Crypto Blockchain

Cristian TOMA, Marius POPA, Alin ZAMFIROIU

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 | www.acs.ase.ro | www.dice.ase.ro

Crypto Blockchain Technology

https://www.insidehighered.com/news/2018/08/13/rising-profile-blockchain-academe

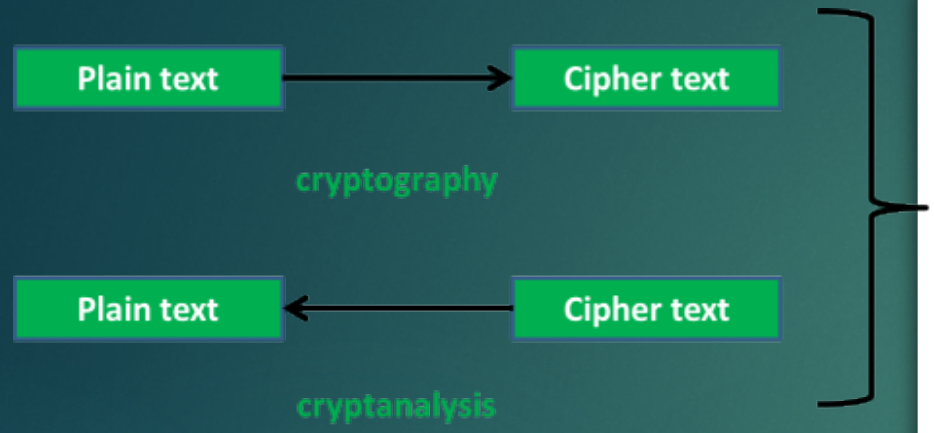
Agenda for the Blockchain

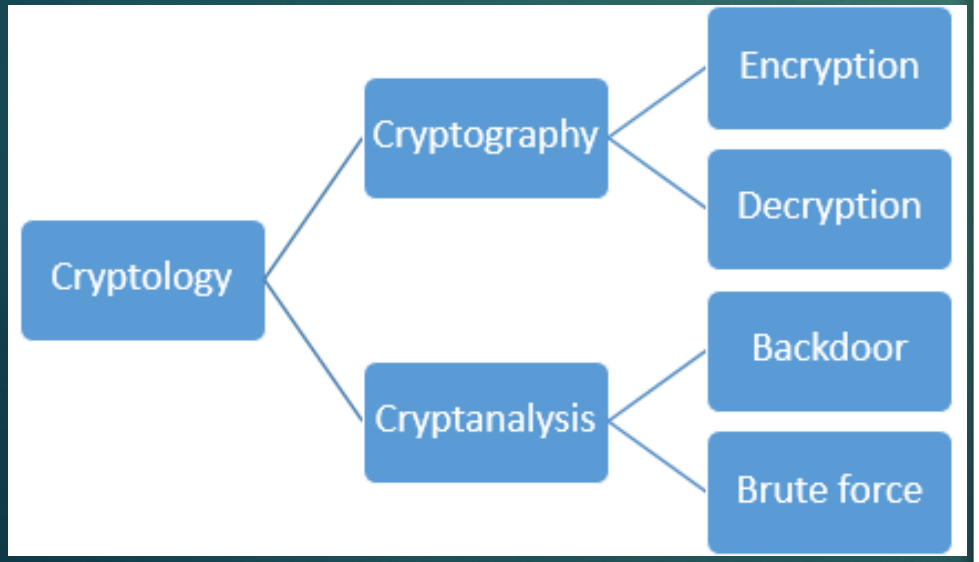
1. Cryptography Intro
2. Blockchain Tech: Bitcoin & Ethereum
3. Exchange Ideas Use Cases

www.dice.ase.ro

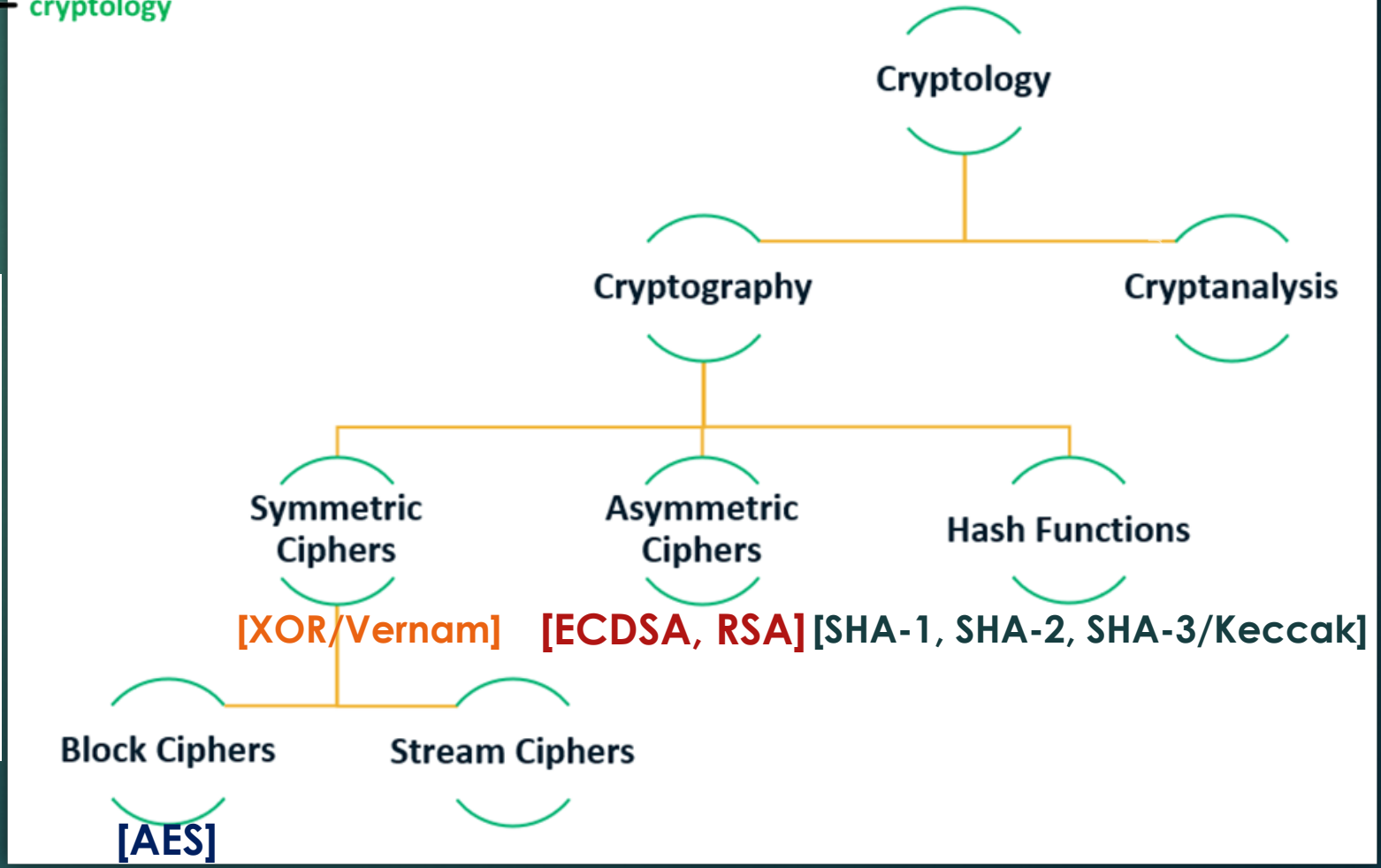
www.ism.ase.ro

01. Crypto Intro – Ninja Lecture Approach





Cryptology vs Cryptography



01. Crypto Intro – XOR with symmetric key

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

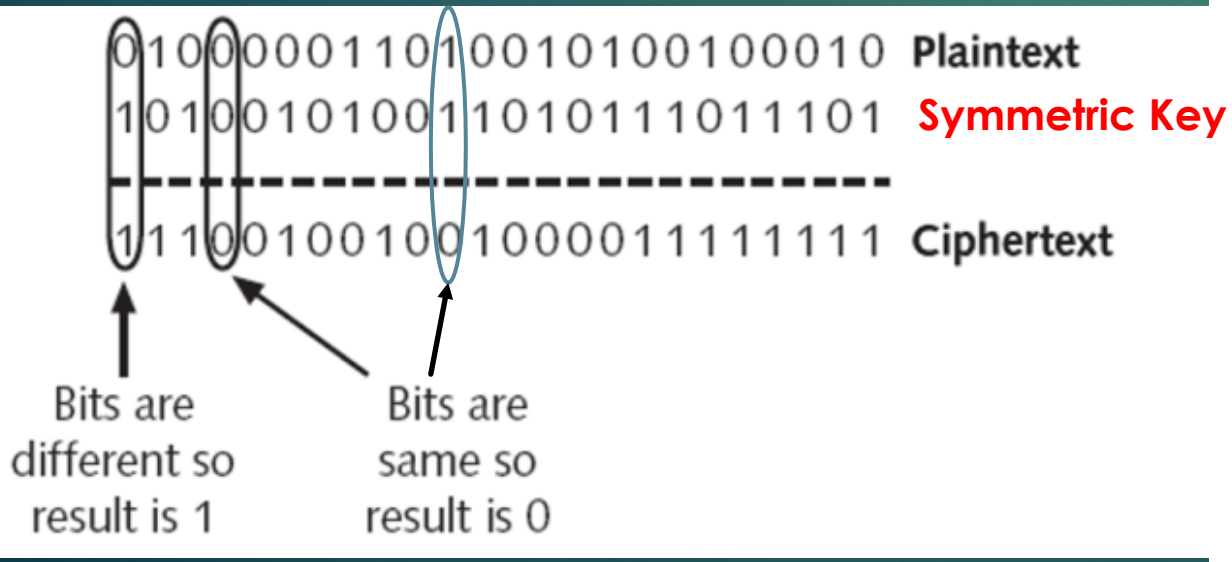
XOR Truth Table

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

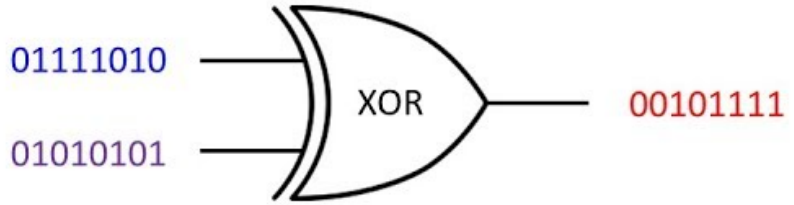
NOT Truth Table

A	B
0	1
1	0

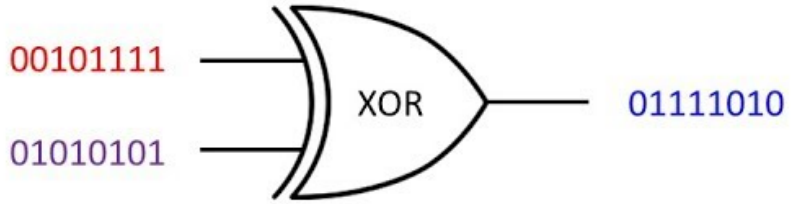
Exclusive disjunction essentially means 'either one, but not both nor none'. In other words, the statement is true if and only if one is true and the other is false. For example, if two horses are racing, then one of the two will win the race, but not both of them.



ENCRYPTION



DECRYPTION



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).

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.

AND Truth Table

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

OR Truth Table

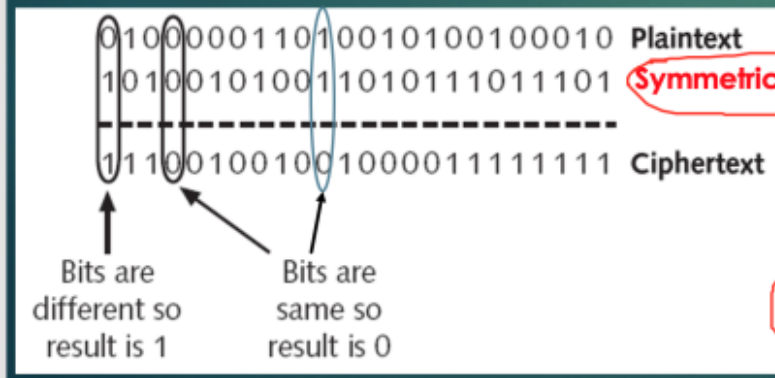
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

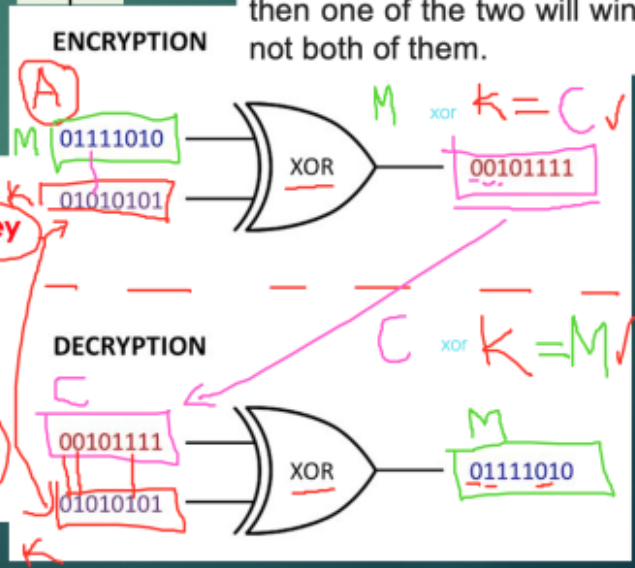
XOR Truth Table

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

NOT Truth Table

A	B
0	1
1	0





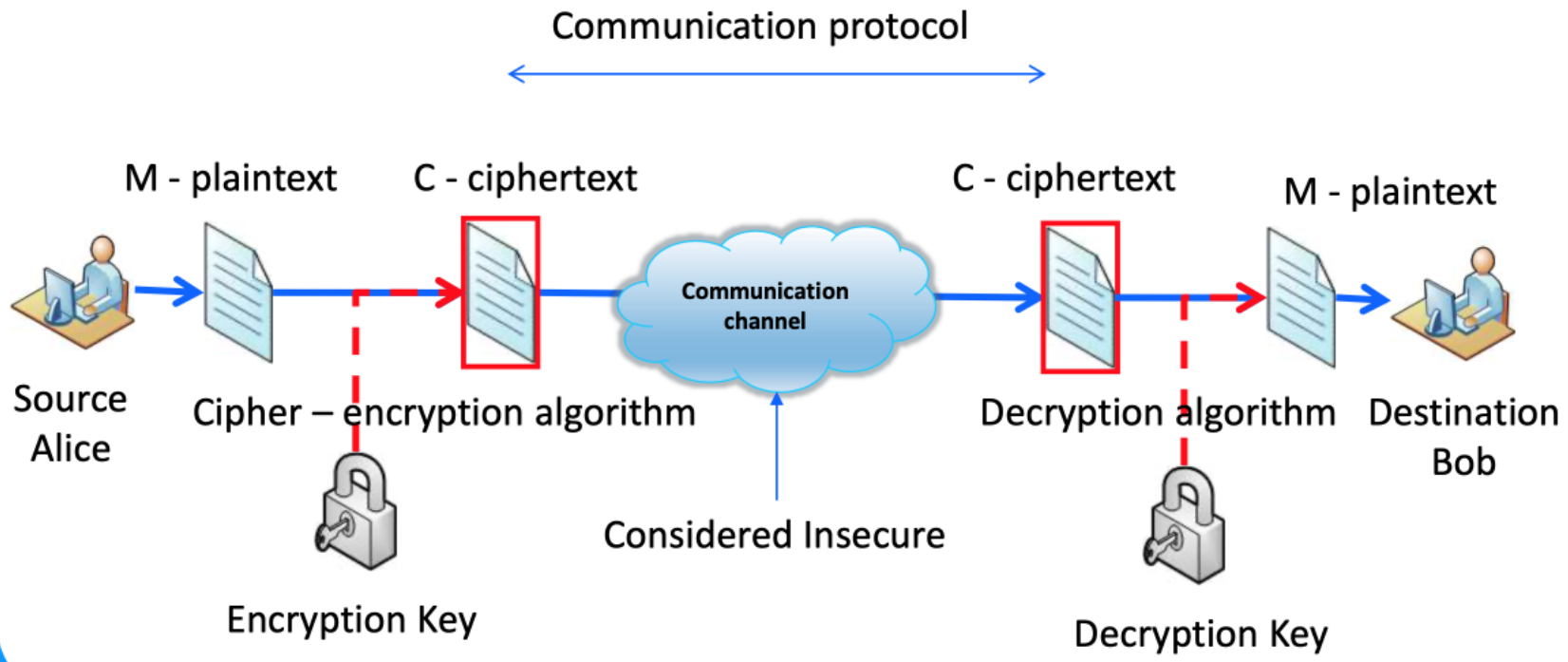
Click to add notes

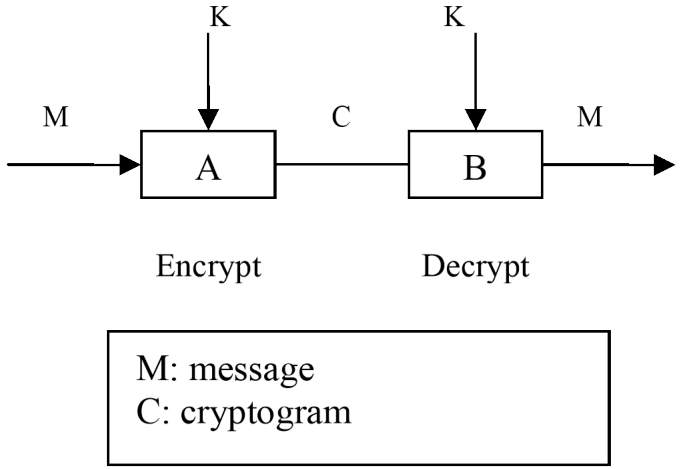
Slide 5 of 134 English

Notes Comments 130%

01. Crypto Intro – Symmetric Keys

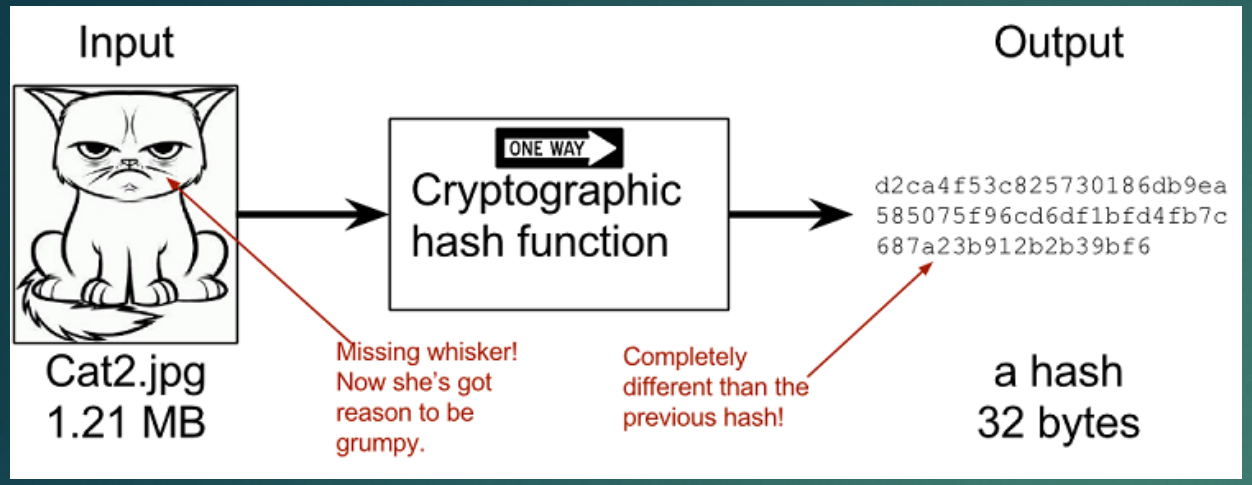
Key ingredients of a cryptographic system

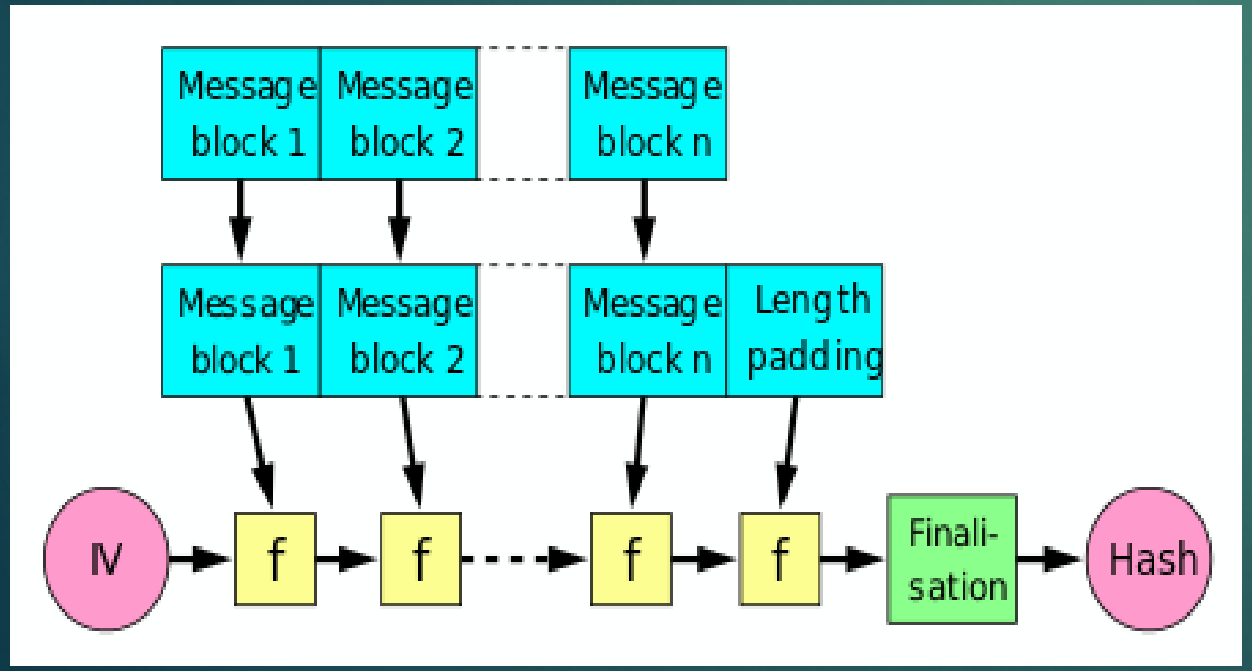


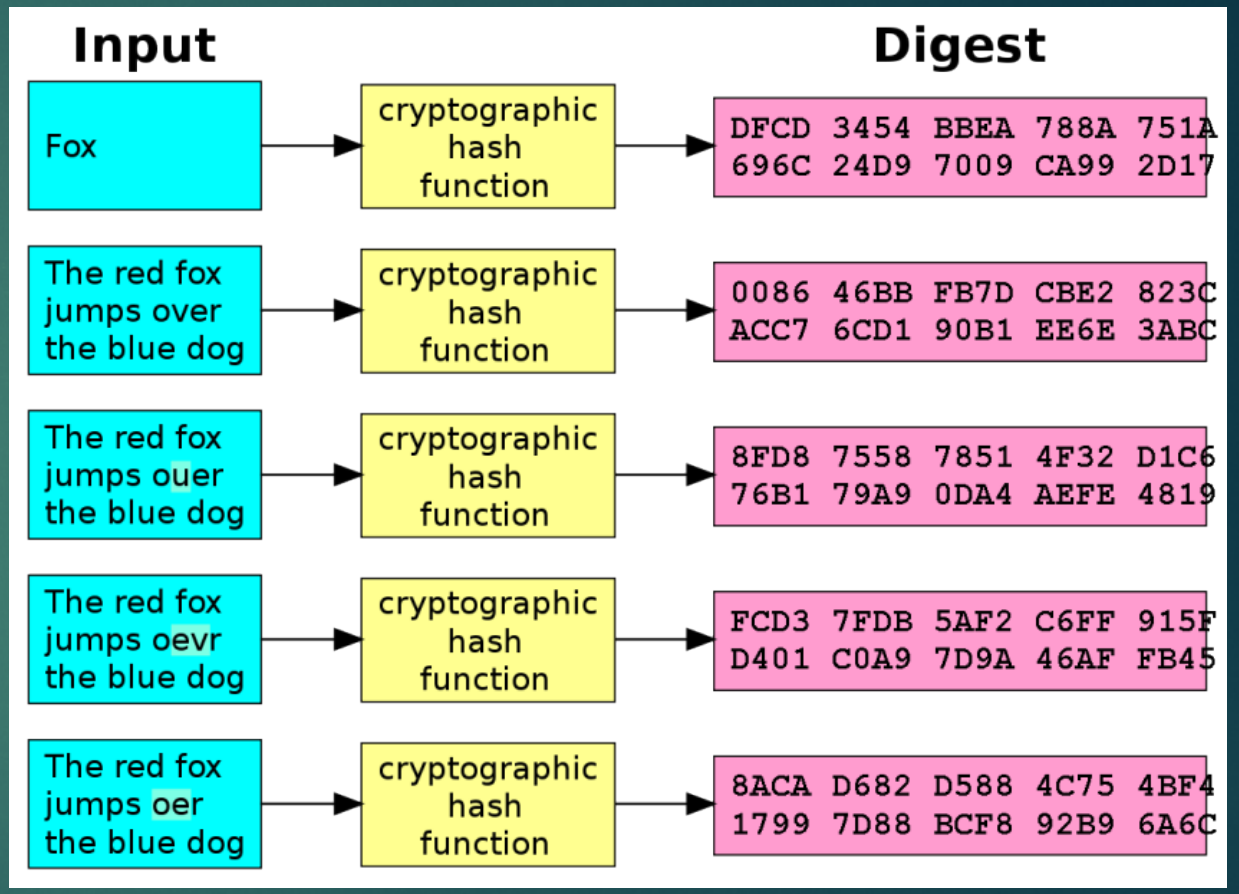


01. Crypto Intro - Hash

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

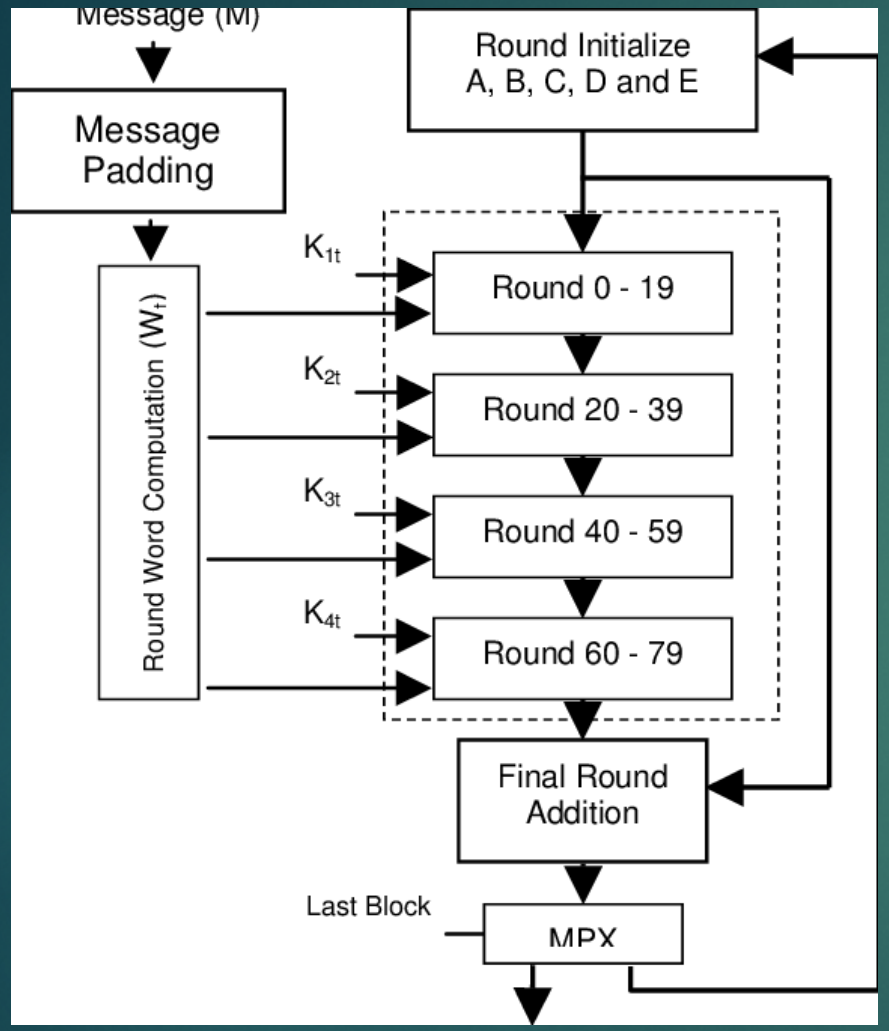




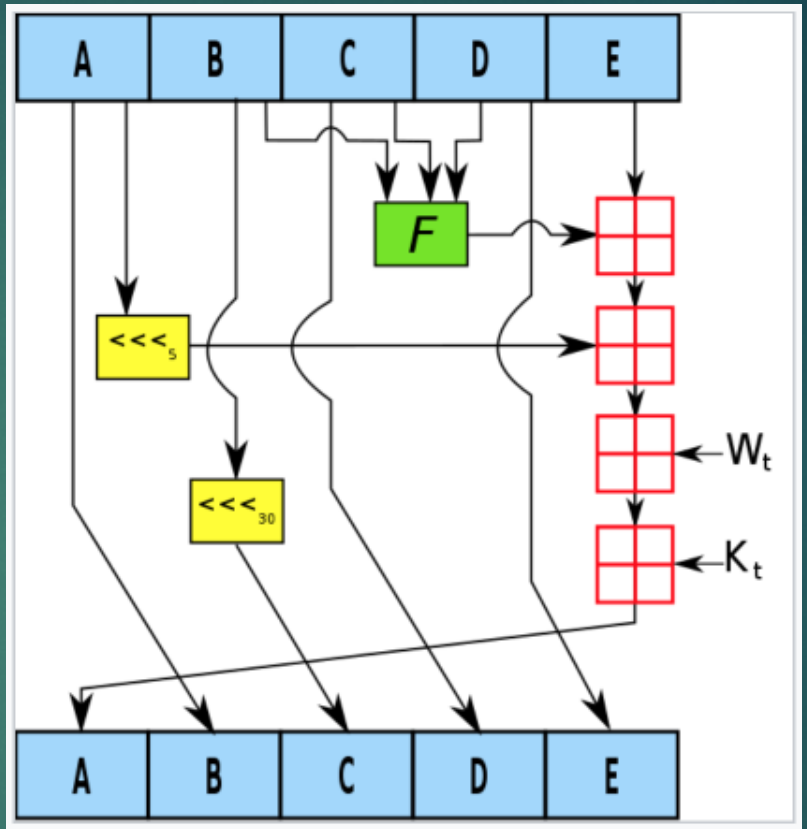


01. Crypto Intro - Hash

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

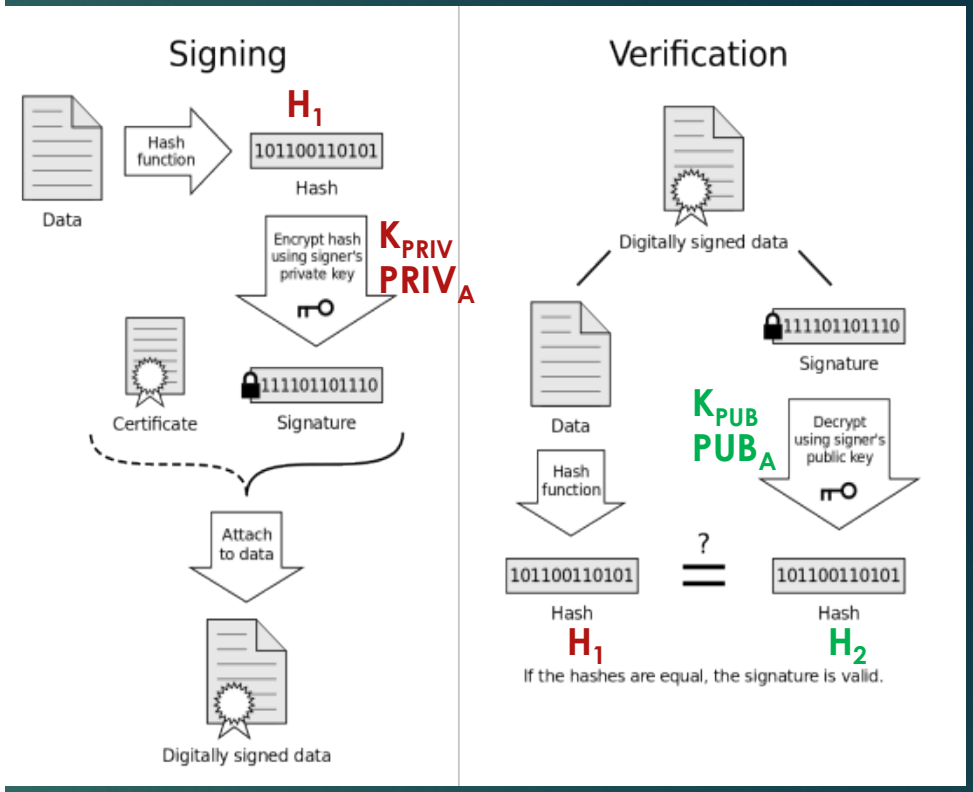


https://www.slideshare.net/ruchimaurya487/sha-secure-hashing-algorithm



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;
$\lll_n$ denotes a left bit rotation by *n* places;
n varies for each operation;
W_t is the expanded message word of round t;
K_t is the round constant of round t;
$\boxplus$ denotes addition modulo 2^{32}.

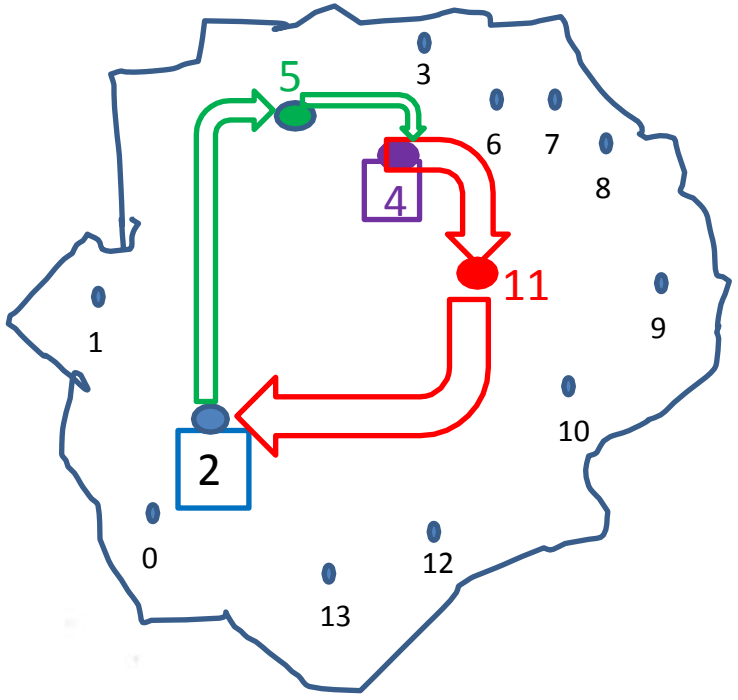
01. Crypto Intro – Asymmetric Keys

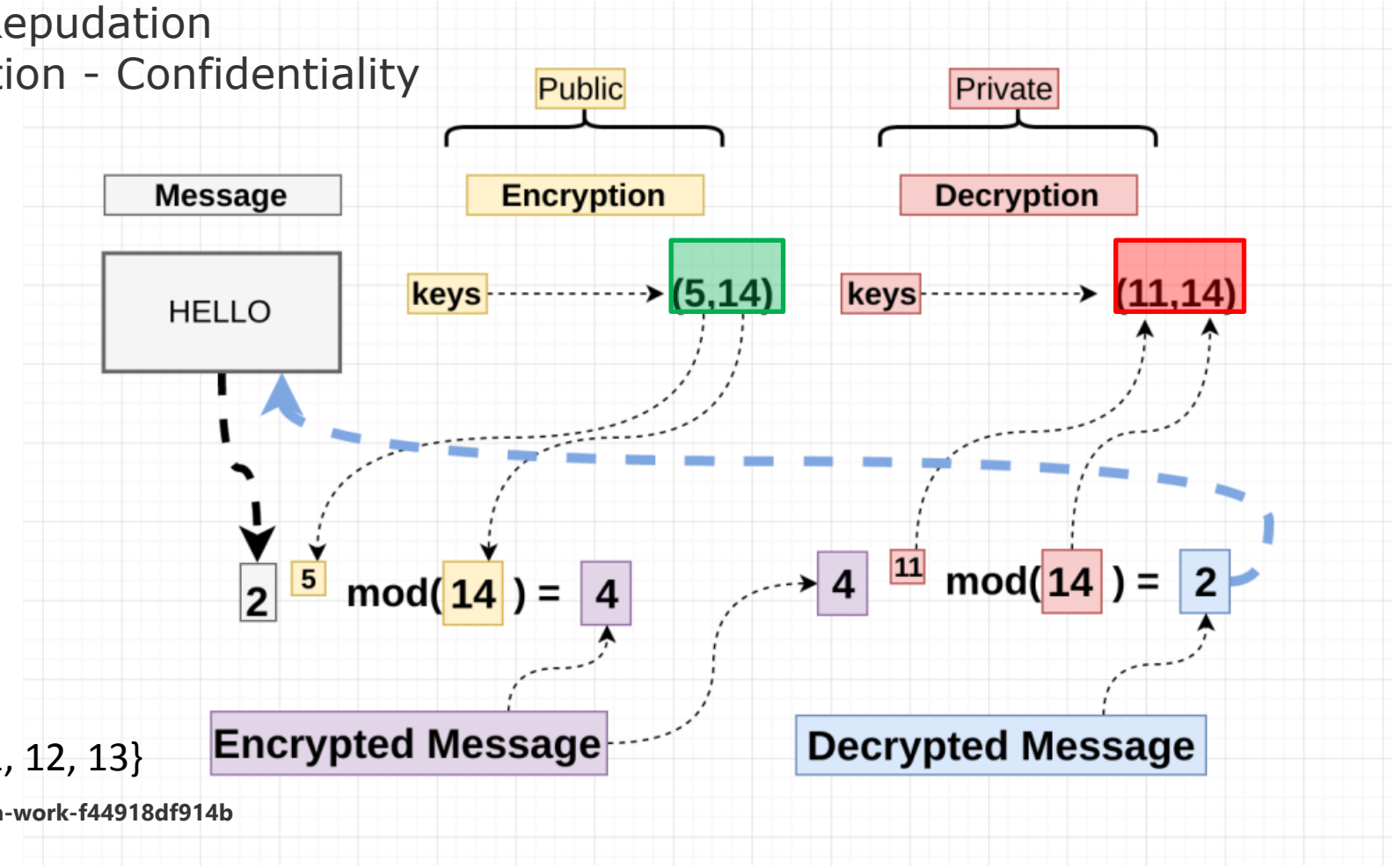


https://commons.wikimedia.org/wiki/File%3ADigital_Signature_diagram.svg

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

- How does RSA work:
  - Signature – Non- Repudation
  - Encryption/Decryption - Confidentiality



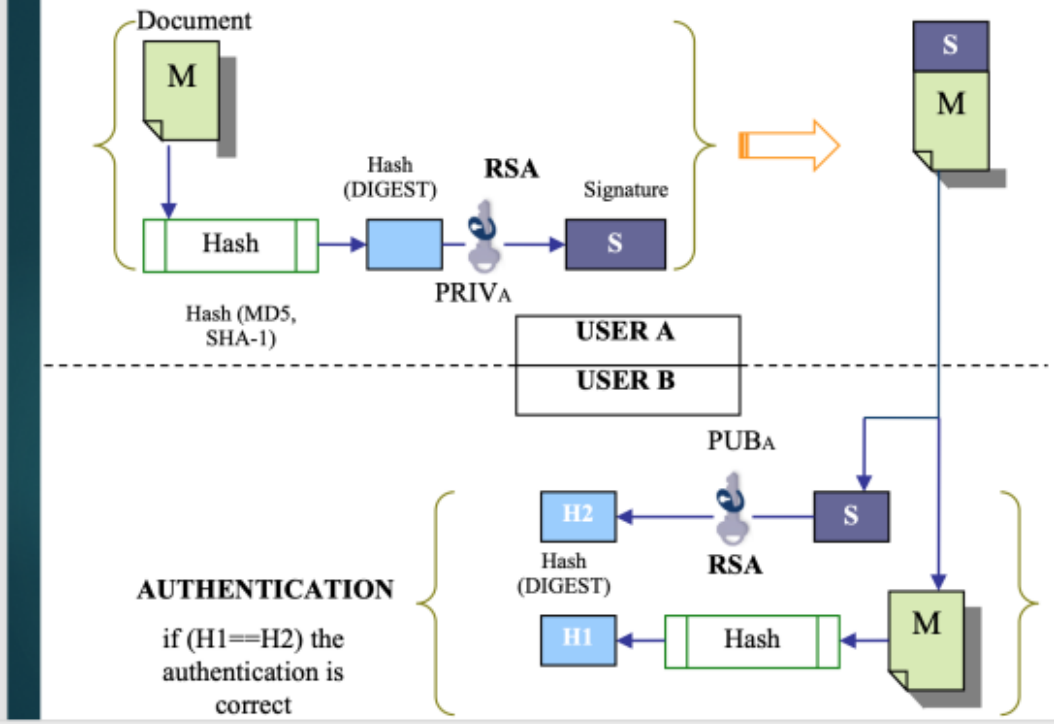


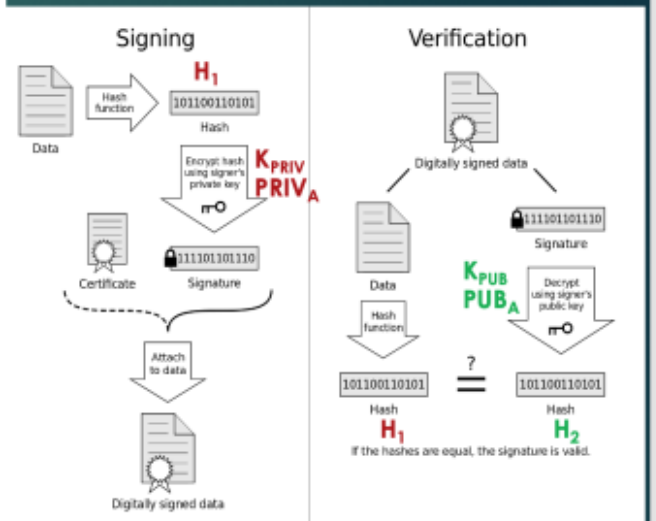
Z_{14} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}

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

01. Crypto Intro – Asymmetric Keys

9



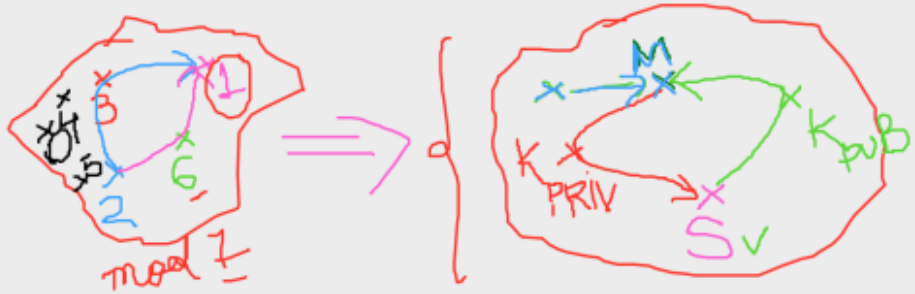


(2 ^ 6) mod 7 = (2 ^ 3) mod 7

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

M = 2
KPRIV = 3
KPUB = 6

(M ^ KPRIV) mod 7 = 2^3 mod 7 = 8 mod 7 = 1
2 ^ 6 mod 7 = 64 mod 7 = [(2^3) * (2^3)] mod 7 = [(2^3) mod 7] * [(2^3) mod 7] = 1 * 1 = 1



01. Crypto Intro – RSA simple numeric sample

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*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:
  - **Φ(n) = (p-1)*(q-1) = 52*60 = 3120.**
- Is chosen a secret private key **$PRIV_A$ = 71**; (meaning a number relatively prime with 3120 and in the period [max(p,q)+1,n-1]
- Is calculated the public key **PUB_A = 791** as multiplied reverse *mod 3120* of $PRIV_A$:
  - **(71*791) mod 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

- 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 - $PRIV_A$ = (71, 3233) = (EXP_PRIV_A, MOD_A)**:
  - **(1302^71) mod 3233 = 1984;**
  - **(1426^71) mod 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 - PUB_A = (791, 3233) = (EXP_PUB_A, MOD_A)**:
  - **(1984^791) mod 3233 = 1302**
  - **(2827^791) mod 3233 = 1426;**
  - **So H1 = 1302 1426;**
- and will calculate H(M) on the received message:

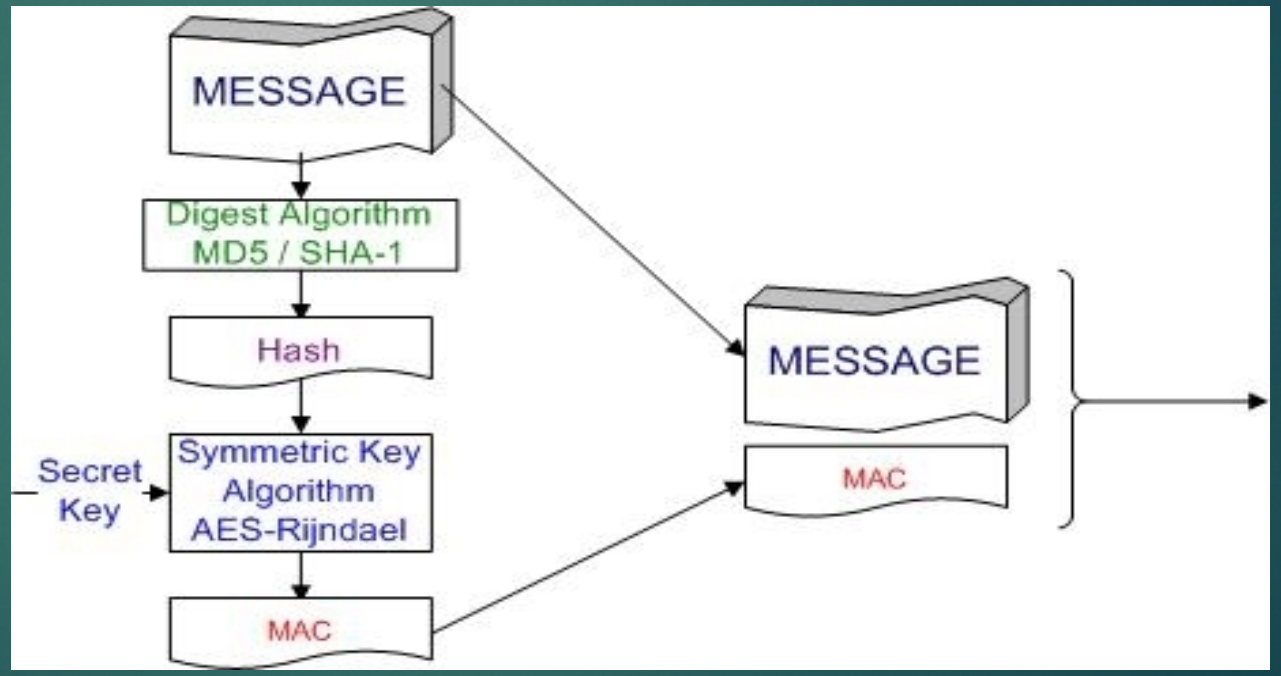
  H2 = H(M) = 1302 1426;

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

This definition is taken from RFC 2104:

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

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

where

H is a cryptographic hash function

m is the message to be authenticated

K is the secret key

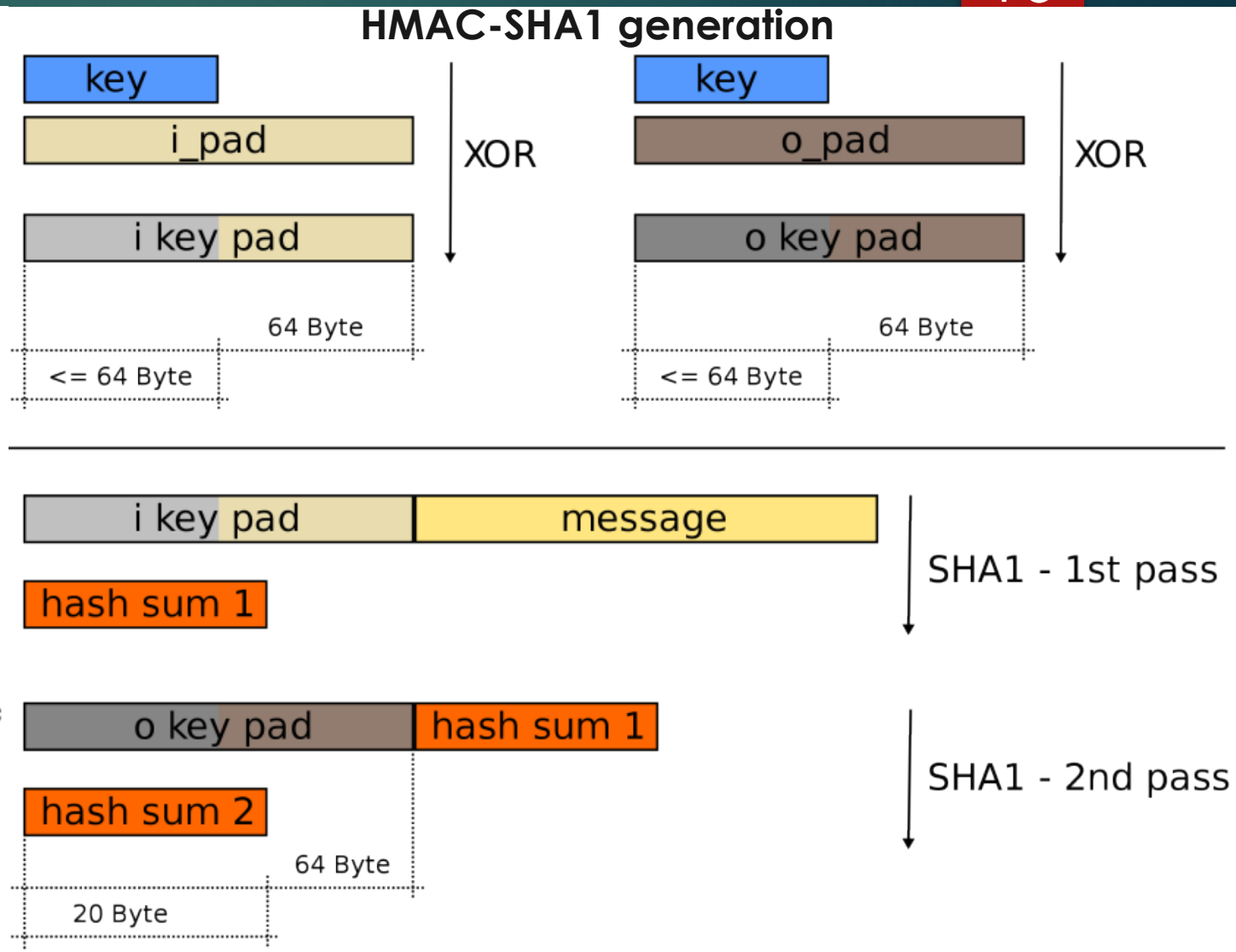
K' is a block-sized key derived from the secret key, *K*; either by padding to the right with 0s up to the block size, or by hashing down to less than or equal to the block size first and then padding to the right with zeros

|| denotes concatenation

⊕ 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



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

01. Crypto Intro a little bit more details

What is cryptology?

