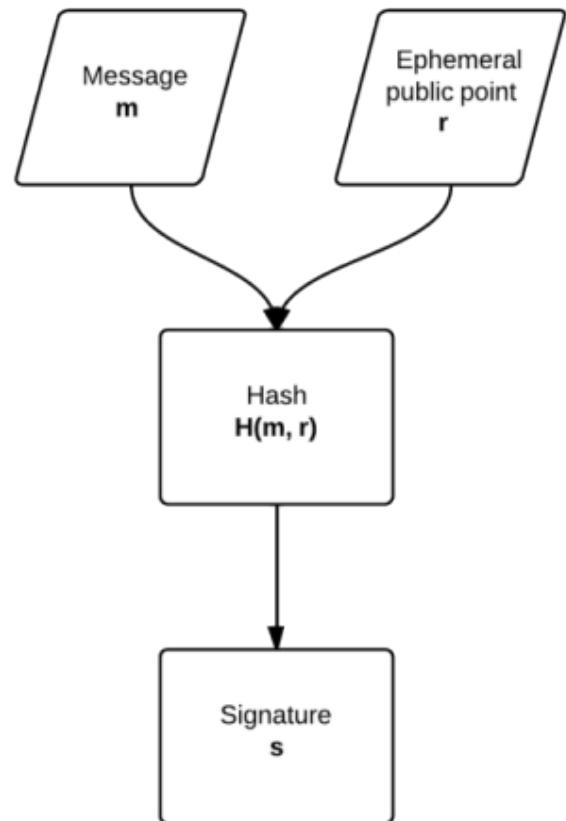


Image: secp256k1 (left) versus Curve25519 (right)

## DSA

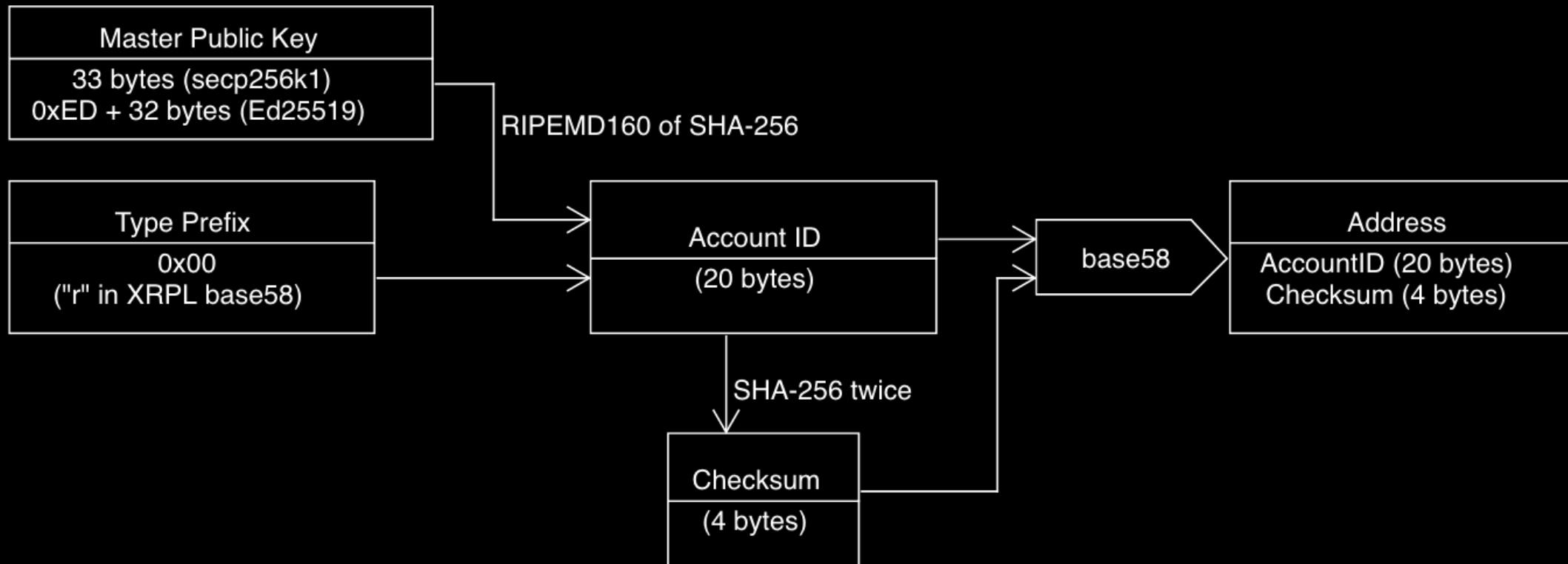


## Schnorr

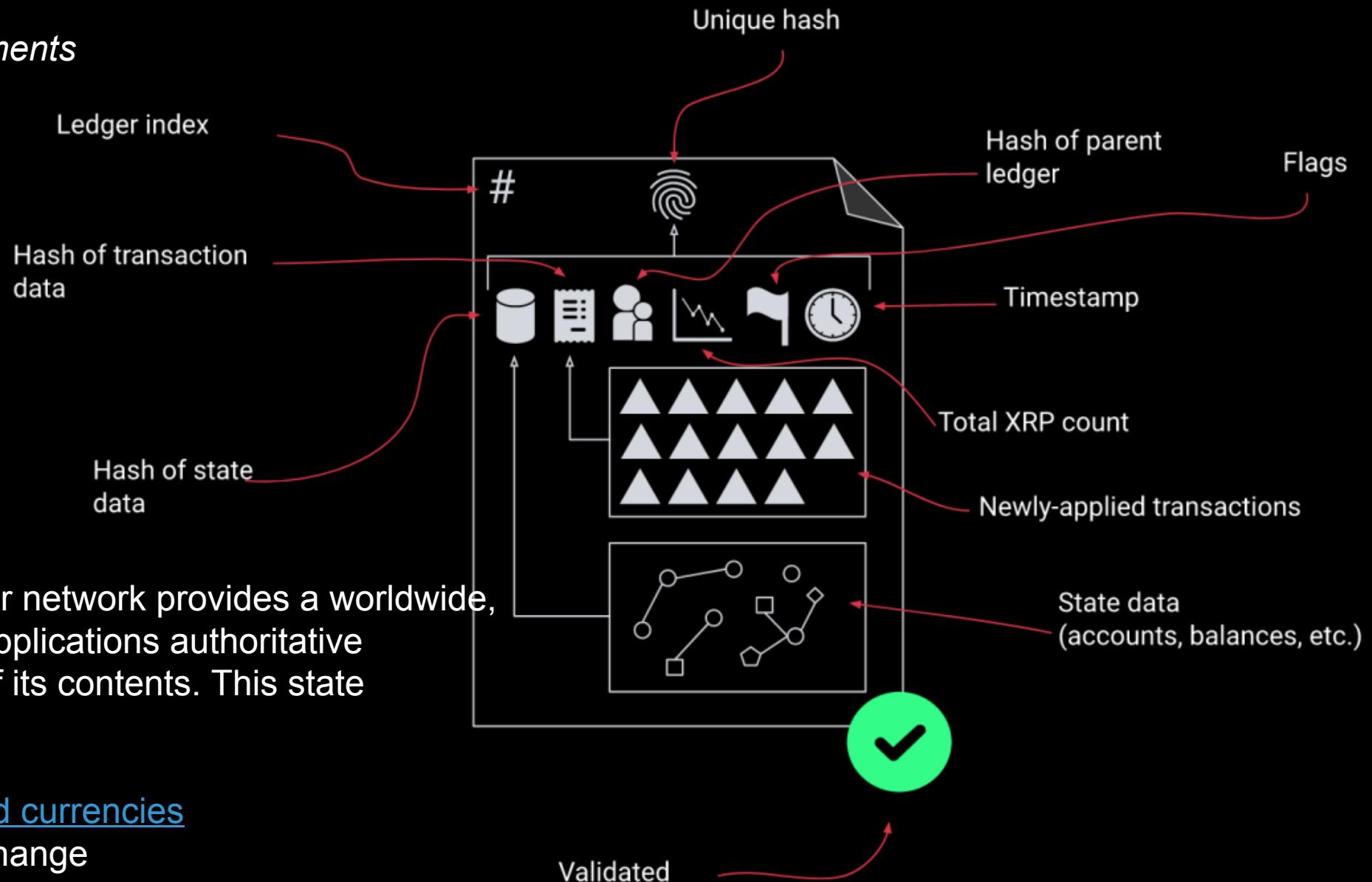


# #XRP: XRP Address and Demos

## Address Encoding



# #XRP: XRP Ledger Elements

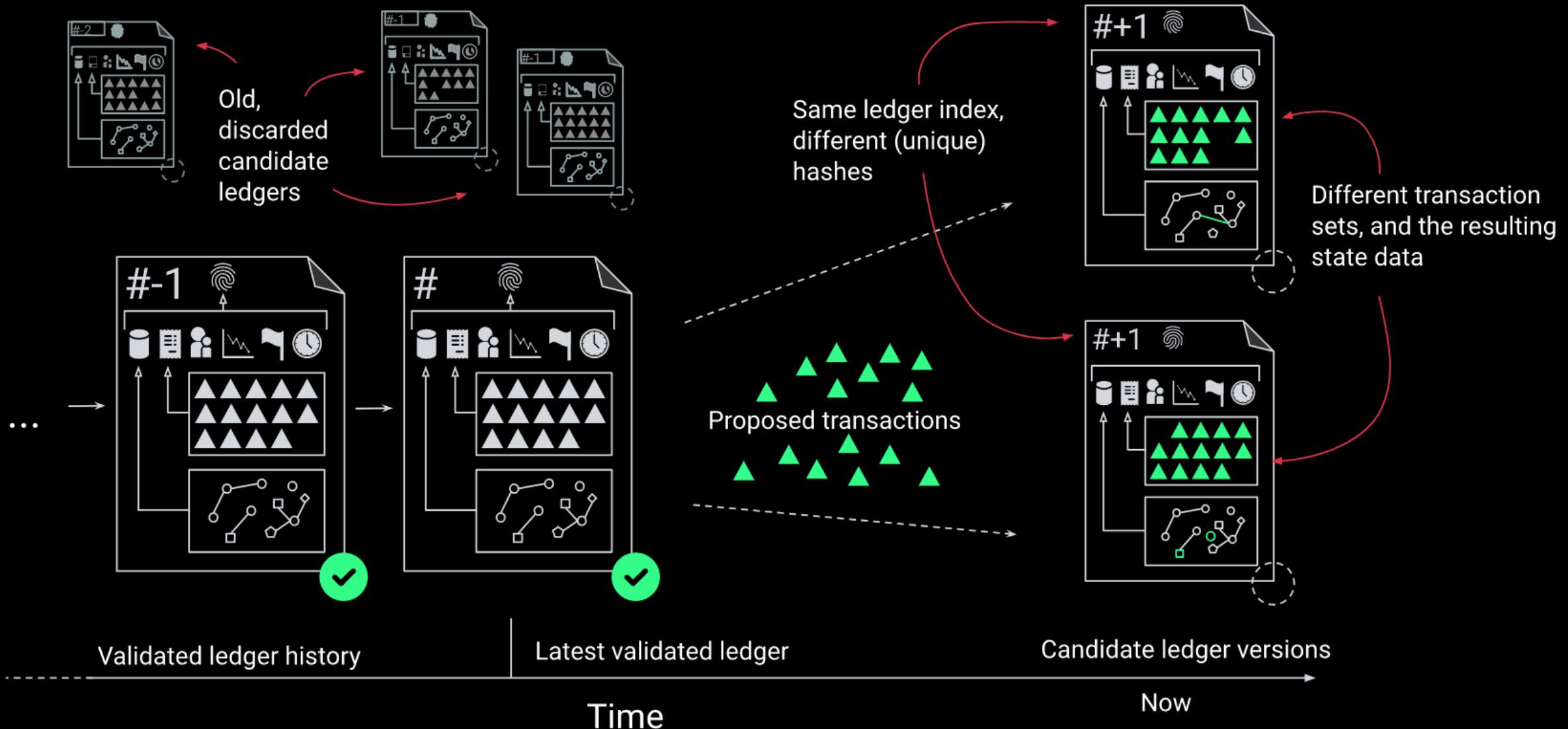


The peer-to-peer XRP Ledger network provides a worldwide, shared ledger, which gives applications authoritative information about the state of its contents. This state information includes:

- settings for each [account](#)
- balances of XRP and [issued currencies](#)
- offers in the distributed exchange
- network settings, such as [transaction costs](#) and [reserve](#) amounts
- a timestamp

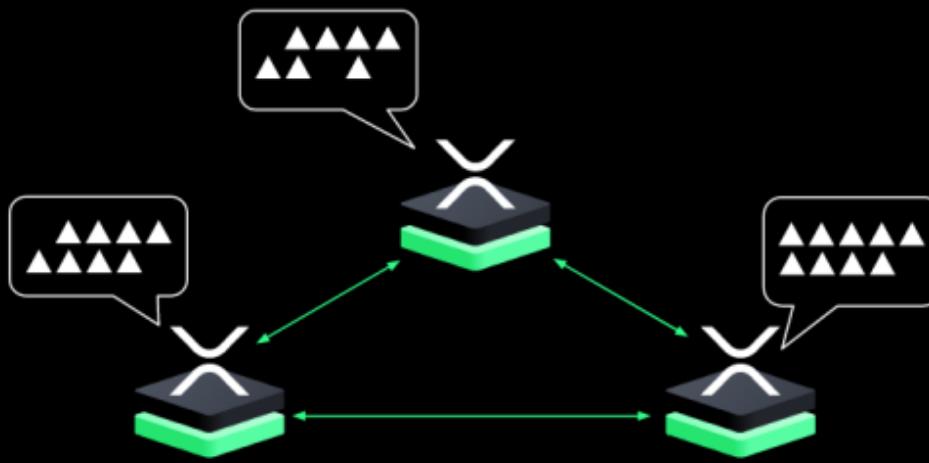
For a full, technical description of which data is included in a