The meaning

- „kryptós“ („hidden, secret“) and „lógos“ („writing“, however in this context „lógos“ means „study“).
- Cryptology is about techniques and protocols making information available only for authorized persons.
- The fields: **cryptography** and **cryptanalysis**.

Cryptography (*kryptós* and *gráphein*, “to write”)

- Science of encryption systems guaranteeing secure and confidential storage and exchange of information (e.g. between computers).
- A secure exchange of the encryption keys and integrity checking, e.g. for online banking, for electronic elections, or for electronic money.
- Based on (unsolved/difficult) mathematical problems.

Source: https://www.cryptool.org/en/jct/

What is cryptology?

Cryptanalysis (*kryptós* and *analýein*, “to loosen” or “to untie”)

- The counter part to cryptography and studies theories and techniques for testing and breaking cryptographic methods.
- It tries e.g. to derive information about the original plaintext or the used encryption key by investigating a ciphertext (the result of an encryption process).
- Maths and computer science are used (e.g. statistical tests, entropy, analysis of frequency and structure, complexity considerations, brute-force algorithms and much more).

Source: https://www.cryptool.org/en/jct/

History of cryptography and cryptanalysis

Classic Cryptography

- Transposition Ciphers rearrange the order of letters in a message (e.g., 'hello world' becomes 'ehlol owrdl').
- Substitution Ciphers replace letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet).
- Ciphertexts reveal statistical information about the plaintext, and that information can often be used to break the cipher.
- Caesar cipher, Steganography (a message tattooed on a slave's shaved head and concealed under the regrown hair).

Source: https://en.wikipedia.org/wiki/Cryptography

History of cryptography and cryptanalysis

Computer Era

- Extensive use of mathematics, including aspects of information theory, computational complexity, statistics, combinatorics, abstract algebra, number theory, and finite mathematics generally.
- Development of digital computers and electronics helped in cryptanalysis, it made possible much more complex ciphers.
- Many computer ciphers can be characterized by their operation on binary bit sequences (sometimes in groups or blocks), unlike classical and mechanical schemes, which generally manipulate traditional characters (i.e., letters and digits) directly.
- Good modern ciphers have stayed ahead of cryptanalysis; it is typically the case that use of a quality cipher is very efficient.

Source: https://en.wikipedia.org/wiki/Cryptography

History of cryptography and cryptanalysis

Advent of Modern Cryptography

- Cryptanalysis of the new mechanical devices proved to be both difficult and laborious.
- Extensive open academic research into cryptography is relatively recent. Cryptography systems based on mathematical problems that are easy to state but have been found difficult to solve.
- Cryptographic algorithm and system designers must also sensibly consider probable future developments while working on their designs.

Source: https://en.wikipedia.org/wiki/Cryptography

Modern cryptography

Symmetric-Key Cryptography

- Encryption methods in which both the sender and receiver share the same key.
- Symmetric key ciphers are implemented as either block ciphers or stream ciphers.
- Stream ciphers create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad.
- Cryptographic hash functions take a message of any length as input, and output a short, fixed length hash. For good hash functions, an attacker cannot find two messages that produce the same hash.
- Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key can be used to authenticate the hash value upon receipt.
- Data Encryption Standard (DES), Advanced Encryption Standard (AES), SHA algo family (designed by NSA)

Source: https://en.wikipedia.org/wiki/Cryptography

Modern cryptography

Public-Key Cryptography

- A significant disadvantage of symmetric ciphers is the key management necessary to use them securely.
- Asymmetric key cryptography use two different but mathematically related keys: a public key and a private key.
- In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret.
- In a public-key encryption system, the public key is used for encryption, while the private or secret key is used for decryption.
- Diffie–Hellman key exchange protocol, a solution that is now widely used in secure communications to allow two parties to secretly agree on a shared encryption key.
- RSA (Ronald Rivest, Adi Shamir, and Len Adleman) algorithm, Elliptic Curve techniques.

Source: https://en.wikipedia.org/wiki/Cryptography

Modern cryptography

Public-Key Cryptography

- Used for implementing digital signature schemes.
- Digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. Two algorithms: one for signing (secret key is used to process the message or a hash), and one for verification (the matching public key is used to check the validity of the signature).
- Digital signatures are central to the operation of public key infrastructures and many network security schemes.
- RSA and DSA are two of the most popular digital signature schemes.

Source: https://en.wikipedia.org/wiki/Cryptography

Modern cryptography

Cryptanalysis

- The goal of cryptanalysis is to find some weakness or insecurity in a cryptographic scheme, thus permitting its subversion or evasion.

- Most ciphers, apart from the one-time pad, can be broken with enough computational effort by brute force attack, but the amount of effort needed may be exponentially dependent on the key size.

- Cryptanalytic attacks: ciphertext-only attack (access only to the ciphertext), known-plaintext attack (access to a ciphertext and its corresponding plaintext), chosen-plaintext attack (choose a plaintext and learn its corresponding ciphertext), gardening (encouraging a target to use known plaintext in an encrypted message), man-in-the-middle attack (gets in between the sender)and the recipient, accesses and modifies the traffic and then forwards it to the recipient).

Source: https://en.wikipedia.org/wiki/Cryptography

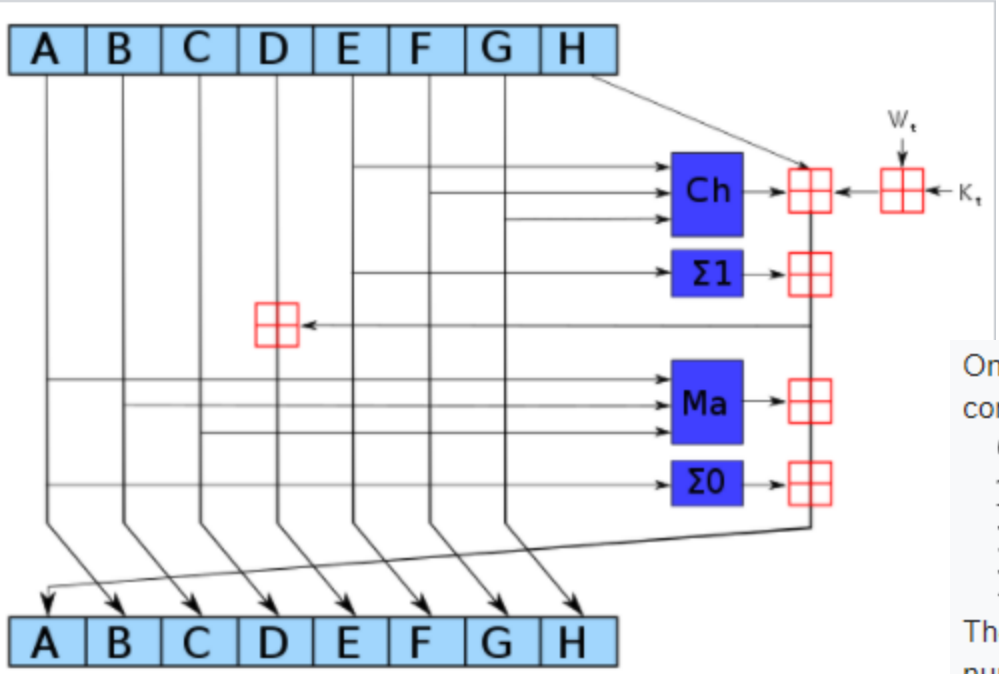
Hash Functions

- 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.

Source: https://en.wikipedia.org/wiki/Hash_function

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:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

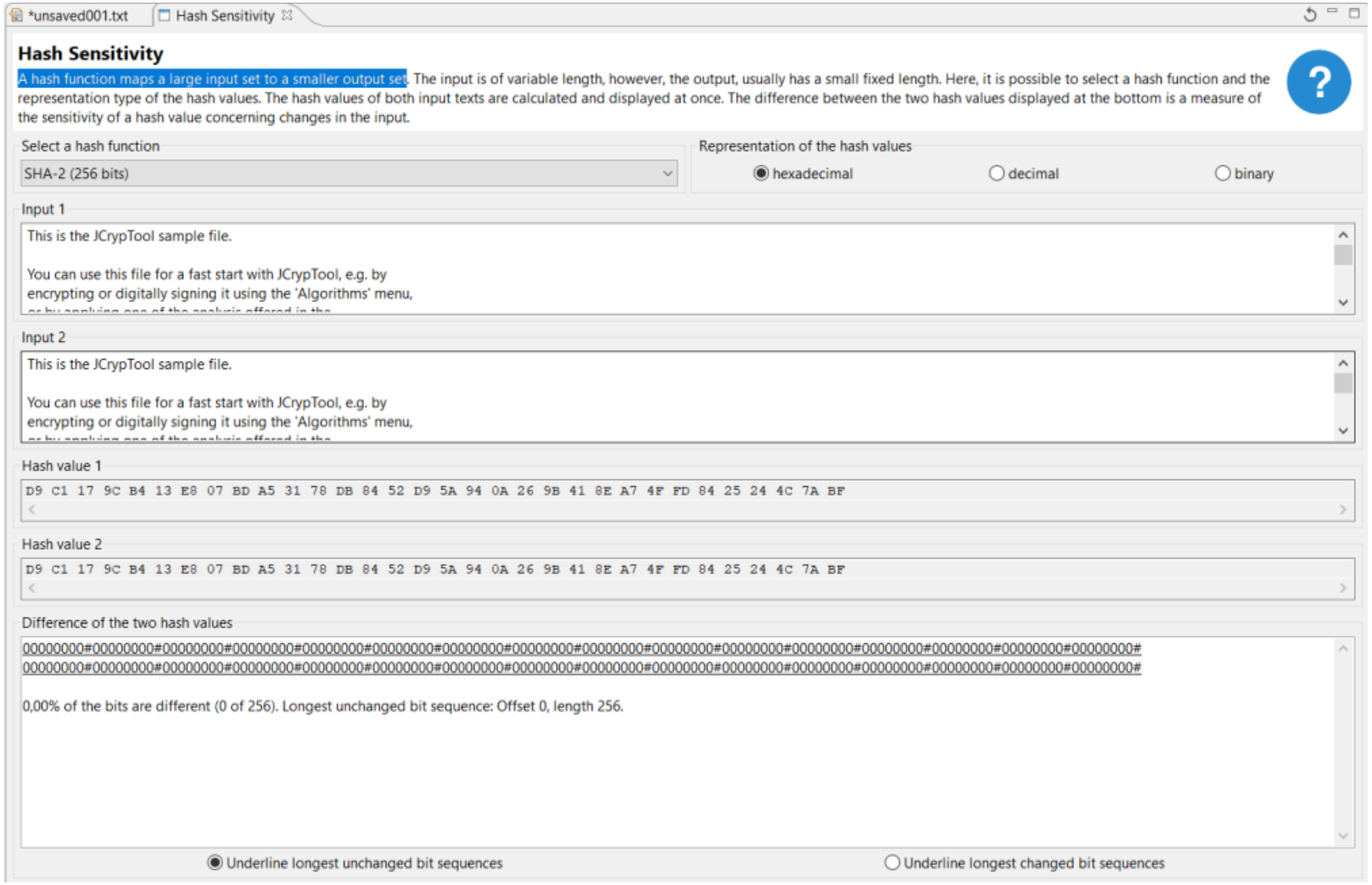
The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.

The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

Source: https://en.wikipedia.org/wiki/SHA-2

Cryptography Elements used in Blockchains

JCrypTool DEMO - Hash Sensitivity



Source: https://www.cryptool.org/en/jct/

Symmetric-Key Cryptography - Advanced Encryption Standard (AES)

- Defined in each of:
  - FIPS PUB 197: Advanced Encryption Standard (AES).
  - ISO/IEC 18033-3: Block ciphers.
- Based on a design principle known as a substitution–permutation network.
- A fixed block size of 128 bits, and a key size of 128, 192, or 256 bits.
- The key size specifies the number of transformation rounds that convert the plaintext into the ciphertext:
  - 10 rounds for 128-bit keys.
  - 12 rounds for 192-bit keys.
  - 14 rounds for 256-bit keys.

Source: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric-Key Cryptography - Advanced Encryption Standard (AES)

High-level description of the algorithm [edit]

1. `KeyExpansion` – round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
    1. `AddRoundKey` – each byte of the state is combined with a byte of the round key using bitwise xor.
3. 9, 11 or 13 rounds:
    1. `SubBytes` – a non-linear substitution step where each byte is replaced with another according to a lookup table.
    2. `ShiftRows` – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    3. `MixColumns` – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
    4. `AddRoundKey`
4. Final round (making 10, 12 or 14 rounds in total):
    1. `SubBytes`
    2. `ShiftRows`
    3. `AddRoundKey`

Source: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric-Key Cryptography - Advanced Encryption Standard (AES)

AES key schedule for a 128-bit key

Also define RotWord as a one-byte left circular shift:[note 6]

$$\mathrm{RotWord}([\,b_0 \quad b_1 \quad b_2 \quad b_3\,]) = [\,b_1 \quad b_2 \quad b_3 \quad b_0\,]$$

and SubWord as an application of the AES S-box to each of the four bytes of the word:

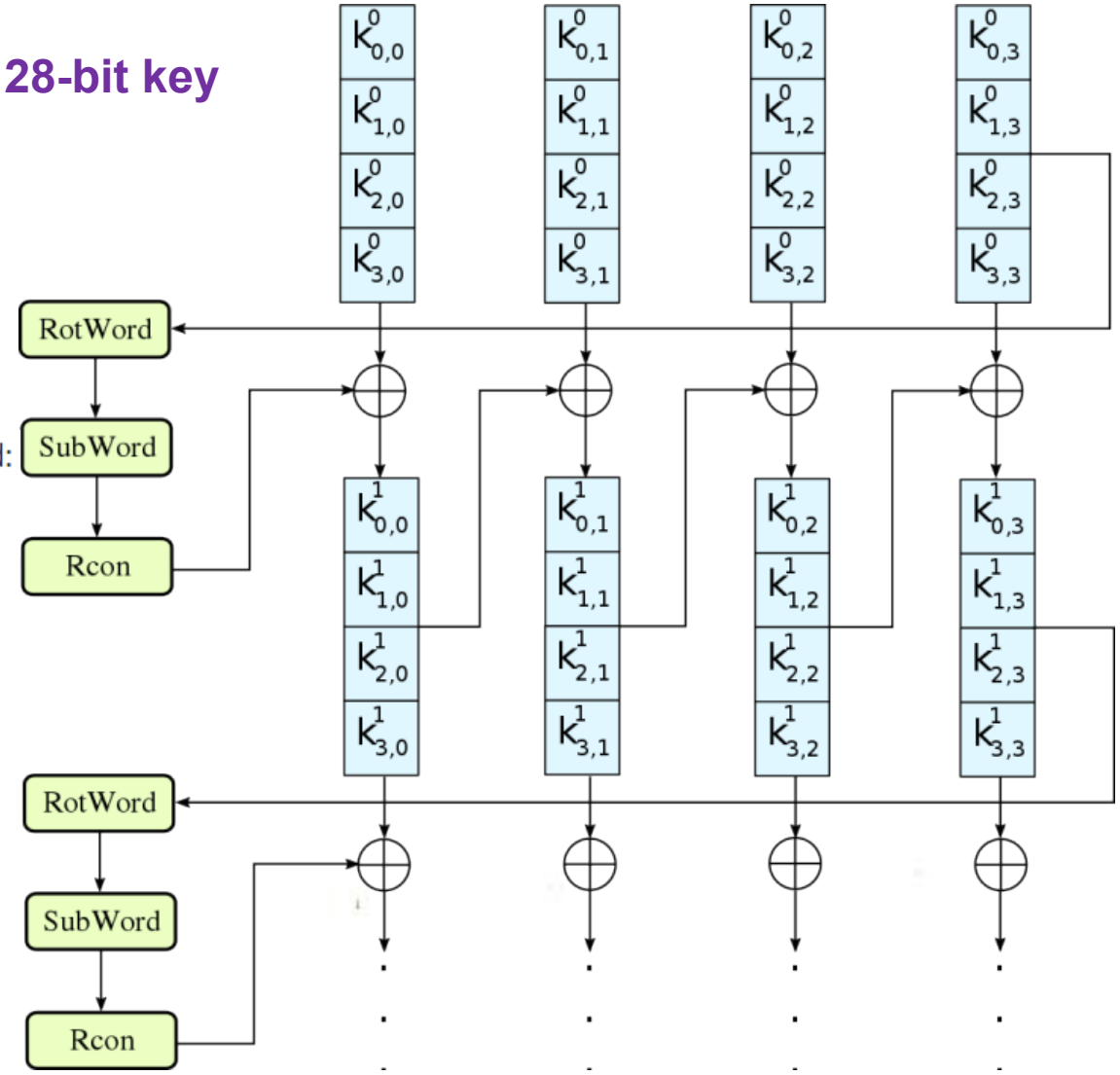
$$\mathrm{SubWord}([\,b_0 \quad b_1 \quad b_2 \quad b_3\,]) = [\,\mathrm{S}(b_0) \quad \mathrm{S}(b_1) \quad \mathrm{S}(b_2) \quad \mathrm{S}(b_3)\,]$$

The round constant $rcon_i$ for round i of the key expansion is the 32-bit word:

$$rcon_i = [\,rc_i \quad 00_{16} \quad 00_{16} \quad 00_{16}\,]$$

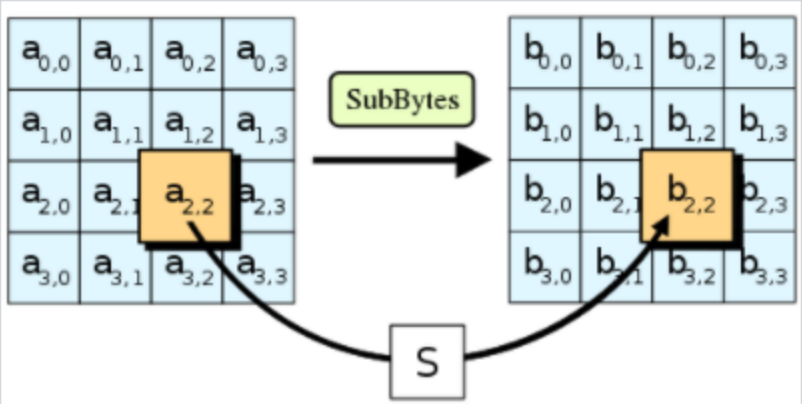
where rc_i is an eight-bit value defined as:

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11\mathrm{B}_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

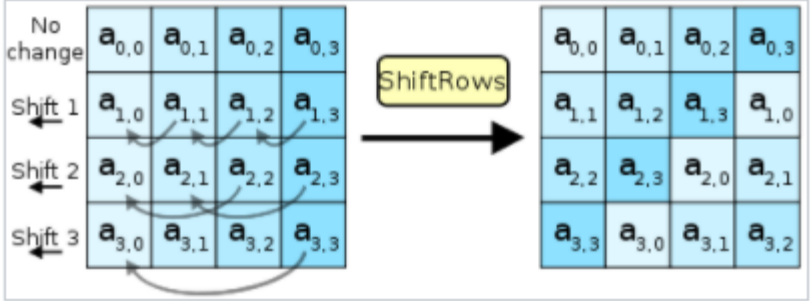


Source: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



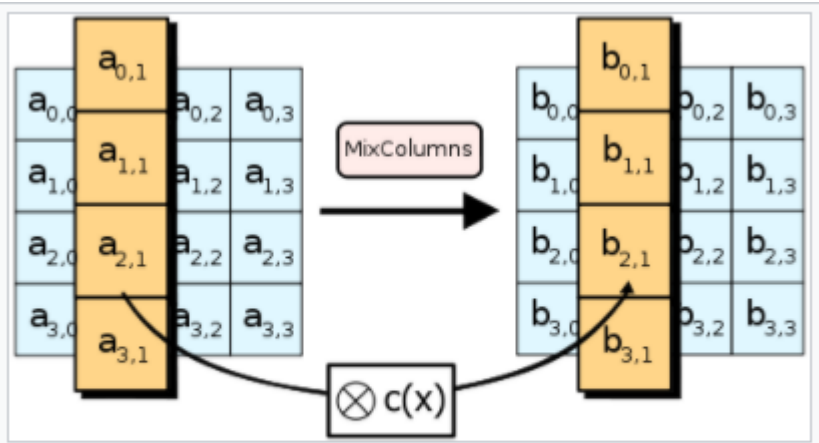
In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S; $b_{ij} = S(a_{ij})$.



In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs incrementally for each row.

Source: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

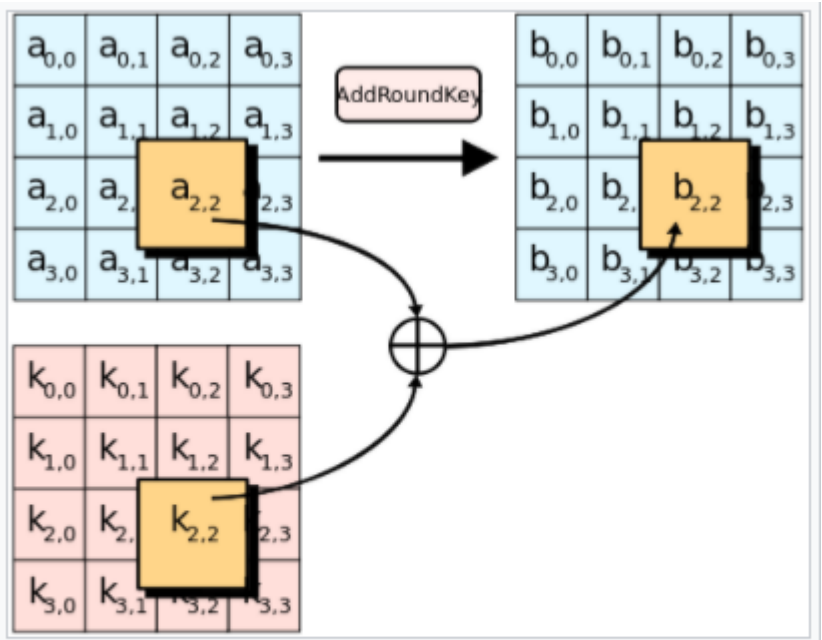
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



In the MixColumns step, each column of the state is multiplied with a fixed polynomial $c(x)$.

Multiplication is modulo irreducible polynomial

$$x^8 + x^4 + x^3 + x + 1.$$



In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation ($\oplus$).

Source: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric-Key Cryptography
- Advanced Encryption Standard (AES)

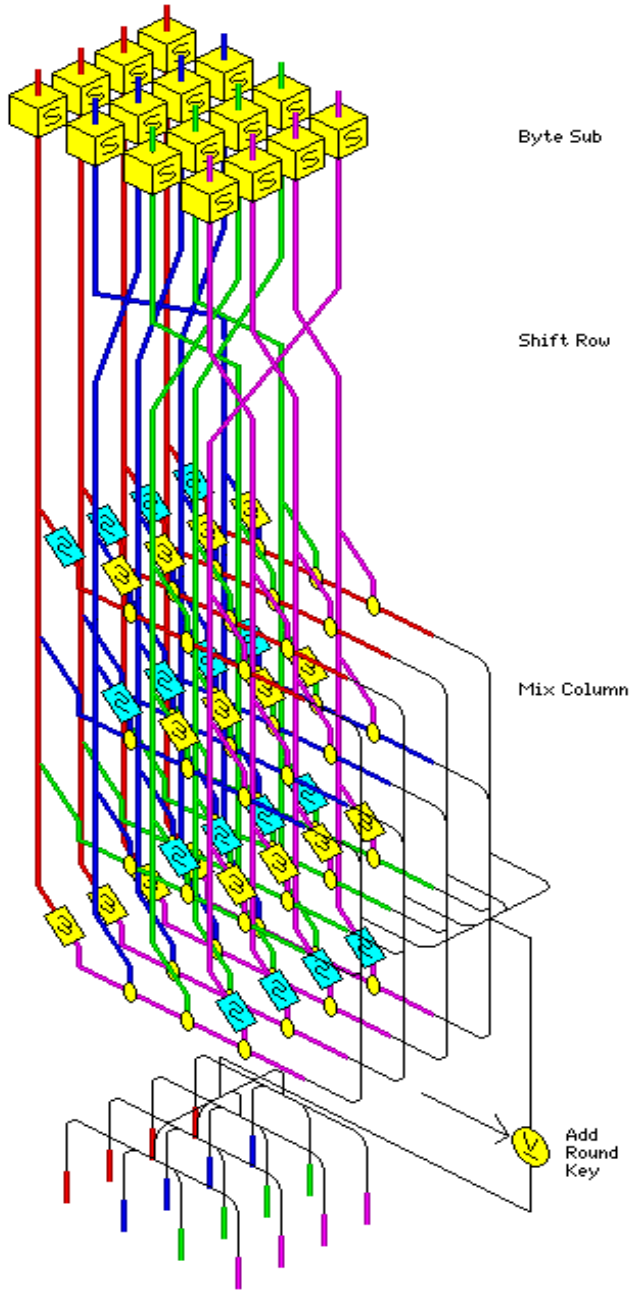
```
Rijndael(State,CipherKey) {
    KeyExpansion(CipherKey,ExpandedKey) ;

    AddRoundKey(State,ExpandedKey);

        for( i=1 ; i<Nr ; i++ )
            Round(State,ExpandedKey + Nb*i);
        FinalRound(State,ExpandedKey + Nb*Nr);
        }
}
```

```
//funcţie folosită doar dacă Nk<=6
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)]) {
        for(i = 0; i < Nk; i++)
    W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);

                for(i = Nk; i < Nb * (Nr + 1); i++)
                {
                        temp = W[i - 1];
                        if (i % Nk = = 0)
                 temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
                        W[i] = W[i - Nk] ^ temp;
                }
}
```

```
/*
//funcţie folosită doar dacă Nk>6
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)]) {
        for(i = 0; i < Nk; i++)
        W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);

                for(i = Nk; i < Nb * (Nr + 1); i++)
                {
                        temp = W[i - 1];
                        if (i % Nk == 0)
                temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
         else if (i % Nk == 4) temp = SubByte(temp);

         W[i] = W[i - Nk] ^ temp;
                }
}
*/
```

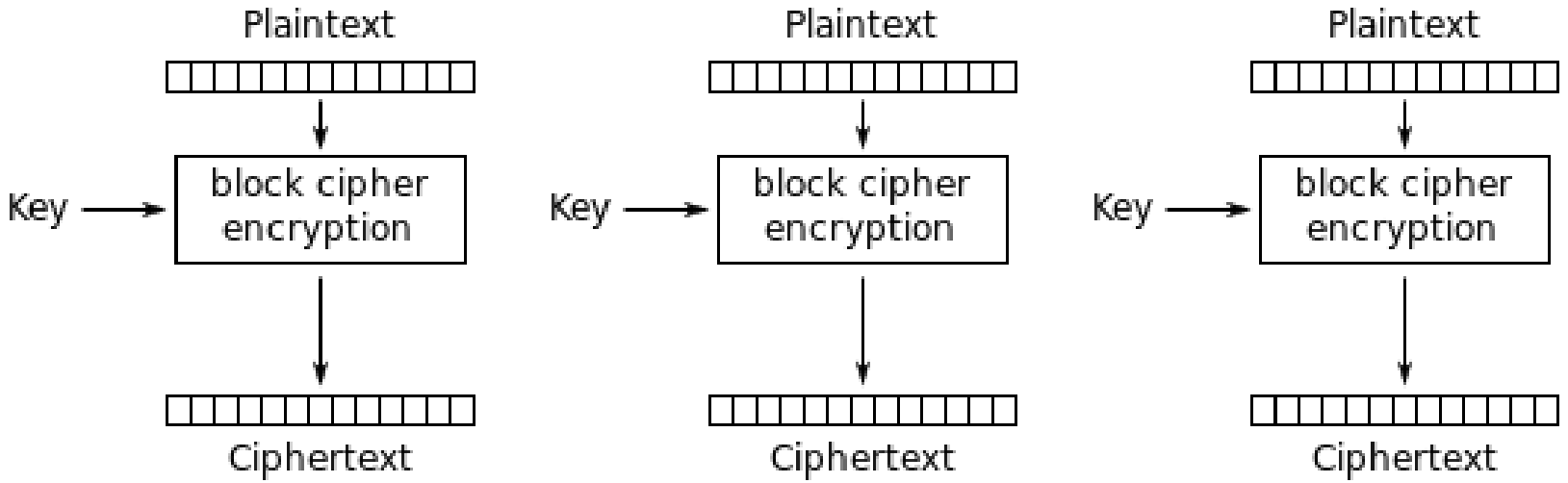
Symmetric-Key Cryptography
- Advanced Encryption Standard (AES)

```
Round(State,RoundKey) {
        ByteSub(State);
        ShiftRow(State);
        MixColumn(State);
        AddRoundKey(State,RoundKey);
}

FinalRound(State,RoundKey) {
        ByteSub(State) ;
        ShiftRow(State) ;
        AddRoundKey(State,RoundKey);
}

AddRoundKey(State,ExpandedKey) {
        State = (State ^ ExpandedKey);
}
```



A round in Rijndael algorithm realized by the **Round(State, RoundKey)** function

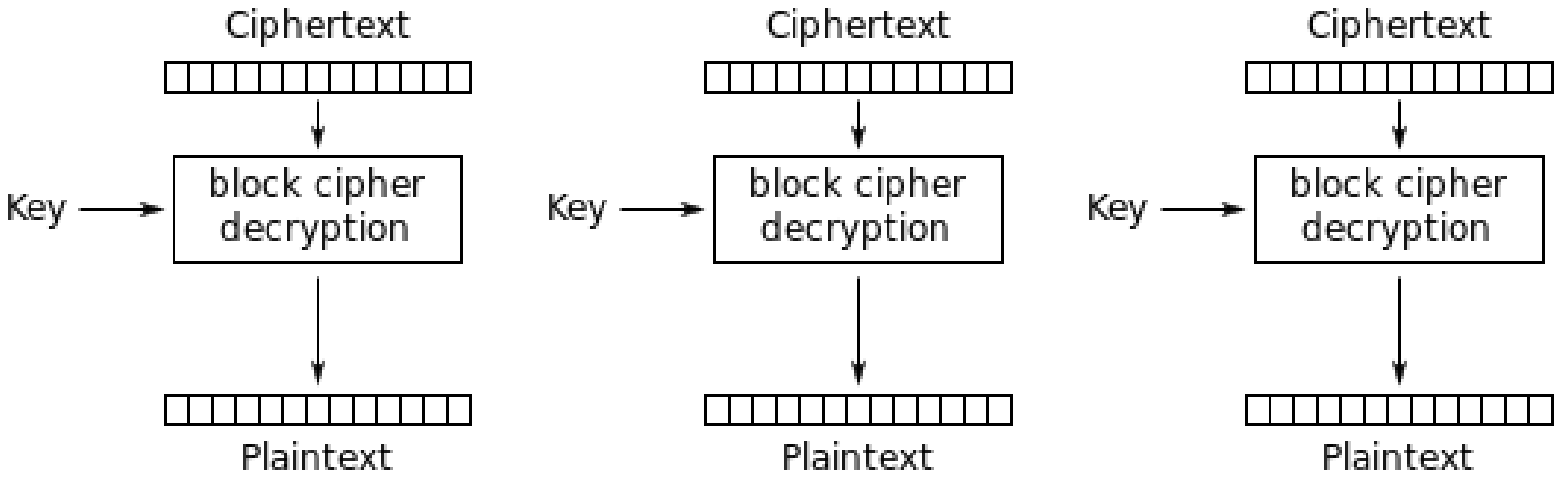
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Electronic Codebook (ECB) mode encryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

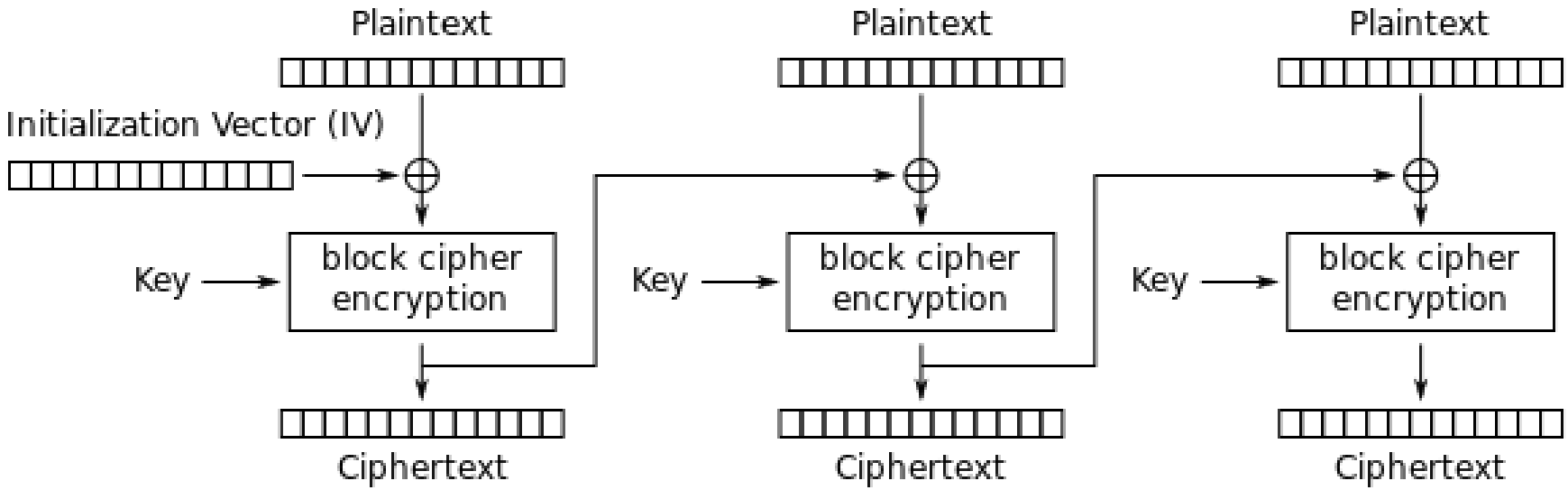
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Electronic Codebook (ECB) mode decryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

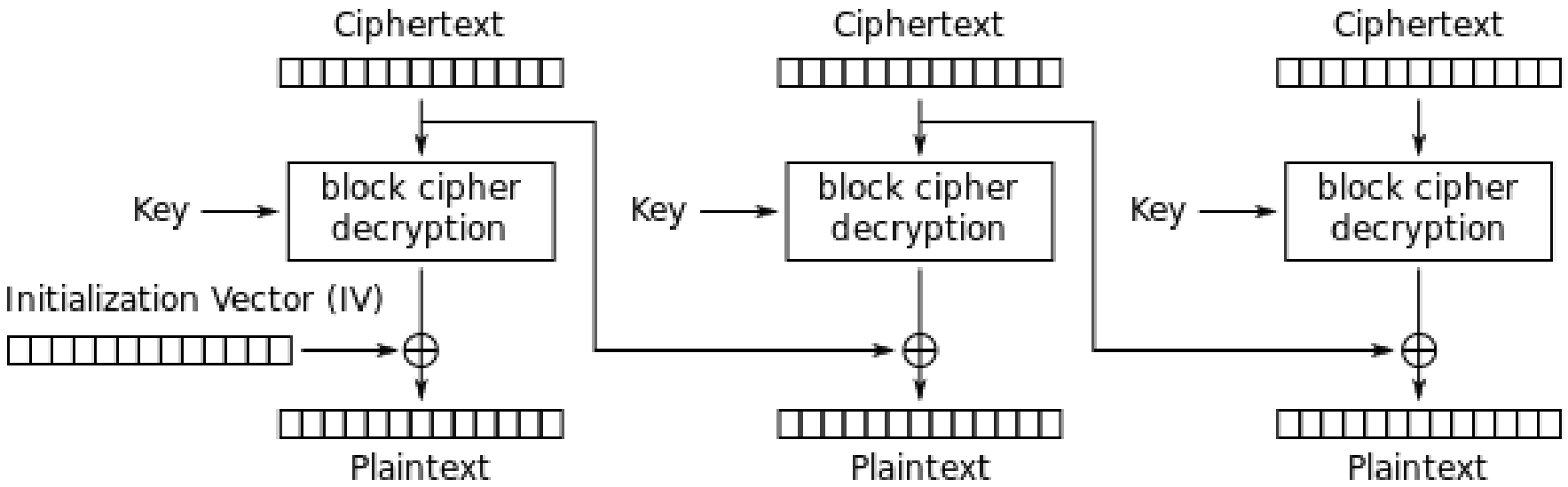
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Cipher Block Chaining (CBC) mode encryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

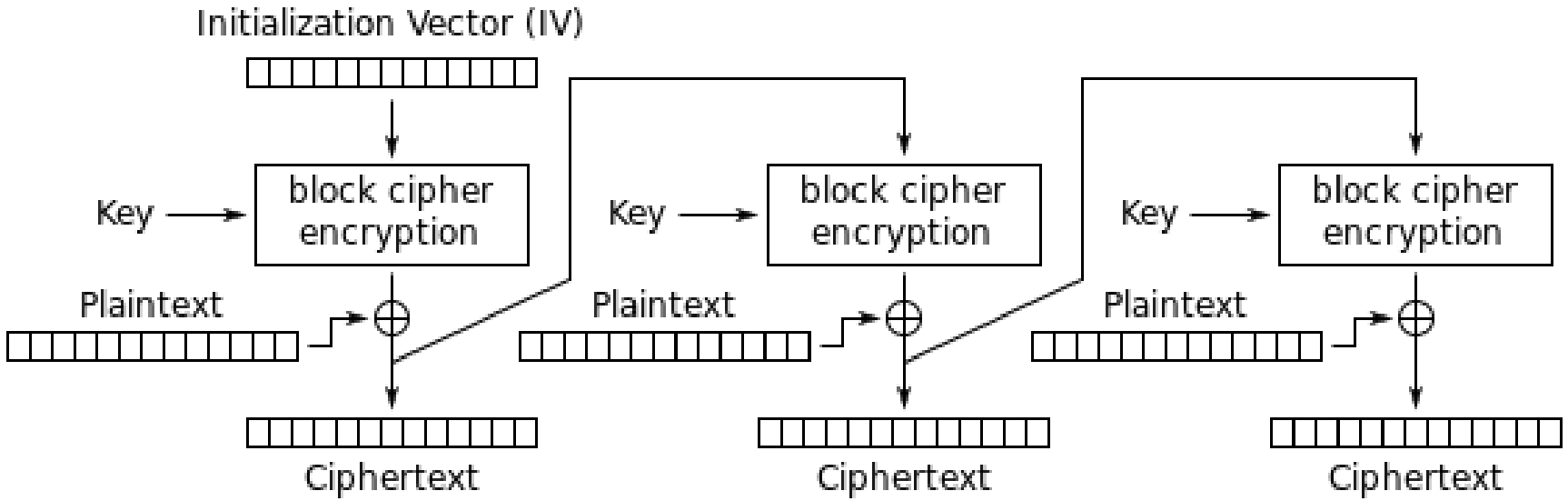
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Cipher Block Chaining (CBC) mode decryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

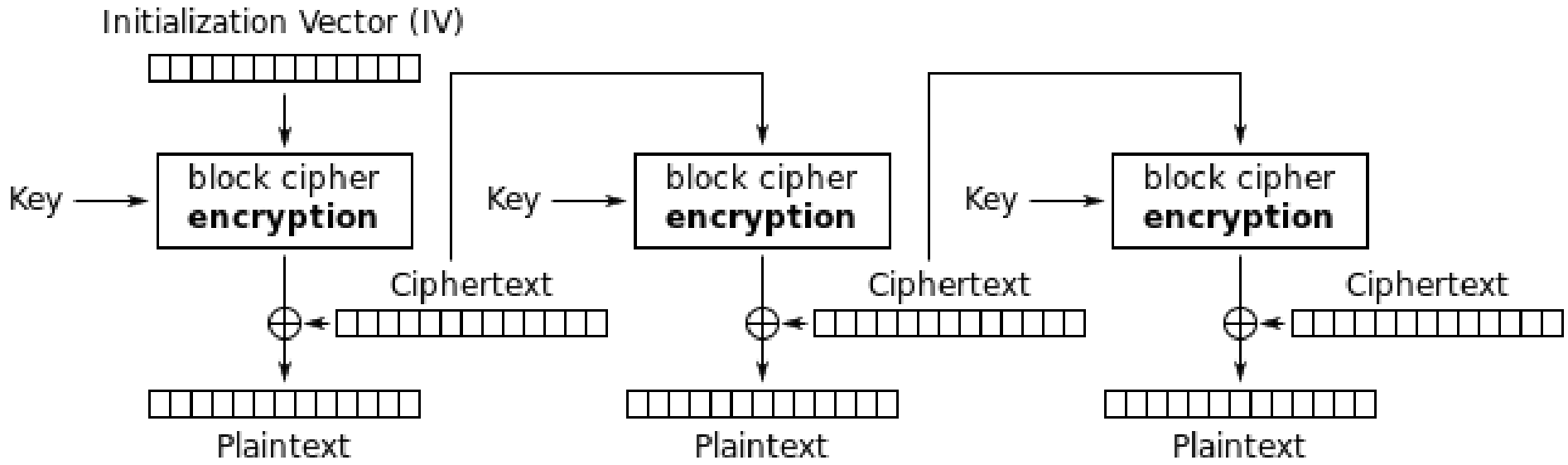
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Cipher Feedback (CFB) mode encryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

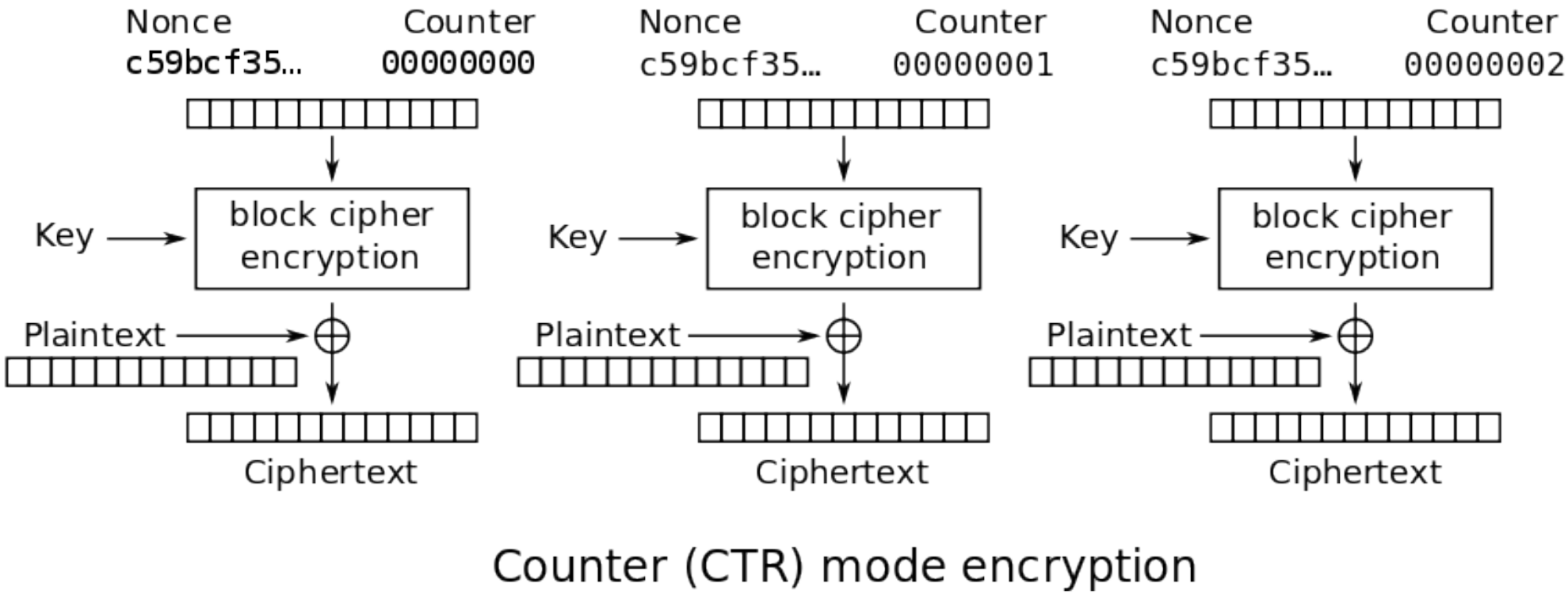
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Cipher Feedback (CFB) mode decryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

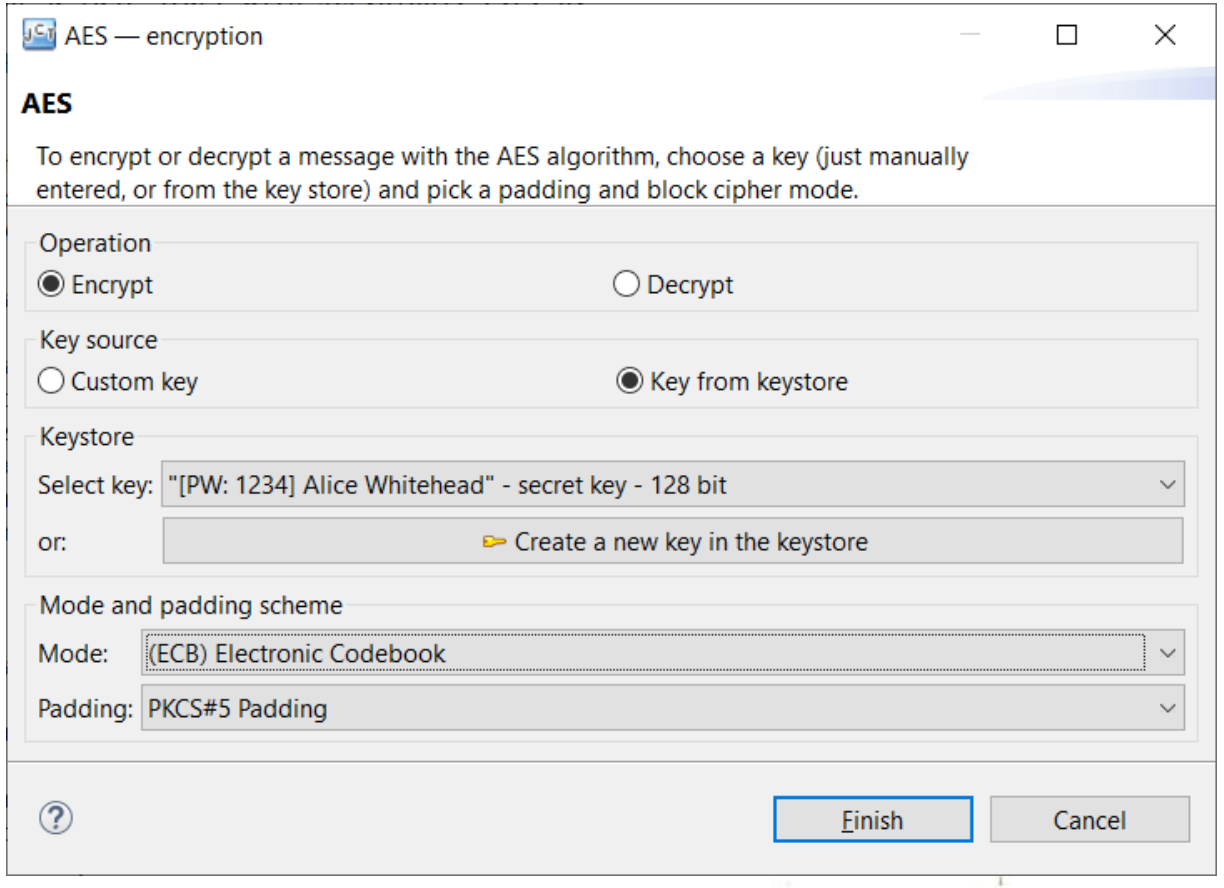
Symmetric-Key Cryptography - Advanced Encryption Standard (AES)



Counter (CTR) mode encryption

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

JCrypTool DEMO – AES Encryption



Source: https://www.cryptool.org/en/jct/

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

- First public-key cryptosystems.
- Widely used for encryption and signing.
- RSA key generation:

1 - Generate two primes p and q and compute their product N.