# #XRP: XRP Ledger History



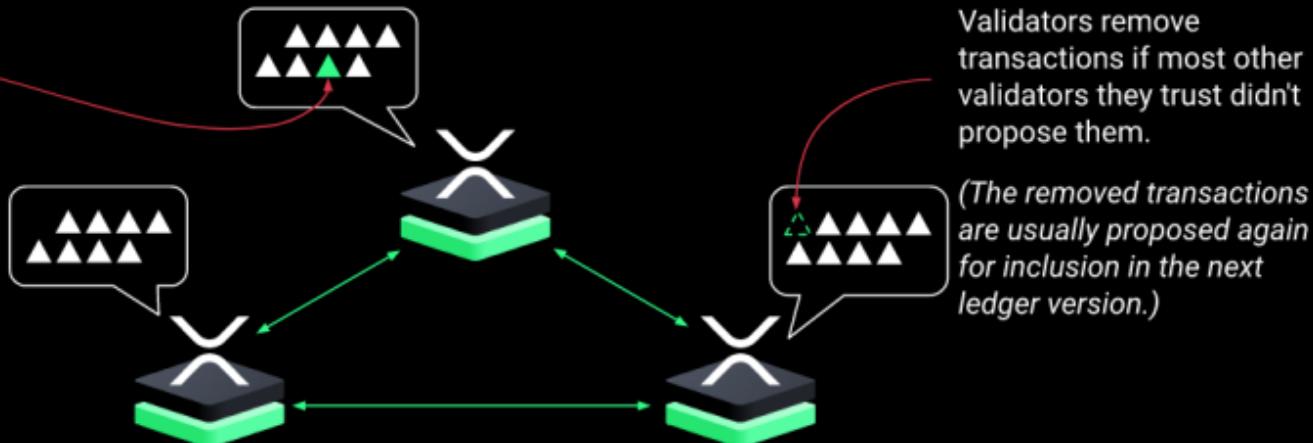
# #XRP:

Validators each propose a set of transactions to be included in the next ledger version.



Round 1

Validators add transactions to their proposals if most other validators they trust proposed those transactions



Round 2

Validators remove transactions if most other validators they trust didn't propose them.

(The removed transactions are usually proposed again for inclusion in the next ledger version.)

*Validators Propose and Revise Transaction Sets — At the start of consensus, validators may have different sets of transactions. In later rounds, servers modify their proposals to match what their trusted validators proposed. This process determines which transactions they should apply to the ledger version currently being discussed, and which they should postpone for later ledger versions.*

# 03. Blockchain Use-cases in Fin-Tech

129

- ▶ Introduction
- ▶ **Use case 1** – AML – Anti Money Laundry
- ▶ **Use case 2** – KYC – Know Your Customer
- ▶ XRP Ripple as SWIFT replacement

# Introduction



- ▶ A blockchain is a data structure that makes it possible to create a digital ledger of transactions and share it amongst a distributed network of computers.
- ▶ It uses cryptography to allow each participant on the network to manipulate the ledger in a secure way without the need for a central authority.

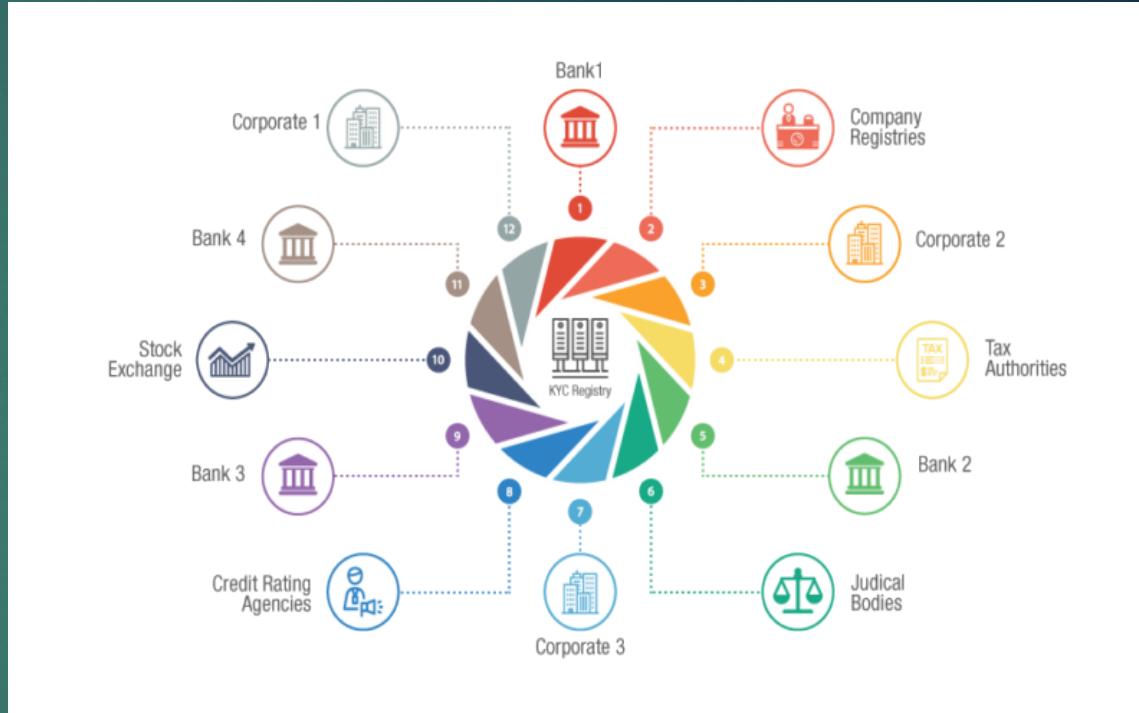
# Introduction

- ▶ Blockchain technology is emerging as the way to let companies make and verify financial transactions on a network instantaneously without a central authority
- ▶ In banking there are many existing and developing use cases to implement blockchain.



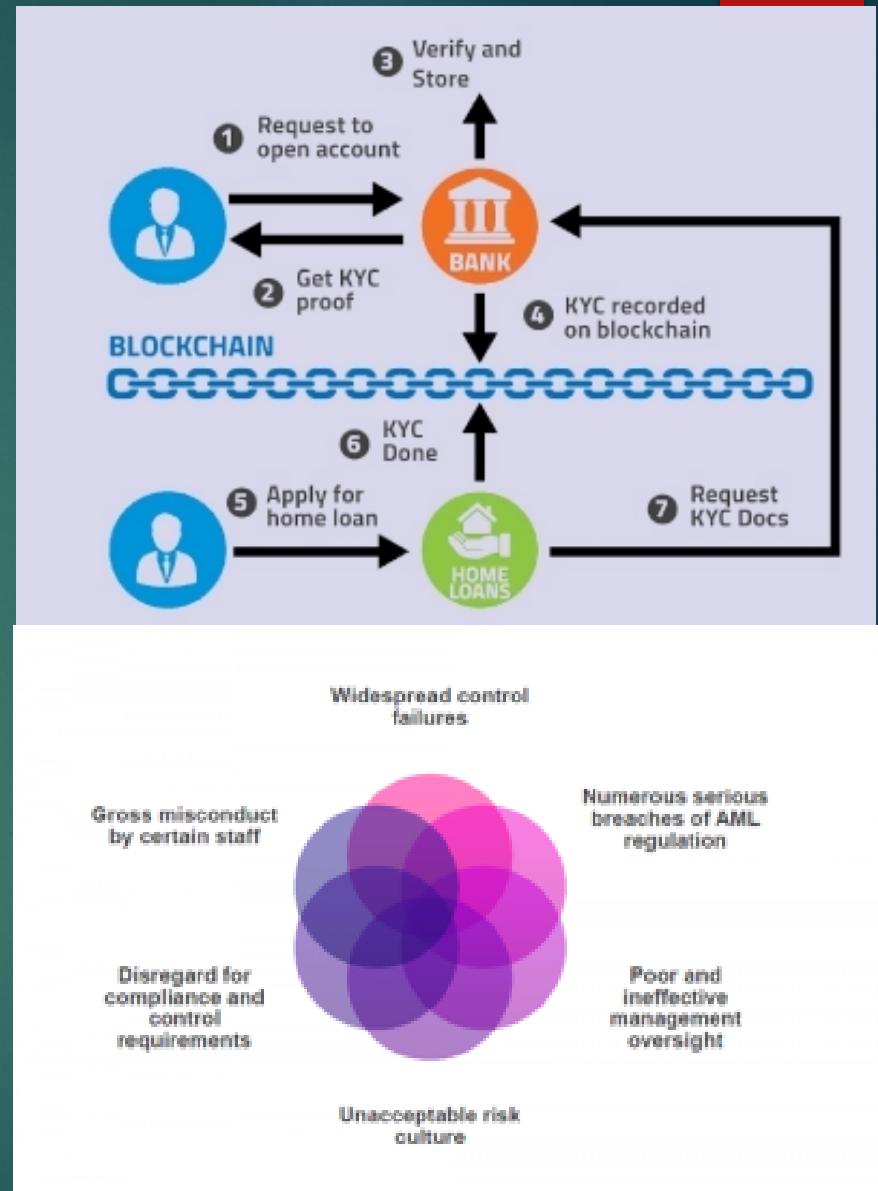
# Use case 1 – AML

- ▶ Since the inception of cryptocurrency in 2009, thanks to Bitcoin, criminals have repeatedly tried to use it to launder large amounts of money obtained from illicit activity. Since their identities are hidden, it becomes harder for regulatory bodies like the [FATF](#) (Financial Action Task Force) to catch them.
- ▶ **Money laundering, in a nutshell, is the process by which money obtained through illicit activity, say bribery or terrorist financing, is ‘cleansed’ by processing it in a sequence of banking transactions so as to make it appear as if the money came from a legitimate source.**
- ▶ As a matter of fact, the total amount of money laundered globally every single year amounts to a whopping **estimated total of US \$2 trillion**. That's roughly **5% of the world's entire GDP!** To combat money laundering, blockchain technology could be an ideal tool because every transaction done leaves behind a permanent trail that's impossible to be altered – making it easier for authorities to track the original source of the money.
- ▶ Currently, digital exchanges and financial institutions all around the world are developing frameworks that increase compliance and introduce greater levels of regulations in their system to **comply with KYC and new AML (Anti-Money Laundering) guidelines** as the old ones have been criticized to be ineffective.
- ▶ **Unlike current banking models where consumer data is given to large corporations without much consideration for personal security and control, blockchain technology is a much more trusted network where data would only be shared to verified and trusted sources – e.g. via Smart Contracts.**