2 - Pick e such that $\gcd(e, \phi(N)) = 1$, where $\phi(n) = (p-1)(q-1)$.

3 - Compute d such that $d = e^{-1} \bmod \phi(N)$.

$\boxed{\text{Public Key} = \langle e, N \rangle}$ $\boxed{\text{Private Key} = \langle d, N \rangle}$

Source: https://www.cryptool.org/en/jct/
Source: Cesar Alison MonteiroPaixao: "An efficient variant of the RSA cryptosystem", eprint paper, 2003

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

- RSA encryption / decryption:

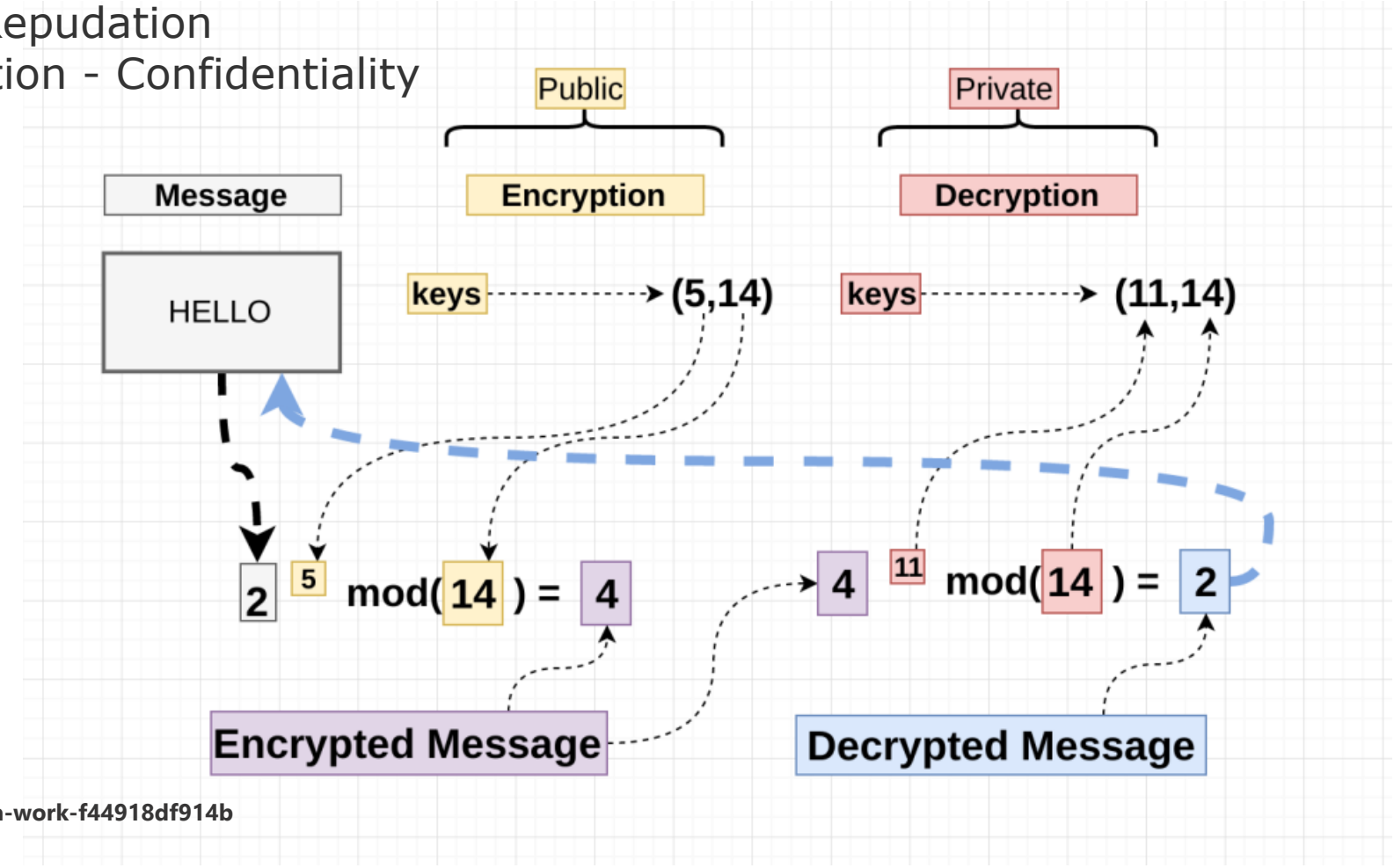
Encryption: Given a plaintext M and the public key $\langle e, N \rangle$, compute the ciphertext $C = M^e \bmod N$.

Decryption: Given a ciphertext C and the private key $\langle d, N \rangle$, compute the plaintext $M = C^d \bmod N$.

Source: https://www.cryptool.org/en/jct/
Source: Cesar Alison MonteiroPaixao: "An efficient variant of the RSA cryptosystem", eprint paper, 2003

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

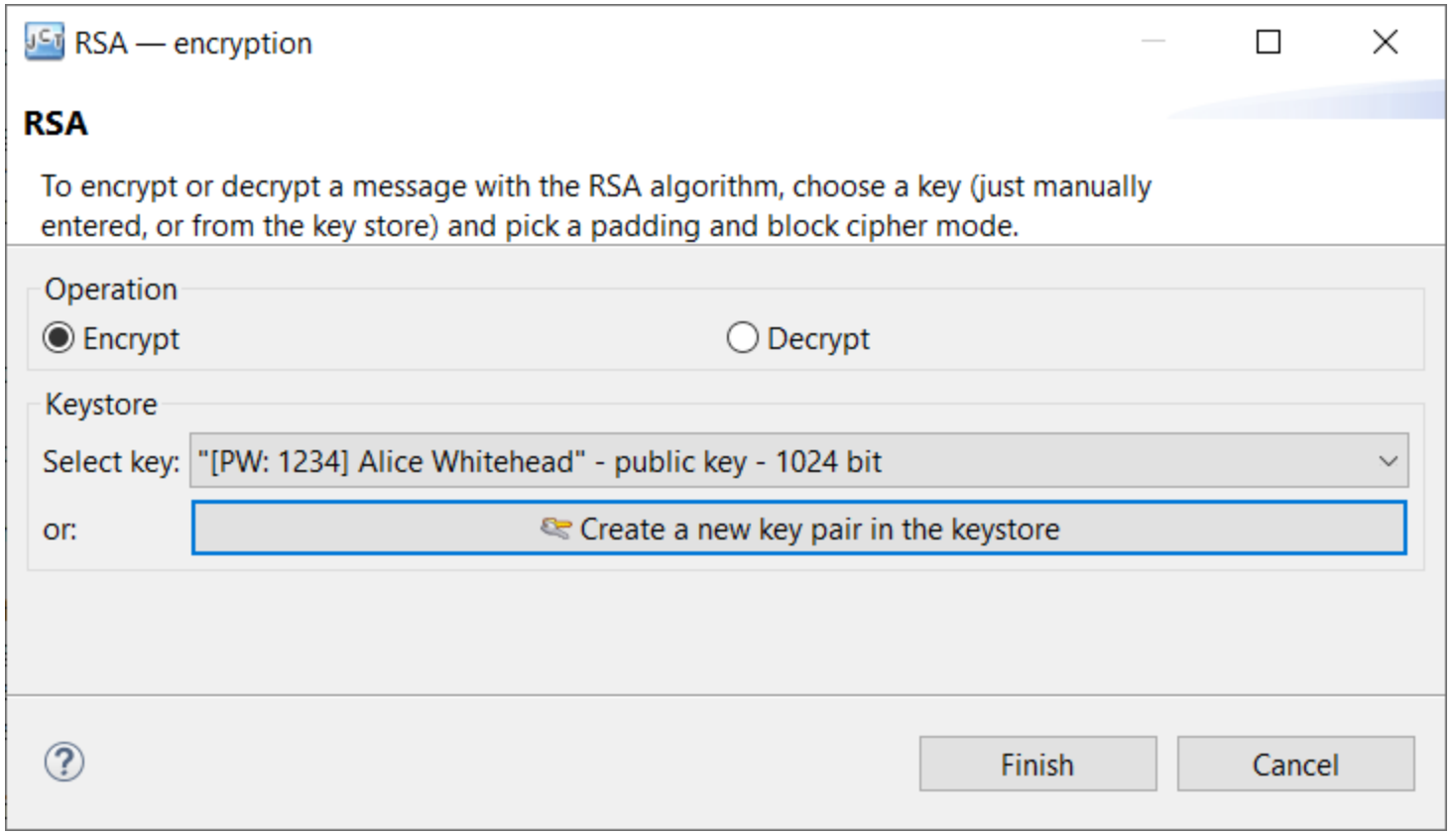
- How does RSA work:
  - Signature – Non- Repudation
  - Encryption/Decryption - Confidentiality



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

JCrypTool DEMO – RSA Encryption

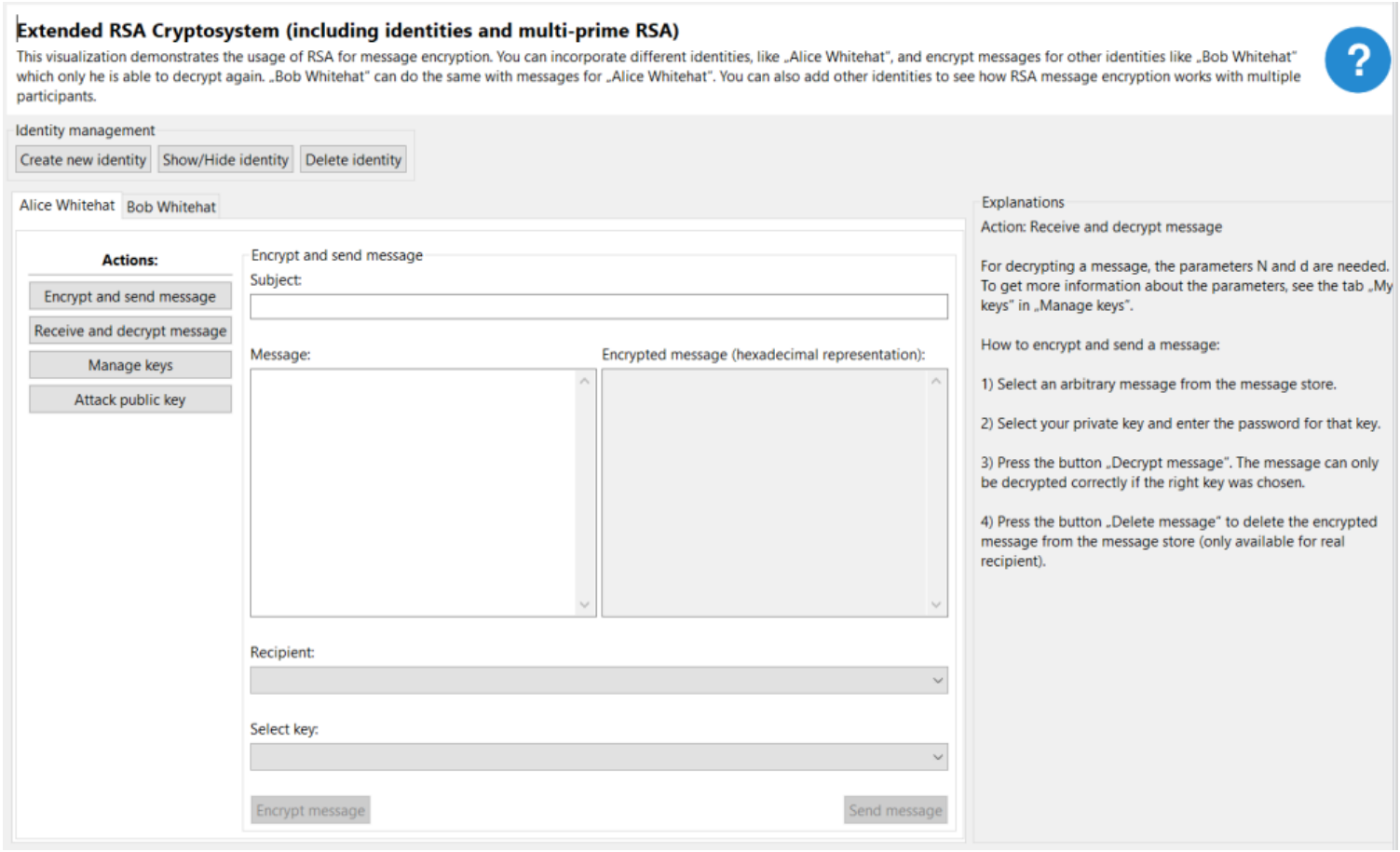
Default Perspective > Algorithms > Asymmetric > RSA



Source: https://www.cryptool.org/en/jct/

JCrypTool DEMO – RSA Encryption

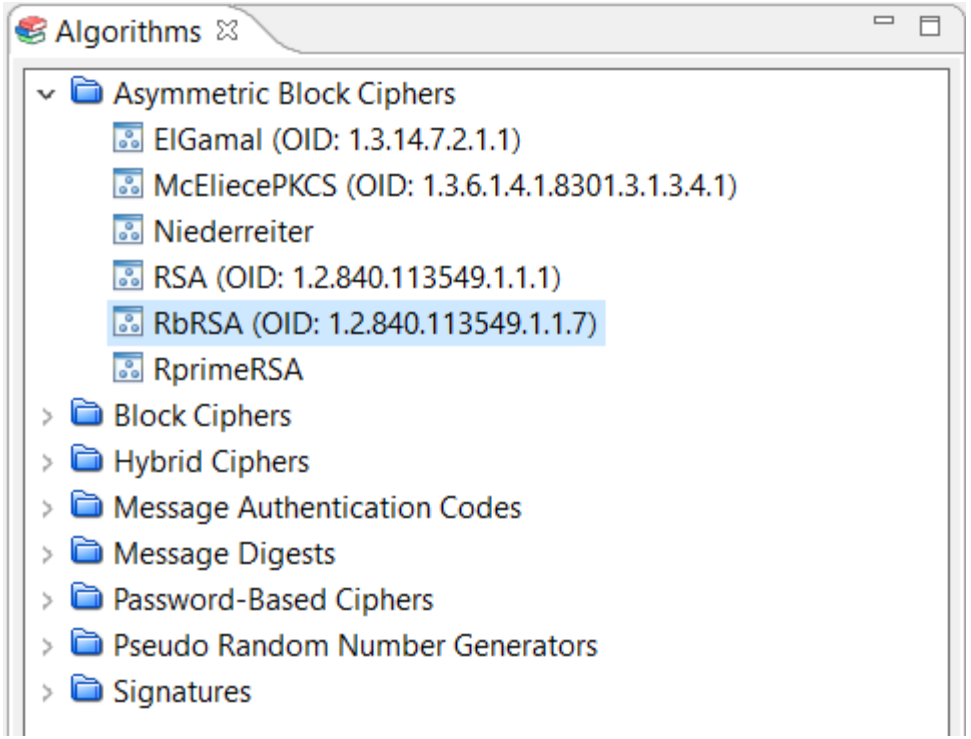
Default Perspective > Visuals > Extended RSA Cryptosystem



Source: https://www.cryptool.org/en/jct/

JCrypTool DEMO – RSA Encryption

Algorithms Perspective > Algorithms tab



Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

- Based on the algebraic structure of elliptic curves over finite fields.
- Allows smaller keys compared to non-EC cryptography (based on plain Galois fields) to provide equivalent security.
- Cryptographic schemes:

Several discrete logarithm-based protocols have been adapted to elliptic curves, replacing the group $(\mathbb{Z}_p)^\times$ with an elliptic curve:

- The Elliptic Curve Diffie–Hellman (ECDH) key agreement scheme is based on the Diffie–Hellman scheme,
- The Elliptic Curve Integrated Encryption Scheme (ECIES), also known as Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme,
- The Elliptic Curve Digital Signature Algorithm (ECDSA) is based on the Digital Signature Algorithm,
- The deformation scheme using Harrison's p-adic Manhattan metric,
- The Edwards-curve Digital Signature Algorithm (EdDSA) is based on Schnorr signature and uses twisted Edwards curves,
- The ECMQV key agreement scheme is based on the MQV key agreement scheme,
- The ECQV implicit certificate scheme.

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

Asymmetric-Key Cryptography - Elliptic-curves

- Standard documents for defining standard curves (named curves):

  - NIST, Recommended Elliptic Curves for Government Use
  - SECG, SEC 2: Recommended Elliptic Curve Domain Parameters
  - ECC Brainpool (RFC 5639), ECC Brainpool Standard Curves and Curve Generation

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

Asymmetric-Key Cryptography - Elliptic-curves

Domain parameters

- Agreement on all the elements defining the elliptic curve ---> the domain parameters of the scheme.
- They are (p, a, b, G, n, h) for prime case, and (m, f, a, b, G, n, h) for binary case.
- The field is defined by p (prime) case and the pair of m and f (binary case).
- Constants a and b used in its defining equation.
- Cyclic subgroup is defined by its generator (base point) G.
- The order of G, that is the smallest positive number n such that nG= {the point at infinity of the curve, and the identity element}. It is normally prime.
- The number $h = \frac{1}{n}|E(\mathbb{F}_p)|$ is an integer.

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

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the real number space

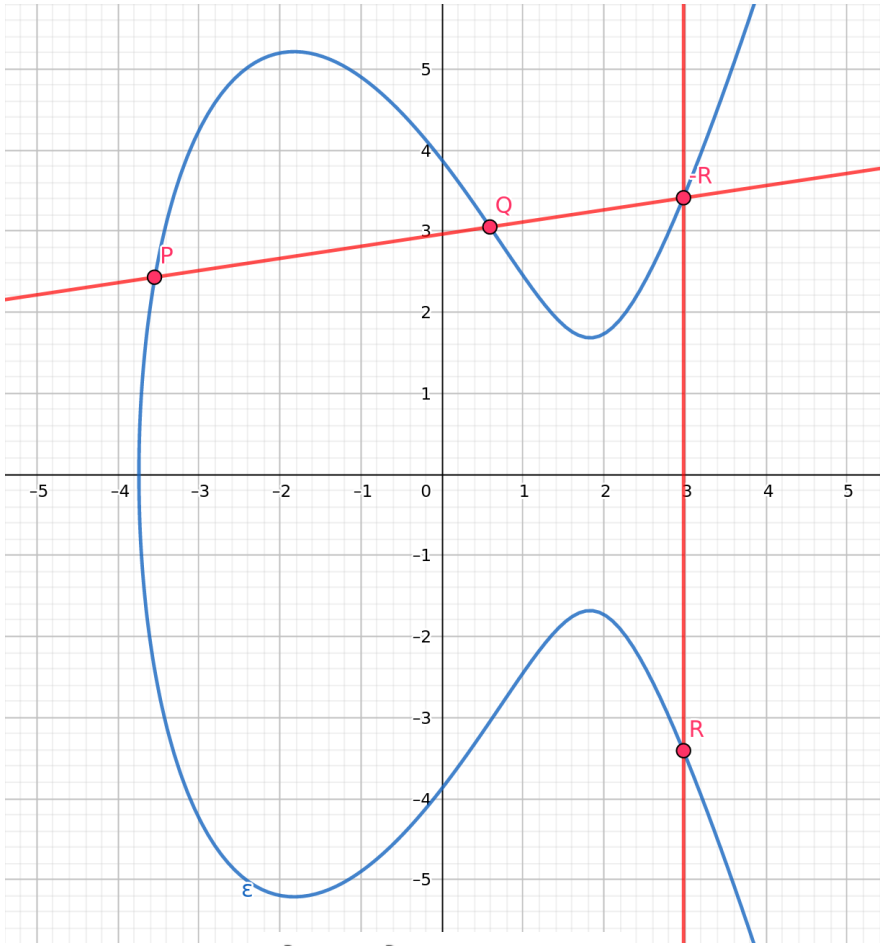
- Equation:

$$y^2 = x^3 + a \times x + b$$

Where a, b, x and y are real numbers.

- Point addition:

1. Select a point P and a point Q.
2. Draw a line through P and Q and continue until you cross the curve for the third time (P and Q being the first two times). This point is $-R$.
3. Point R is the point $-R$ with inverted y value (mirrored on the x axis).



$y^2 = x^3 - 10x + 15$

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the real number space

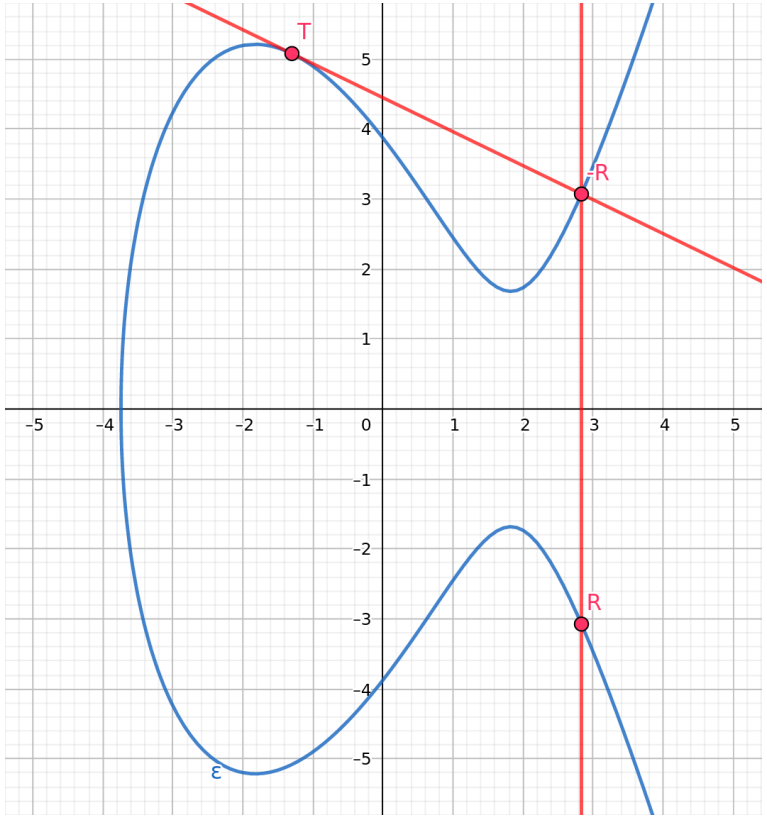
- Three simple formulas:

$$m = \frac{Q_y - Q_y}{Q_x - P_x}$$
$$R_x = m^2 - P_x - Q_x$$
$$R_y = -P_y + s(P_x - R_x)$$

- Point doubling and multiplication:
  - Multiply points by adding them up.
  - Double a point calculate a tangent for that point

$$m = \frac{3x^2 + a}{2y}$$

$$R_x = m^2 - 2x$$
$$R_y = -y + m(x - R_x)$$



$$y^2 = x^3 - 10x + 15$$

Source: https://www.cryptool.org/en/jct/

Cryptography Elements used in Blockchains

Asymmetric-Key Cryptography - Elliptic-curves

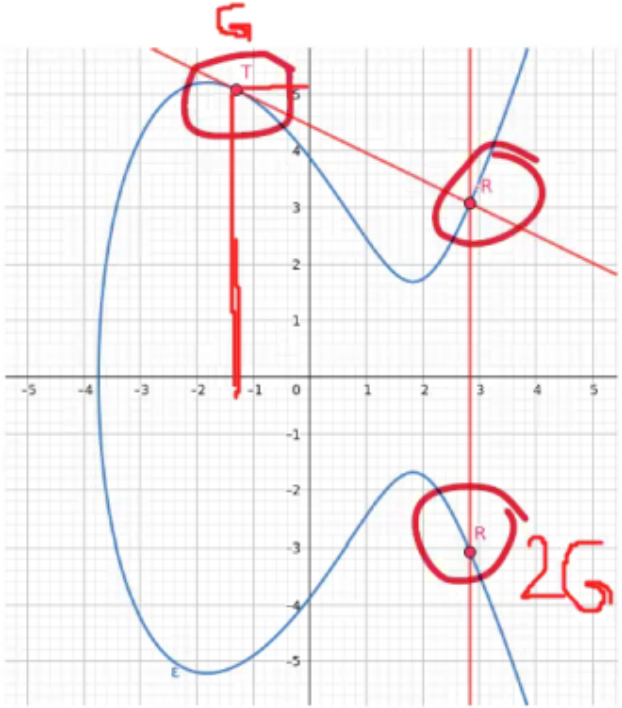
Curves in the real number space

- Three simple formulas:

$$m = \frac{Q_y - Q_y}{Q_x - P_x}$$
$$R_x = m^2 - P_x - Q_x$$
$$R_y = -P_y + s(P_x - R_x)$$

- Point doubling and multiplication:
  - Multiply points by adding them up.
  - Double a point calculate a tangent for that point

$$m = \frac{3x^2 + a}{2y}$$
$$R_x = m^2 - 2x$$
$$R_y = -y + m(x - R_x)$$



$$y^2 = x^3 - 10x + 15$$

Source: https://www.cryptool.org/en/jct/

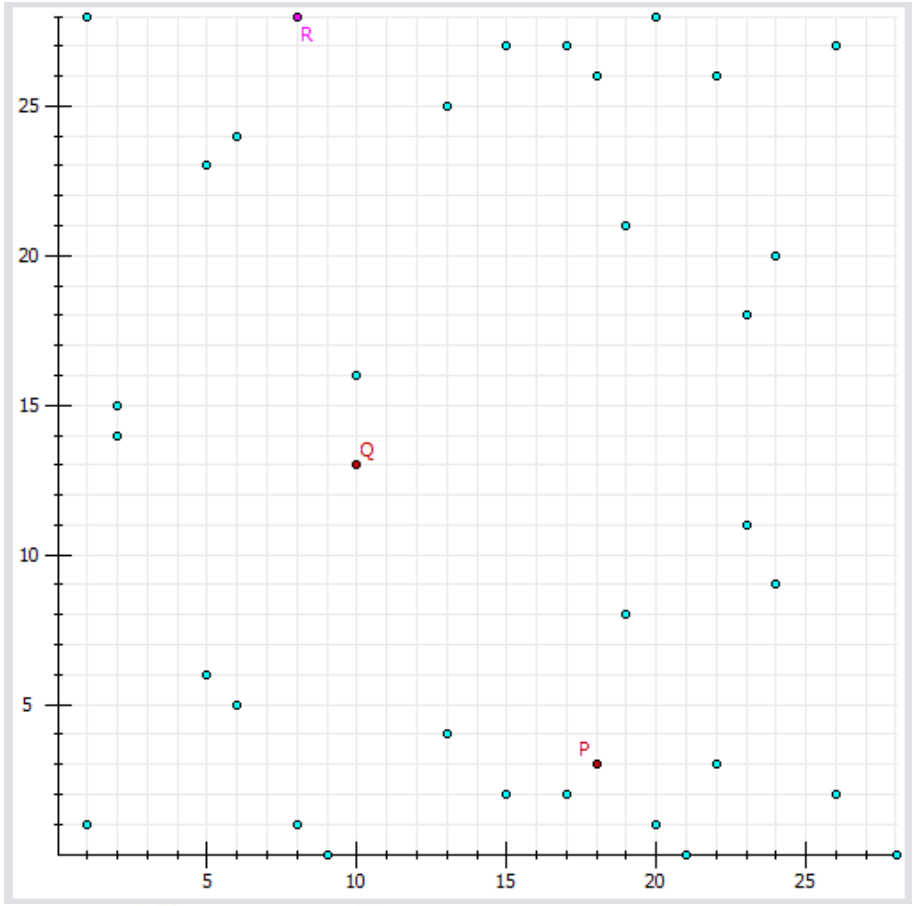
Asymmetric-Key Cryptography - Elliptic-curves

Curves over finite fields Fp

- Equation: $y^2 = x^3 + ax + b \pmod{p}$

where a and b are real numbers parameters of EC.

- There are a maximum of p^2 points on the curve.
- Each point (x, y) is element of the group of points in Fp where p is a prime number. Not all points are valid ones.
- The form of the elliptic curve is not recognizable anymore



An elliptic curve over F_{29}

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves over finite fields Fp

- Point addition:
  - Very similar to curves with real numbers

$$
\begin{aligned}
m &= (Q_y - P_y) \cdot (Q_x - P_x)^{-1} \mod p \\
R_x &= m^2 - P_x - Q_x \mod p \\
R_y &= -P_y + m(P_x - R_x) \mod p
\end{aligned}
$$

- Point doubling and multiplication:
  - Formulas for point doubling

$$
\begin{aligned}
m &= (3x^2 + a) \cdot (2y)^{-1} \mod p \\
R_x &= m^2 - 2x \mod p \\
R_y &= -y + m(x - R_x) \mod p
\end{aligned}
$$

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Equation $y^2 + xy = x^3 + ax^2 + b.$
- Additional parameters:
  - Polynomial / bit length m
  - a and b for the elliptic curve
  - Generator G: an irreducible primary polynomial with a length of $m + 1$. Irreducible polynomials have similar properties as prime numbers. They cannot be divided by other polynomials. G is used to generate points on the curve
  - Reduction polynomial $f(x)$: a polynomial through which is divided for the modulo calculation

Source: https://www.cryptool.org/en/jct/

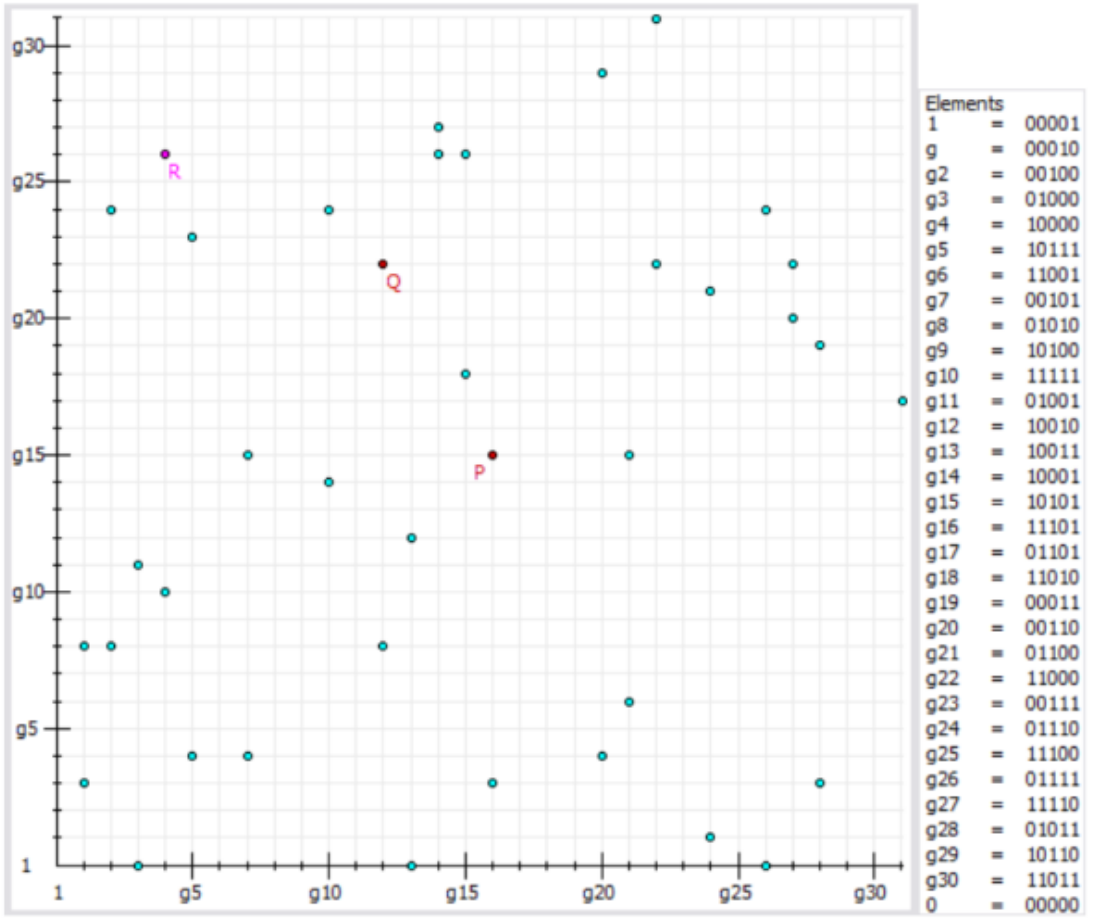
Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Rules and definitions:

  - Given a point $P = (x, y)$, its negative is $-P(x, y) := (x, x + y)$.

  - There exists a neutral element O for which applies: $-O := O$, and for each point P of the elliptic curve applies $P + O := O + P := P$.



An elliptic curve over $\mathbb{F}_{2^5}$ with the parameters:

$y^2 + xy = x^3 + g9 \cdot x^2 + g3$ and the generator $G = 110111$

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

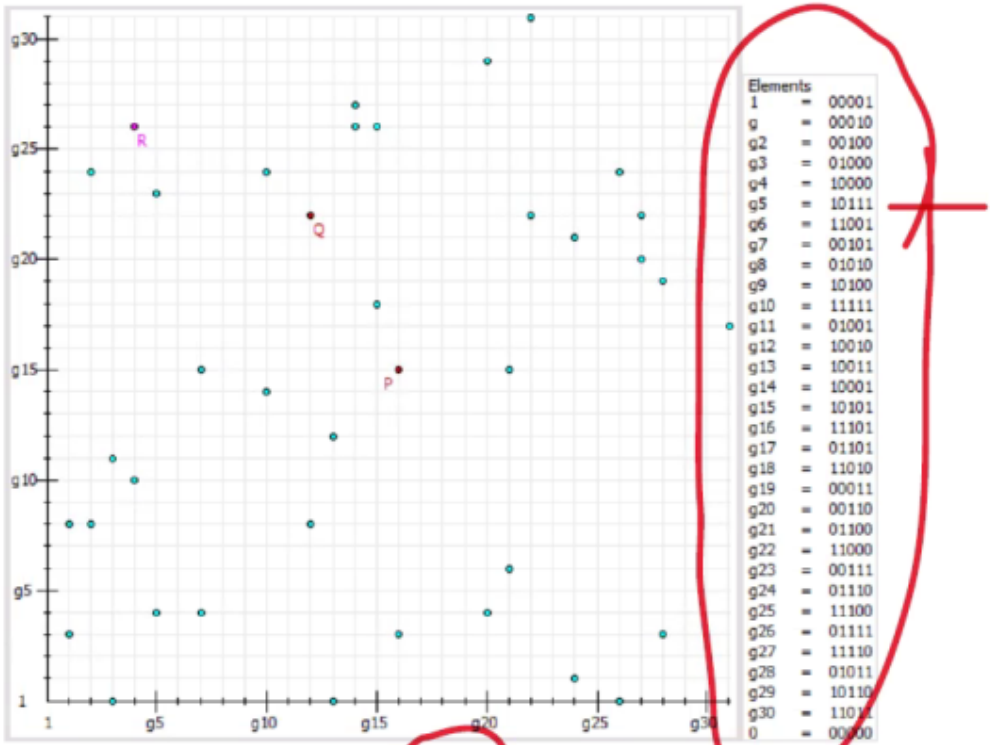
Curves in the field $\mathbb{F}_{2^m}$

- Rules and definitions:

- Given a point $P = (x, y)$, its negative is $-P(x, y) := (x, x + y)$.

- There exists a neutral element O for which applies: $-O := O$, and for each point P of the elliptic curve applies $P + O := O + P := P$.

0000 0010
0000 0100



An elliptic curve over $\mathbb{F}_{2^5}$ with the parameters:

$y^2 + xy = x^3 + g9 \cdot x^2 + g3$ and the generator $G = 110111$

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Calculations
  - Addition - simple bitwise XOR, also called "carryless addition

	Polynomial	Bitstring
p	$x^8+x^4+x^2+1$	100010101
q	x^4+x+1	000010011
$p+q$	x^8+x^2+x	100000110

  - Substraction - exactly the same operation as the addition, a bitwise XOR

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Calculations
  - Multiplication - single term polynomials (monomial), left shift of k bits

Polynomial	Bitstring
$(x^8 + x^4 + x^2 + 1) \cdot x$	$100010101 \cdot 10$
$(x^9 + x^5 + x^3 + x)$	1000101010

Polynomial	Bitstring
$(x^8 + x^4 + x^2 + 1) \cdot x^4$	$100010101 \cdot 10000$
$(x^{12} + x^8 + x^6 + x^4)$	1000101010000

  - Multiplication – for two polynomials results from the distributive law

Polynomial	Bitstring
$(x^8 + x^4 + x^2 + 1) \cdot (x^4 + x + 1) =$	$100010101 \cdot 10011$
$(x^8 + x^4 + x^2 + 1) \cdot x^4 \quad +$	1000101010000
$(x^8 + x^4 + x^2 + 1) \cdot x \quad +$	1000101010
$(x^8 + x^4 + x^2 + 1) \cdot 1$	100010101
	1001001101111
$(x^{12} + x^9 + x^6 + x^5 + x^3 + x^2 + x + 1)$	1001001101111

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Calculations
  - Division – done with remainder for applying normal polynomial division rules

Polynomial	Bitstring
$(x^8 + x^4 + x^2 + 1) / (x^4 + x + 1) = x^4 + x$	$100010101/10011 = 10010$
$-(x^8 + x^5 + x^4)$	$-(10011)$
$x^5 + x^2$	00010010
$-(x^5 + x^2 + x)$	$-(10011)$
$x + 1$ Remainder	000011 Remainder

Source: https://www.cryptool.org/en/jct/

Asymmetric-Key Cryptography - Elliptic-curves

Curves in the field $\mathbb{F}_{2^m}$

- Point addition

$$
\begin{aligned}
m &= (Q_y + P_y) \cdot (Q_x + P_x)^{-1} \mod f(x) \\
R_x &= m^2 + m + P_x + Q_x + a \mod f(x) \\
R_y &= m(P_x + R_y) + R_x + P_y \mod f(x)
\end{aligned}
$$

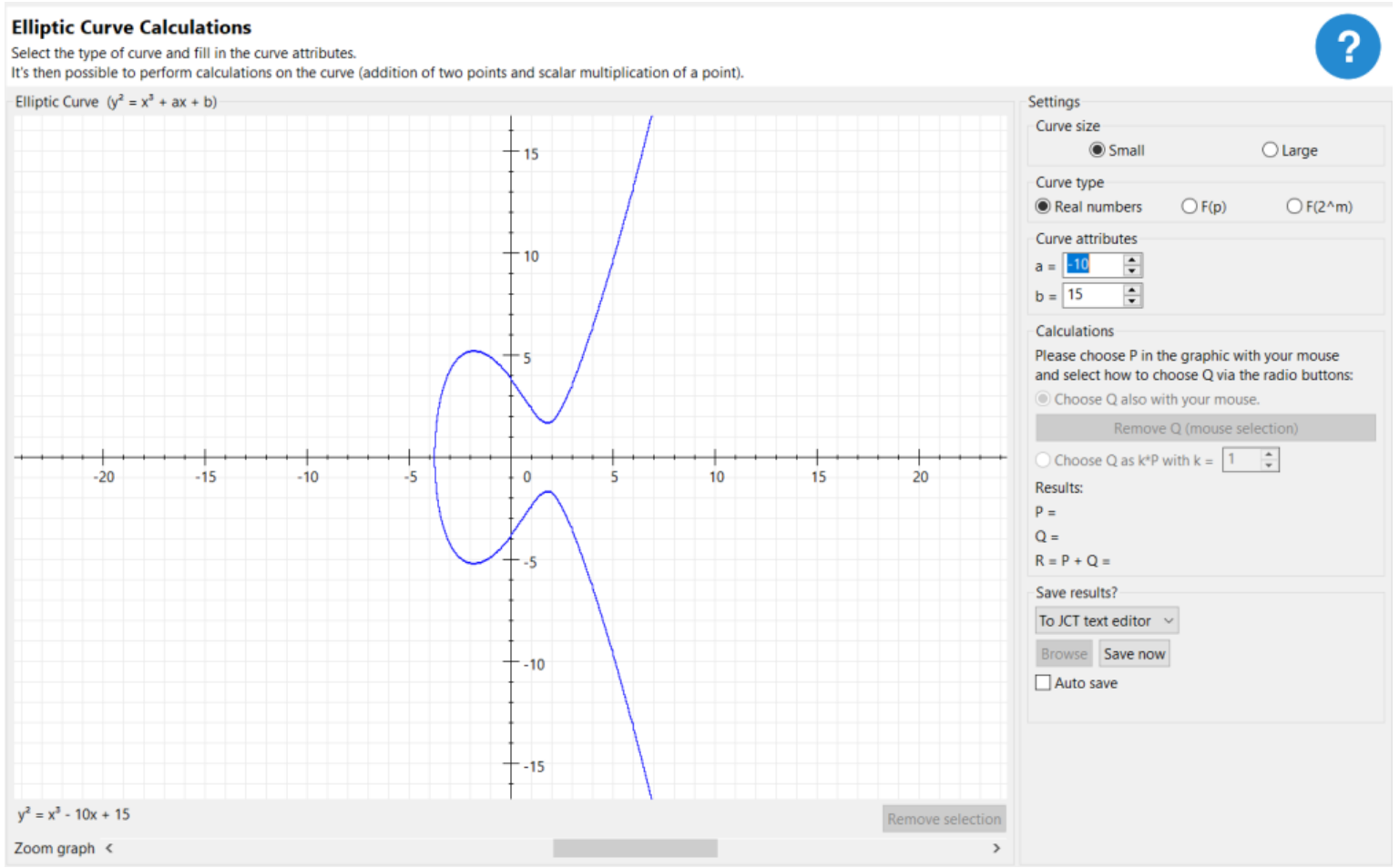
- Point doubling

$$
\begin{aligned}
m &= P_x + P_y \mod f(x) \\
R_x &= m^2 + m + a \mod f(x) \\
R_y &= P_x^2 + R_x \cdot (m + 1) \mod f(x)
\end{aligned}
$$

Source: https://www.cryptool.org/en/jct/

JCrypTool DEMO – ECC

Default Perspective > Visuals > Elliptic Curve Calculations



Source: https://www.cryptool.org/en/jct/

Digital Signature Schemes and Algorithms - DSA

- A cryptographically secure standard for digital signatures (signing messages and signature verification).

- Based on the math of the modular exponentiations and discrete logarithms and the difficulty of the discrete logarithm problem (DLP).

- **DSA signing**:
  - Compute a ***message hash***.
  - Generate a ***random integer*** k and computes the signature.
  - The ***signature*** is a pair of integers {r, s}, where r is computed from k and s is computed using the message hash + the private key exponent + the random number k.
  - Due to randomness, the signature is **non-deterministic**.

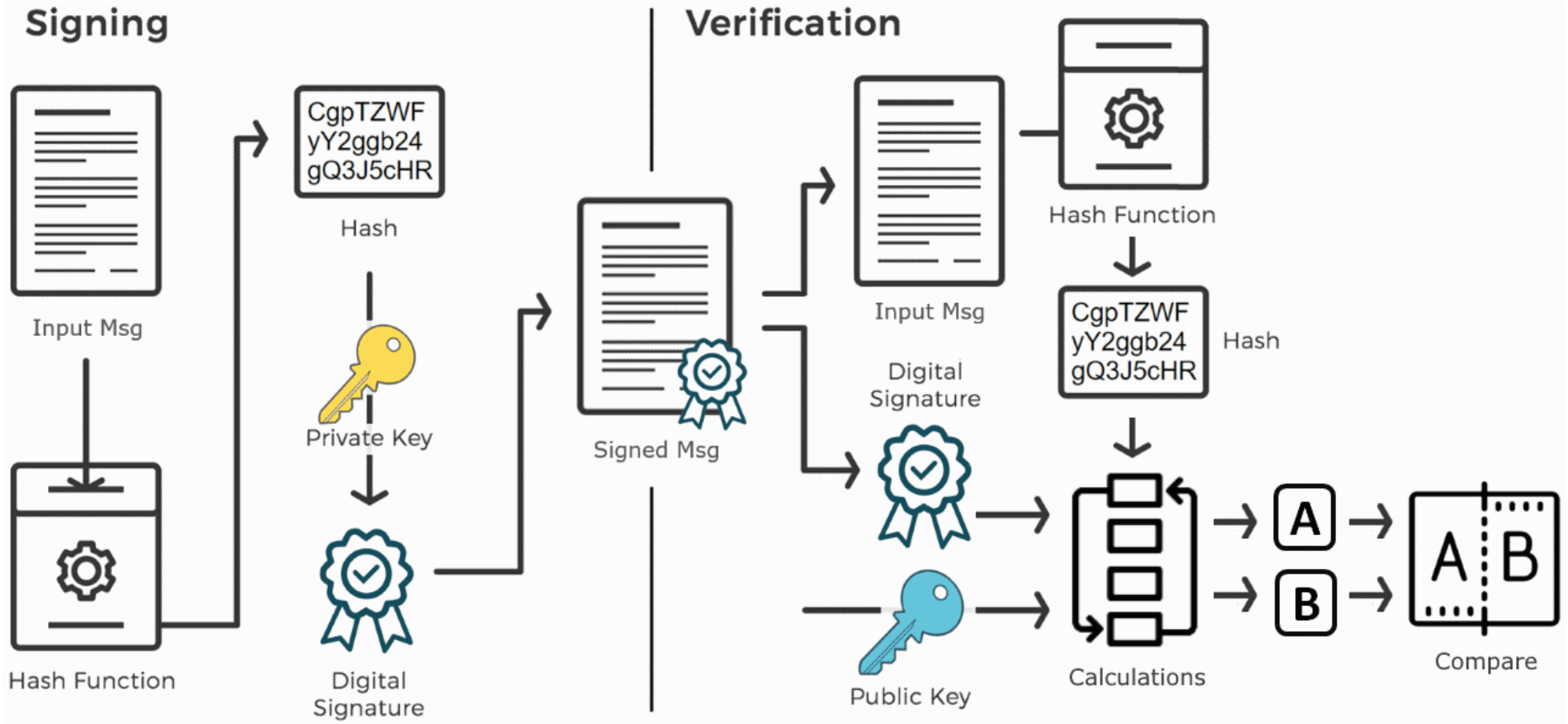
Source: https://wizardforcel.gitbooks.io/practical-cryptography-for-developers-book/content/digital-signatures.html

Digital Signature Schemes and Algorithms - DSA

- **DSA signature verification**:
  - Based on the message **hash** + the **public key** exponent + the **signature** {**r**, **s**}.

Source: https://wizardforcel.gitbooks.io/practical-cryptography-for-developers-book/content/digital-signatures.html

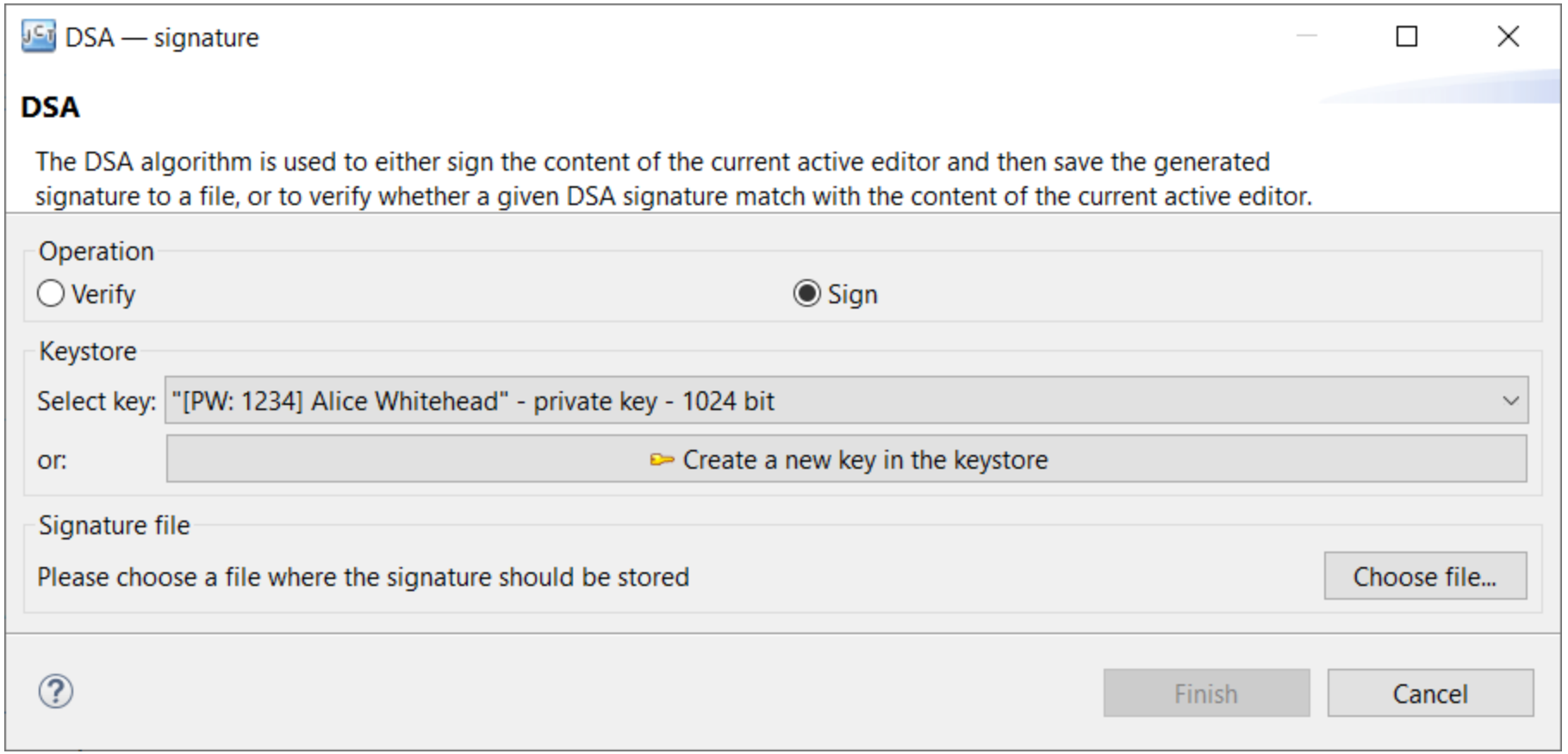
Digital Signature Schemes and Algorithms - DSA



Source: https://wizardforcel.gitbooks.io/practical-cryptography-for-developers-book/content/digital-signatures.html

JCrypTool DEMO – DSA

Default Perspective > Algorithms > Signature > DSA



Source: https://www.cryptool.org/en/jct/

Digital Signature Schemes and Algorithms - ECDSA

- Variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.
- Needed EC parameters:

Parameter	
CURVE	the elliptic curve field and equation used
G	elliptic curve base point, a point on the curve that generates a subgroup of large prime order n
n	integer order of G, means that $n \times G = O$, where O is the identity element.
d_A	the private key (randomly selected)
Q_A	the public key (calculated by elliptic curve)
m	the message to send

Source: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

Digital Signature Schemes and Algorithms - ECDSA

- Sign a message m by the sender:

1. Calculate $e = \mathrm{HASH}(m)$. (Here HASH is a cryptographic hash function, such as SHA-2, with the output converted to an integer.)
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n. (Note that z can be *greater* than n but not *longer*.[1])
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s). (And $(r, -s \mod n)$ is also a valid signature.)

Source: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

Digital Signature Schemes and Algorithms - ECDSA

- Verify signature by the receiver:
  - Check the public key is a valid elliptic curve point

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n \times Q_A = O$

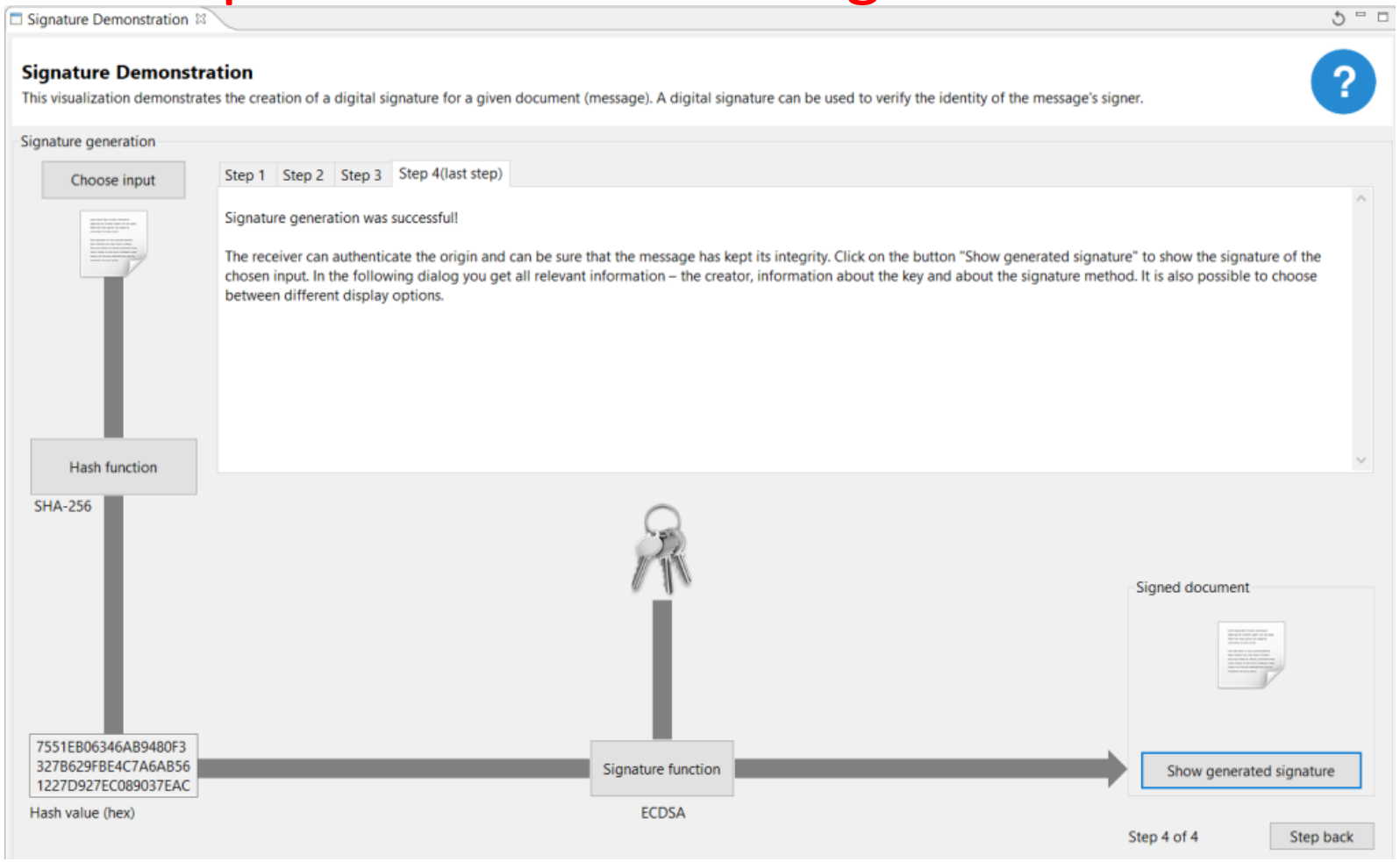
  - Check the DSA signature

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $u_1 = zs^{-1} \mod n$ and $u_2 = rs^{-1} \mod n$.
5. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. If $(x_1, y_1) = O$ then the signature is invalid.
6. The signature is valid if $r \equiv x_1 \pmod n$, invalid otherwise.

Source: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

JCrypTool DEMO – DSA

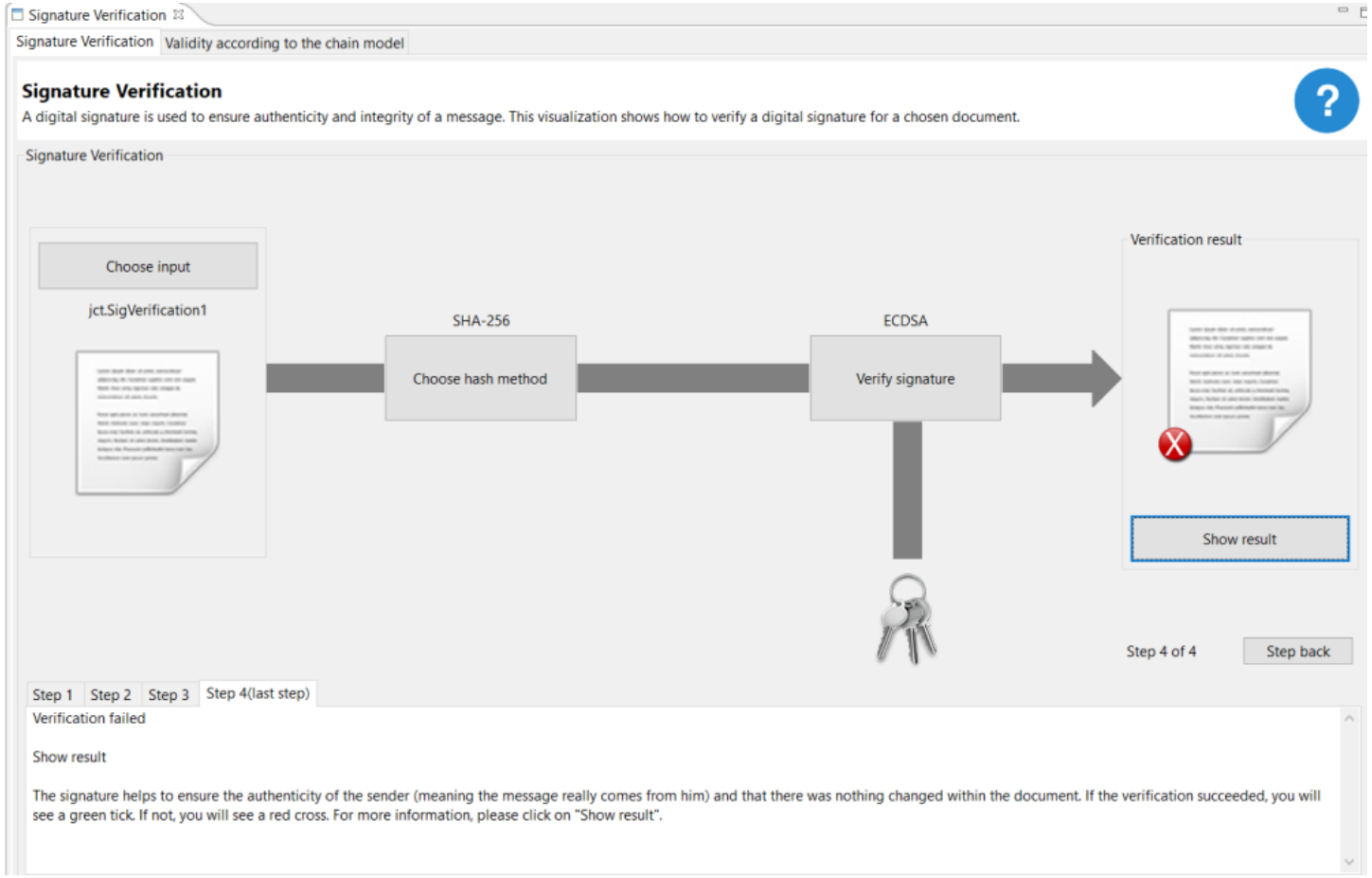
Default Perspective > Visuals > Signature Demonstration



Source: https://www.cryptool.org/en/jct/

JCrypTool DEMO – DSA

Default Perspective > Visuals > Signature Verification



Source: https://www.cryptool.org/en/jct/

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: $g=h^{(p-1)/q} \bmod p$, where h is a integer relatively prime with p, h in (0..p) so that: $h^{(p-1)/q} \bmod p>1$.
  - 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: **PRIV**, an integer in (0..q);
  - public key: **PUB**, an integer, calculated so that: $PUB=g^{PRIV} \bmod p$
- 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:
  - $r = (g^k \bmod p) \bmod q$ and
  - $s = ((k^{-1})*(H(M)+PRIV_A* r)) \bmod q$

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 or s1 = $s^{-1} \bmod q$ (s must be reversible)***.

The signature is valid if is verified the equation **r=r'**, where r' is:

$r'=(g^{H(M)*s1} *(PUB_A)^{r*s1} \bmod p) \bmod q$

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: $g=3^{78} \bmod 7879 = 170$.

Given the secret key of the user A, $PRIV_A$=75. It results the pair public key: $PUB_A = g^{PRIV_A} \bmod 7879 = 4567$.
It is presumed that sender A, who has the secret key $PRIV_A$, 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: $k^{-1} \bmod 101 = 99$ <= verification: 99*50 mod 101 = 1.

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

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

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

It is calculated:
- H(M) * s1 mod q = 1234*25 mod 101 = 45;
- r*s1 mod q = 94*25 mod 101 = 27

and the verification:
$r'=(170^{45}*4567^{27} \bmod 7879) \bmod 101 = 2518 \bmod 101 = 94$.

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/

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 -> ECDSA[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*** = hash(***msg***)
2. Generate securely a **random** number ***k*** in the range [1..***n***-1]
   - In case of **deterministic-ECDSA**, the value ***k*** is HMAC-derived from ***h*** + ***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*** = $k^{-1} * (h + r * privKey) \pmod n$
   - 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.

Source: https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages

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*** = 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.

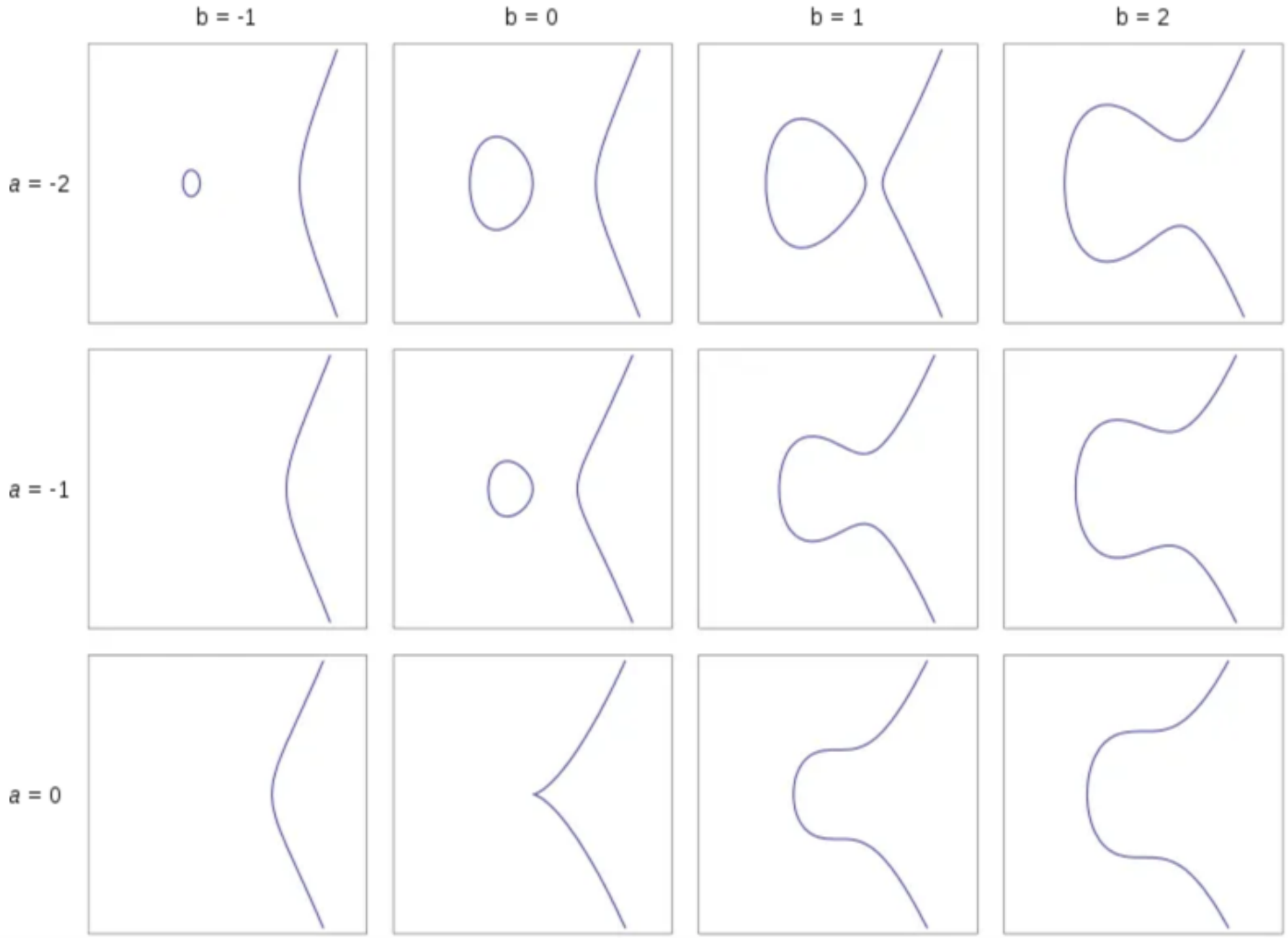
Source: https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages

Cryptography Elements used in Blockchains – ECDSA over Finite Field

Elliptic curves have a general form of:

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

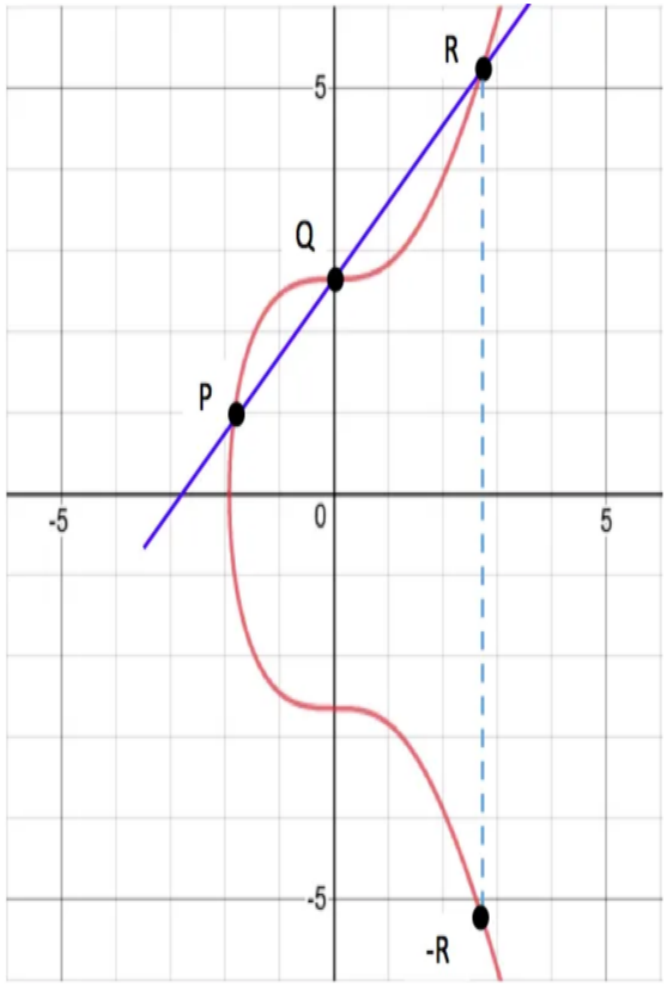
and this is what they can look like:



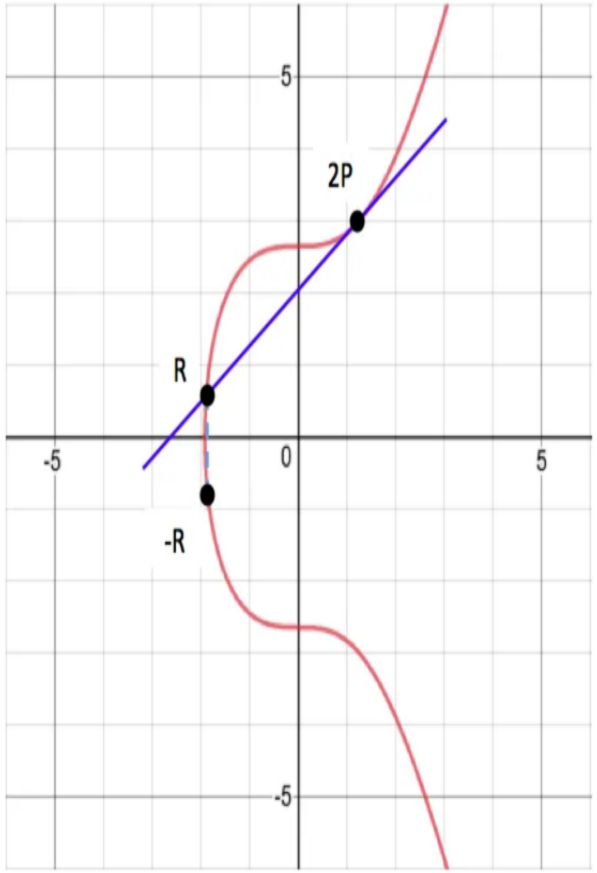
https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/

Cryptography Elements used in Blockchains – ECDSA over Finite Field

However, this just gets us P + Q + R = 0. We want P + Q = '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 as the same characteristic as if P = Q, or 2P = -R. This is also known as 'point doubling':



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:

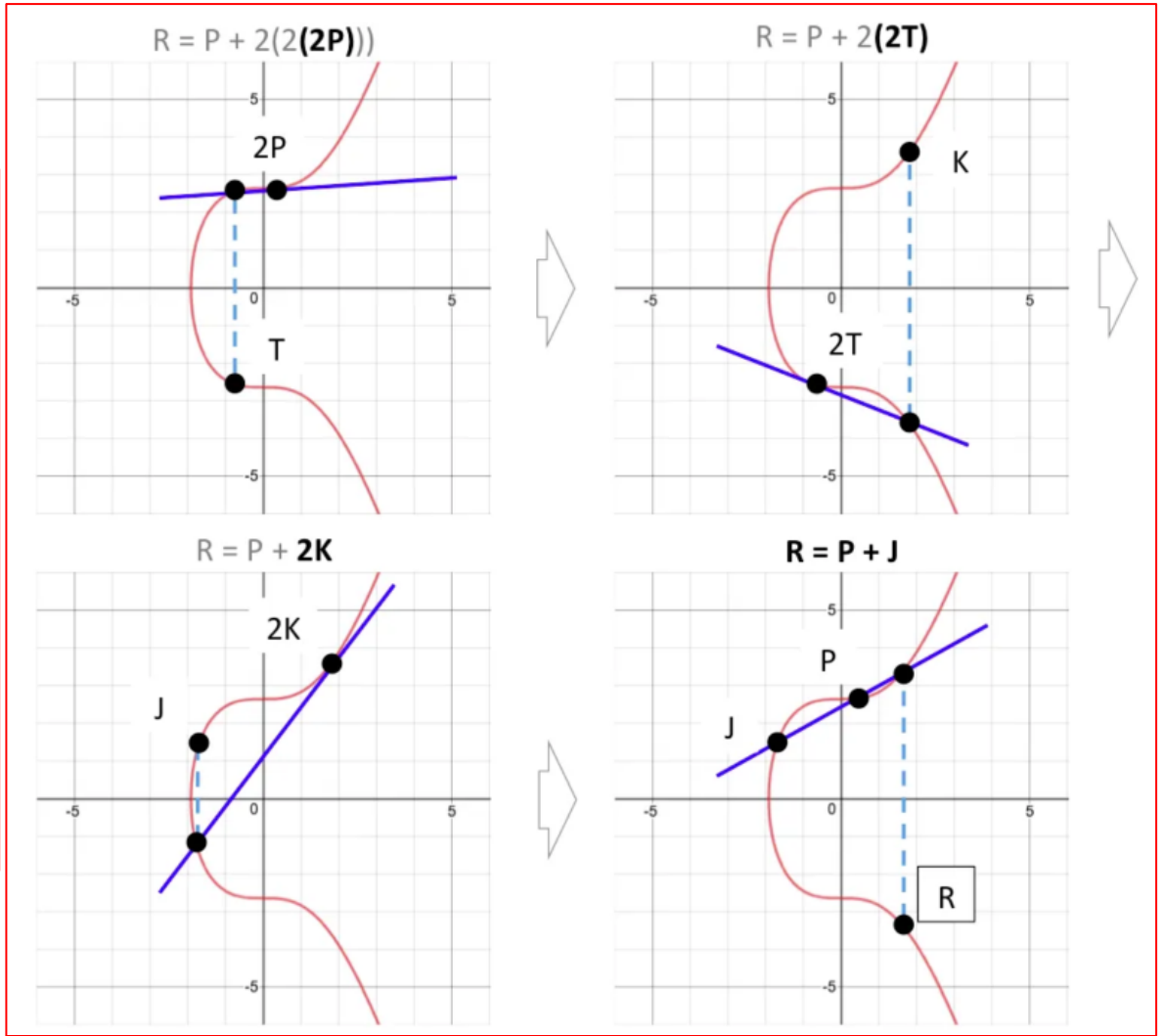
*Public = Private * Base → R = 9P*

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.

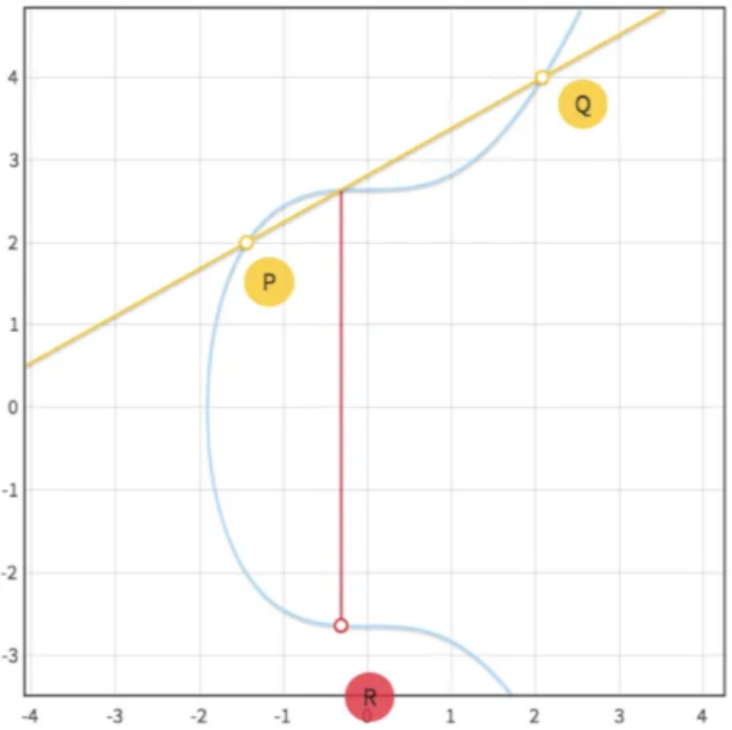


Cryptography Elements used in Blockchains – ECDSA over Finite Field

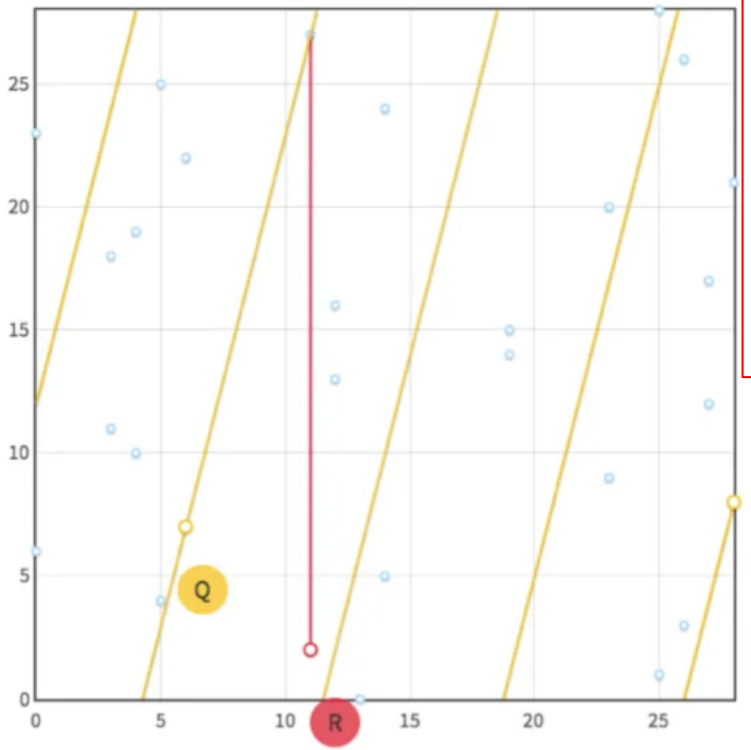
A Short Look at Finite Fields

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 the slope m assumes the form:

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

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

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

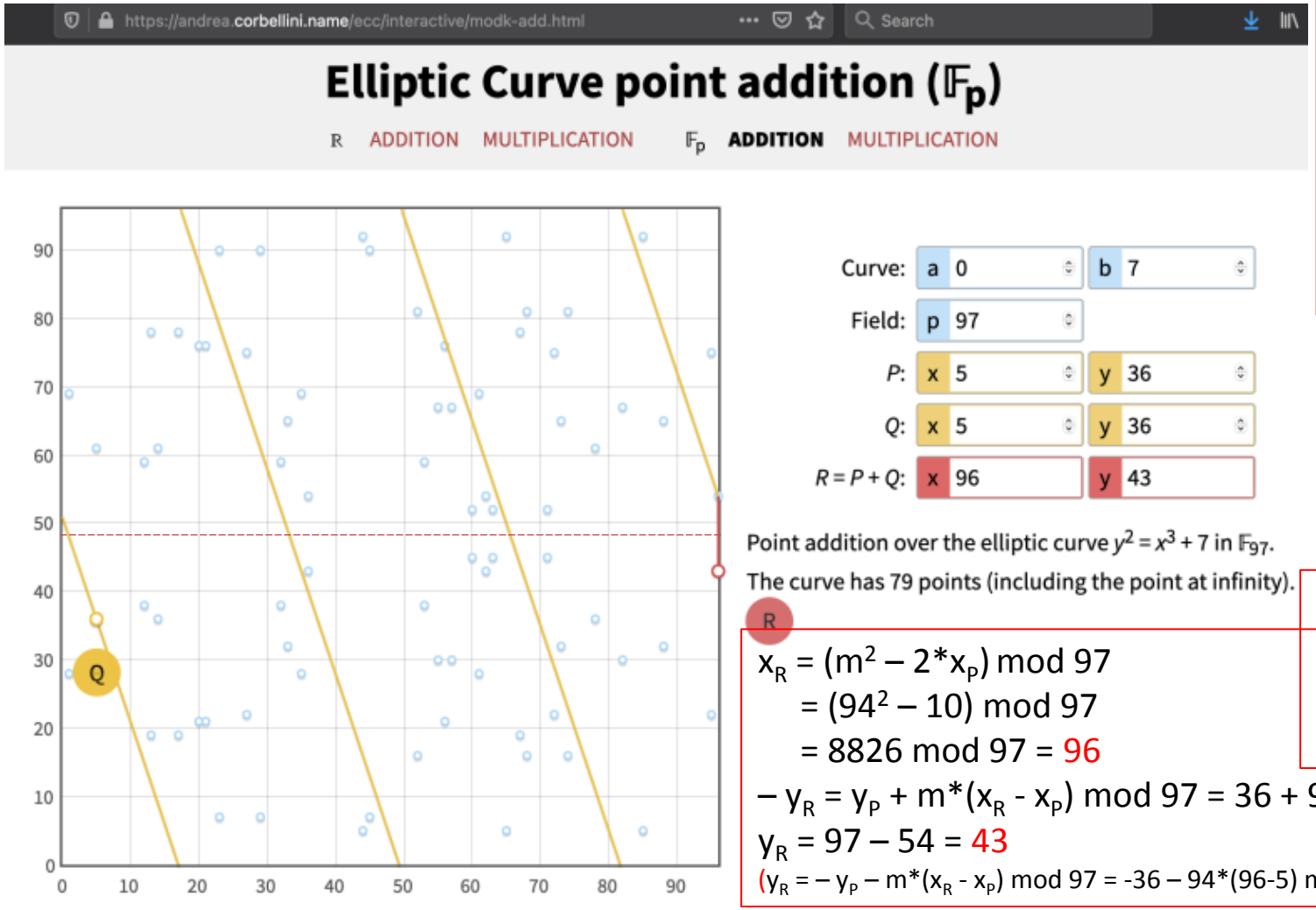
For P=(5|36), a=0, b=7, calculate 2*P = P + P

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 => 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



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 the slope m assumes the form:

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

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

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

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

For $(72)^{-1}$ mod 97 - Modular Multiplicative Inverse:
https://planetcalc.com/3311/
For modulo:
https://miniwebtool.com/modulo-calculator/

$x_R = (m^2 - 2*x_P) \bmod 97$
$= (94^2 - 10) \bmod 97$
$= 8826 \bmod 97 = 96$

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

$y_R = 97 - 54 = 43$

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

Cryptography Elements used in Blockchains – ECDSA over Finite Field

The Process of Signing Data

The basic process of singing data goes like this:

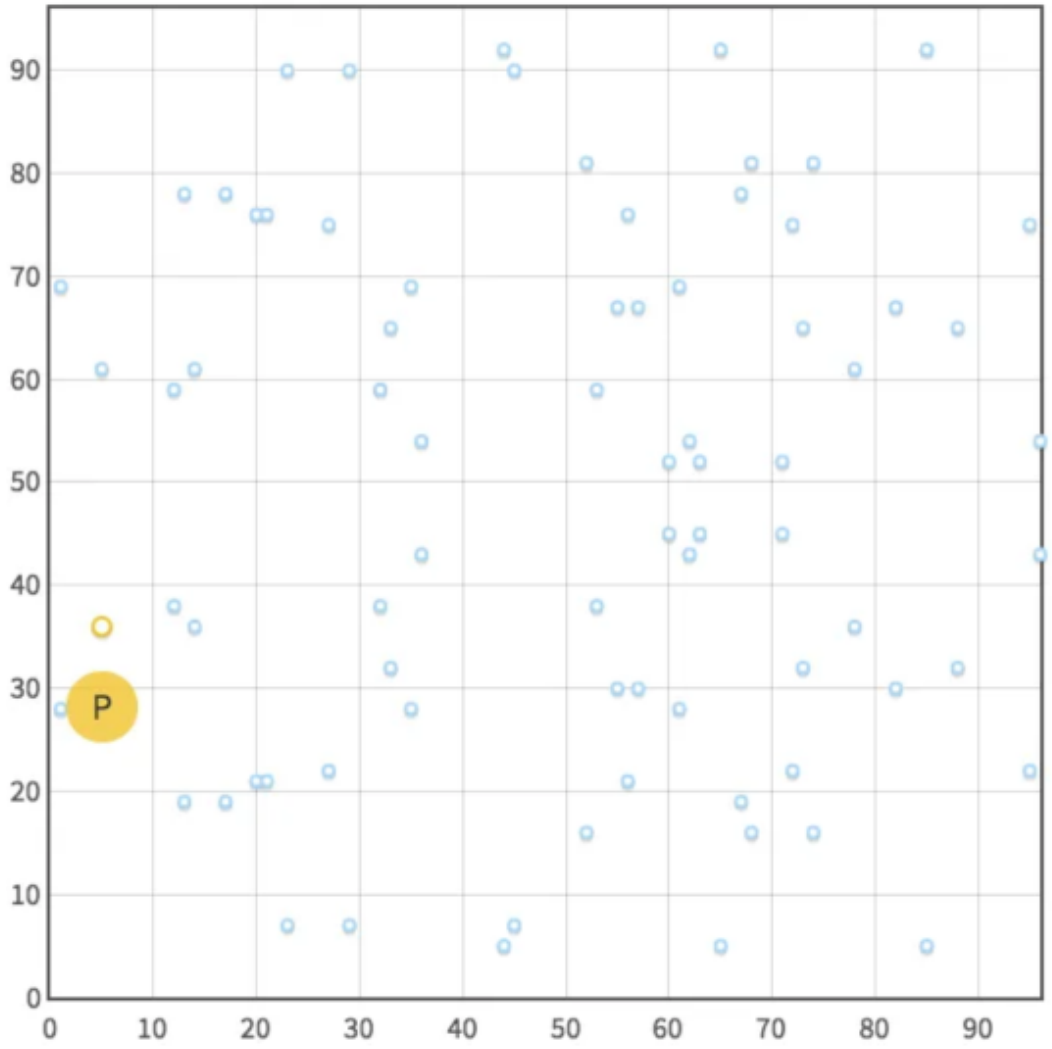
- Choose a private key
- Create a public key
- Create the data you want to sign
- 'Sign' the data by combining the data with your private key to find a point on an elliptic curve

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

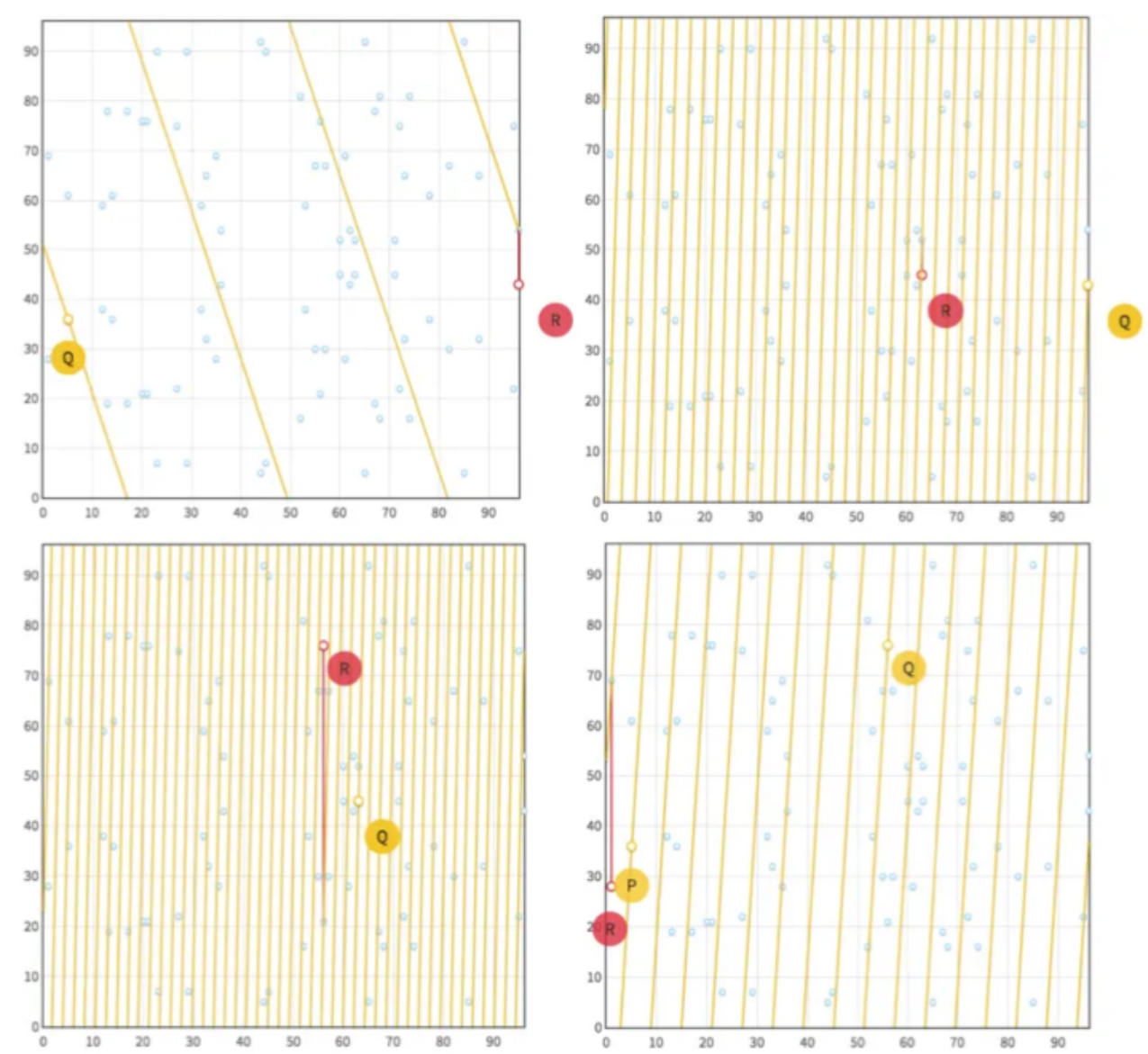
Finding the Public Key

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:

*Public = 9 * Base →*

*Public = Base + 2(2(2 * Base)))*

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



So, doing the math:

*Public = (5,36) + 2(2(2 * (5,36)))) →*

*Public = (5,36) + 2(2 * (96,43)) →*

*Public = (5,36) + 2 * (63,45) →*

Public = (5,36) + (56,76) →

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:

$$(x,y) = k * Base \rightarrow$$
$$(x,y) = 4 * (5,36) \rightarrow$$
$$(x,y) = (63,45)$$

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:

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

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

*Number's Inverse * Number mod n = 1*

So the inverse of 4 would be 20, as 4 * 20 = 80 and 80 mod 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.

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 *(x2 , y2):*

$$(x_2 , y_2) = u_1 * Base + u_2 * Public$$

u_1 and u_2 are found with these:

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

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

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

$$(x_2 , y_2) = u_1 * Base + u_2 * (Private * Base)$$

and we can expand u_1 and u_2….

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

collecting like terms….

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

and replacing s with its definition above.

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

The inverses cancel out and we are left this:

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

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

$$(x,y) = k * Base \bmod n \rightarrow \quad r = x$$

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 *(x2 , y2):*

$$(x_2 , y_2) = u_1 * Base + u_2 * Public$$

u_1 and u_2 are found with these:

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

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

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

$$(x_2 , y_2) = u_1 * Base + u_2 * (Public)$$

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

Proceeding with calculating the example:

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

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

$$\mathbf{u_1 = 40}$$

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

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

$$\mathbf{u_2 = 75}$$

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

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

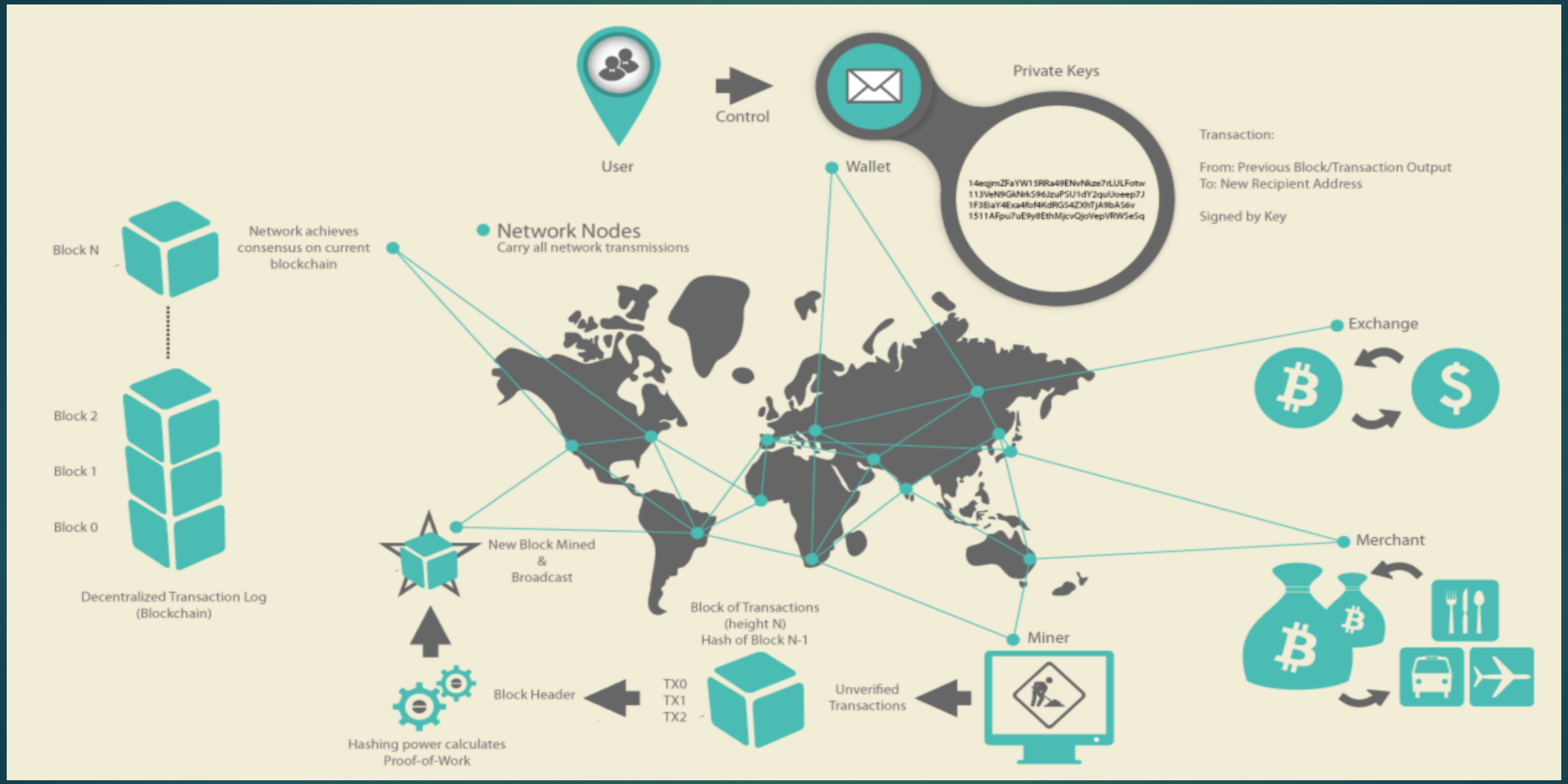
$$(x_2 , y_2) = (82,30) + (67,78) \rightarrow$$

$$\mathbf{(x_2 , y_2) = (63,45)}$$

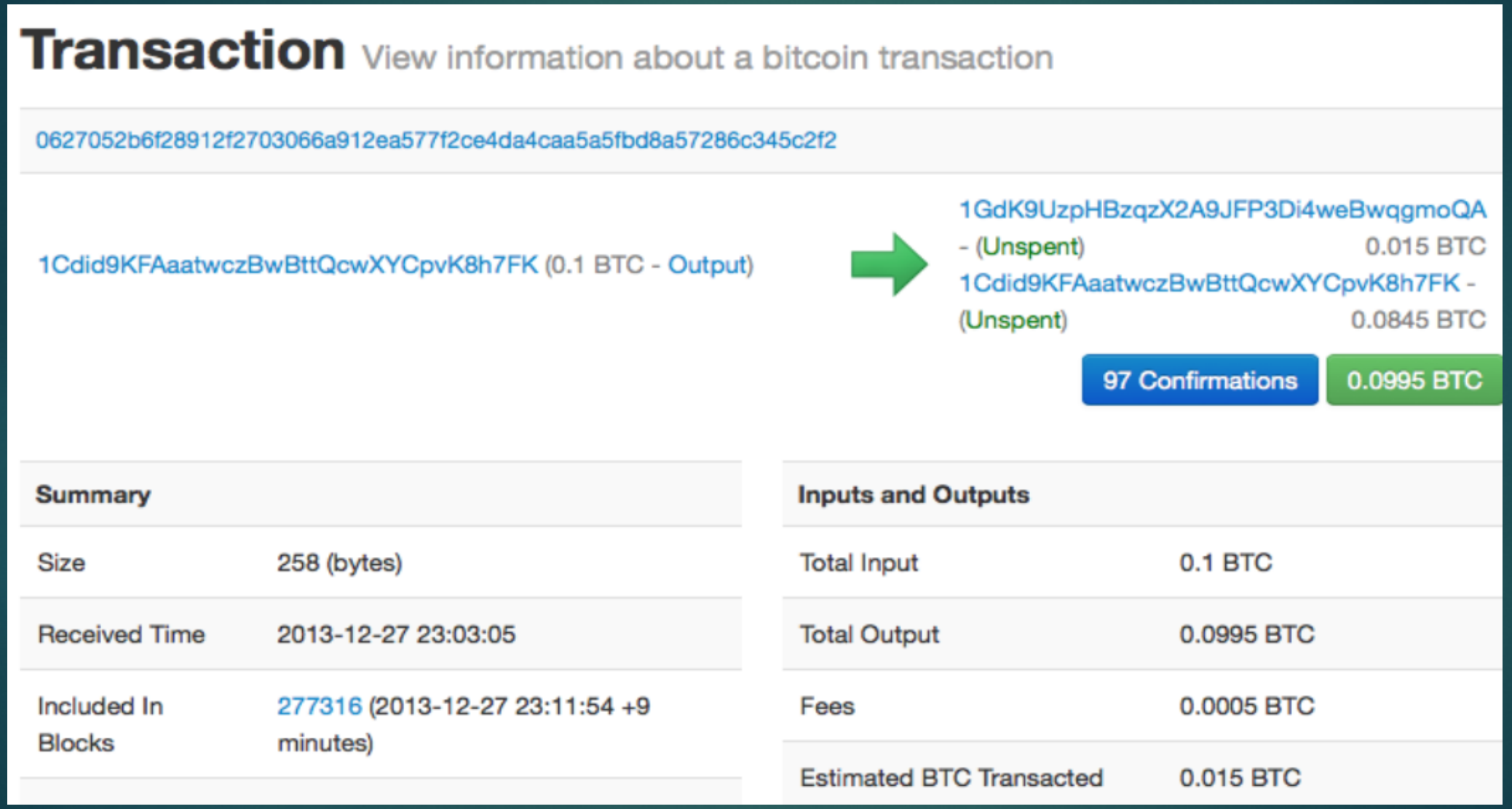
x_2 = 63, and r = 63.

Consider our signature verified.