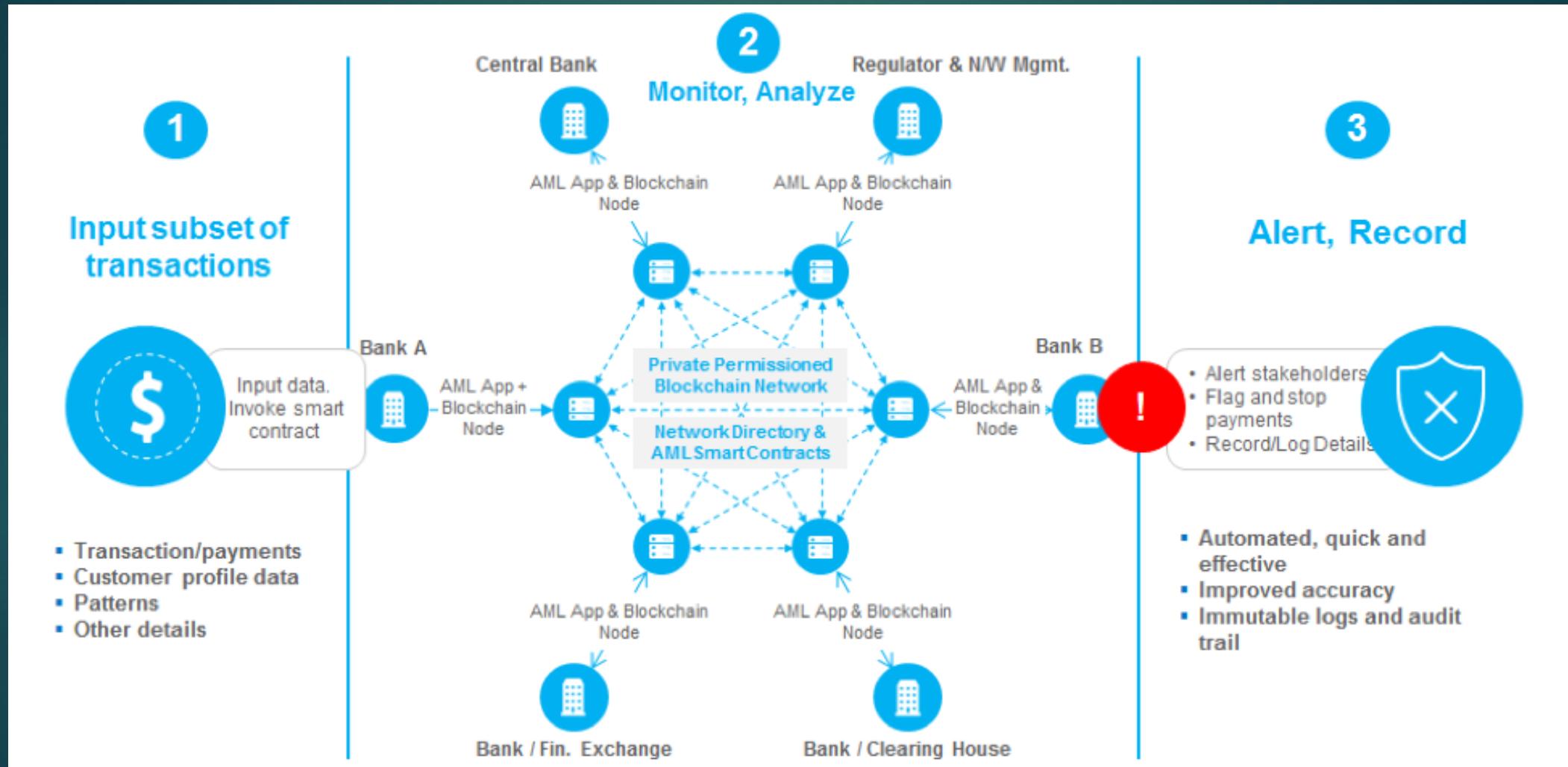


# Use case 1 – AML

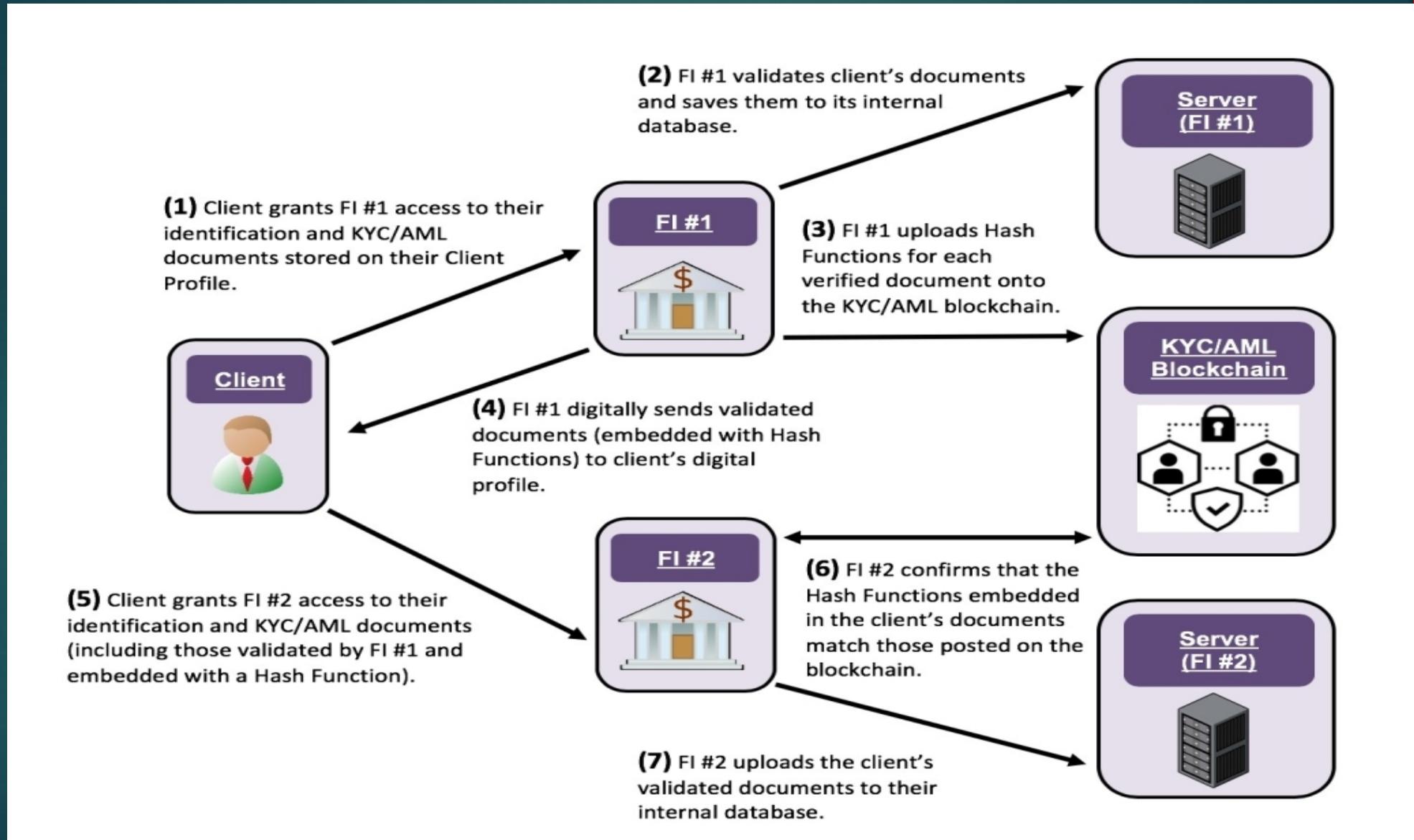
- ▶ The global financial model is changing, out of innovation but also out of necessity. The older AML guidelines and policies have been ineffective, resulting in poor management and low compliance which puts consumers at a risk of being compromised.
- ▶ Financial institutions and governments around the world have started investing in blockchain technology as an alternate strategy. It is important to note that blockchain isn't necessarily a global platform.
- ▶ It can also be created for private use by individual financial institutions with a predetermined set of users. The participants within the network can seamlessly initiate and undertake transactions and access data.
- ▶ Blockchain is, perhaps, the closest to an ideal solution we have so far to fight crimes like money laundering as its encryption technology allows for greater levels of security of transactions and safety of the parties involved.



# Use case 1 – AML

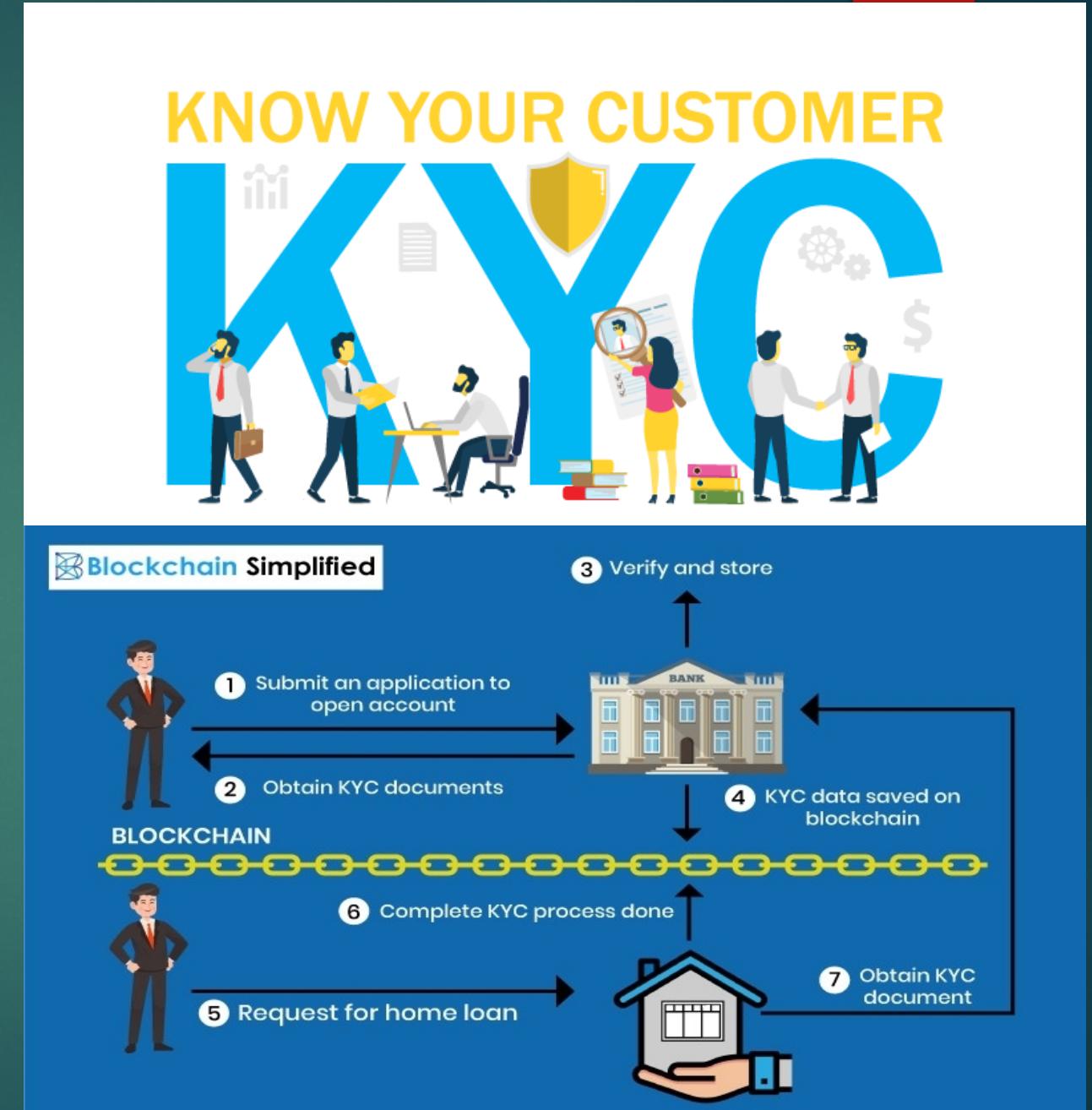


# Use case 1 – AML & KYC



# Use case 2 – KYC

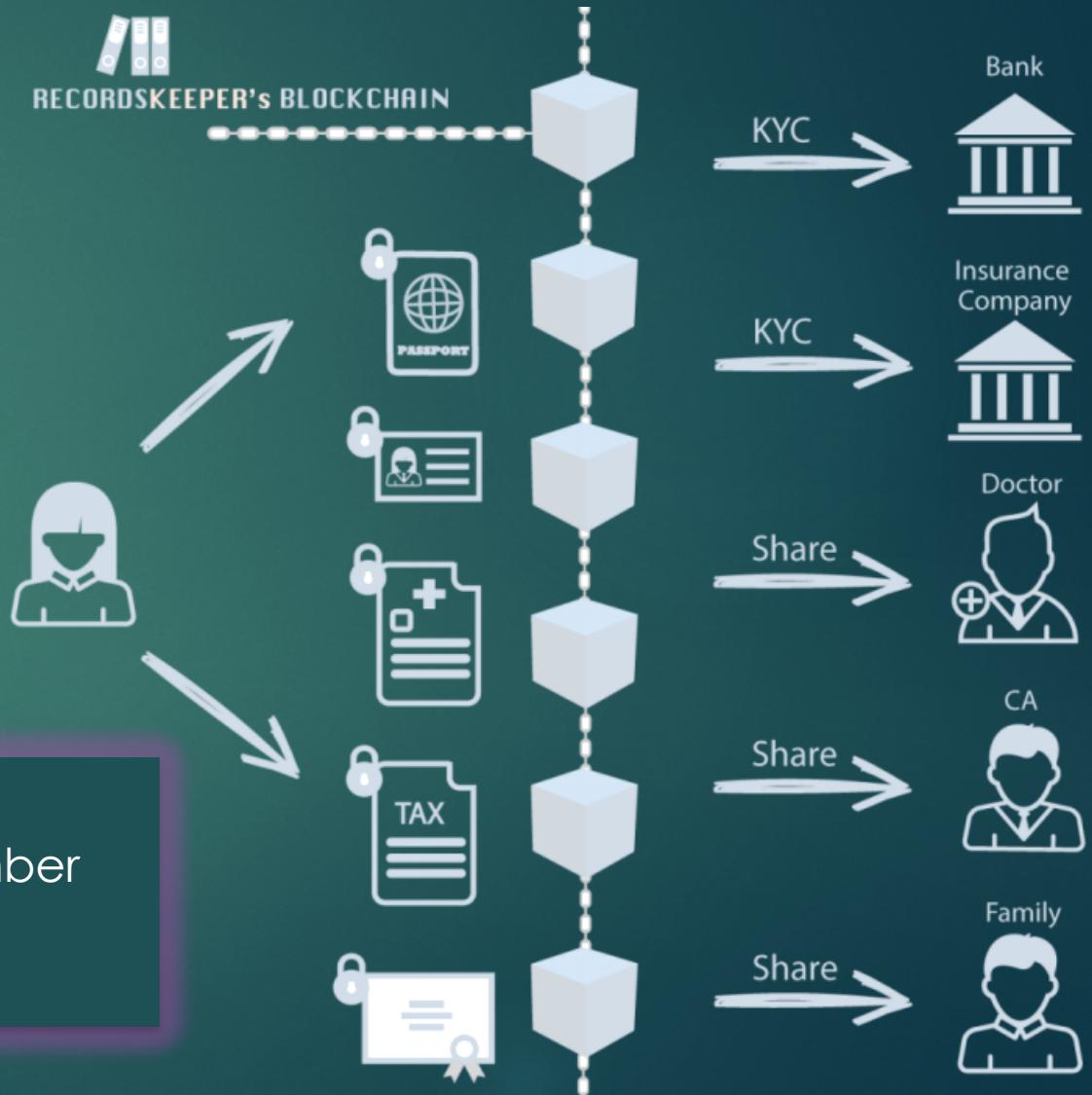
- ▶ Know Your Customer (“KYC”) requests currently can cause delay to banking transactions, typically taking 30 to 50 days to complete to a satisfactory level.



# Use case 2 – KYC

- ▶ On a blockchain system the information about a customer can be used by other banks and other accredited organizations without the need to ask the customer to start the KYC process all over again.

SWIFT has established a KYC Registry with 1,125 member banks sharing KYC documentation



# Use case 2 – KYC



Bank Chain



**SELFKEY**



**SECURE KEY**

# XRP Ripple as SWIFT replacement

- ▶ Ripple is a “real-time gross settlement system” (RTGS), currency exchange and remittance network.