02. Bitcoin | Ethereum Blockchain Technology



02. Bitcoin | Ethereum Blockchain Technology



Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

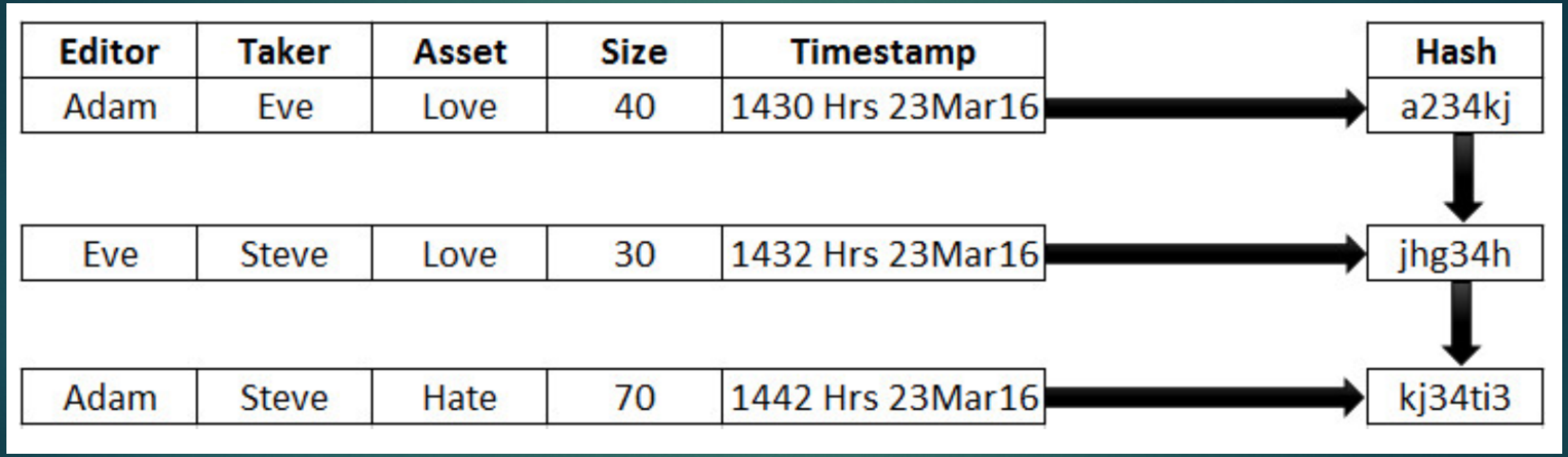
97 Confirmations | 0.0995 BTC

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

02. Bitcoin / Ethereum Blockchain Technology

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

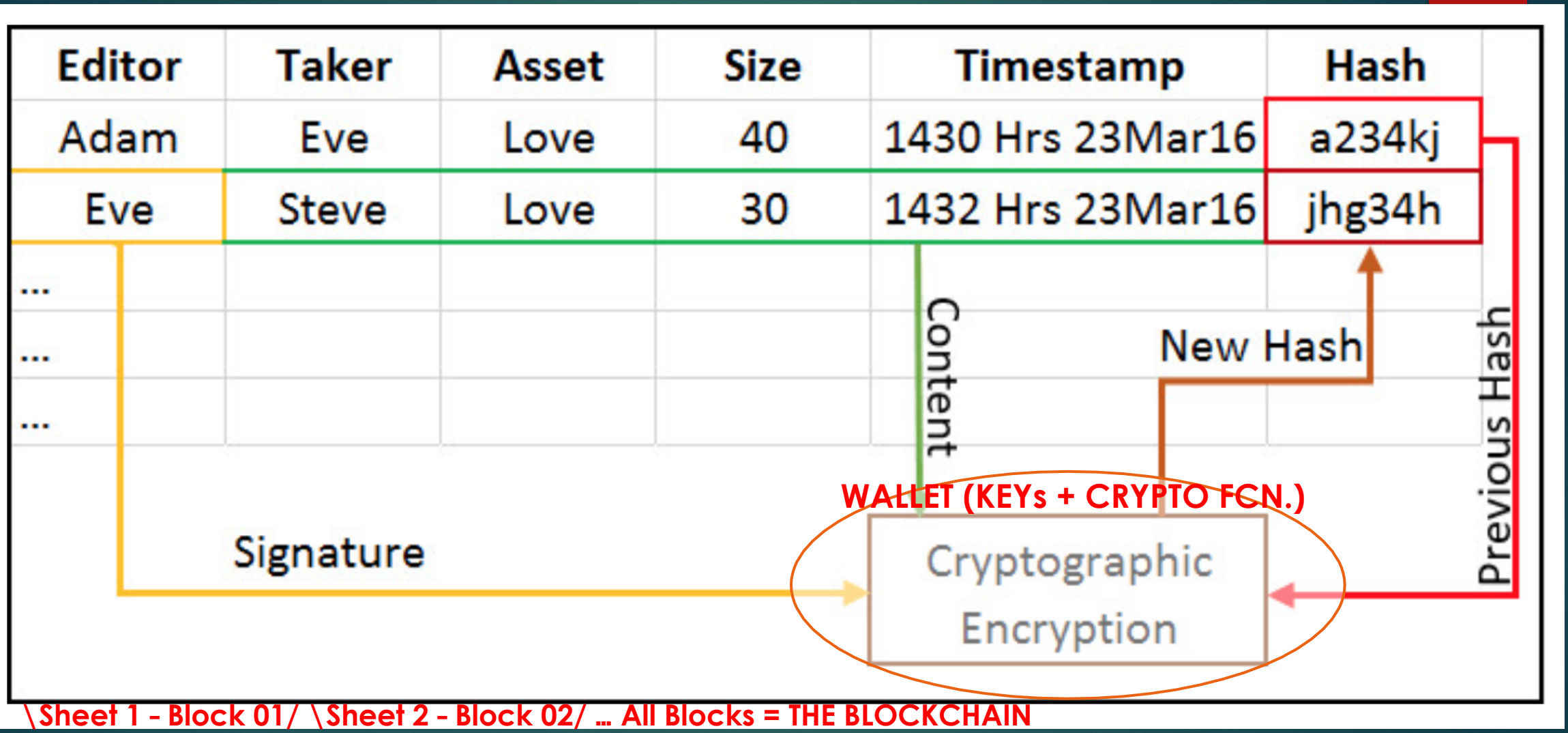


Editor	Taker	Asset	Size	Timestamp	Hash
Adam	Eve	Love	40	1430 Hrs 23Mar16	a234kj
Eve	Steve	Love	30	1432 Hrs 23Mar16	jhg34h
Adam	Steve	Hate	70	1442 Hrs 23Mar16	kj34ti3

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

02. Bitcoin / Ethereum Blockchain Technology

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



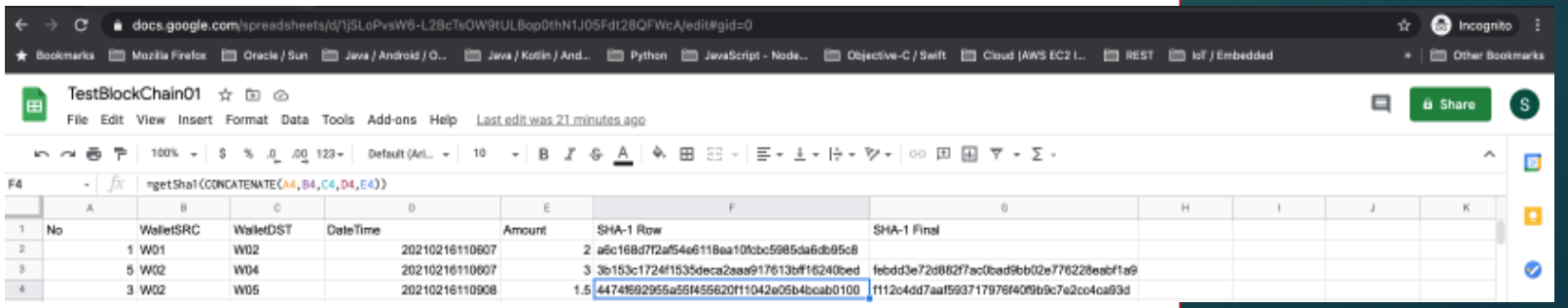
\Sheet 1 - Block 01/ \Sheet 2 - Block 02/ ... All Blocks = THE BLOCKCHAIN