# XRP Ripple as SWIFT replacement

- ▶ Official Monetary and Financial Institutions Forum (OMFIF) presents different reports on blockchain technology and the advantages of Ripple products as an alternative to SWIFT.
- ▶ These reports present blockchain technology as a solution for multiple sectors because can bring improvements in six key points: security, speed, transparency, traceability, cost, and risk management.



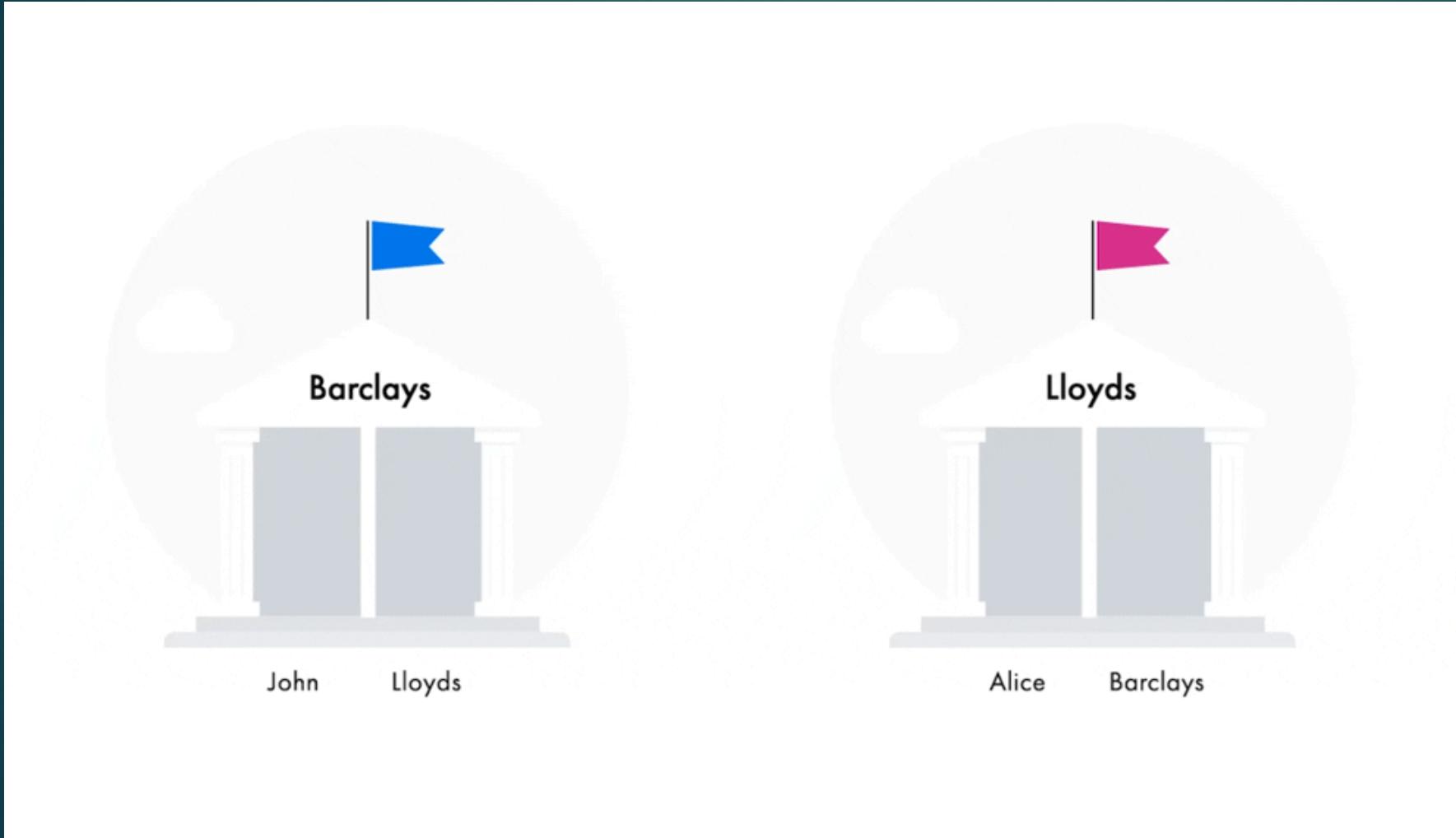
# XRP Ripple as SWIFT replacement

- ▶ Ripple comes up in 2012 with the name OpenCoin. Their objective has been to provide secure payments via a global network.
- ▶ In 2013 their system started to be used in banks for payment systems.
- ▶ In 2014 Ripple helped to create a new cryptocurrency: XRP. This cryptocurrency helps to make transfer money more easier.
- ▶ In February 2020, Ripple made a partnership with AZIMO to make transfer in European money.