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

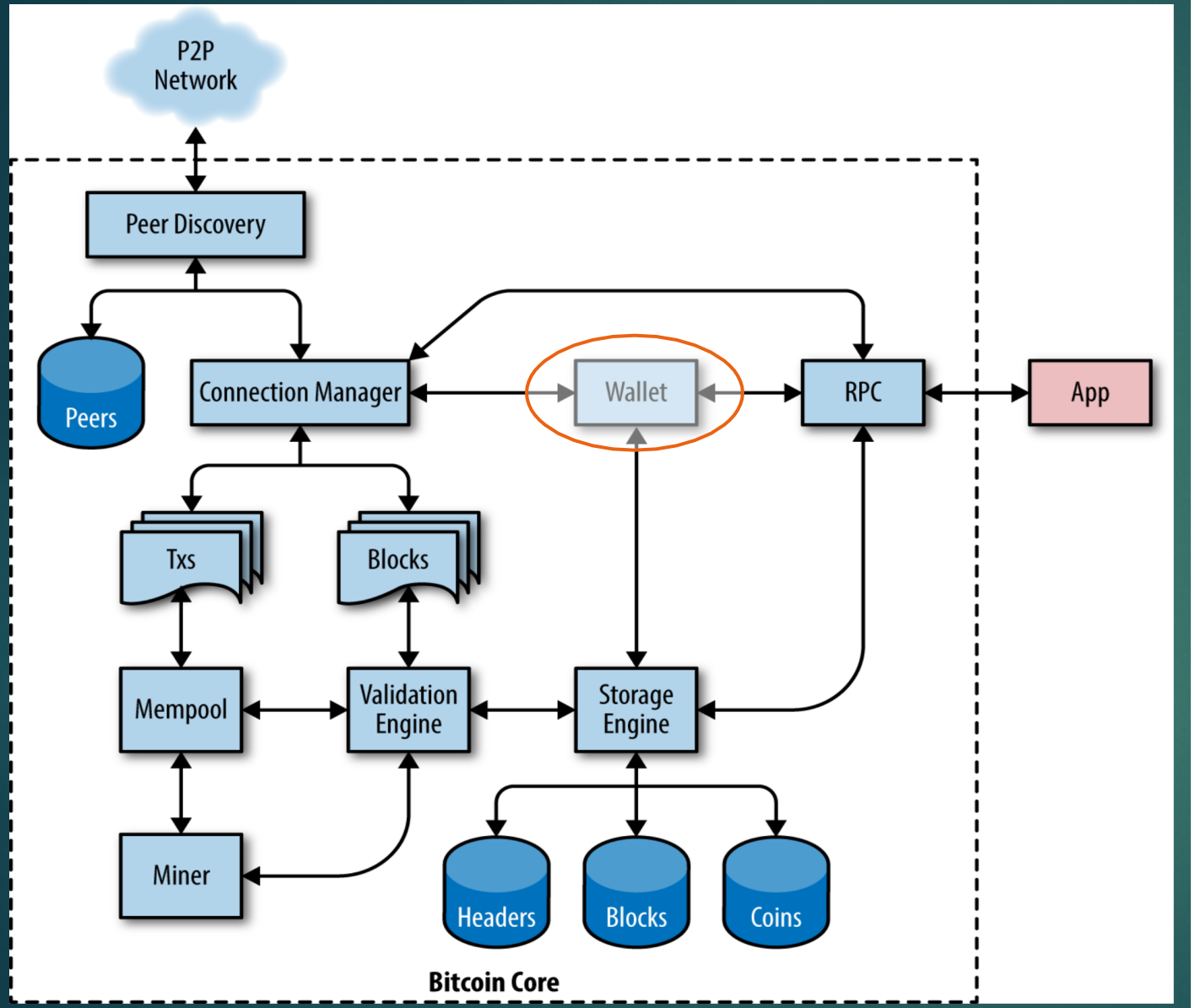
02. Google Sheet DEMO

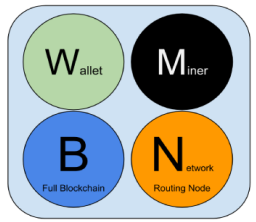

```
// https://docs.google.com/spreadsheets/
// Create Blank Sheet and Tools->Script Editor
//=getSha1(CONCATENATE(A4,B4,C4,D4,E4))

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



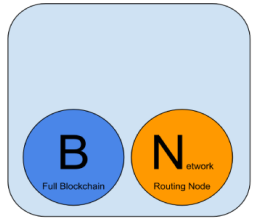
02 Bitcoin | Ethereum Blockchain Technology





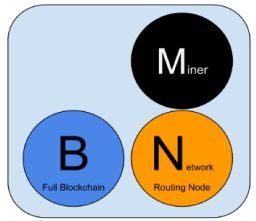
Reference Client (Bitcoin Core)

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



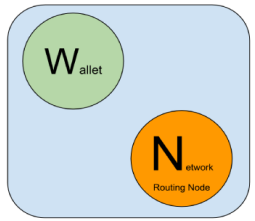
Full Block Chain Node

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



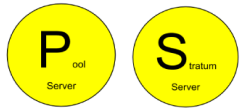
Solo Miner

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.

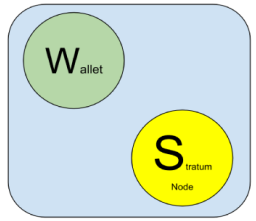


Pool Protocol Servers

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.

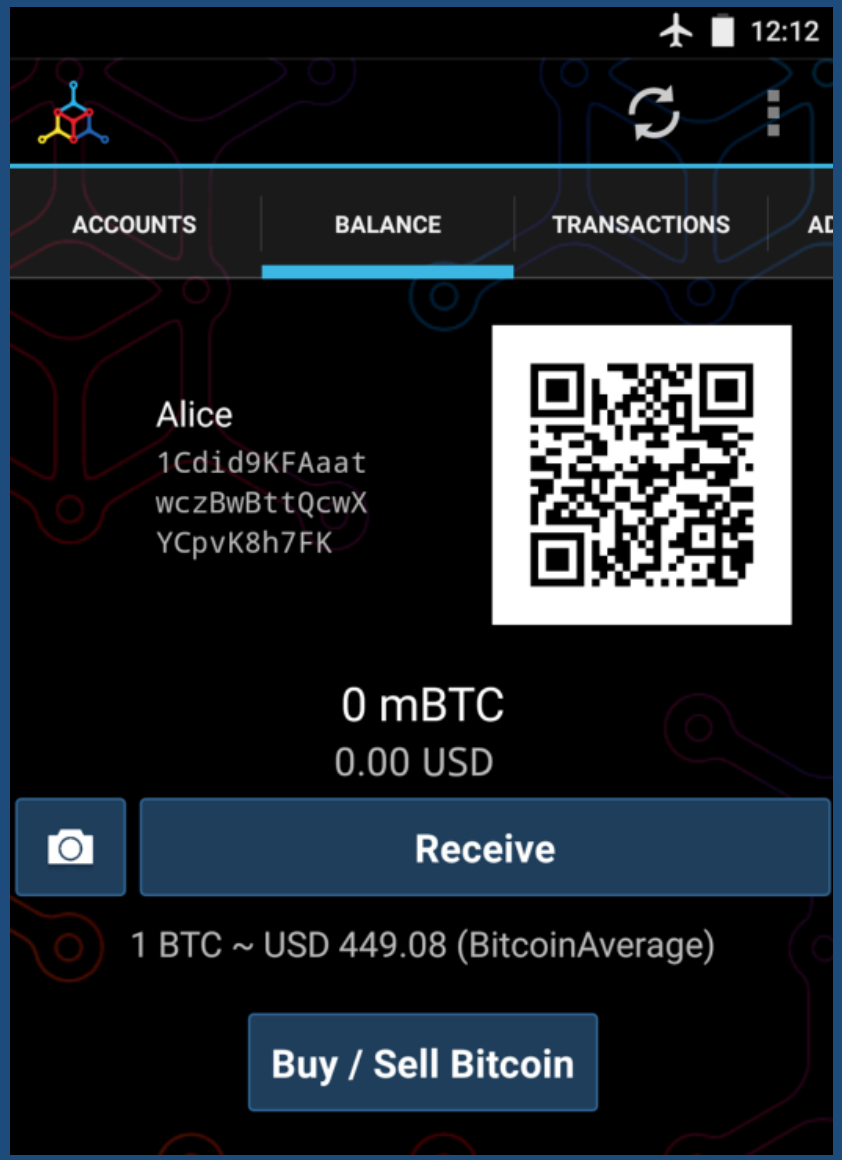
02. Blockchain P2P Network – Consensus Algorithm

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

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

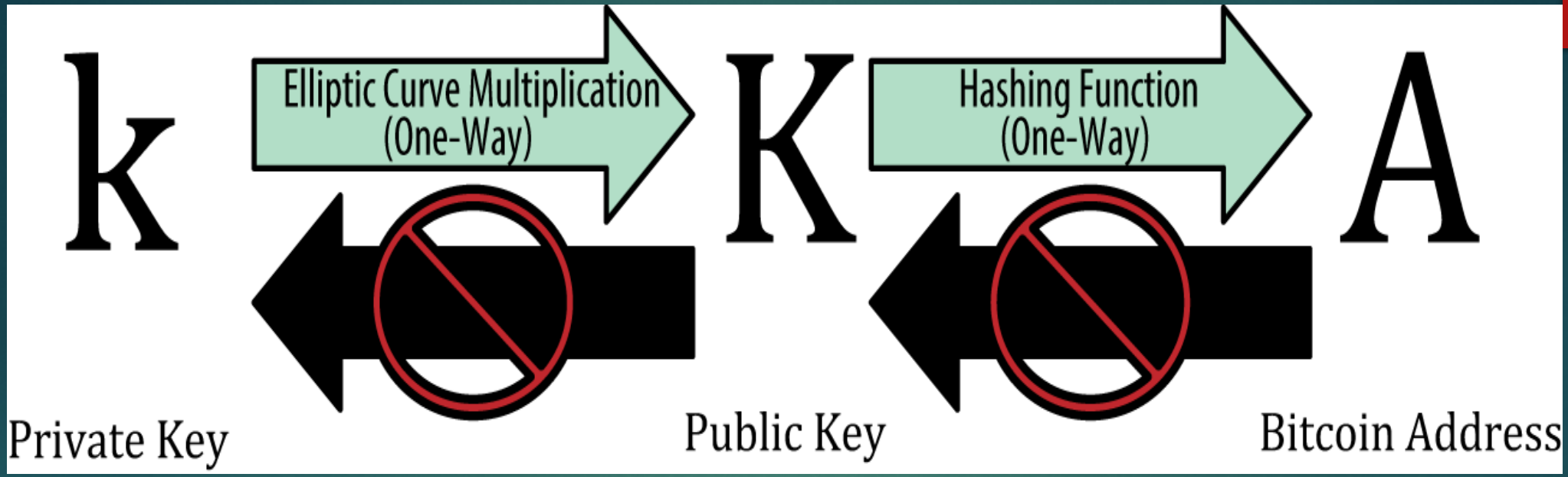
1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

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

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
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

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 MOD P = (1^3 + 7) MOD P
Y = SQRT(8) => Y = ± 2.82
(X=1, Y = ± 2.82) or (X=3, Y = ± 5.83)

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

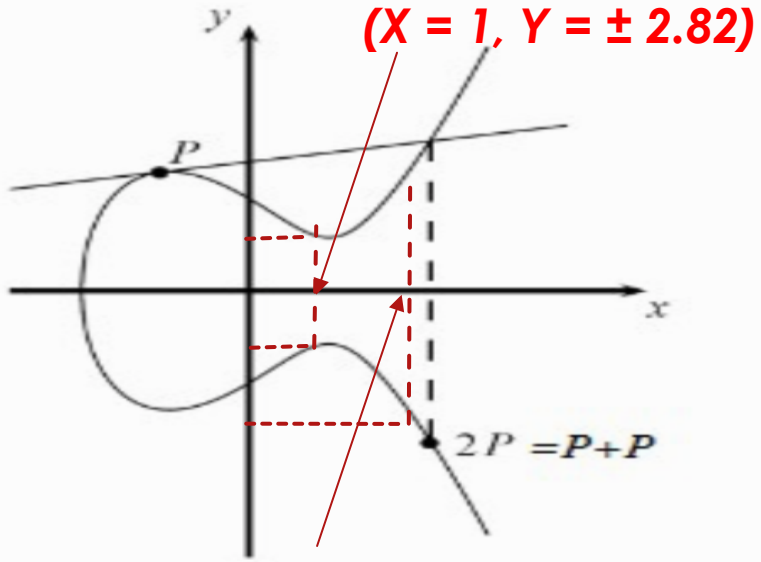
or

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

The *mod 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.



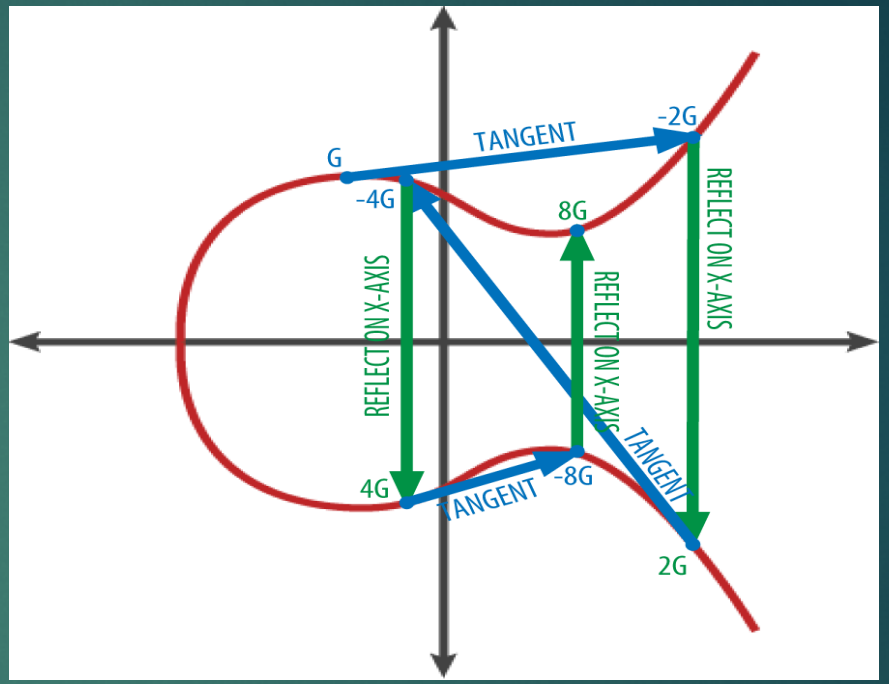
Graph representing point doubling over real numbers

In the case of point doubling, the equation becomes:

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

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

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



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

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

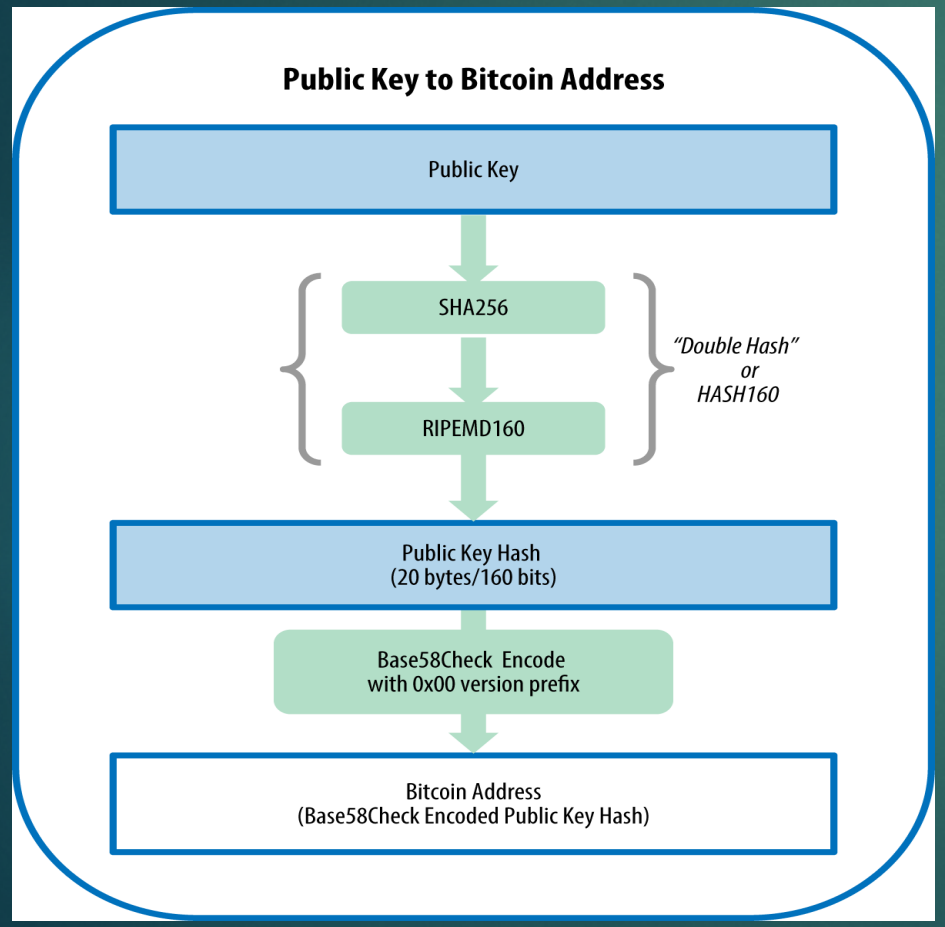
G (uncompressed) = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

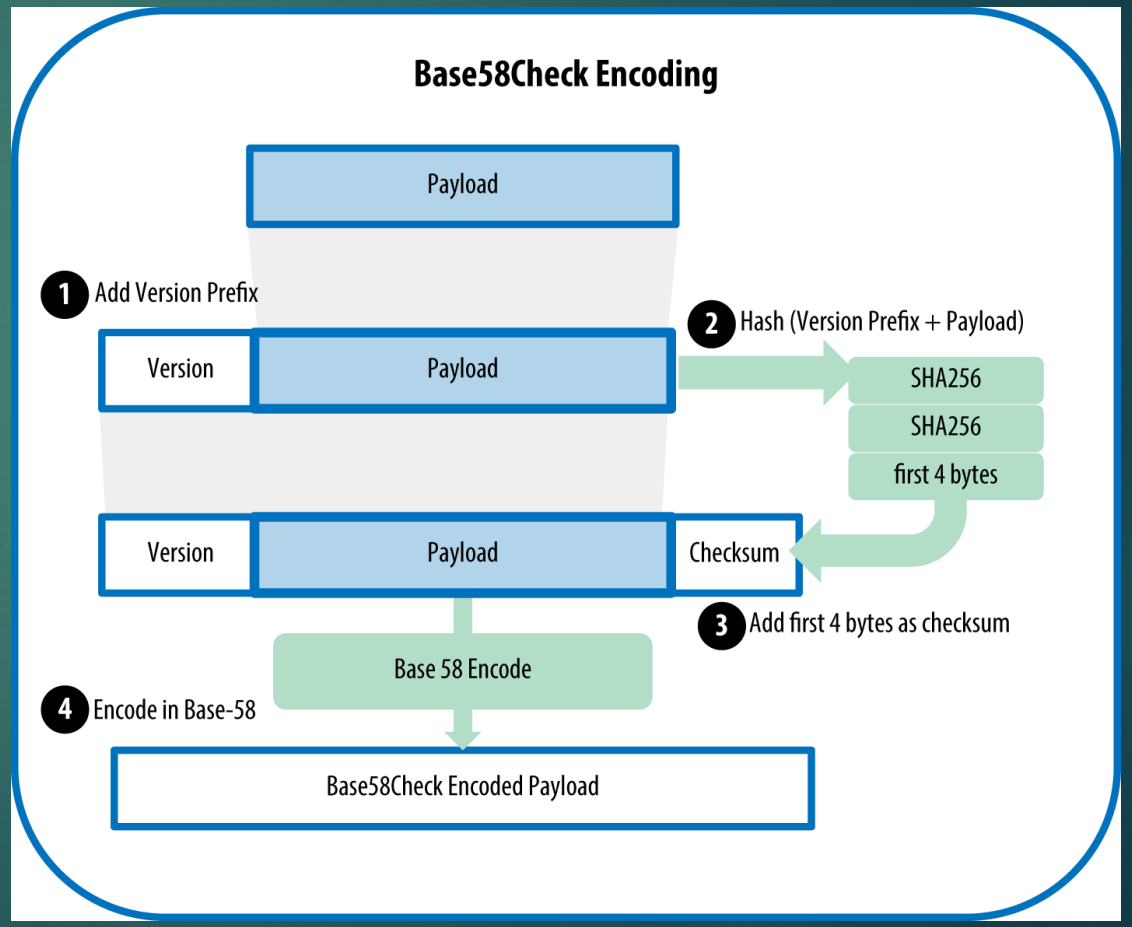
02 Bitcoin / Ethereum Blockchain Technology - Addresses & Keys

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:

$$A = RIPEMD160(SHA256(K))$$

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



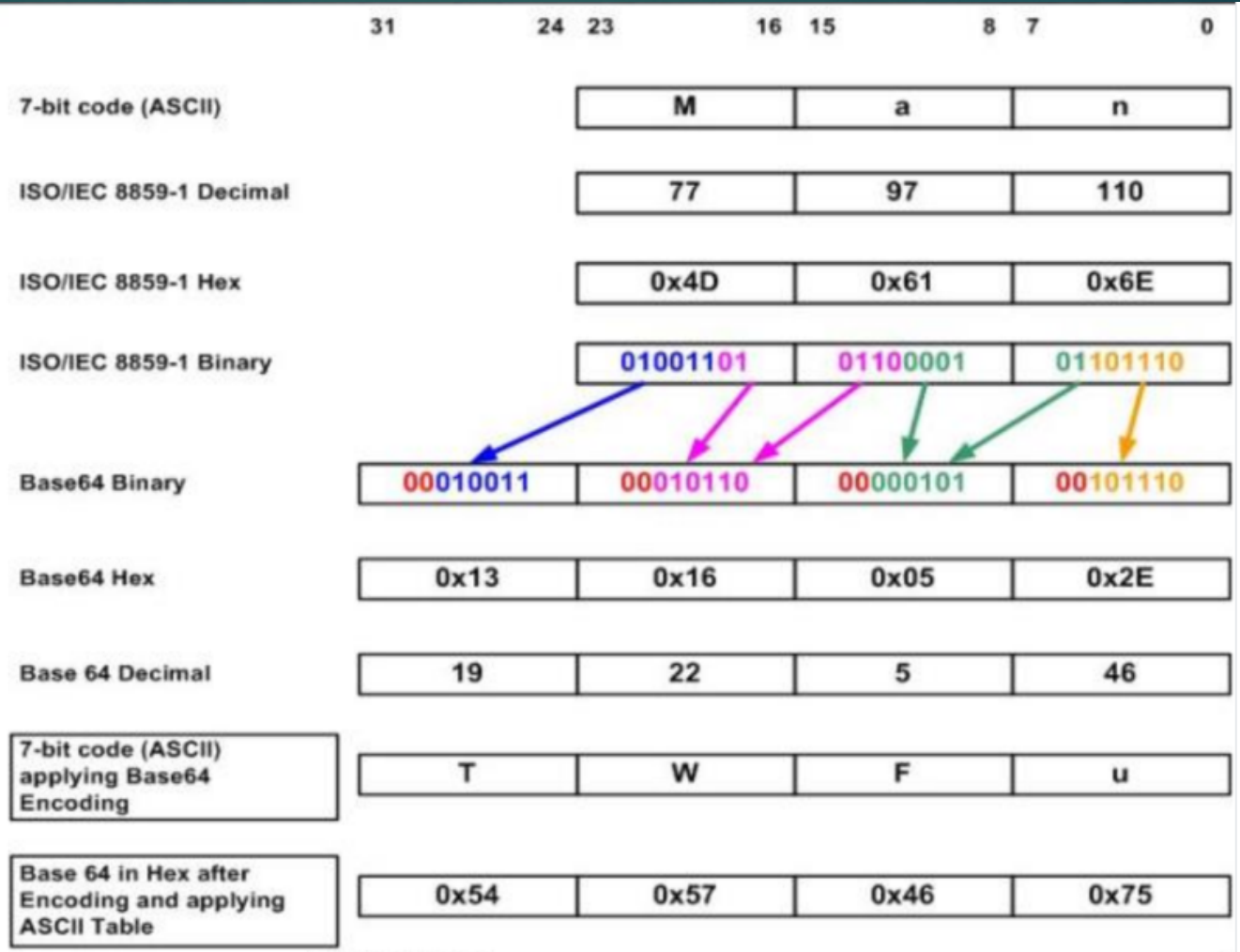


Public key to bitcoin address: conversion of a public key into a bitcoin address.

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.



Base 64 Code

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
0	1	2	3	4	5	6	7	8	9	+	/	=													
52	53	54	55	56	57	58	59	60	61	62	63	64 (64 is used as padding)													

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.

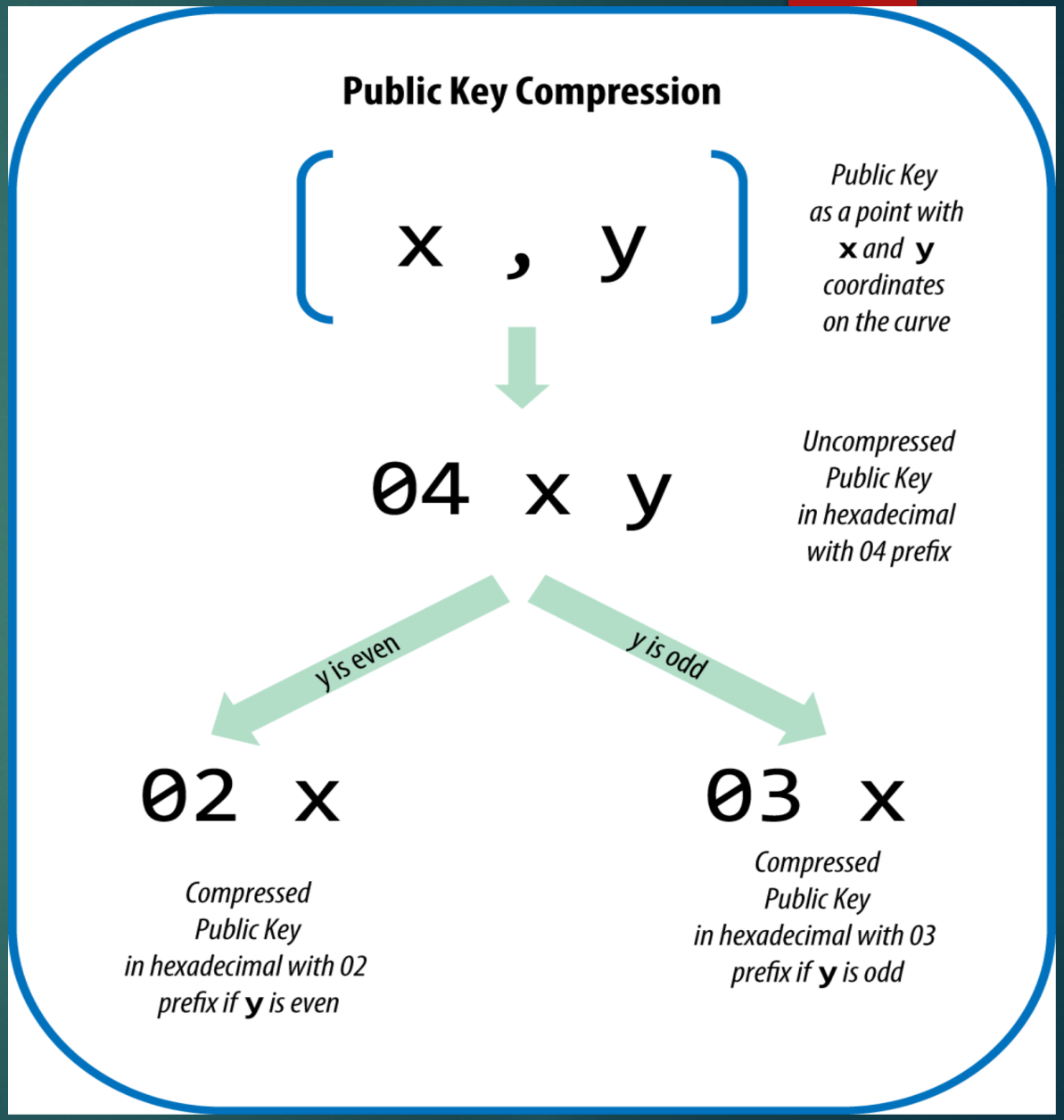
Base64 to Base58 Analogy

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:*

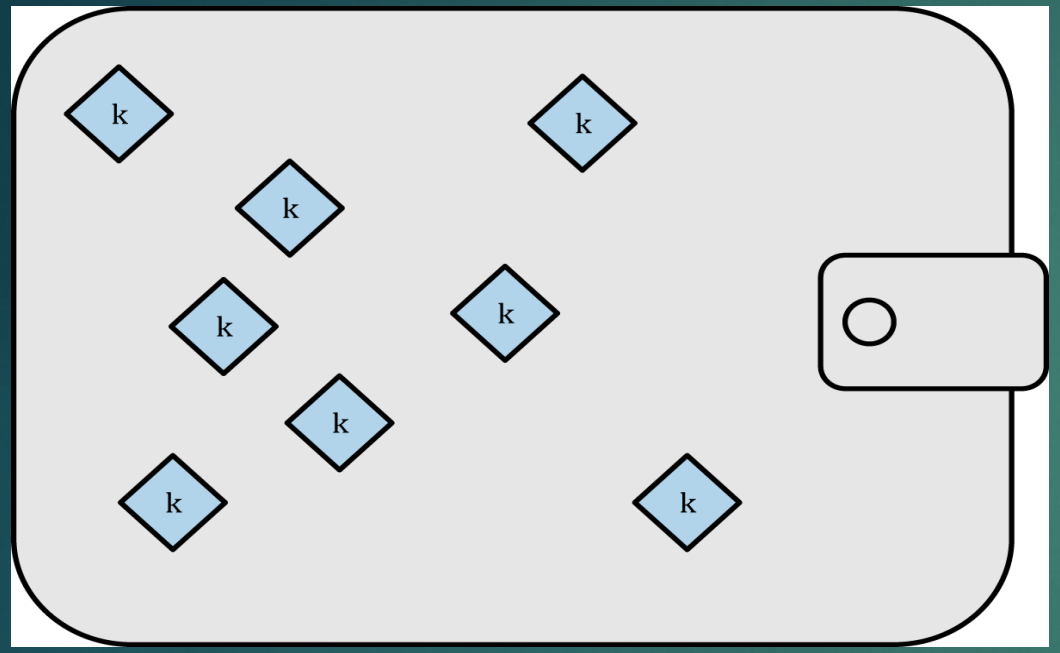
y^2 mod p = (x^3 + 7) mod p.

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!



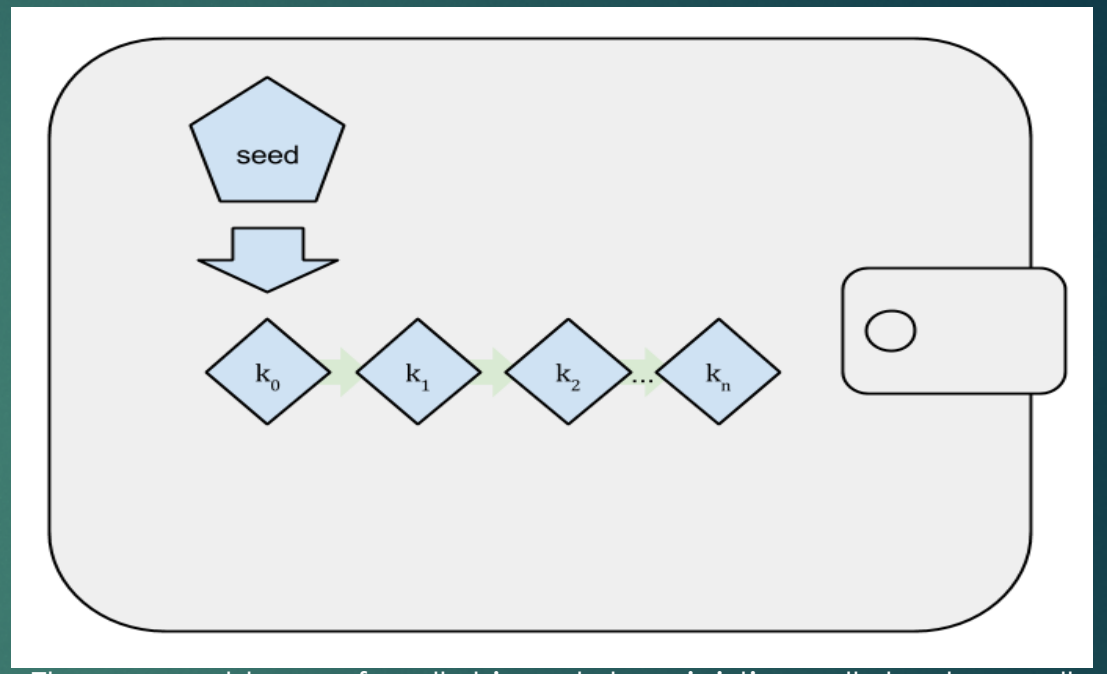
02 Bitcoin / Ethereum Blockchain Technology - Wallets

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."

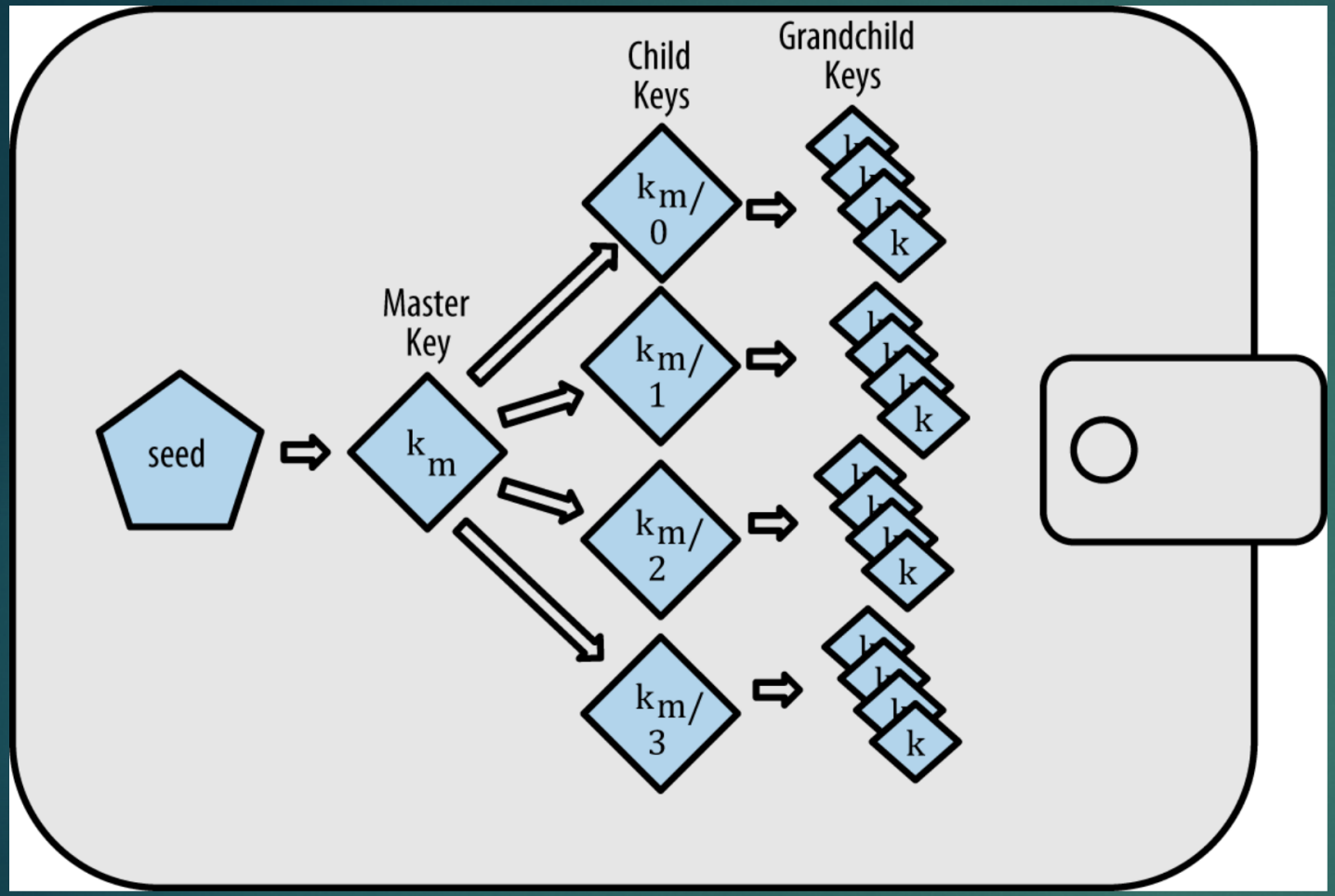
Type-0 nondeterministic (random) wallet: a collection of randomly generated keys



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

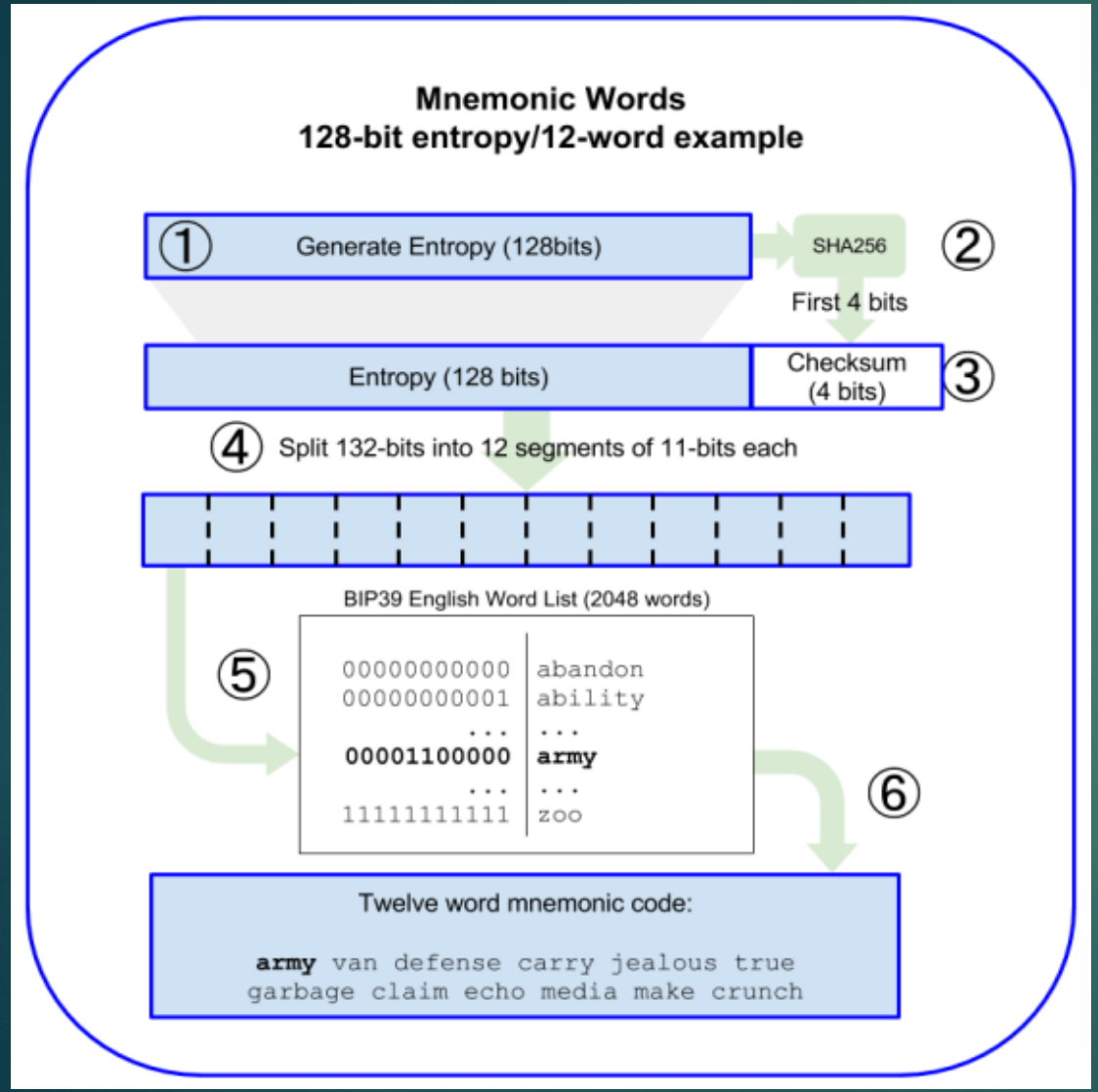
02 Bitcoin / Ethereum Blockchain Technology - Wallets

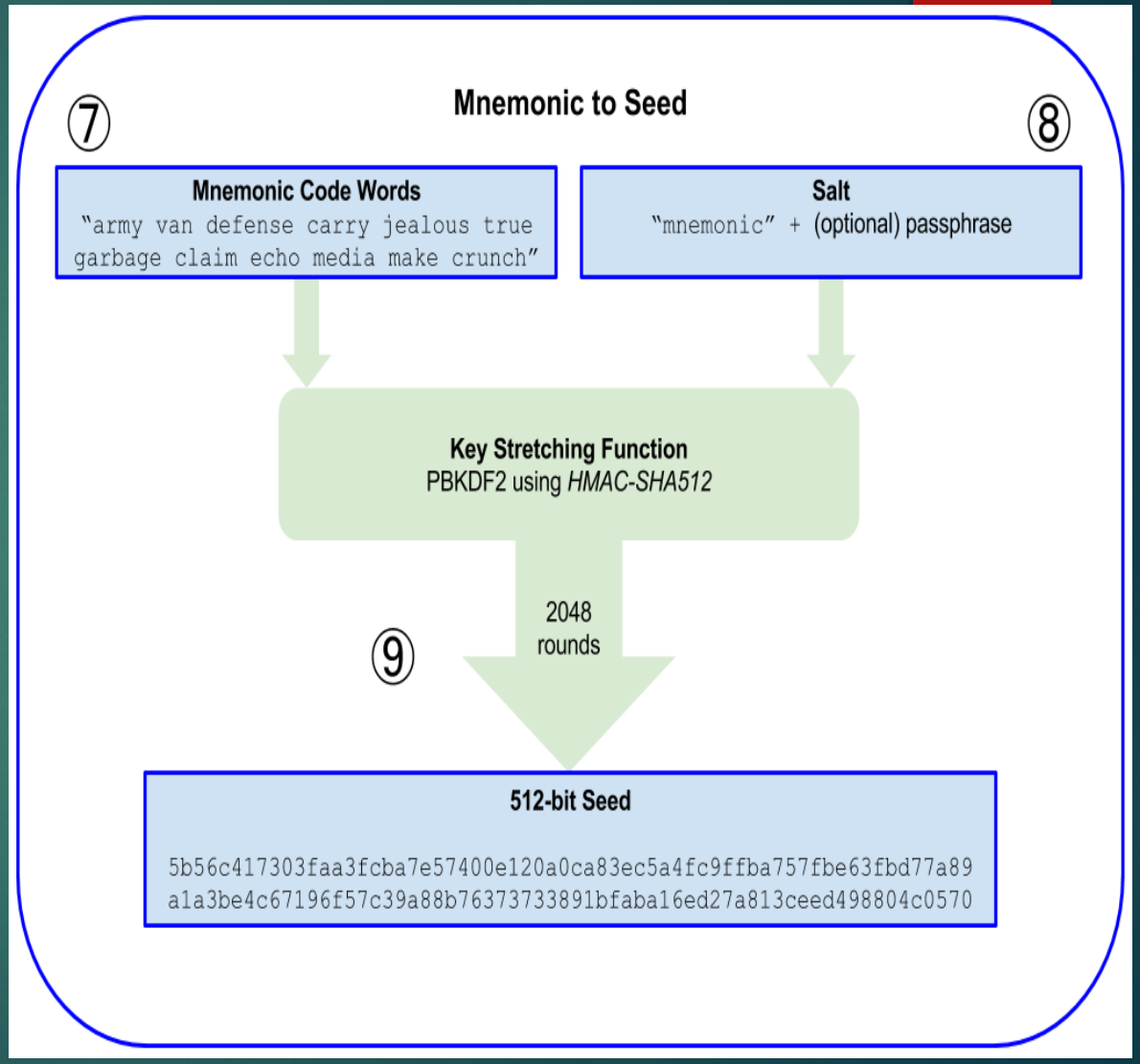


Type-2 HD (*Hierarchical Deterministic*) Wallet: a tree of keys generated from a single seed
HD Wallets (Bitcoin Improvement Proposal: BIP-32/BIP-44)

02 Bitcoin / Ethereum Blockchain Technology - HD Wallets

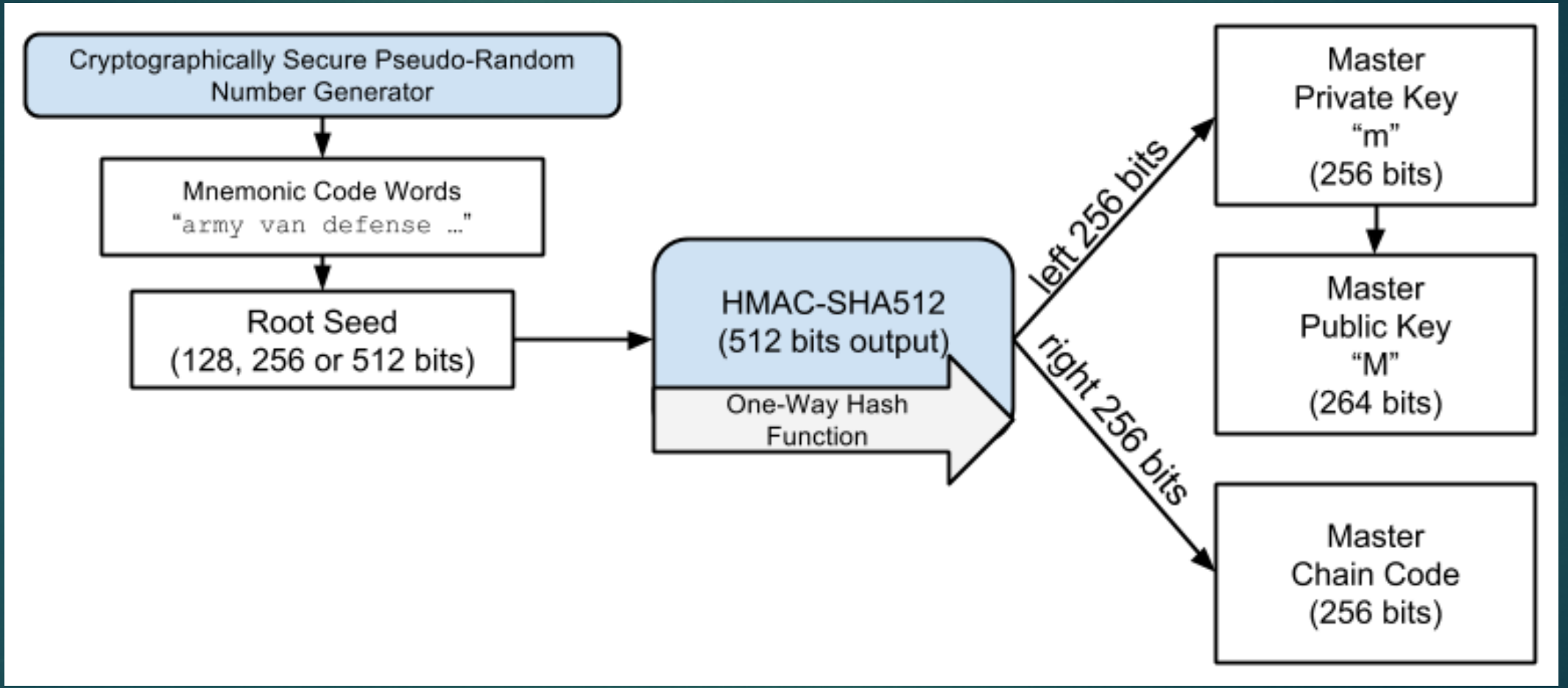
Generating entropy and encoding as mnemonic words





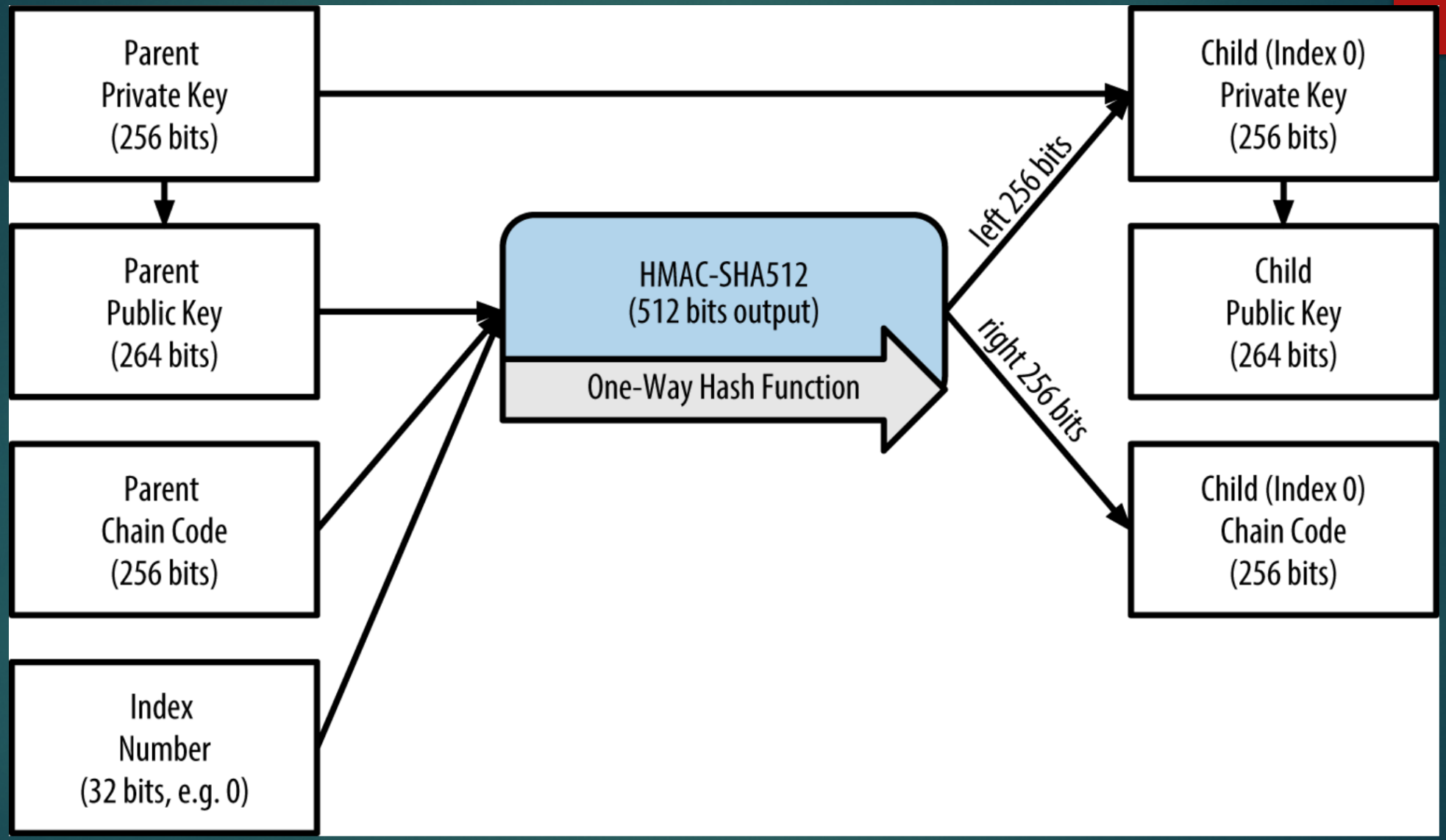
02. Bitcoin / Ethereum Blockchain Technology - HD Wallets

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:



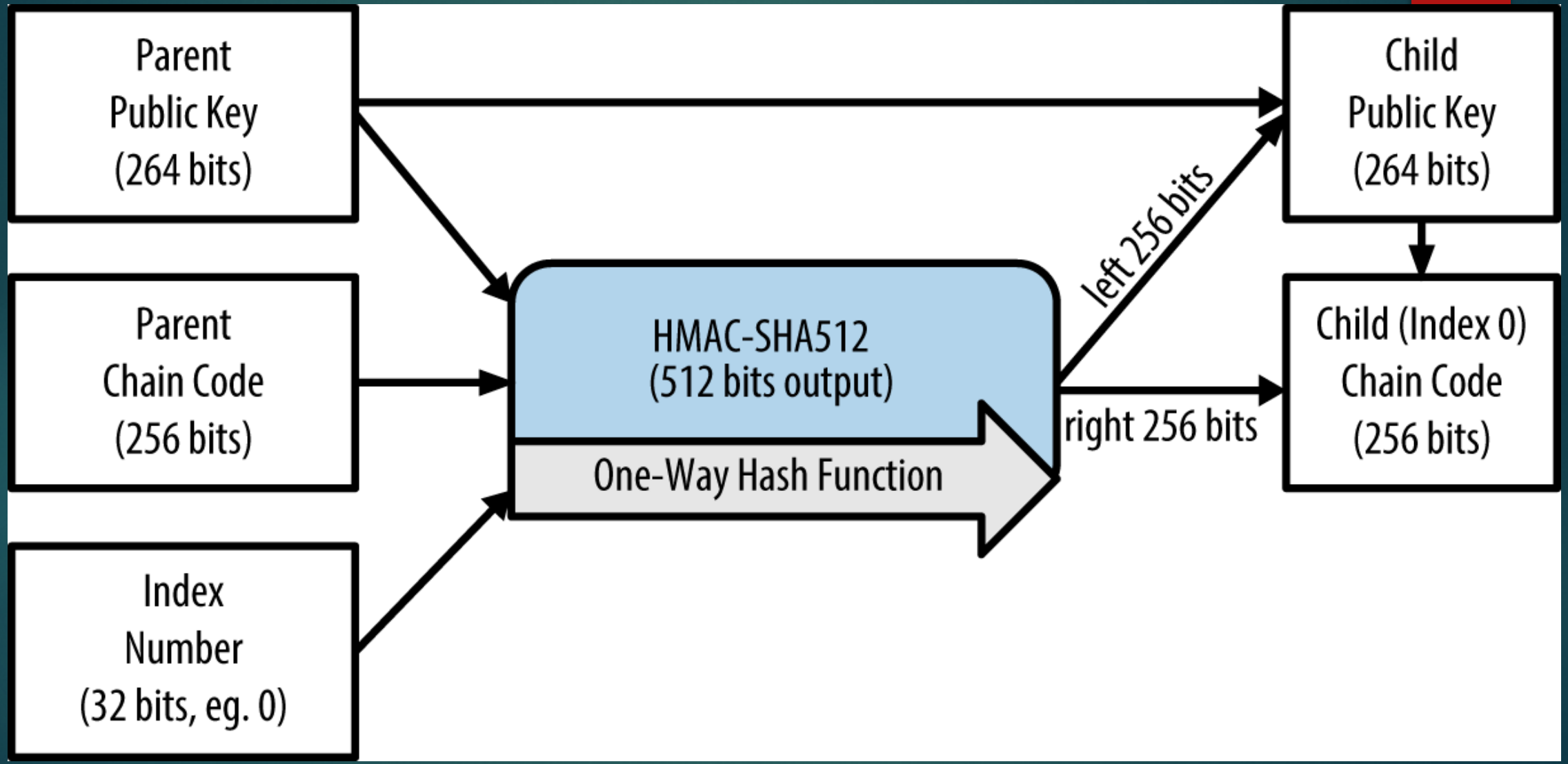
Creating master keys and chain code from a root seed

02. Bitcoin / Ethereum Blockchain Technology - HD Wallets



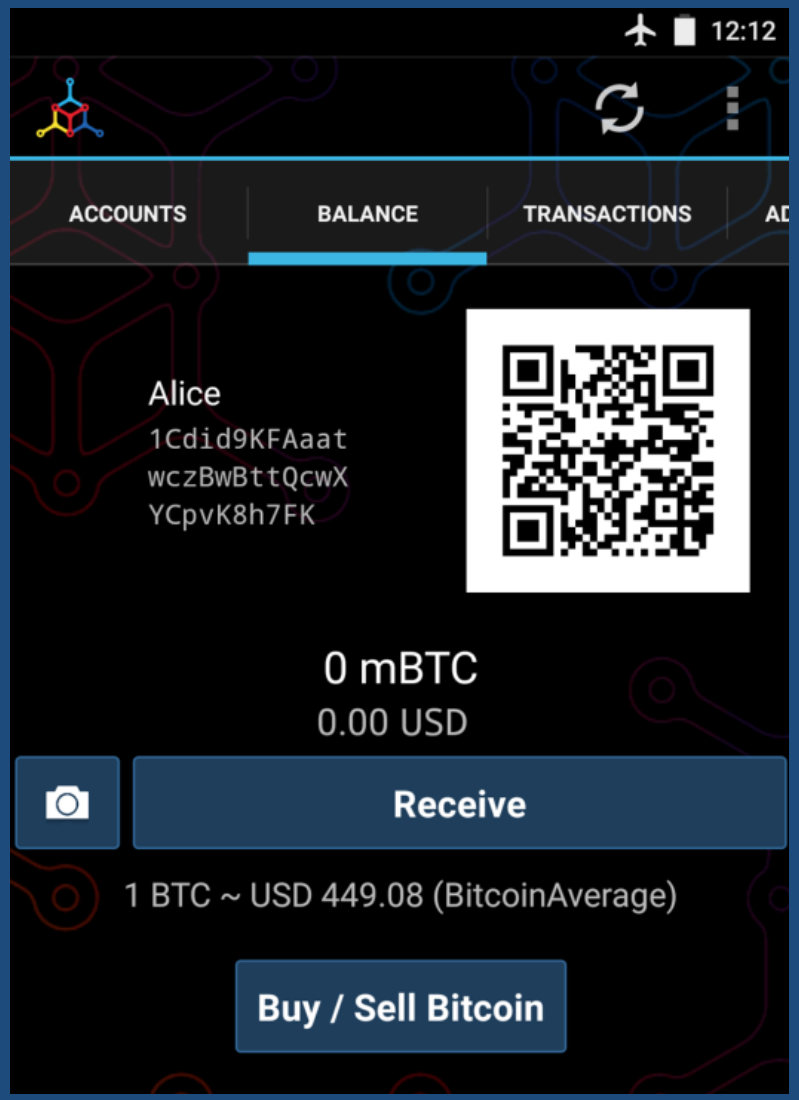
Extending a parent private key to create a child private key

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

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

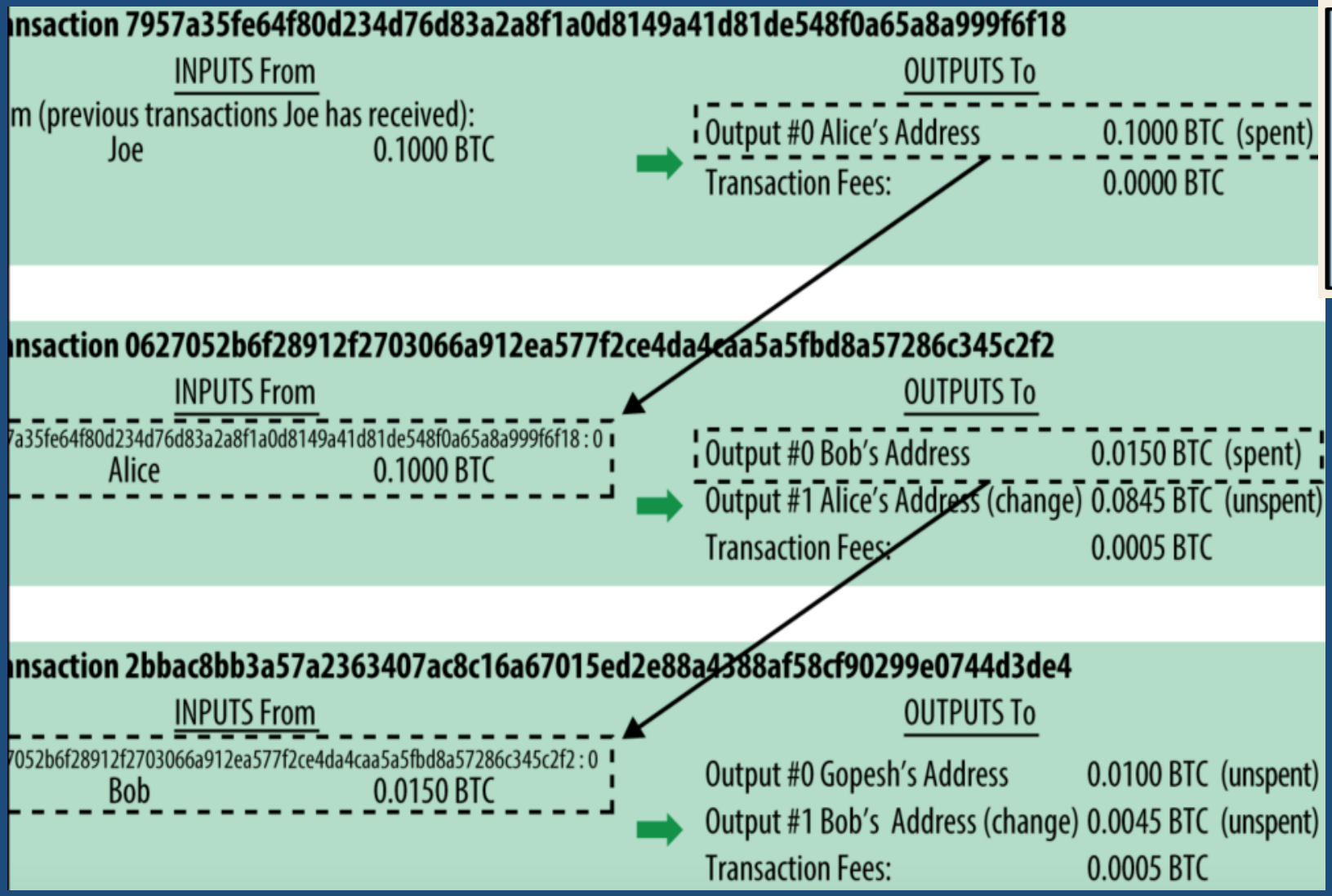
1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

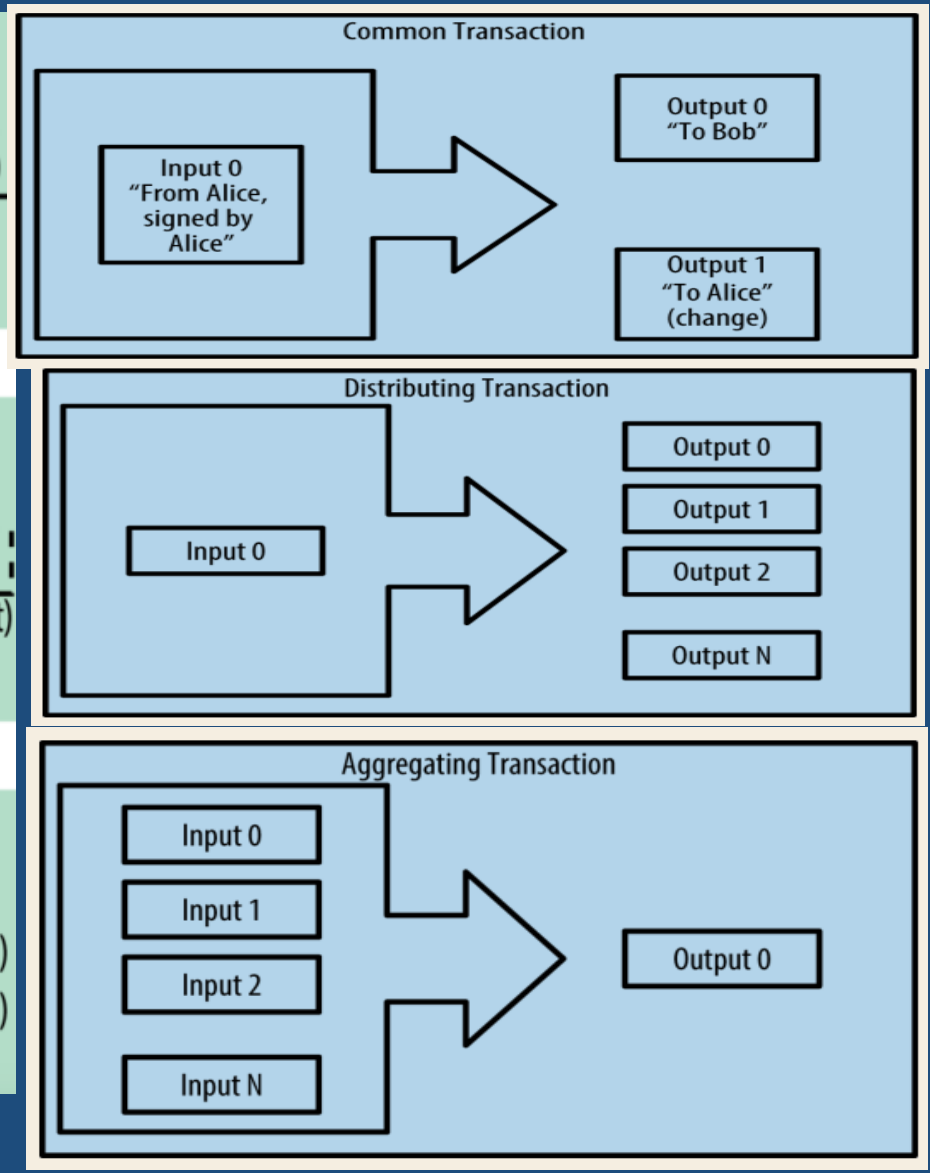
97 Confirmations | 0.0995 BTC

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

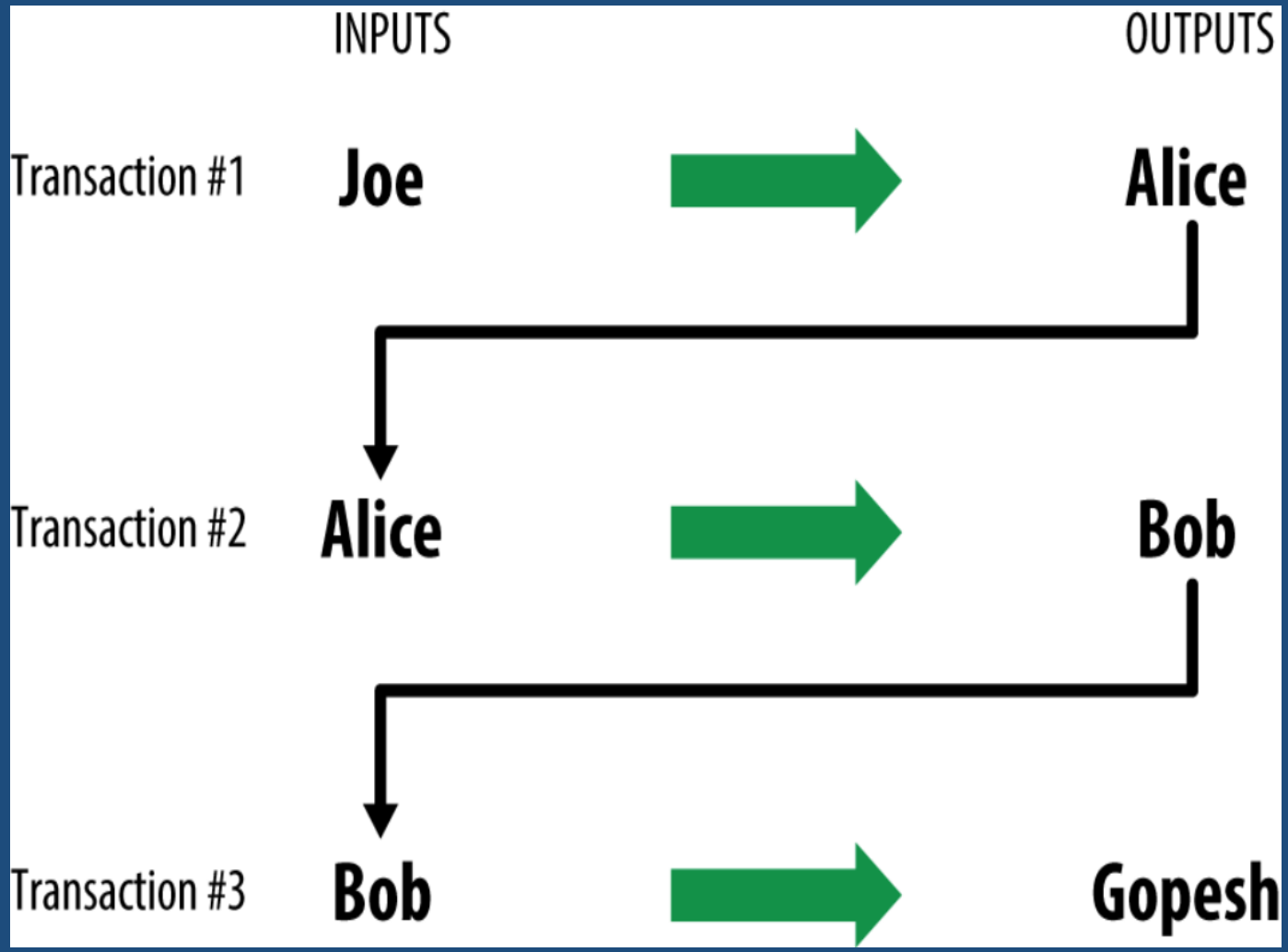
Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

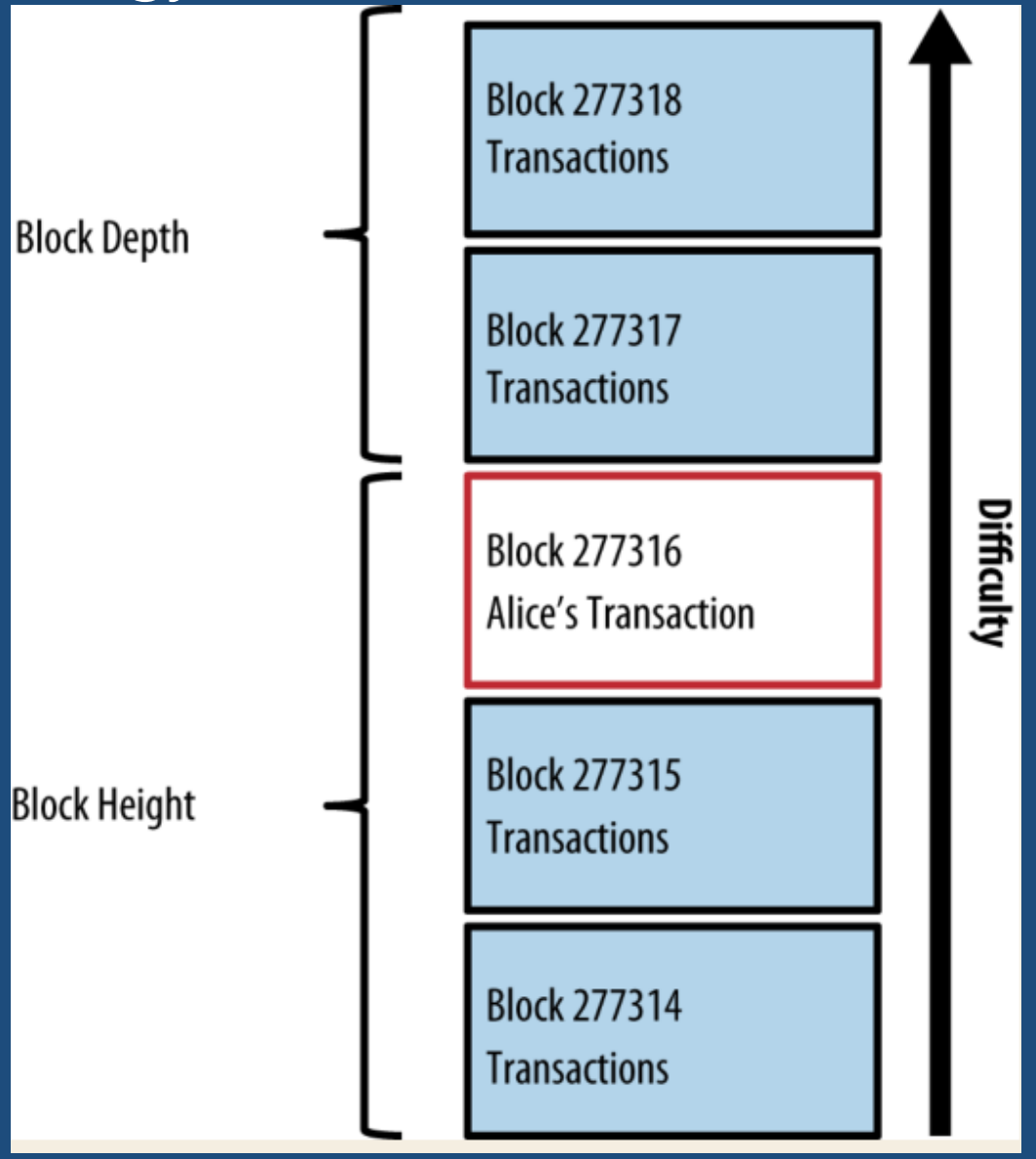
2.1 Bitcoin / Ethereum Blockchain Technology - Transactions





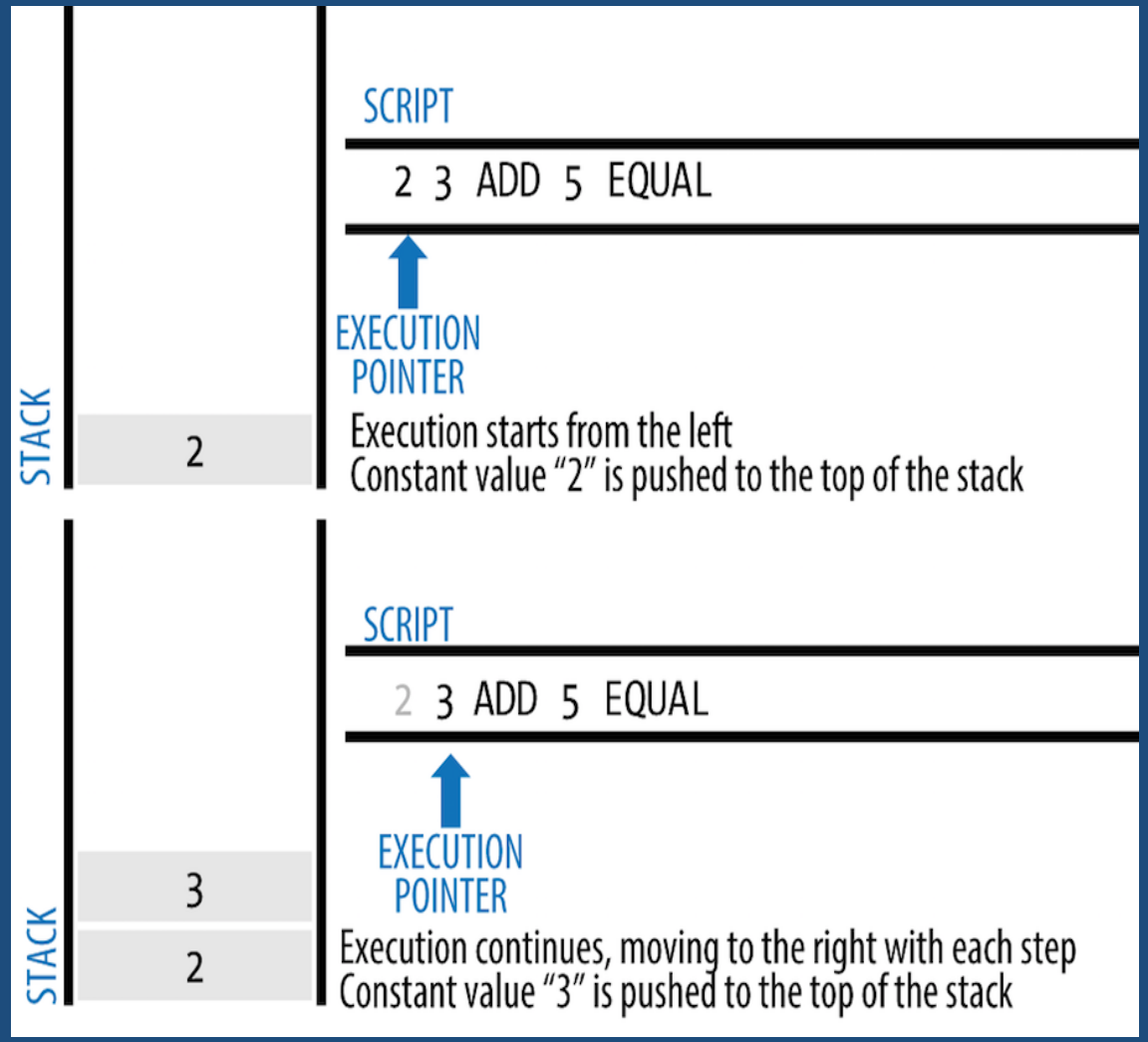
2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

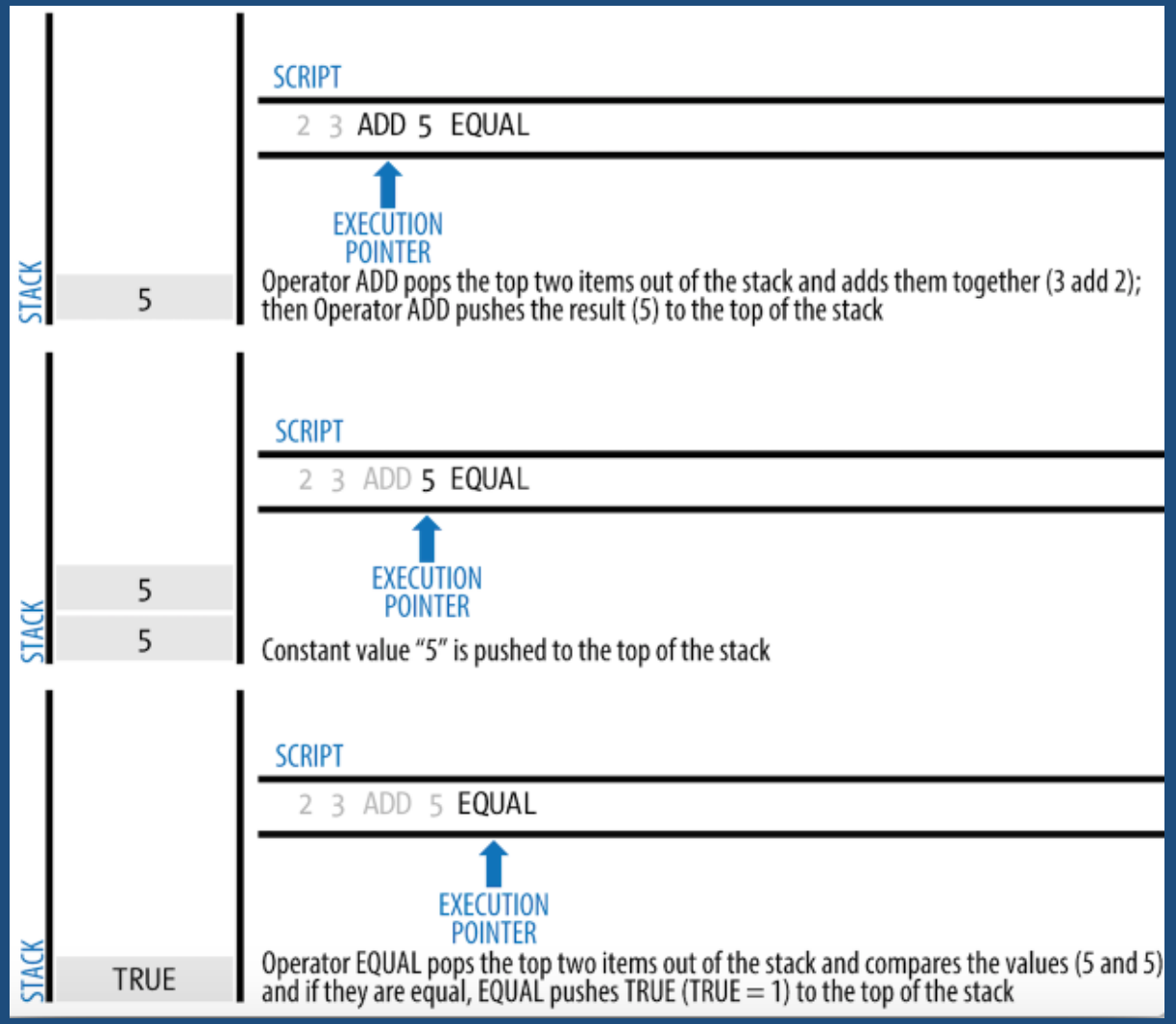




2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

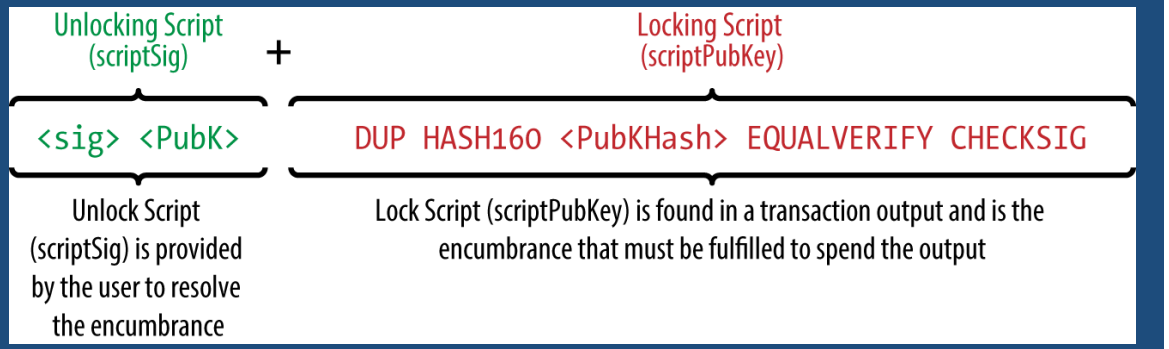
Bitcoin's script validation doing simple math



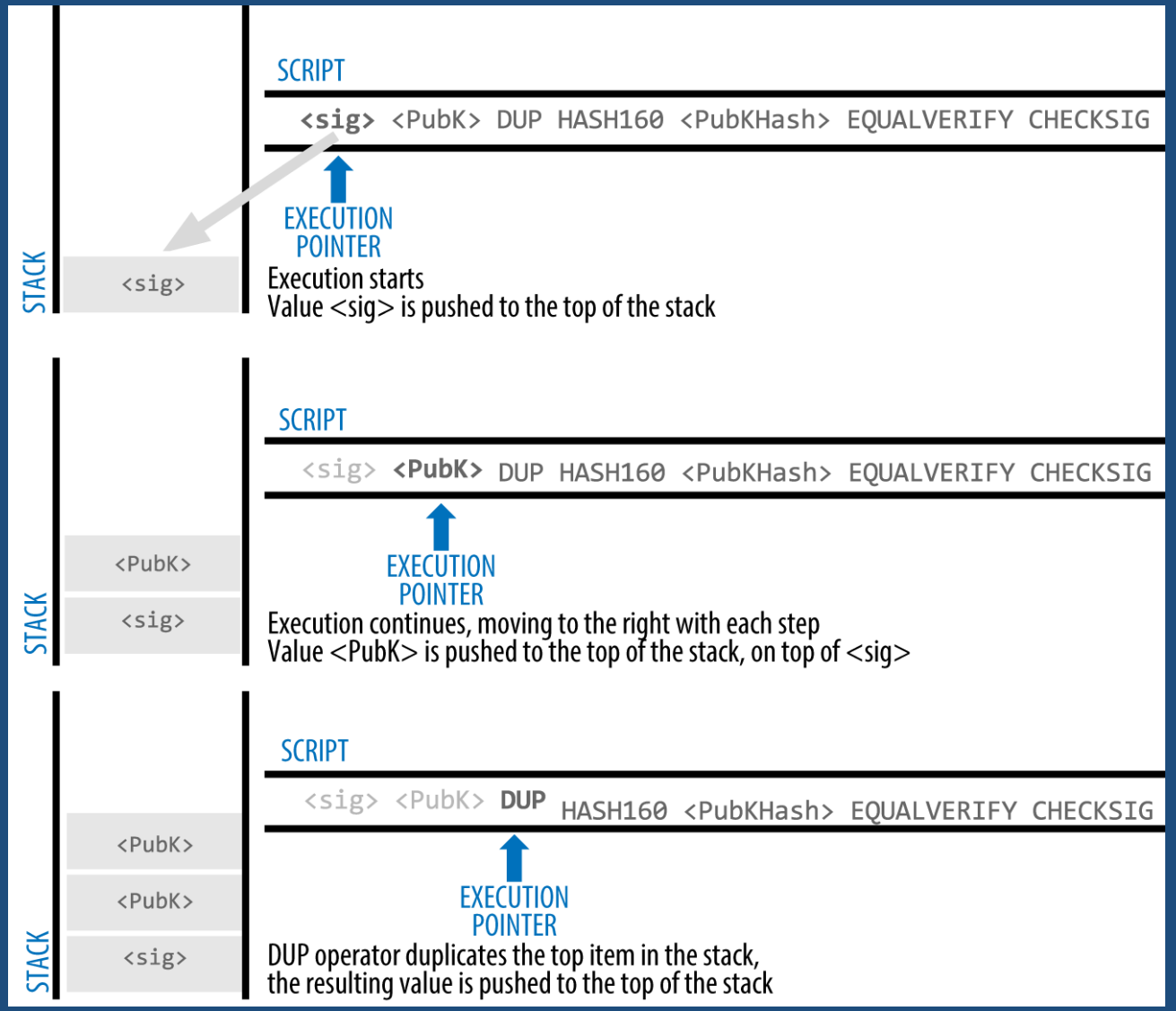


2.1 Bitcoin / Ethereum Blockchain Technology - Transactions

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



010000000**1186f9f998a5aa6f048e51dd8419a14d8a**
0f1a8a2836dd734d2804fe65fa35779000000008
b483045022100884d142d86652a3f47ba4746ec
719bbfbd040a570b1deccbb6498c75c4ae24cb02
204b9f039ff08df09cbe9f6addac960298cad530a
863ea8f53982c09db8f6e381301410484ecc0d46
f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8
416ab9fe423cc5412336376789d172787ec3457e
ee41c04f4938de5cc17b4a10fa336a8d752adfffff
ffff0260e31600000000001976a914ab68025513
c3dbd2f7b92a94e0581f5d50f654e788acd0ef80000
00000001976a9147f9b1a7fb68d60c536c2fd8aeaa5
3a8f3cc025a888ac00000 000

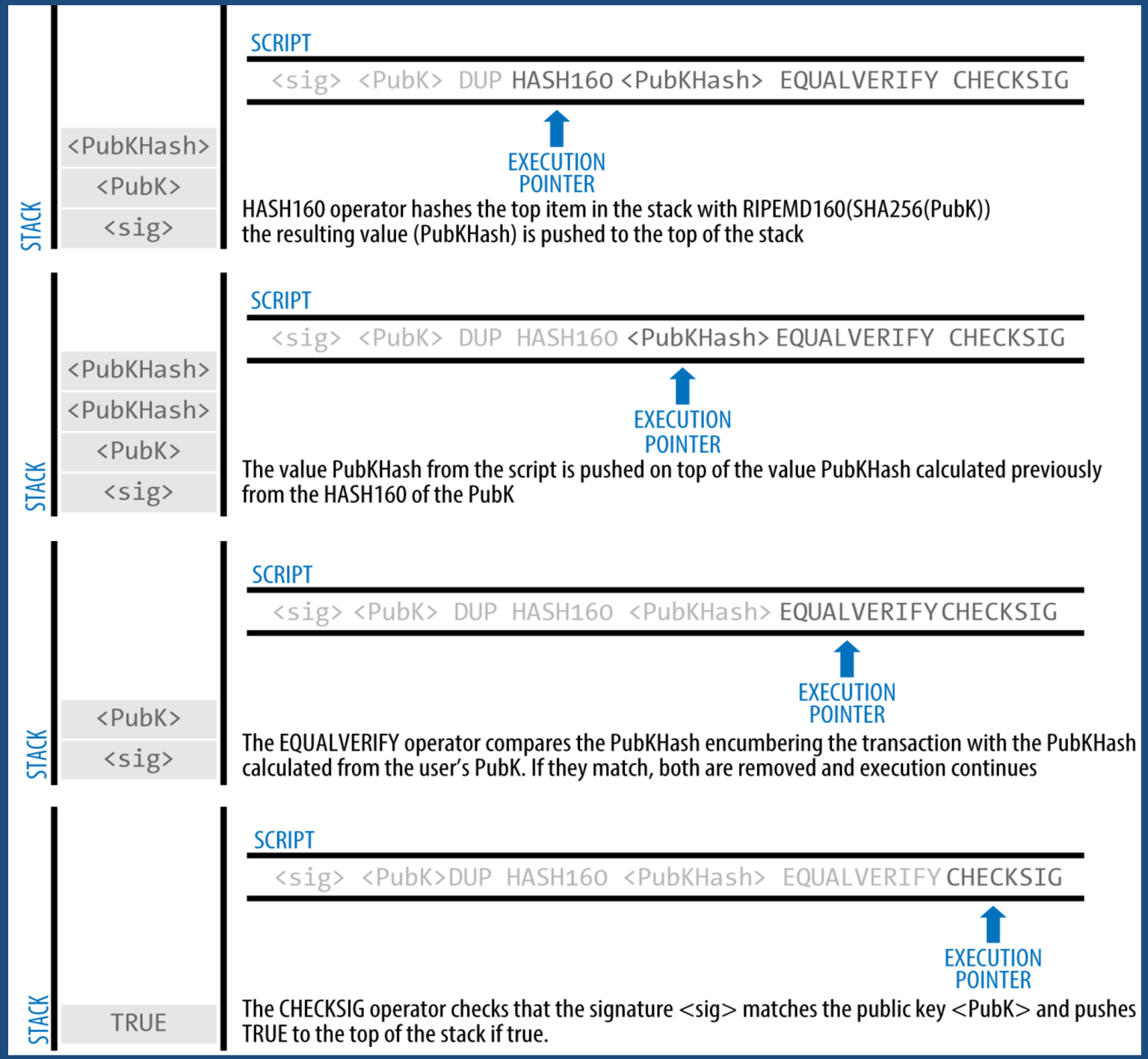


https://www.blockchain.com/explorer

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` — `00884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb`
- `0x02` — another integer follows
- `0x20` — the length of the integer (32 bytes)
- `S` — `4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813`
- 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

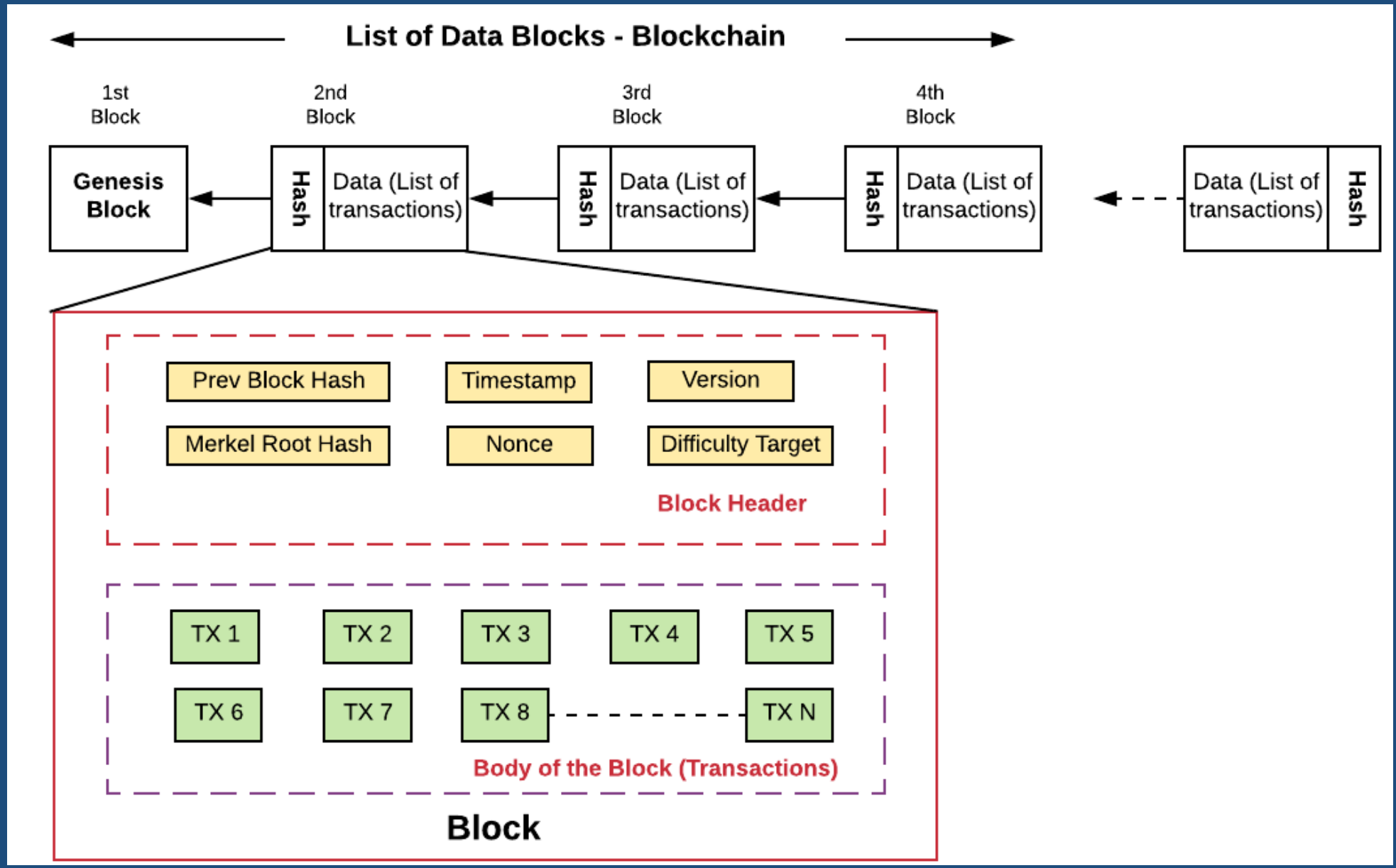
A. The block structure

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

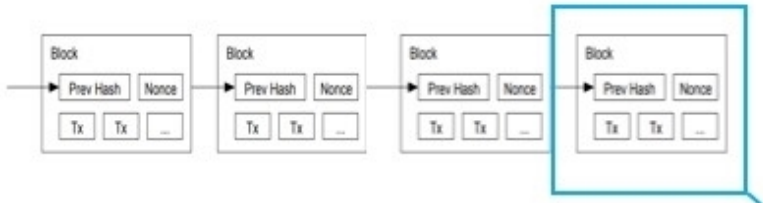
B. The Block Header structure

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



BITCOIN BLOCK

BLOCK HEADER

Hash from previous block

Metadata

Nonce

Merkel root

TRANSACTIONS

Transaction #1

Transaction #2

...

Transaction #n

Block header (80 byte)
SPV save only this information

Hash from previous block (32 byte):
«connection» to previous block

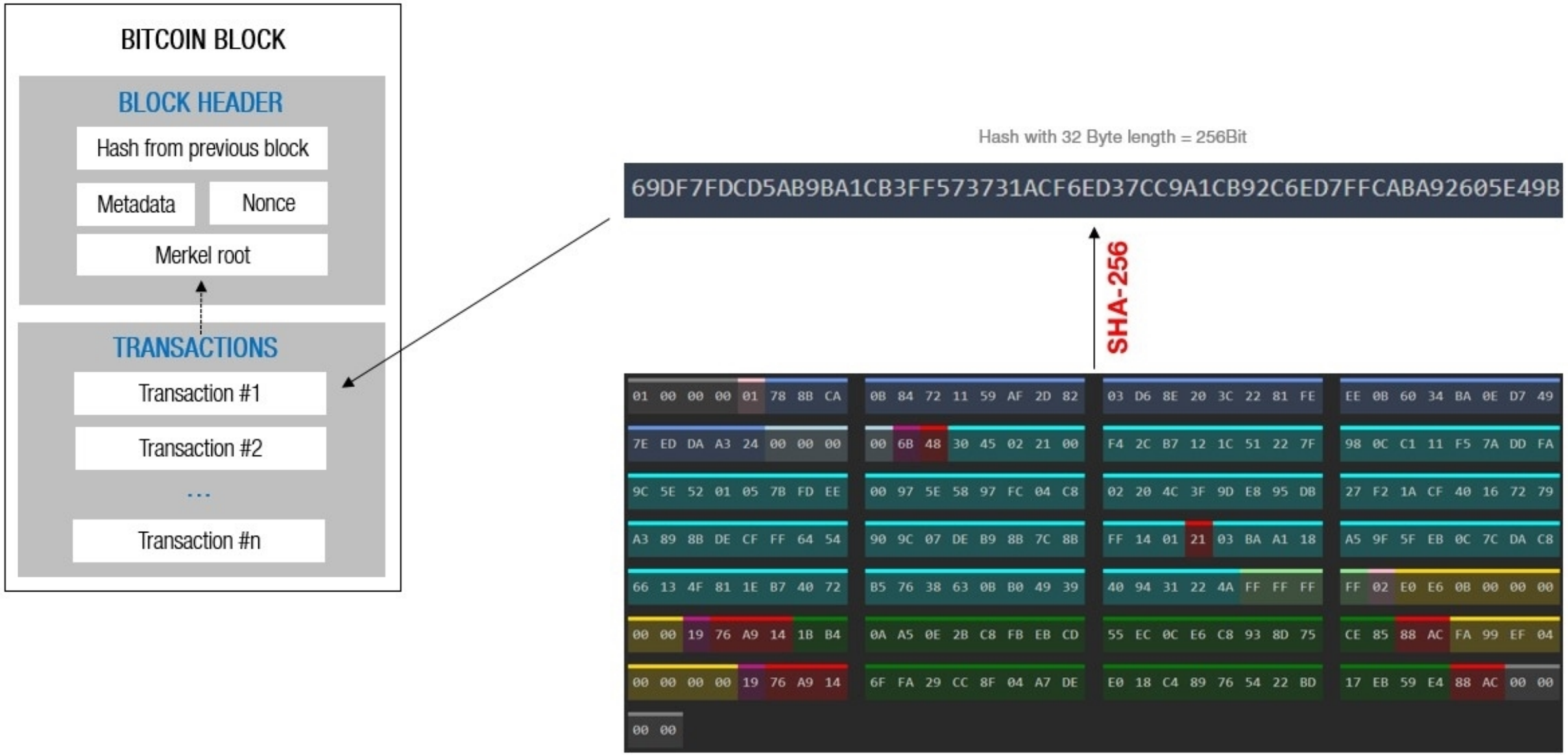
Nonce (4 byte):
«solution» of proof-of-work puzzle

Metadata (12 Byte):
- block version number
- time stamp
- difficulty

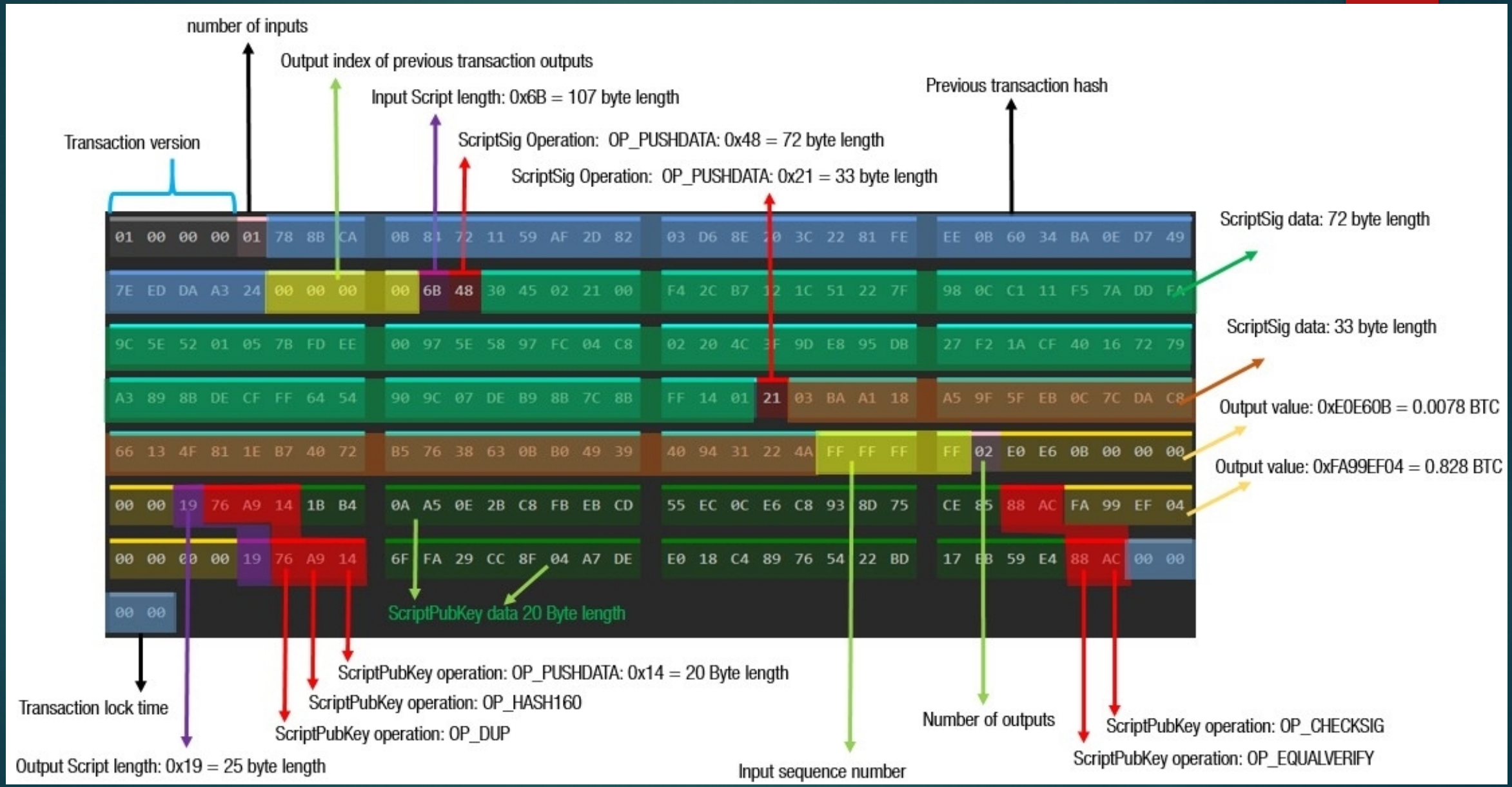
Merkle root (32 Byte Hash):
«summary» of all transactions

List of all transaction per block:
Full nodes saving all of these transactions

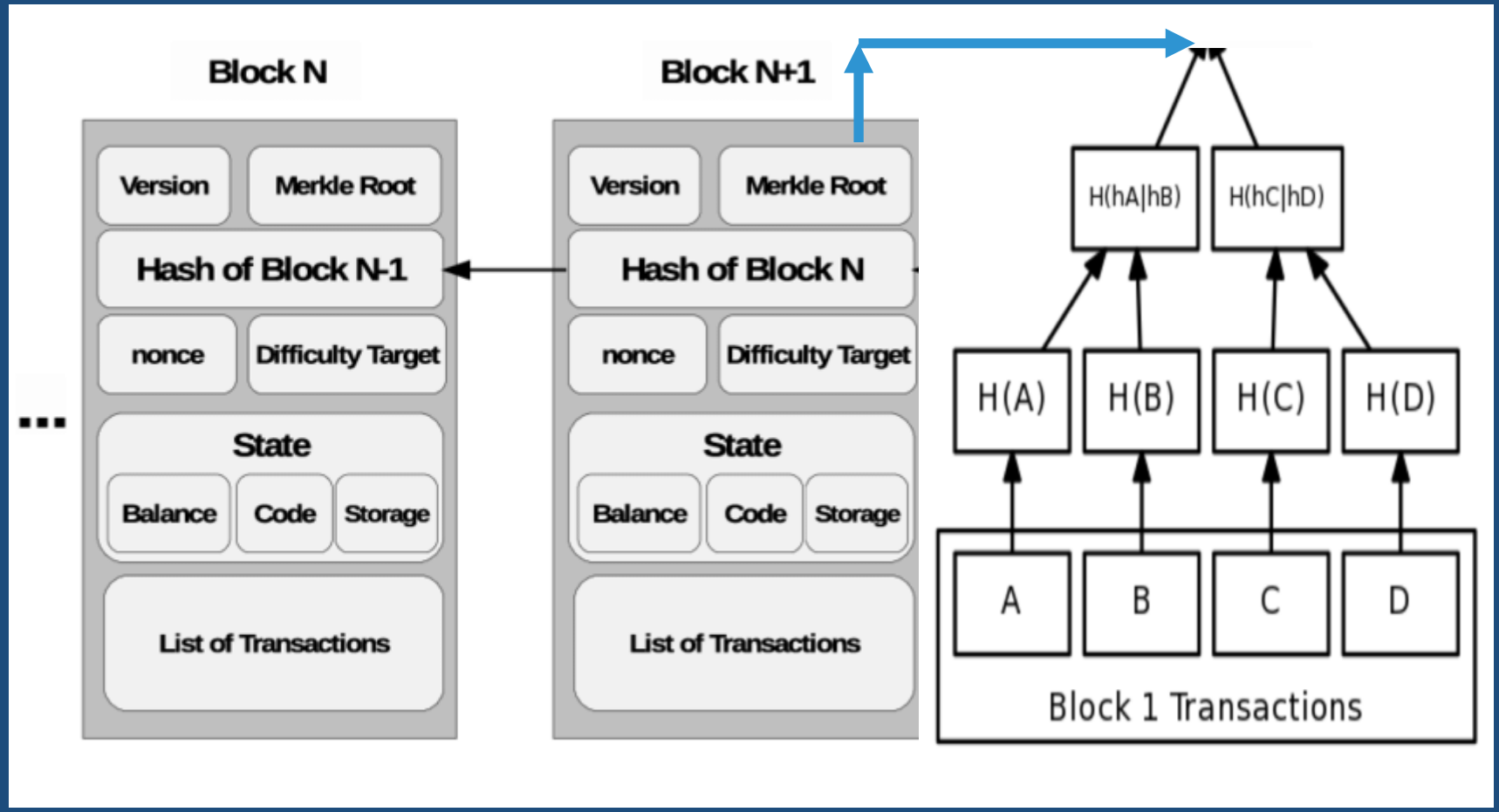
2.1 Bitcoin - Method from a raw Bitcoin HEX transaction into a block



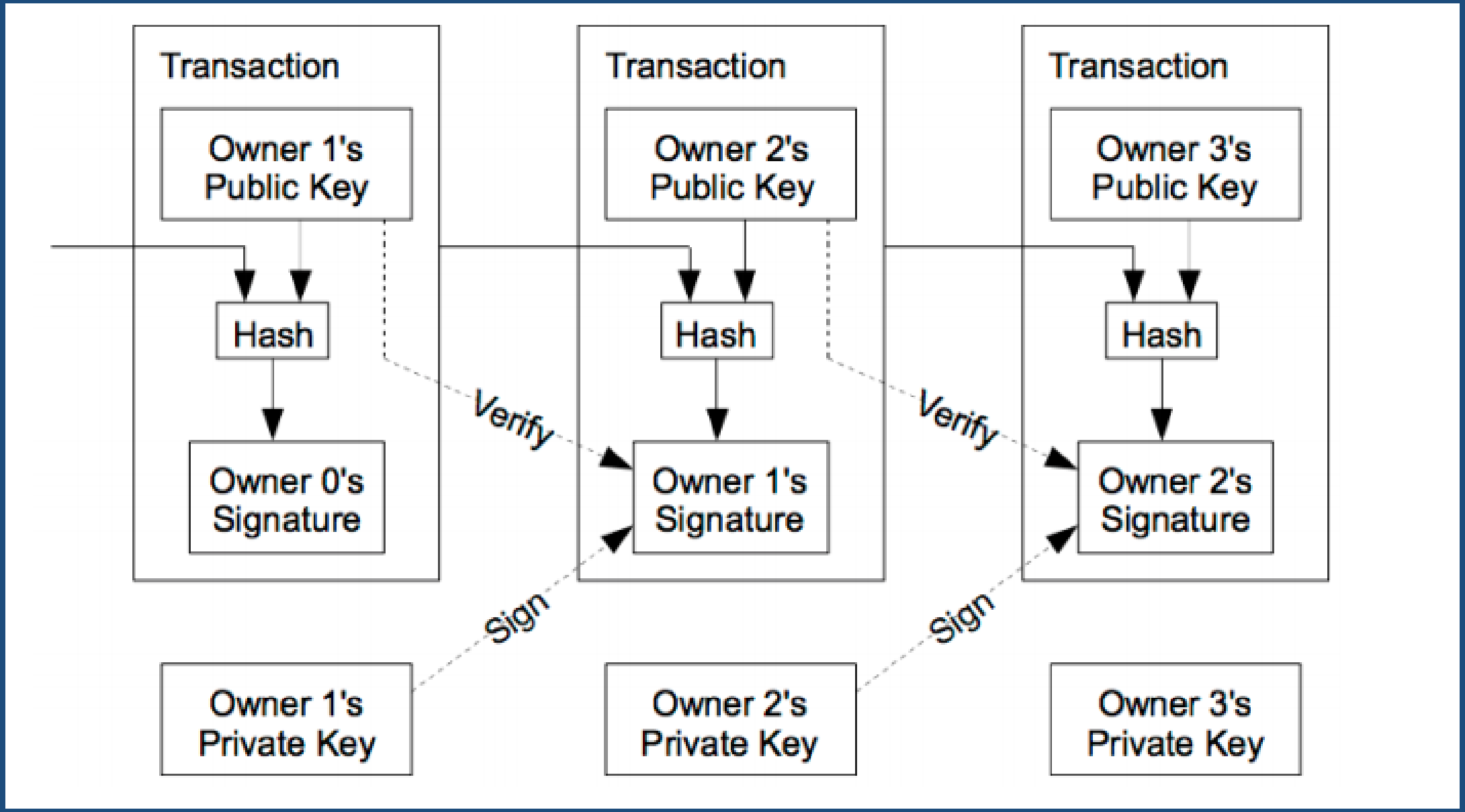
2.1 Detailed analysis of a raw Bitcoin transaction



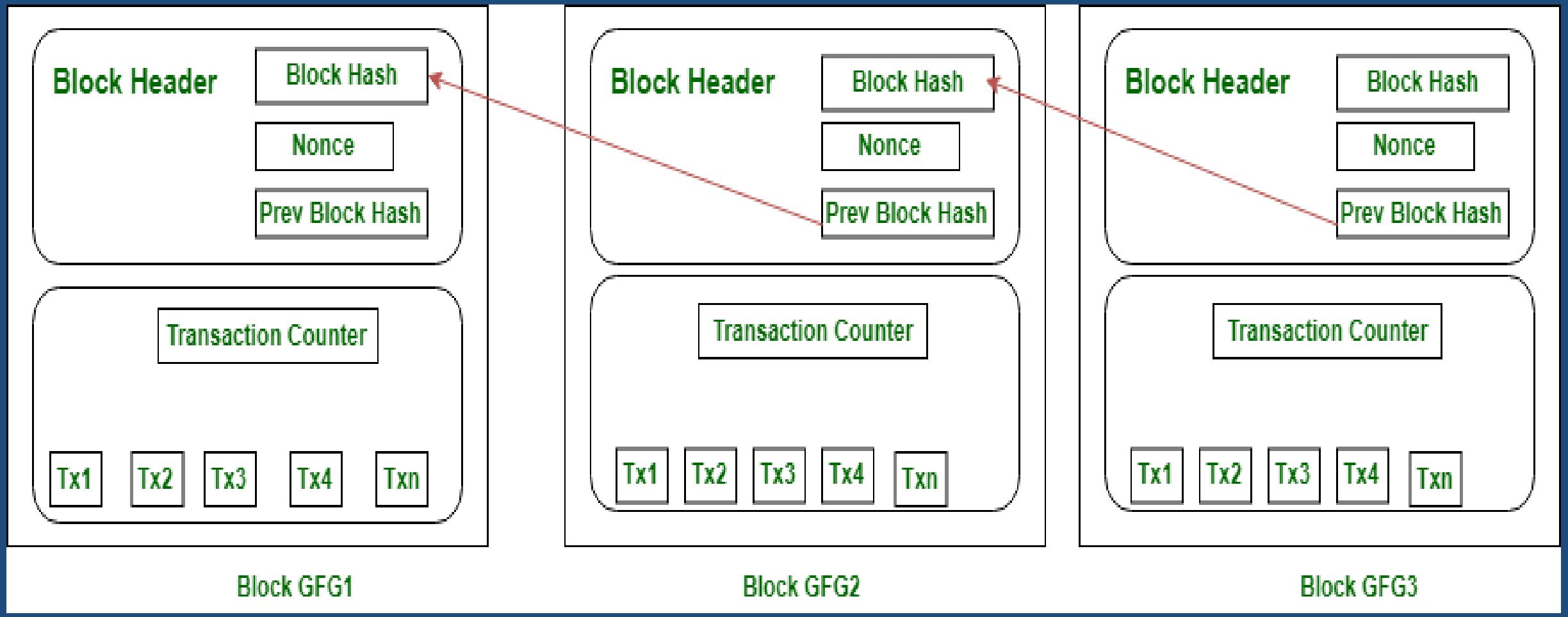
2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain



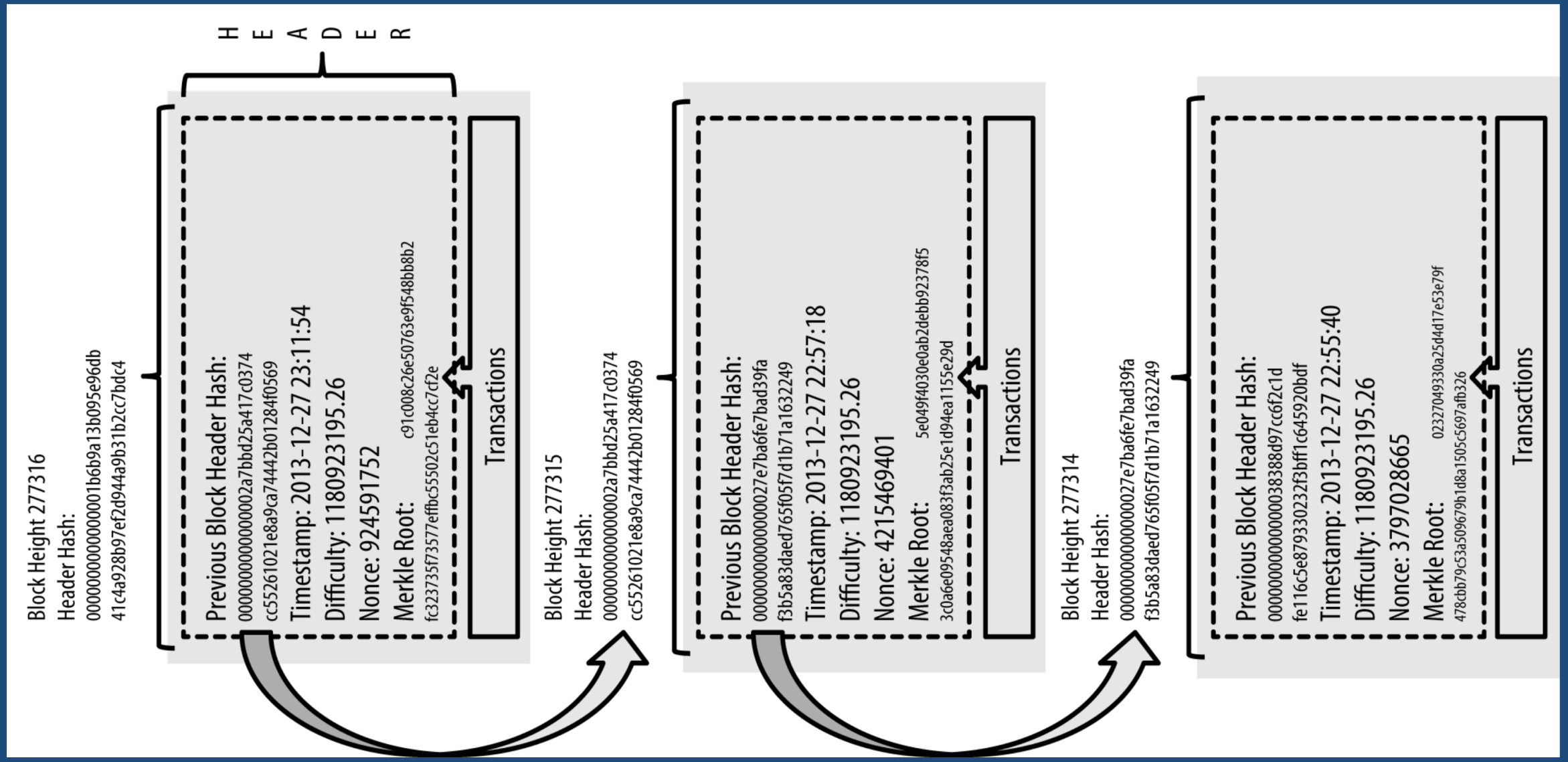
2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain



2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain

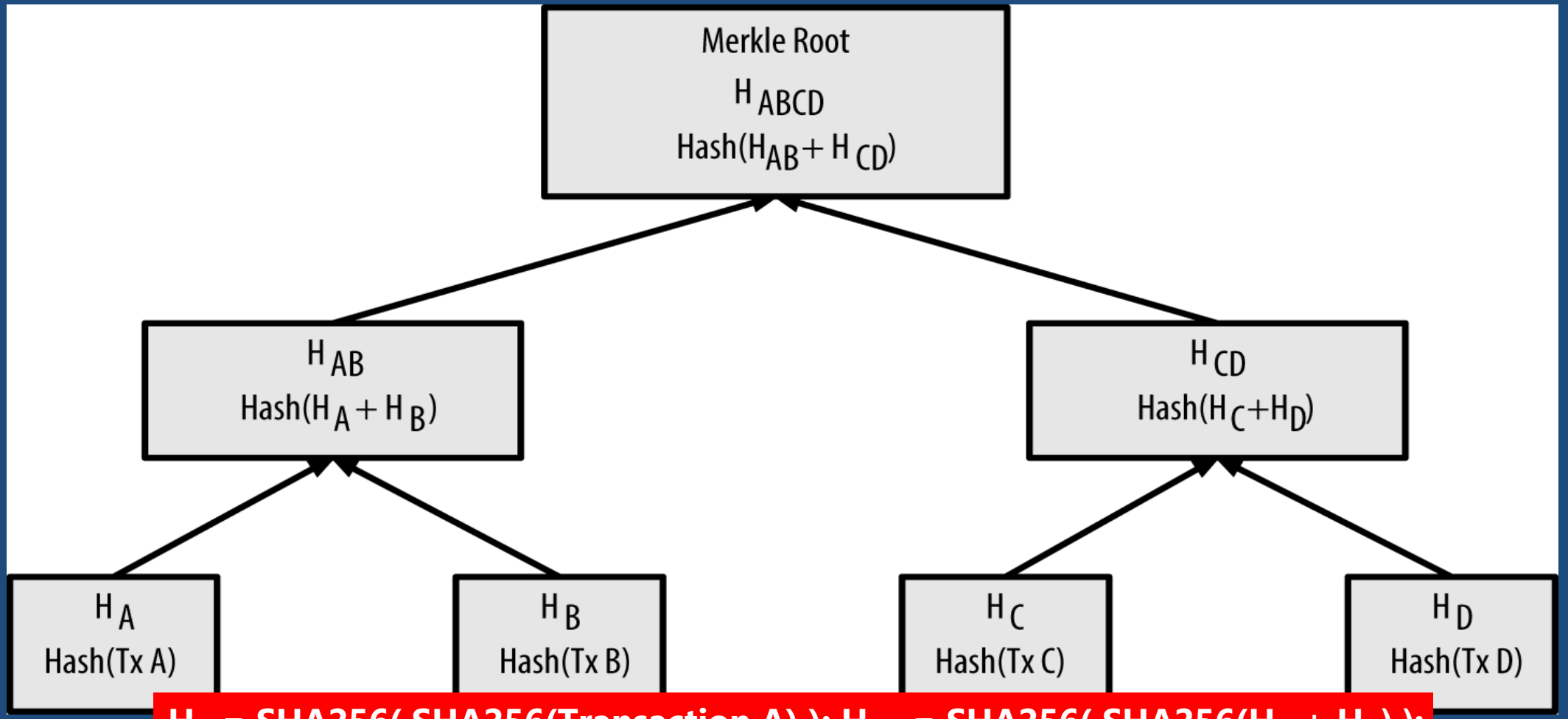


2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain



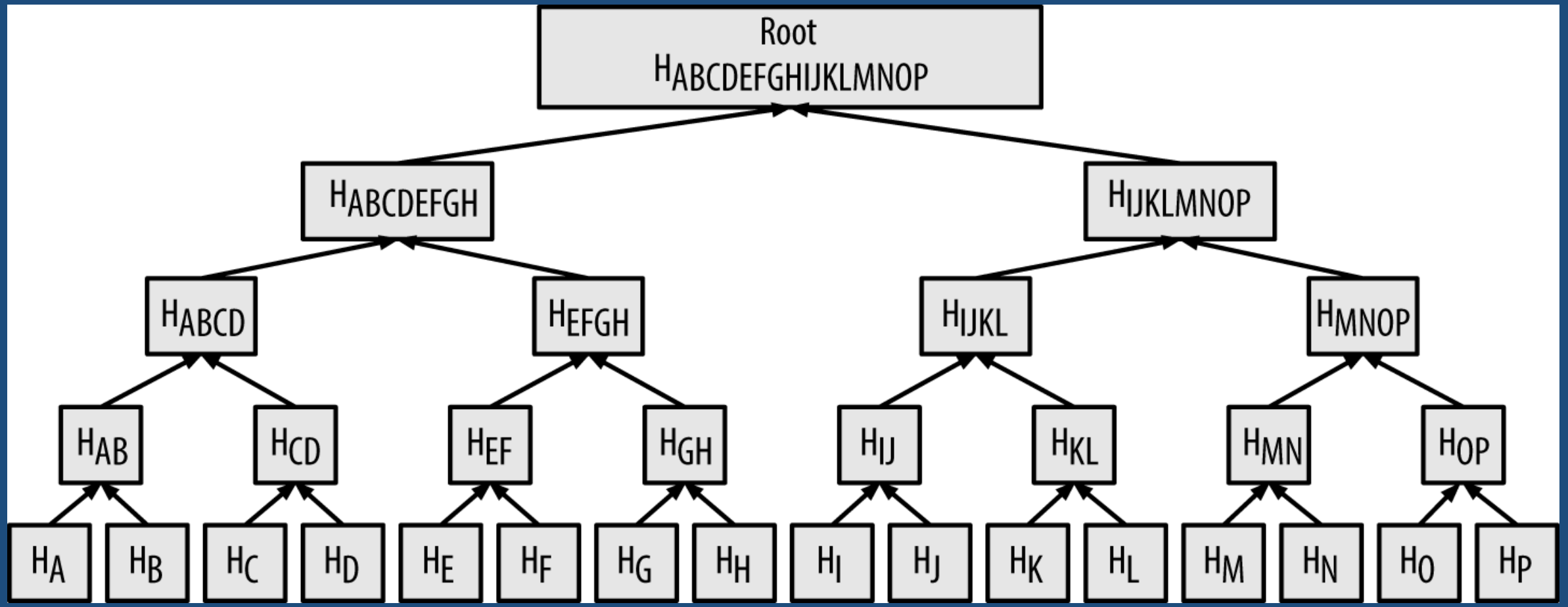
2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain _ Transactions Hashes in Merkle Trees

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:



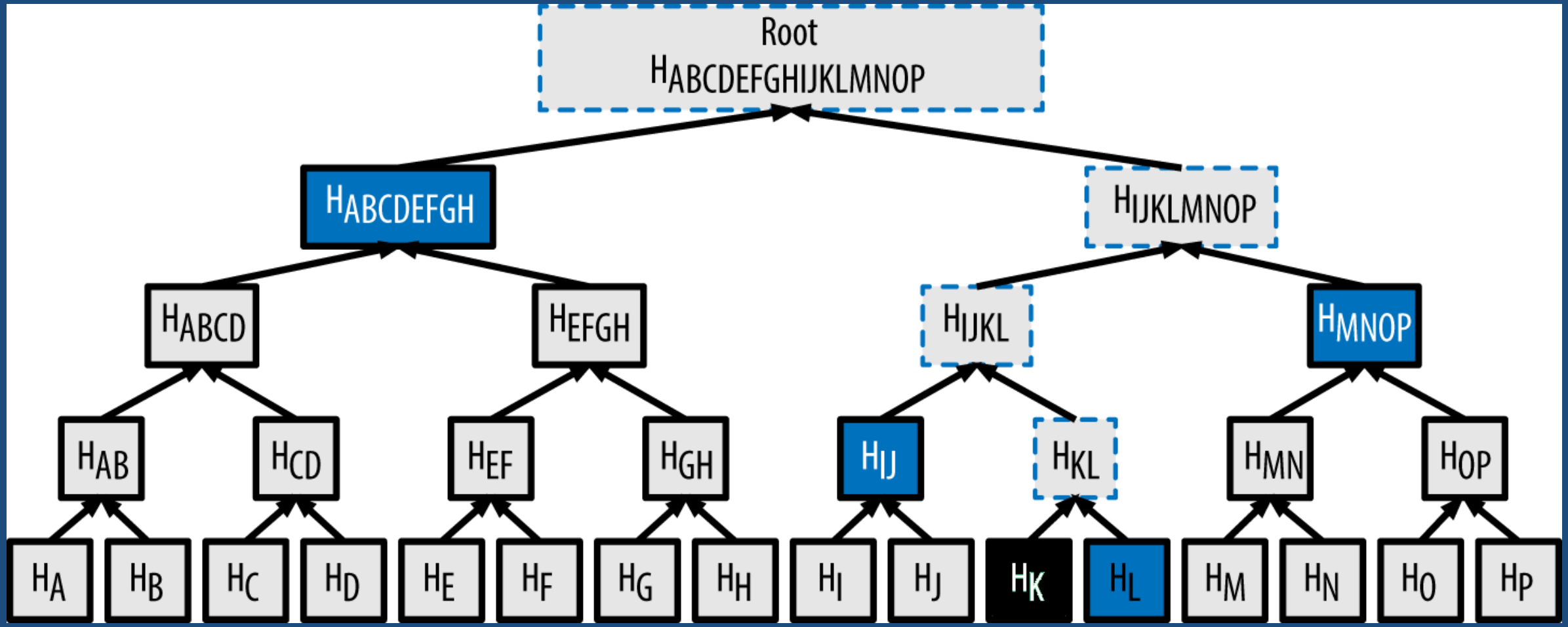
H_A = SHA256(SHA256(Transaction A)); H_{AB} = SHA256(SHA256(H_A + H_B));

2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain _ Transactions Hashes in Merkle Trees



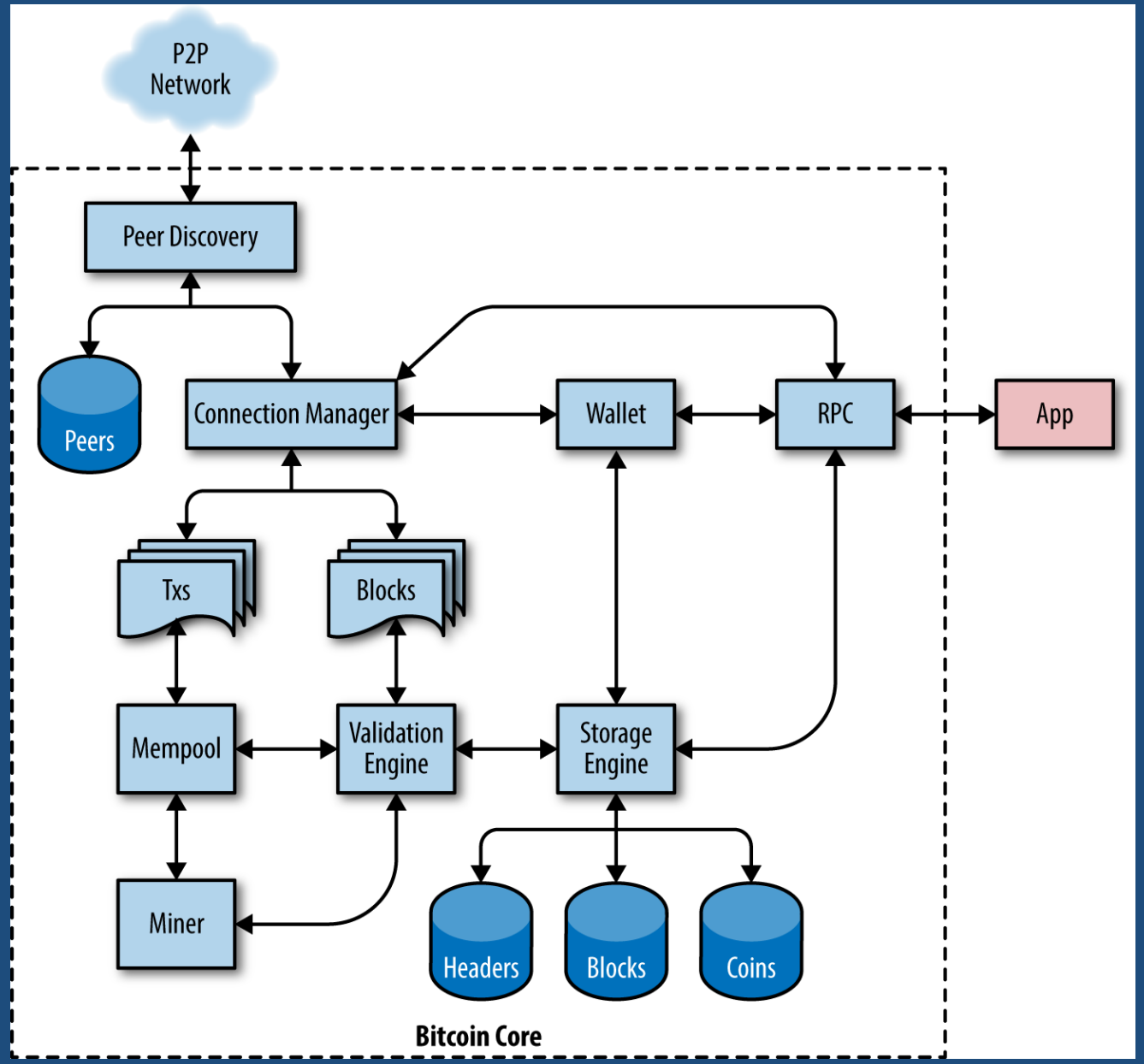
This tree is built from hashes for 16 transactions. Although the root looks bigger than the leaf nodes in the diagram, it is the exact same size, just 32 bytes.

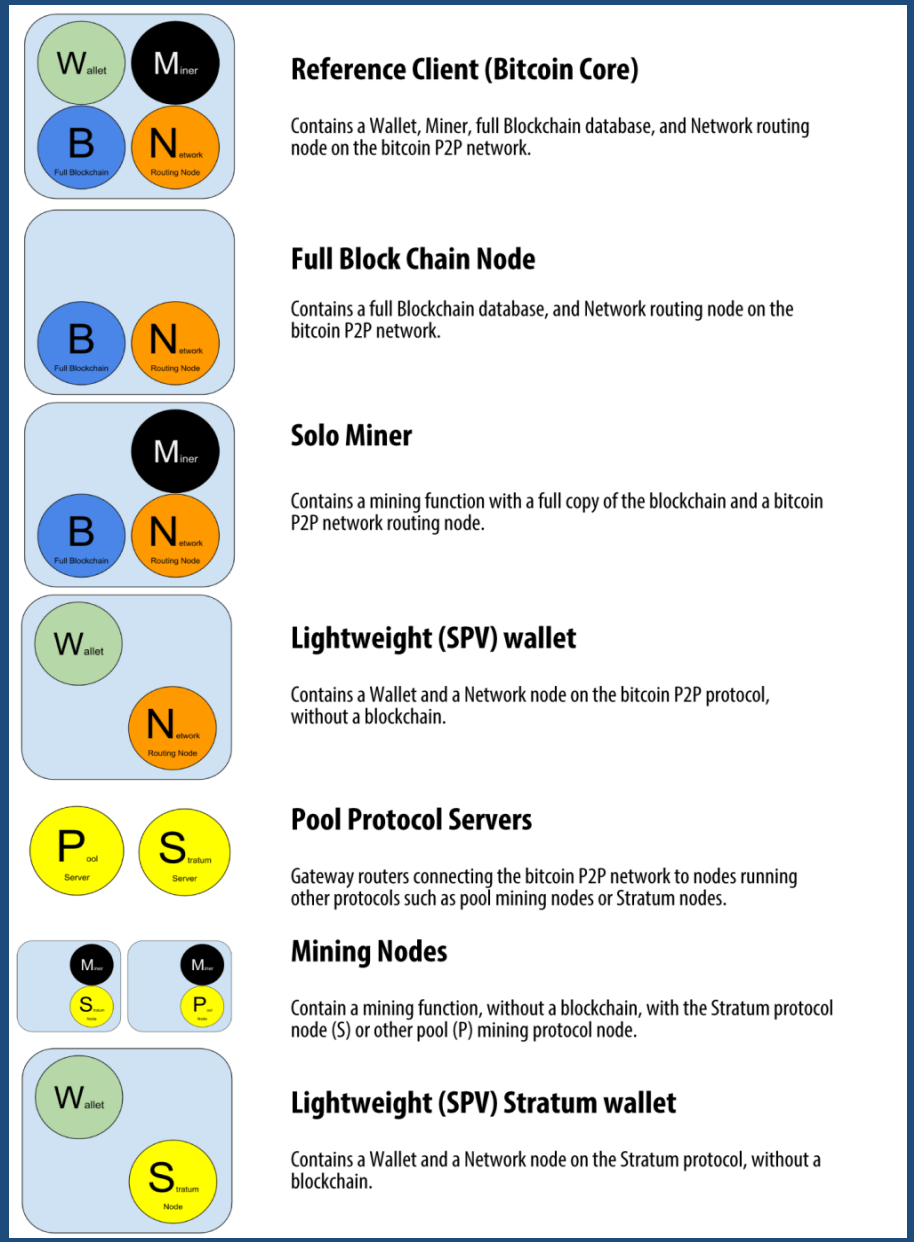
2.1 Bitcoin / Ethereum Blockchain Technology _ Blockchain _ Transactions Hashes in Merkle Trees



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 $H_{ABCDEFGH}$. 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 pair-wise hashes H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, and the Merkle tree root (outlined in a dashed line in the diagram).

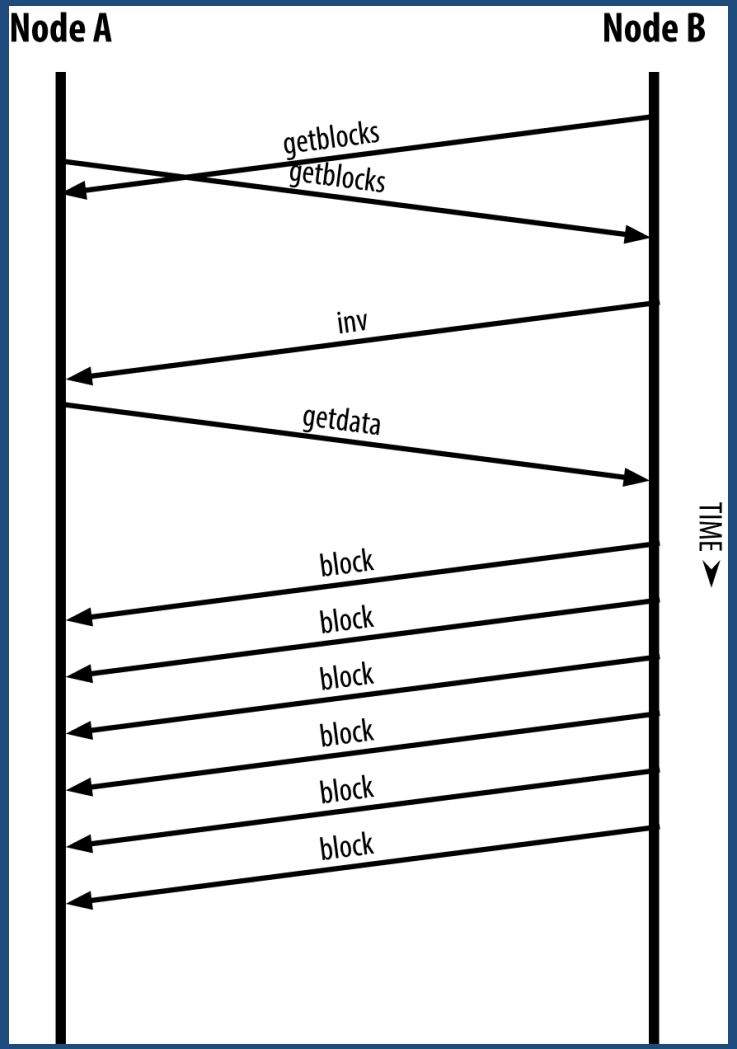
2.1 Bitcoin / Ethereum Blockchain Technology _ Reference Implementation

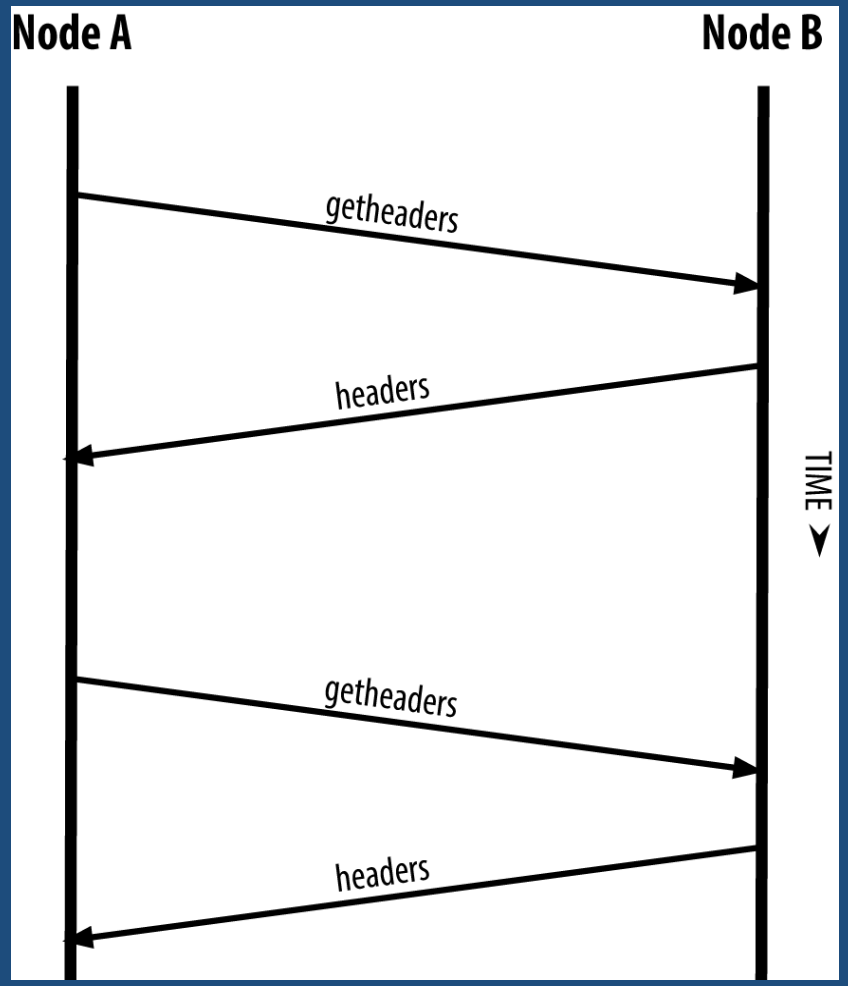




2.1 Bitcoin / Ethereum Blockchain Technology _ P2P Network

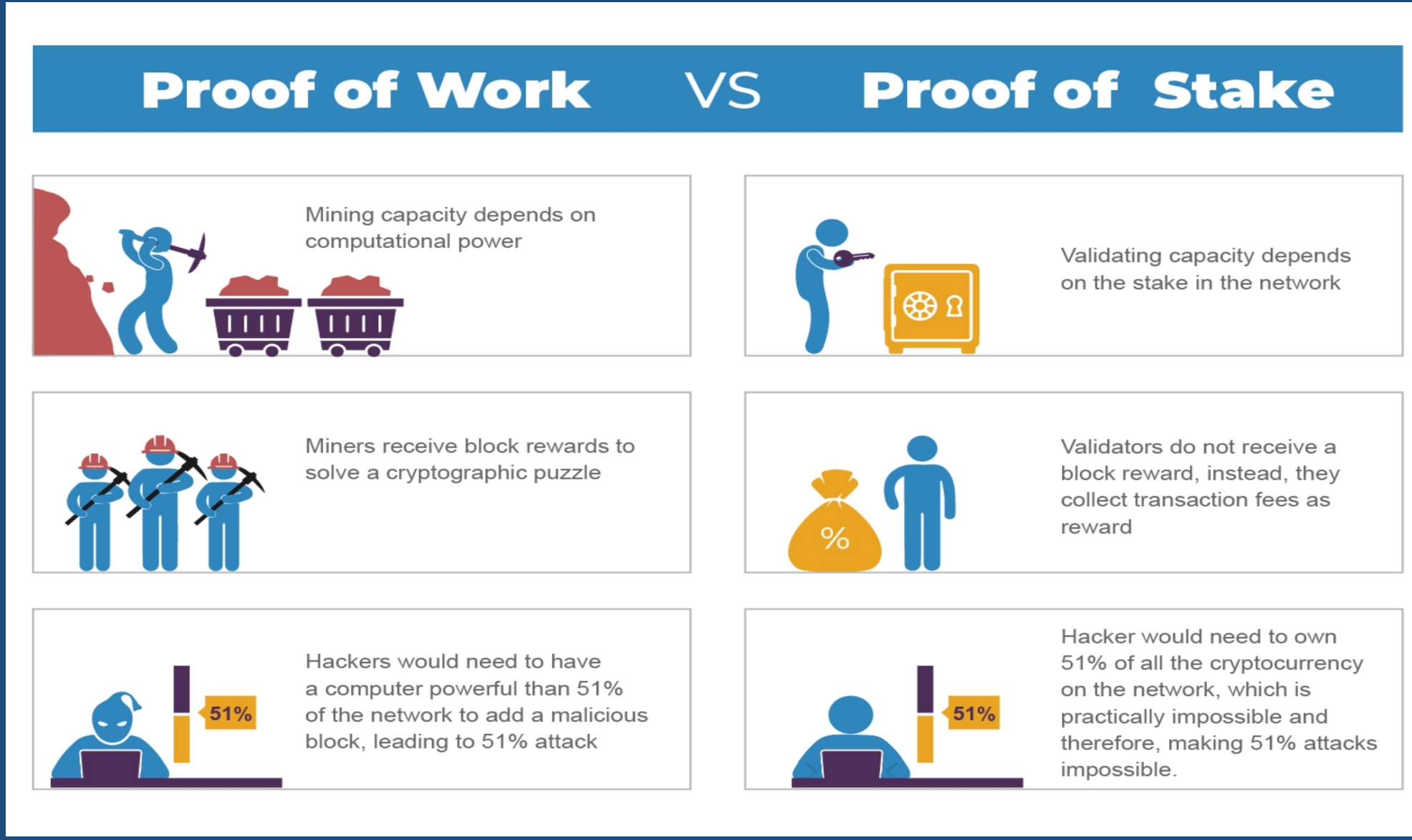
Node synchronizing the blockchain by retrieving blocks from a peer





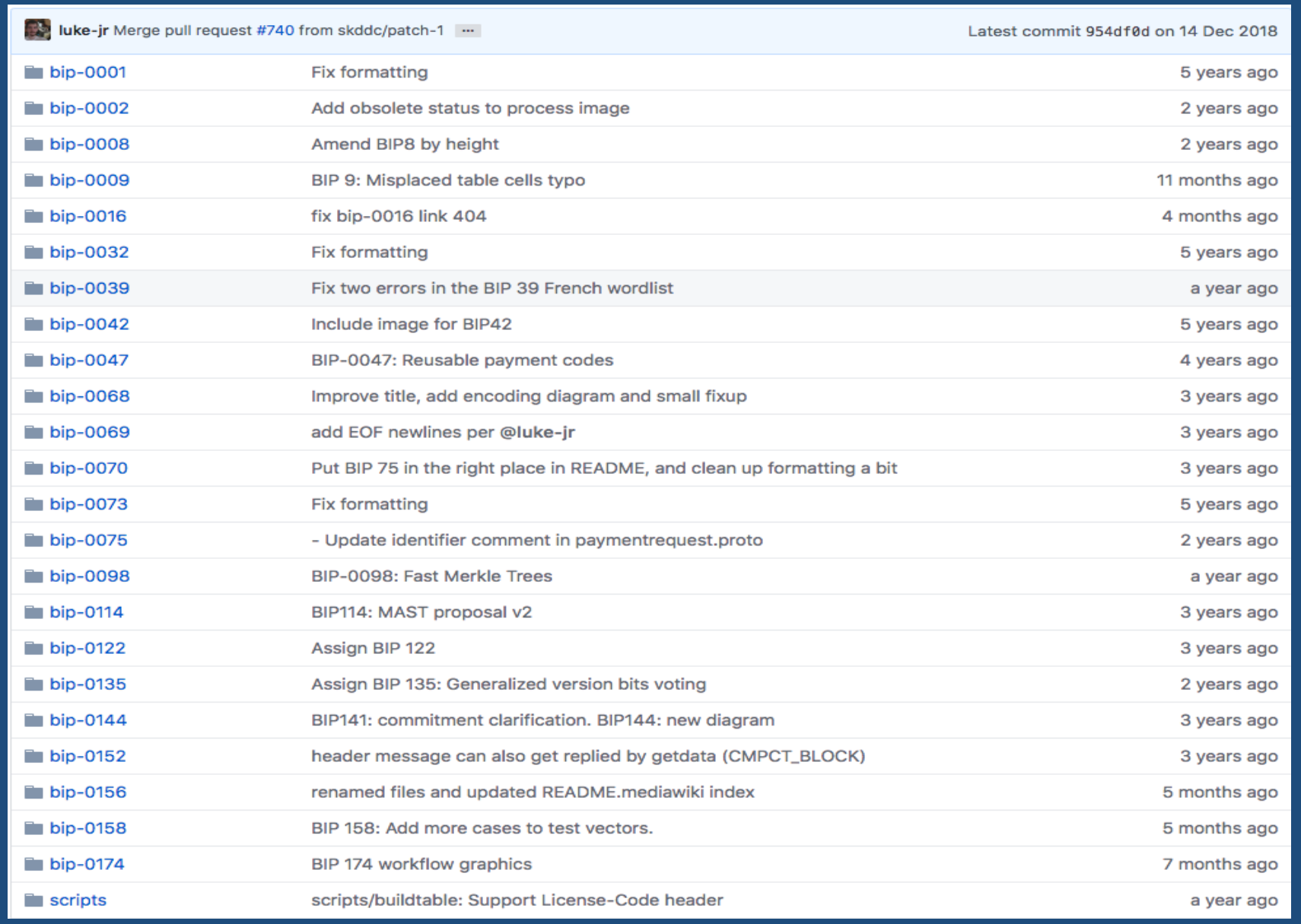
SPV node synchronizing the block headers

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

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

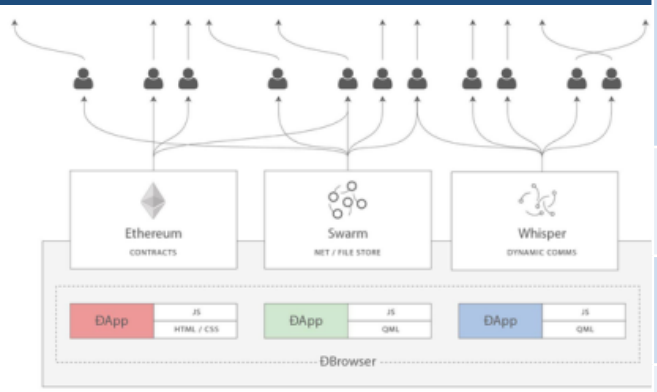
• A **consensus algorithm** that decentralizes control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules

• 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
P2P Network	Ethereum runs on the *Ethereum main network*, which is addressable on TCP port 30303, and runs a protocol called *ĐΞVp2p*. The dynamic communication between the peers is implemented with *Whisper*. **Whisper** is a part of the **Ethereum** P2P protocol suite that allows for messaging between users via the same network that the blockchain runs on.
Consensus rules	Ethereum's consensus rules are defined in the reference specification, the Yellow Paper - https://ethereum.github.io/yellowpaper/paper.pdf
Transactions	Ethereum transactions are network messages that include (among other things) a sender, recipient, value, and data payload.
State machine	Ethereum state transitions are processed by the ***Ethereum Virtual Machine* (EVM),** a stack-based virtual machine that executes *bytecode* (machine-language instructions). **EVM programs**, called "**smart contracts**," are written in high-level languages (e.g., Solidity or Vyper) and compiled to bytecode for execution on the EVM.
Data structures	Ethereum's state is stored locally on each node as a *database* (usually Google's LevelDB), which contains the transactions and system state in a serialized hashed data structure called a *Merkle Patricia Tree*. **Swarm** or IPFS may be used to store the Solidy smart contracts files.
Consensus algorithm	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 **Casper**, in the near future.
Economic security	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.
Clients	Ethereum has several interoperable implementations of the client software, the most prominent of which are Go-Ethereum (Geth) and Parity.

#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:**
  - ĐΞVp2p 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

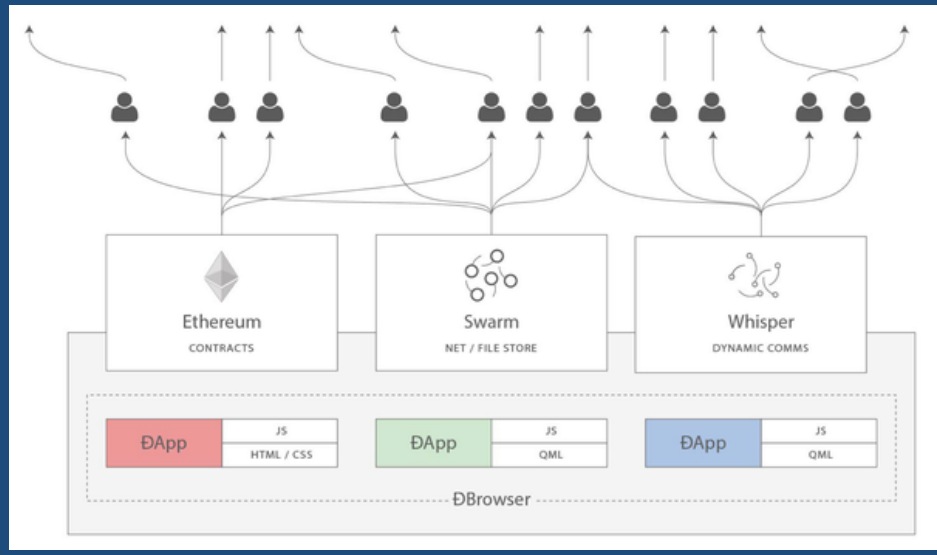
From General-Purpose Blockchains to Decentralized Applications (Dapps / *ĐApps*)

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
Main Ethereum Network	The main public Ethereum blockchain. Real ETH, real value, and real consequences.
Ropsten Test Network	Ethereum public test blockchain and network. ETH on this network has no value.
Kovan Test Network	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.
Rinkeby Test Network	Ethereum public test blockchain and network, using the Clique consensus protocol with proof of authority (federated signing). ETH on this network has no value.
Localhost 8545	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.
Custom RPC	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

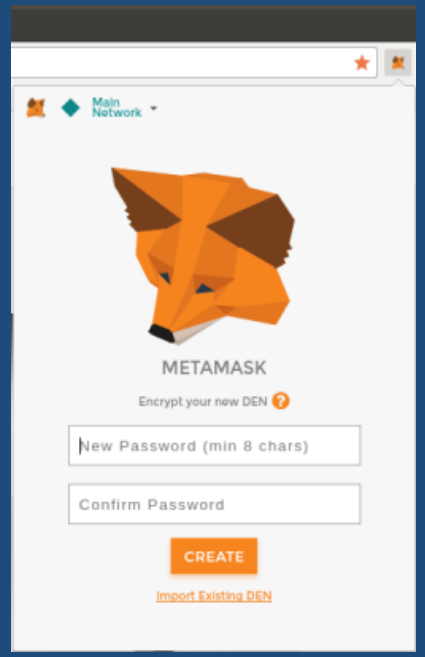
	Ethereum Wallets
MetaMask https://metamask.io/	It is a browser extension wallet that runs in your browser (Chrome, Firefox, Opera, or Brave Browser). It is easy to use and convenient for testing, as it is able to connect to a variety of Ethereum nodes and test blockchains. MetaMask is a web-based wallet.
Jaxx	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.
MyEtherWallet (MEW)	It is a web-based wallet that runs in any browser. It has multiple sophisticated features we will explore in many of our examples.
Emerald Wallet	It is designed to work with the Ethereum Classic blockchain, but is compatible with other Ethereum-based blockchains. It's an open source desktop application and works under Windows, macOS, and Linux. Emerald Wallet can run a full node or connect to a public remote node, working in a "light" mode. It also has a companion tool to do all operations from the command line.

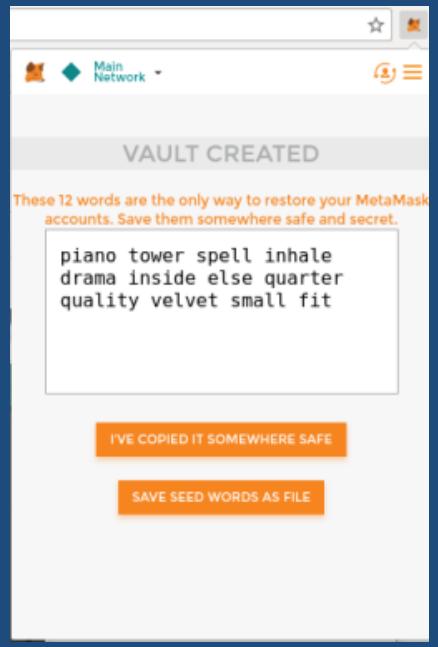
Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000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

reum", Publisher: O'Reilly Media, Inc., Release Date: November 2018, ISBN: ISBN: 9781491971949

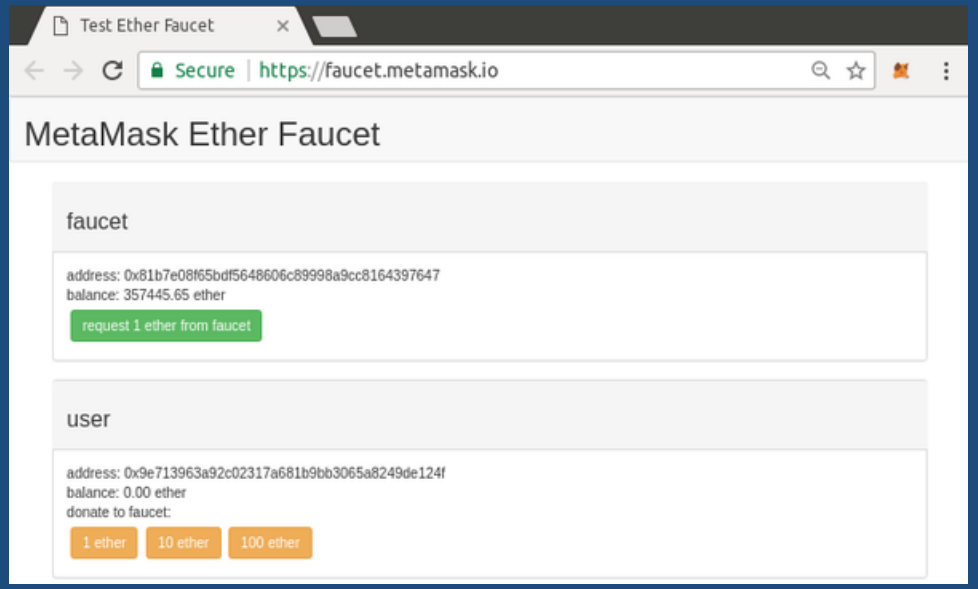
#Ethereum – DEMO with MetaMask and Remix IDE in Mozilla Firefox

D1. MetaMask Password and Mnemonic setup

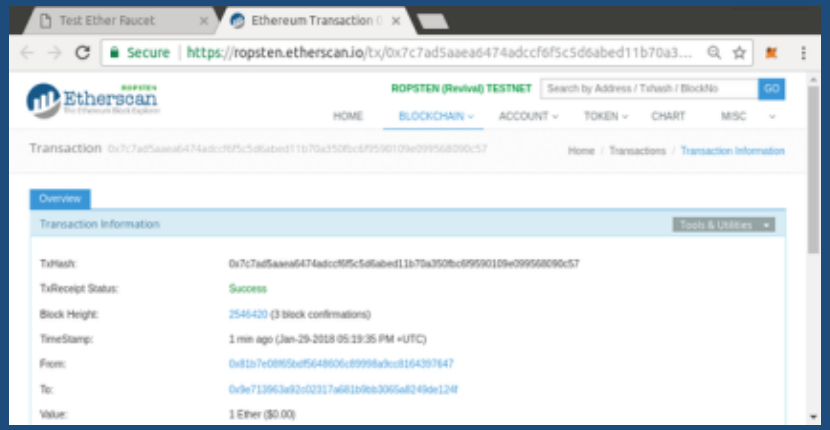




D2. MetaMask buy Ether from Ropsten Test Network's Faucet



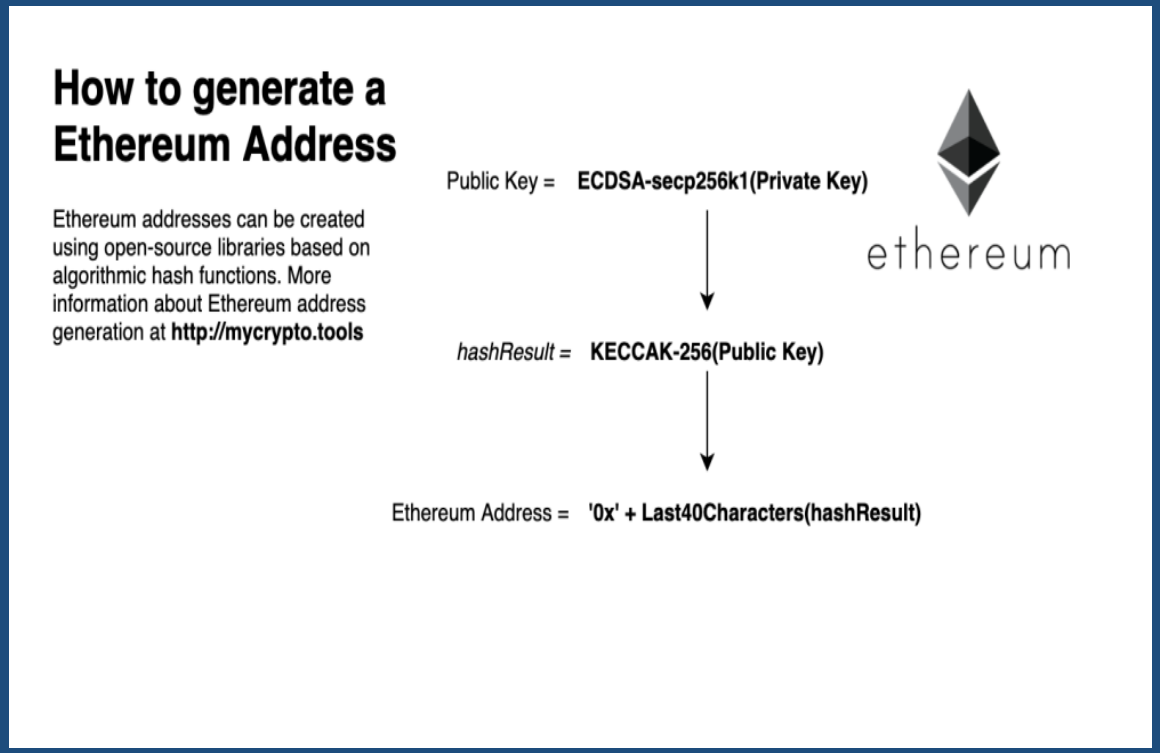
D3. Transaction in RopstenEtherscan.io



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.

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



1 - Generate Private Key using open source libraries such as **Ethereumj** or SHA256 hash function with a randomly generated number.
0: Private Key:
09e910621c2e988e9f7f6ffcd7024f54ec1461fa6e86a4b545e9e1fe21c28866

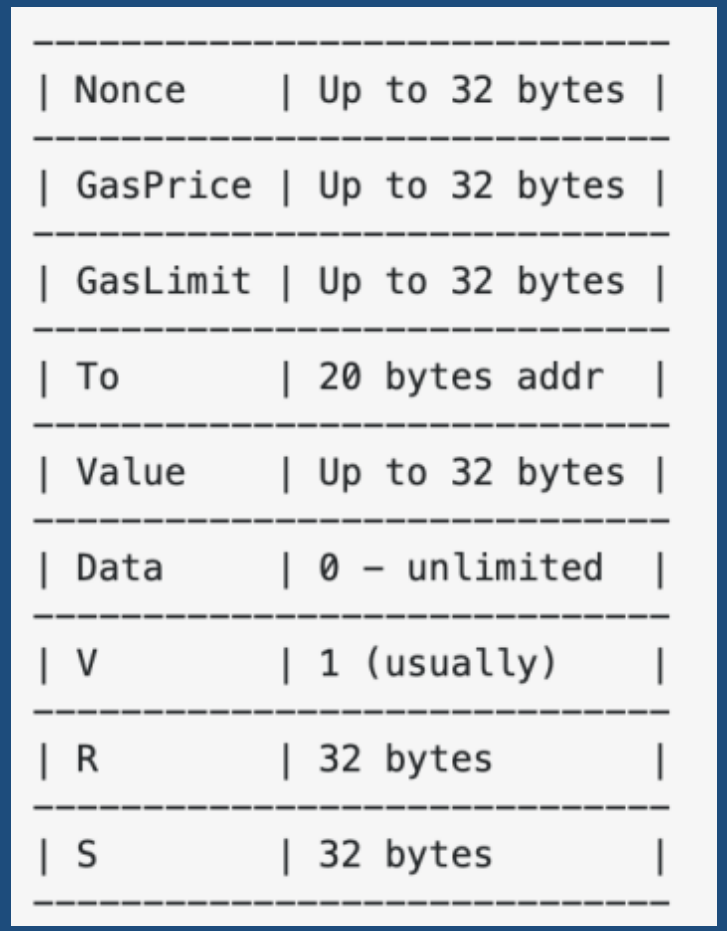
2. - Generate Public key using ECDSA – secp256k1 – algorithm applied to Private Key. The public key is a point on the Elliptic Curve Algorithm. It has x and y coordinates which are used to create Ethereum address.
2.1: Public Key:
048e66b3e549818ea2cb354fb70749f6c8de8fa484f7530fc447d5fe80a1c424e4f5ae648d648c980ae7095d1efad87161d83886ca4b6c498ac22a93da5099014a

-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



Nonce	Up to 32 bytes
GasPrice	Up to 32 bytes
GasLimit	Up to 32 bytes
To	20 bytes addr
Value	Up to 32 bytes
Data	0 – unlimited
V	1 (usually)
R	32 bytes
S	32 bytes

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

eth.getTransaction('0x14a298c1eea89f42285948b7d51eeac2876ca7406c9784b9b90dd3591d156d64').raw

This should give

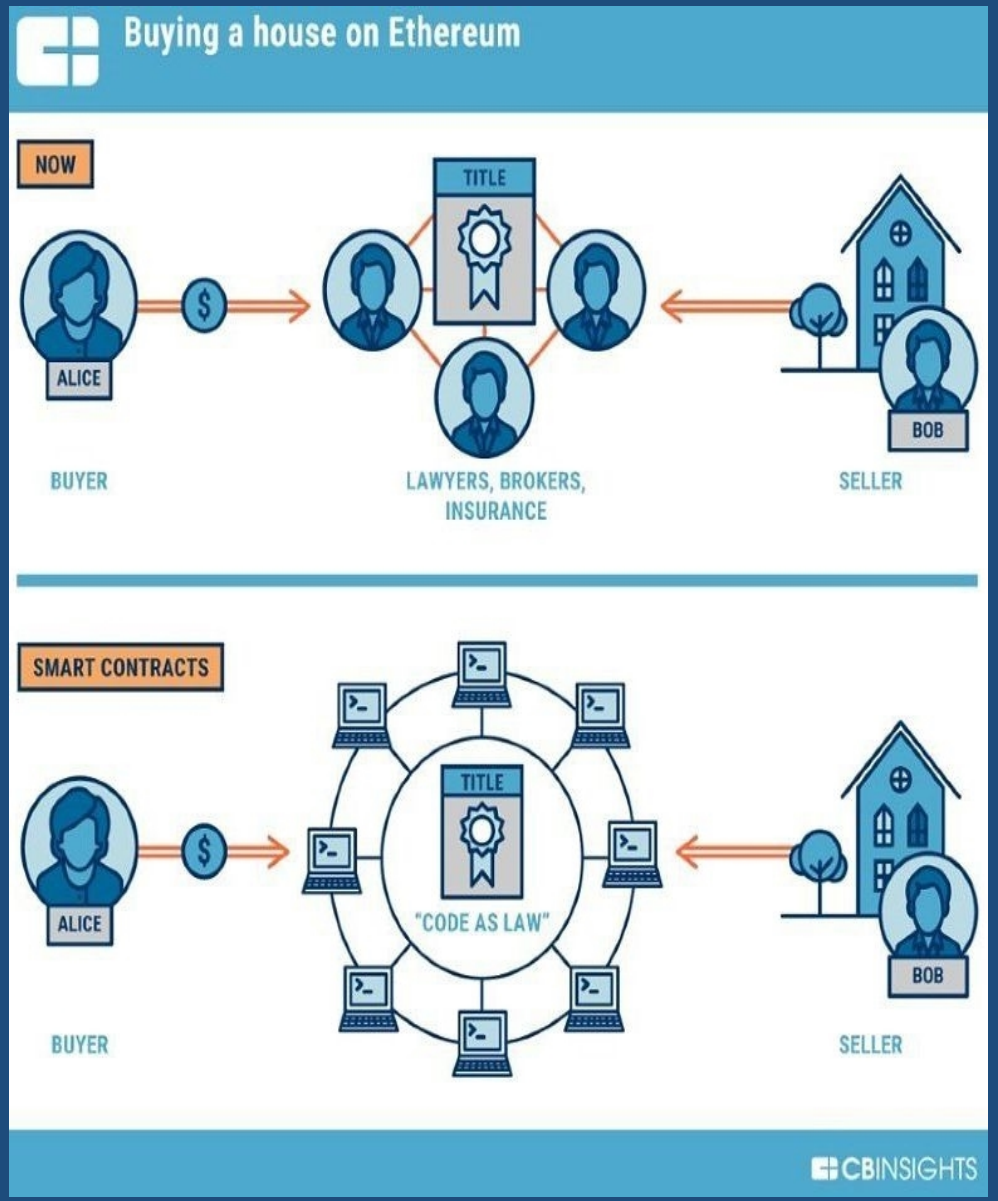
"0xf86b80850ba43b74008252089479 17bc33eea648809c285607579c9919fb864f8f8703baf82d03a0008025a0067940651530790861714b2e8fd8b080361d1ada048189000c07a66848afde46a069b041db7c29dbcc6becf42017ca7ac086b12bd53ec8ee494596f790fb6a0a69"

which is 109 bytes. If we parse the data:

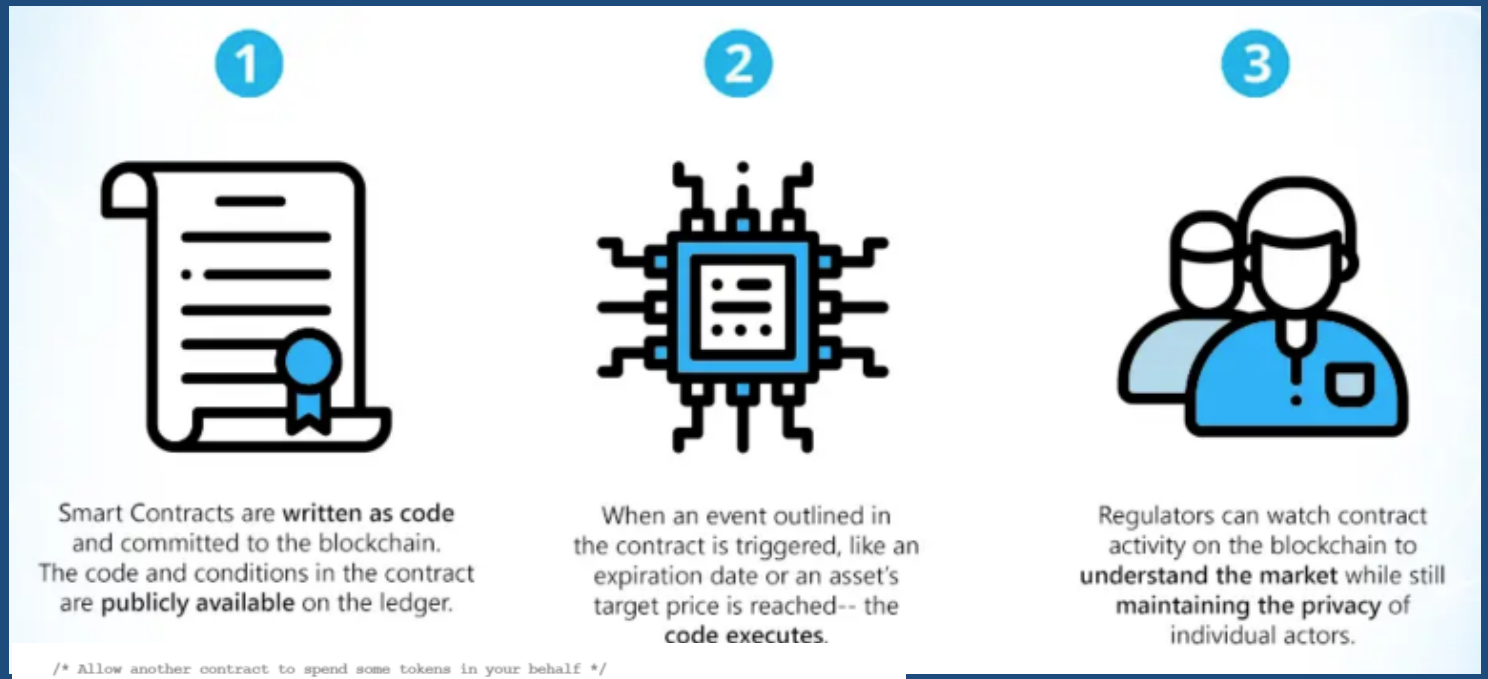
f86b length
80 nonce (0: this is the minimum an account can have)
85 0ba43b7400 gas price
82 5208 gas limit (this is fixed for simple payments)
94 7917bc33eea648809c285607579c9919fb864f8f (address, always 20 bytes)
87 03baf82d03a000 (value, in theory this can be shrunken to zero)
80 (data, already zero length)
25 (V, one byte)
a0 0679406515307908617 14b2e8fd8b080361d1ada048189000c07a66848afde46 (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/