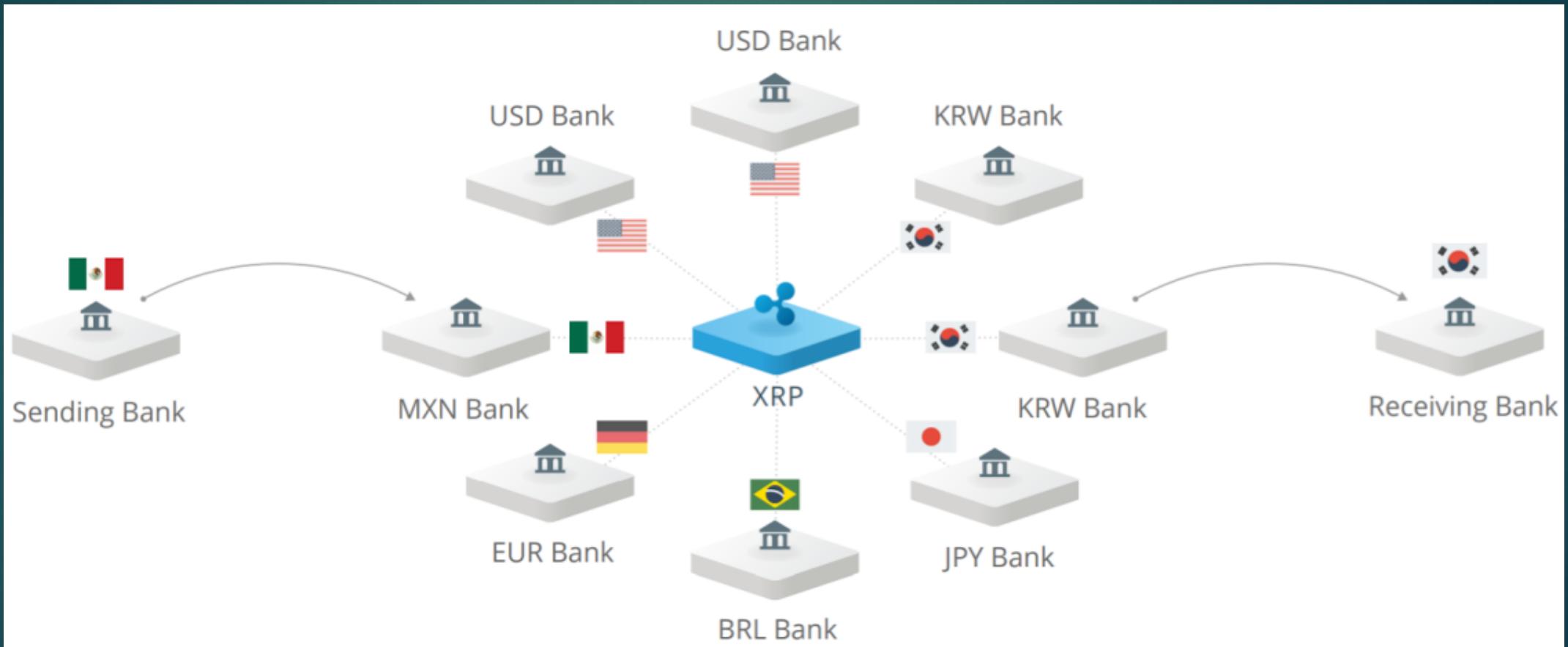
# XRP Ripple as SWIFT replacement

- ▶ Swift was founded in 1973 and has more than 10,000 bank members.
- ▶ Swift is used to send payment orders. The actual transfer of funds is typically made from a pre-funded nostro account that a bank has with a local correspondent bank.
- ▶ Ripple is attempting to change this way of conducting cross-border payments, by using XRP as a ‘bridge’ between fiat currencies.

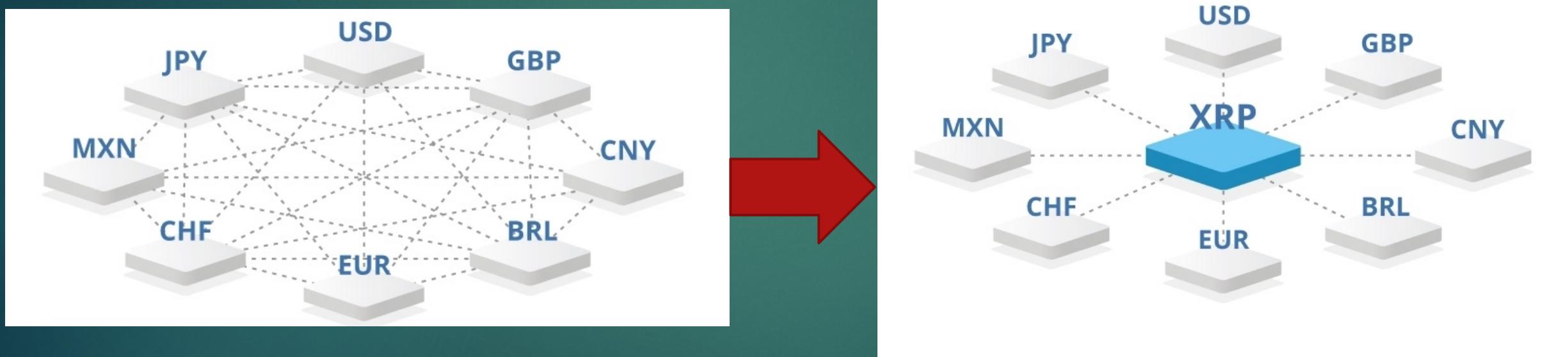
# XRP Ripple as SWIFT replacement



# XRP Ripple as SWIFT replacement



# XRP Ripple as SWIFT replacement



# XRP Ripple as SWIFT replacement

- ▶ The main battleground between Ripple and SWIFT is the competition for banks and institutions to use their services to transfer money. This competition has been ongoing for years now. Ripple only became a real competitor after the development of XRP.
- ▶ SWIFT has made improvements to their messaging system that allows tracking of payments, putting greater emphasis and responsibility on the banks to create a faster payment network.



The screenshot shows a web browser displaying a press release from the U.S. Securities and Exchange Commission (SEC). The URL in the address bar is <https://www.sec.gov/news/press-release/2020-338>. The page features the SEC logo and the text "U.S. SECURITIES AND EXCHANGE COMMISSION". A sidebar on the left lists "Newsroom" categories: Press Releases (selected), Public Statements, Speeches, Testimony, Spotlight Topics, Media Kit, Press Contacts, Events, Webcasts, What's New, and Media Gallery. The main content area is titled "Press Release" and discusses the SEC charging Ripple and two executives with conducting a \$1.3 billion unregistered securities offering. It includes a "FOR IMMEDIATE RELEASE" section and the date "2020-338". The text of the press release states that the SEC has filed an action against Ripple Labs Inc. and its executives Christian Larsen and Bradley Garlinghouse for raising over \$1.3 billion through an unregistered digital asset securities offering. The complaint alleges that Ripple raised funds beginning in 2013 by selling XRP to investors in the U.S. and worldwide. Ripple allegedly distributed billions of XRP in exchange for non-cash consideration like labor and market-making services. The complaint also states that Larsen and Garlinghouse effected personal unregistered sales of XRP totaling approximately \$600 million without registering them.

SEC Charges Ripple and Two Executives with Conducting \$1.3 Billion Unregistered Securities Offering

**FOR IMMEDIATE RELEASE**  
**2020-338**

*Washington D.C., Dec. 22, 2020 —* The Securities and Exchange Commission announced today that it has filed an action against Ripple Labs Inc. and two of its executives, who are also significant security holders, alleging that they raised over \$1.3 billion through an unregistered, ongoing digital asset securities offering.

According to the SEC's complaint, Ripple; Christian Larsen, the company's co-founder, executive chairman of its board, and former CEO; and Bradley Garlinghouse, the company's current CEO, raised capital to finance the company's business. The complaint alleges that Ripple raised funds, beginning in 2013, through the sale of digital assets known as XRP in an unregistered securities offering to investors in the U.S. and worldwide. Ripple also allegedly distributed billions of XRP in exchange for non-cash consideration, such as labor and market-making services. According to the complaint, in addition to structuring and promoting the XRP sales used to finance the company's business, Larsen and Garlinghouse also effected personal unregistered sales of XRP totaling approximately \$600 million. The complaint alleges that the defendants failed to register their offers and sales of XRP or satisfy any exemption from registration, in violation of the registration provisions of the federal securities laws.

# XRP Ripple as SWIFT replacement

LET'S SEND THE MONEY

---

moderated by **Elizabeth Schulze**, *CNBC International*

Brad **Garlinghouse**  
Gottfried **Leibbrandt**

#PFF19

[www.parisfintechforum.com](http://www.parisfintechforum.com)

An event produced and organized by **Altéir**

<https://www.youtube.com/watch?v=SRoCGr4cess>

# Q & A



**www: [ism.ase.ro](http://ism.ase.ro) | [acs.ase.ro](http://acs.ase.ro) | [dice.ase.ro](http://dice.ase.ro)**

**Scan the Tag  
to get the web  
Mobile Address**