```
/* Allow another contract to spend some tokens in your behalf */
function approve(address _spender, uint256 _value)
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    return true;
}

/* Approve and then comunicate the approved contract in a single tx */
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
        return true;
    }
}

/* A contract attempts to get the coins */
function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    if (balanceOf[_from] < _value) throw;                 // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Check for overflows
    if (_value > allowance[_from][msg.sender]) throw;   // Check allowance
    balanceOf[_from] -= _value;                           // Subtract from the sender
    balanceOf[_to] += _value;                             // Add the same to the recipient
    allowance[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
    return true;
}

/* This unnamed function is called whenever someone tries to send ether to it */
function () {
    throw;     // Prevents accidental sending of ether
}
```

Copyright: https://blockgeeks.com/guides/smart-contracts/

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

1 Autonomy

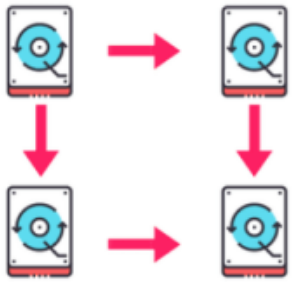
You're the one making the agreement; there's no need to rely on a broker or lawyer

2 Trust

Your documents are encrypted on a shared ledger

3 Backup

On the blockchain,Your documents are duplicated many times over



4 Savings

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

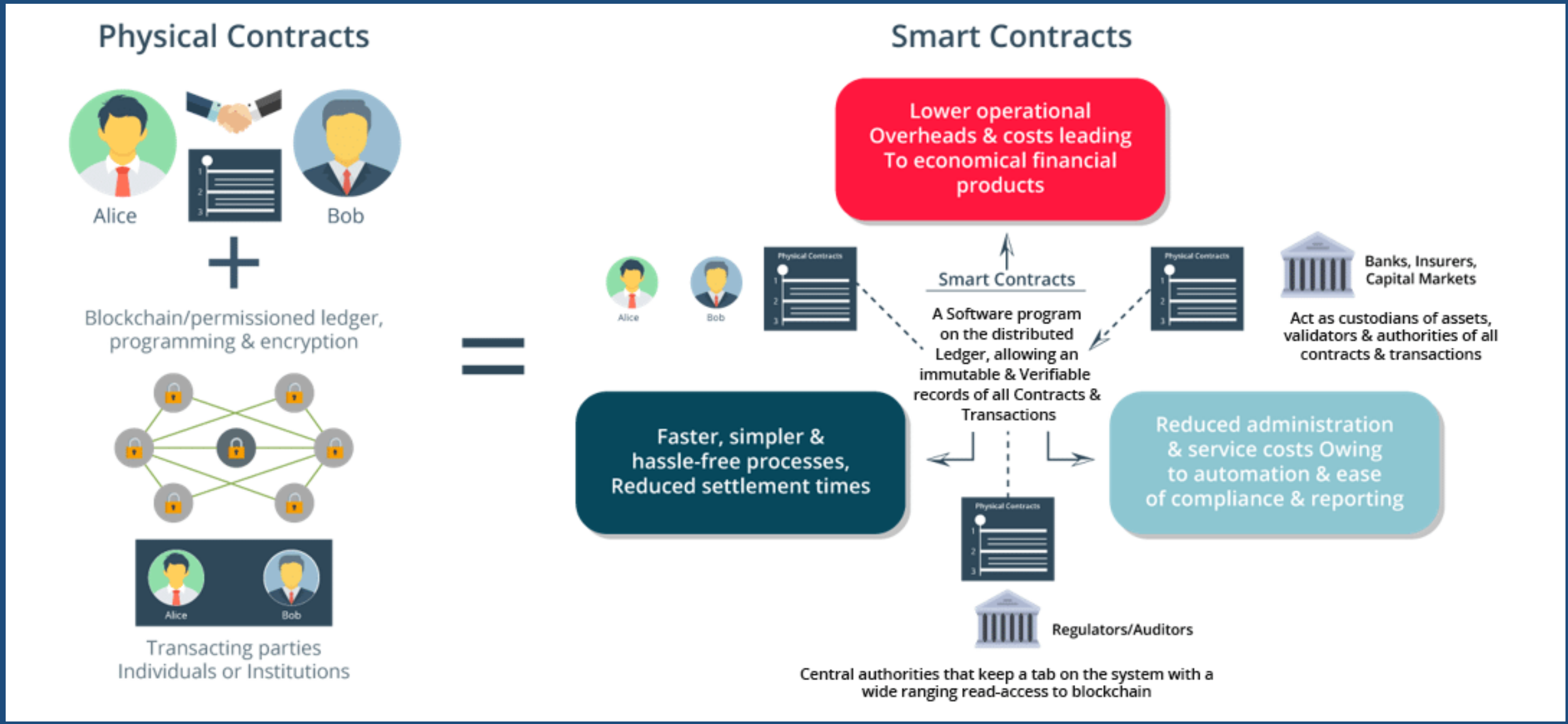
5 Accuracy

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

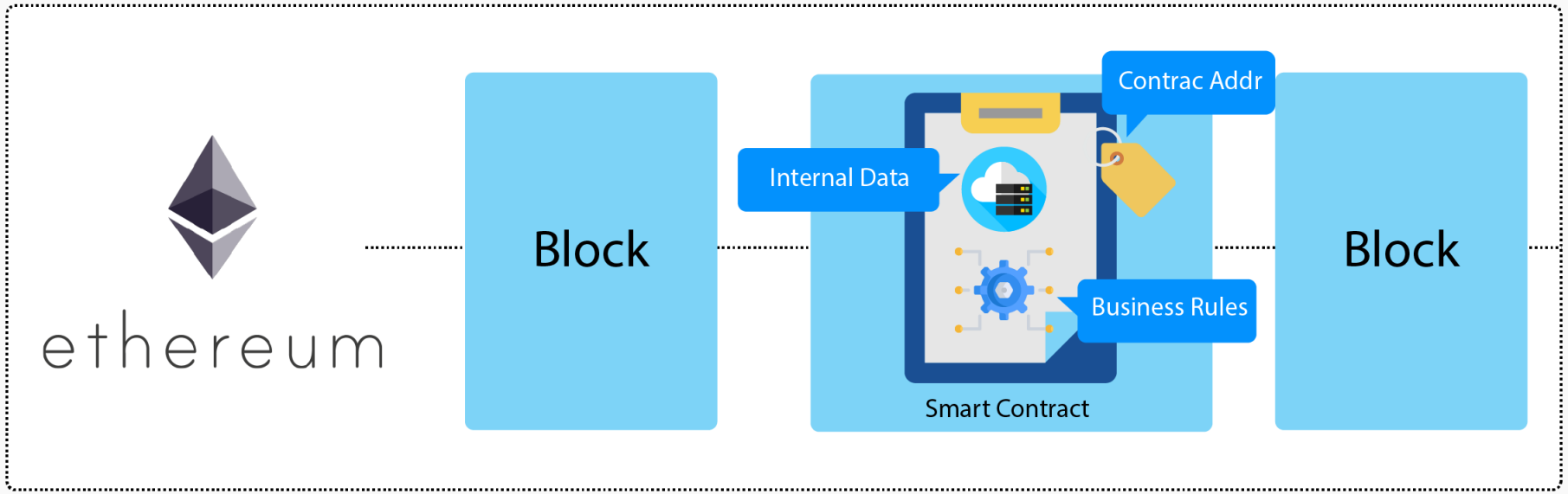
www.Blockgeeks.com

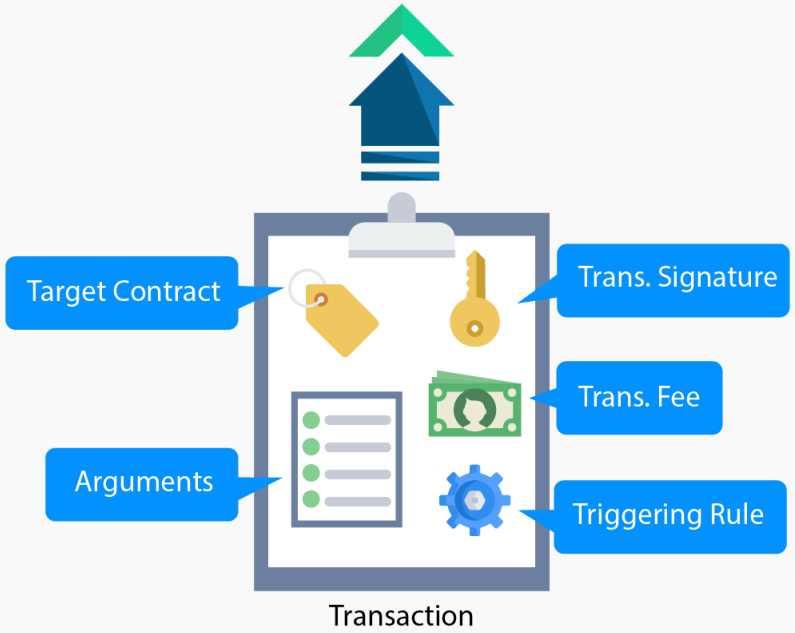
Blockgeeks

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



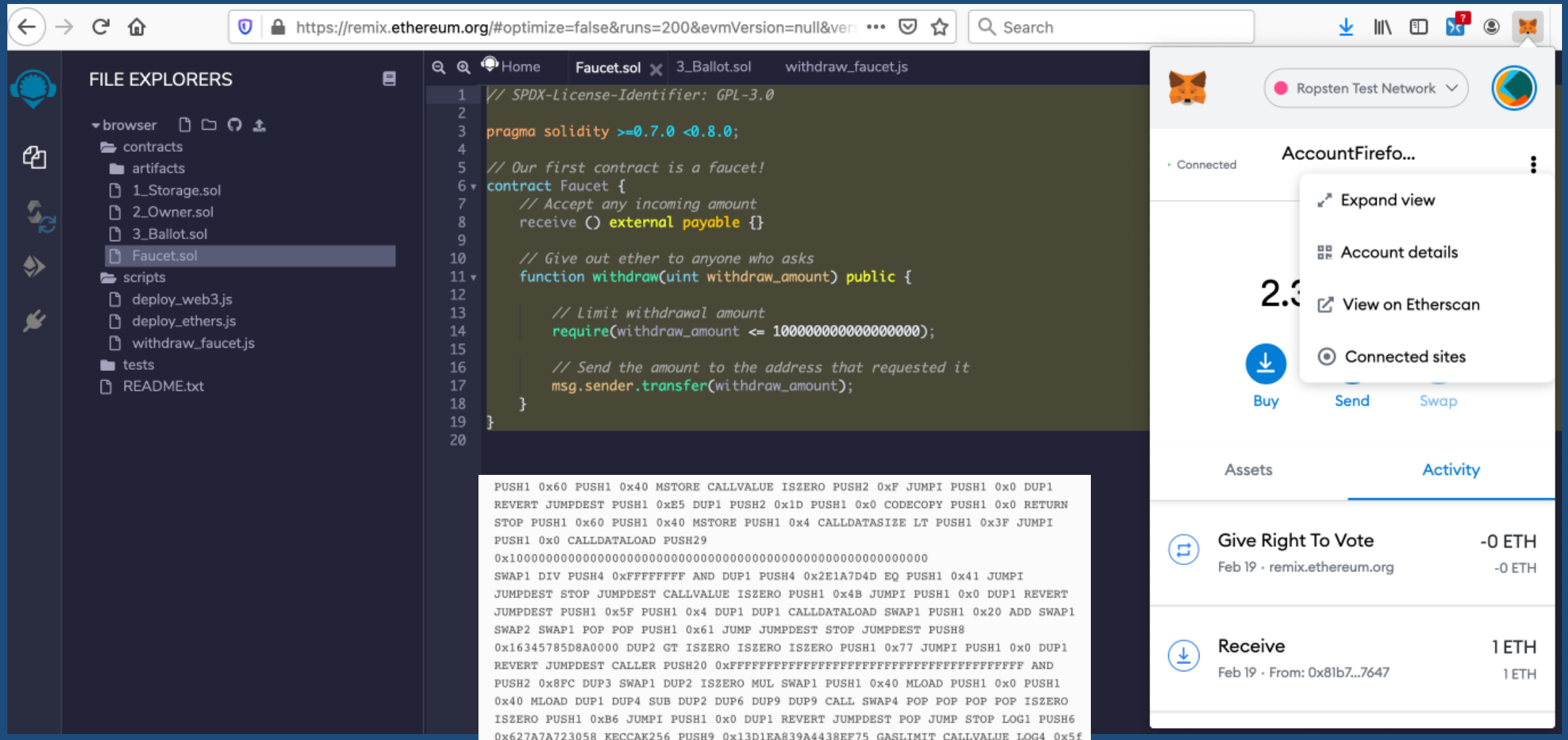
#Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?





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

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



#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.

 This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

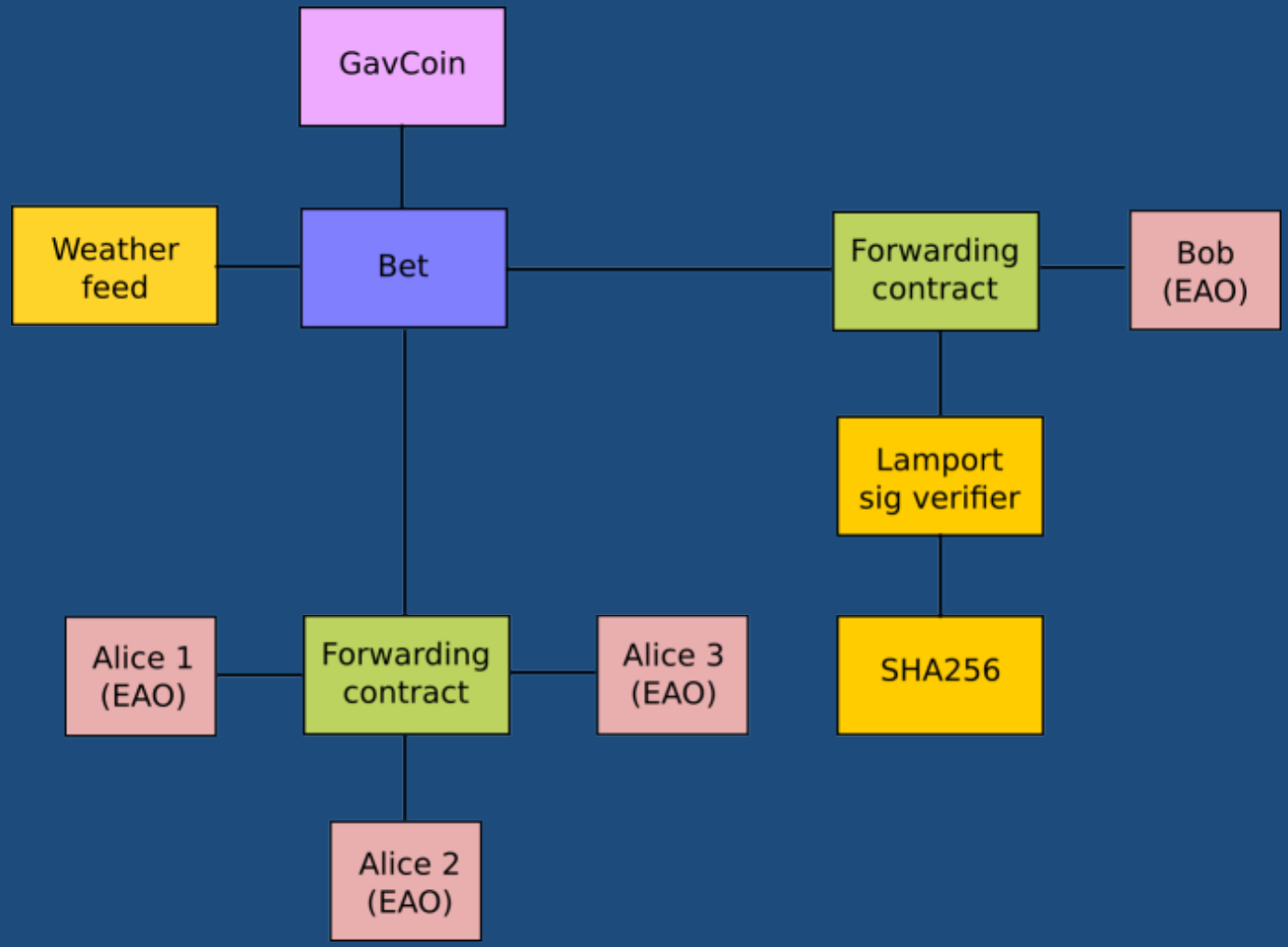
#Ethereum Blockchain Wiki Dev Tutorial – How Smart Contract is working?

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).



 This post was originally written by Vitalik Buterin in 2014, so is more for historical record than something to follow now.

#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.

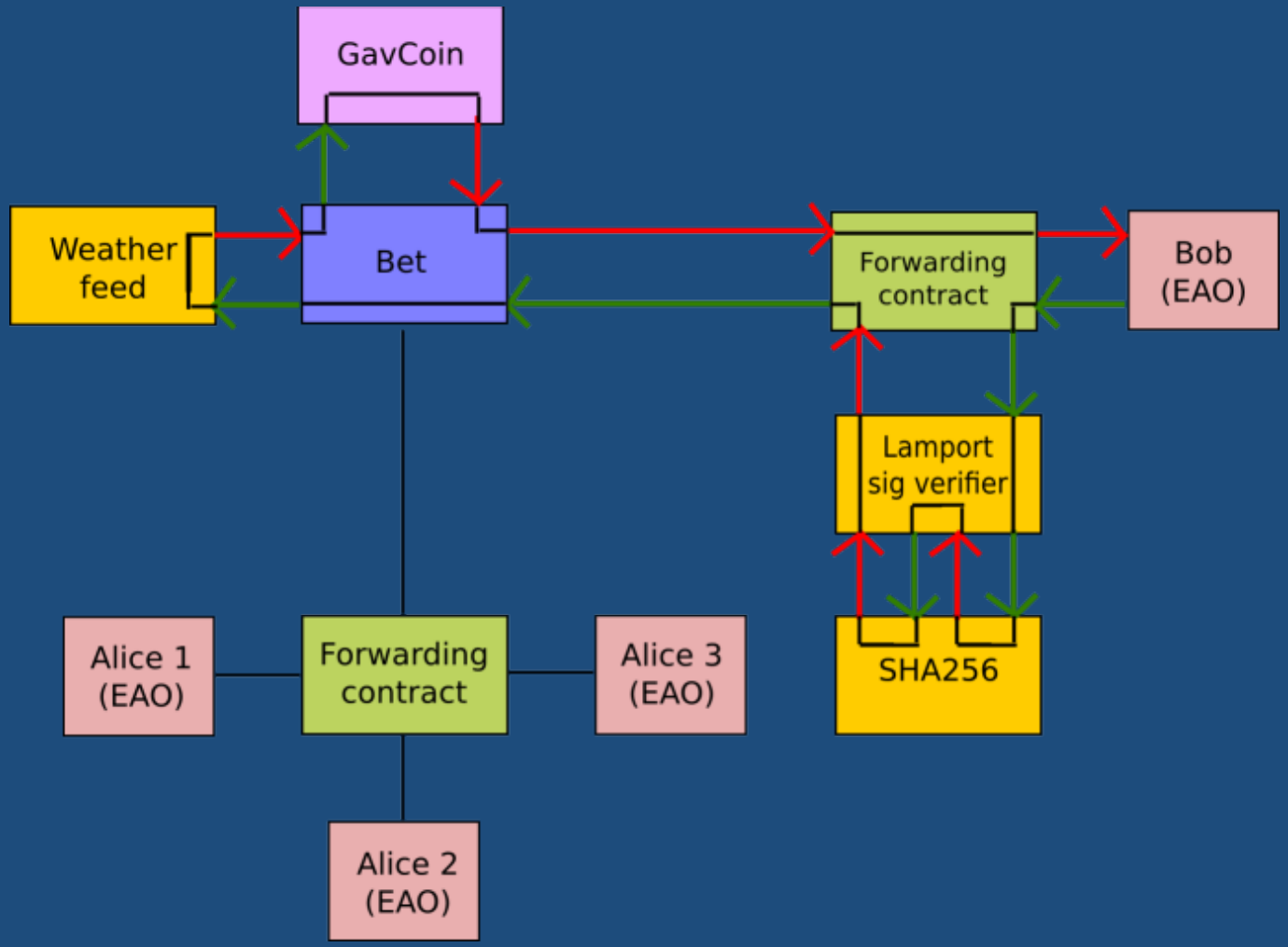
3. The Lamport signature verification library sees that Bob wants a SHA256-based Lamport sig, so it calls the SHA256 library many times as needed to verify the signature.

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 35ºC, so it sends a message to the GavCoin contract to move the GavCoin from its account to Bob's forwarding contract.

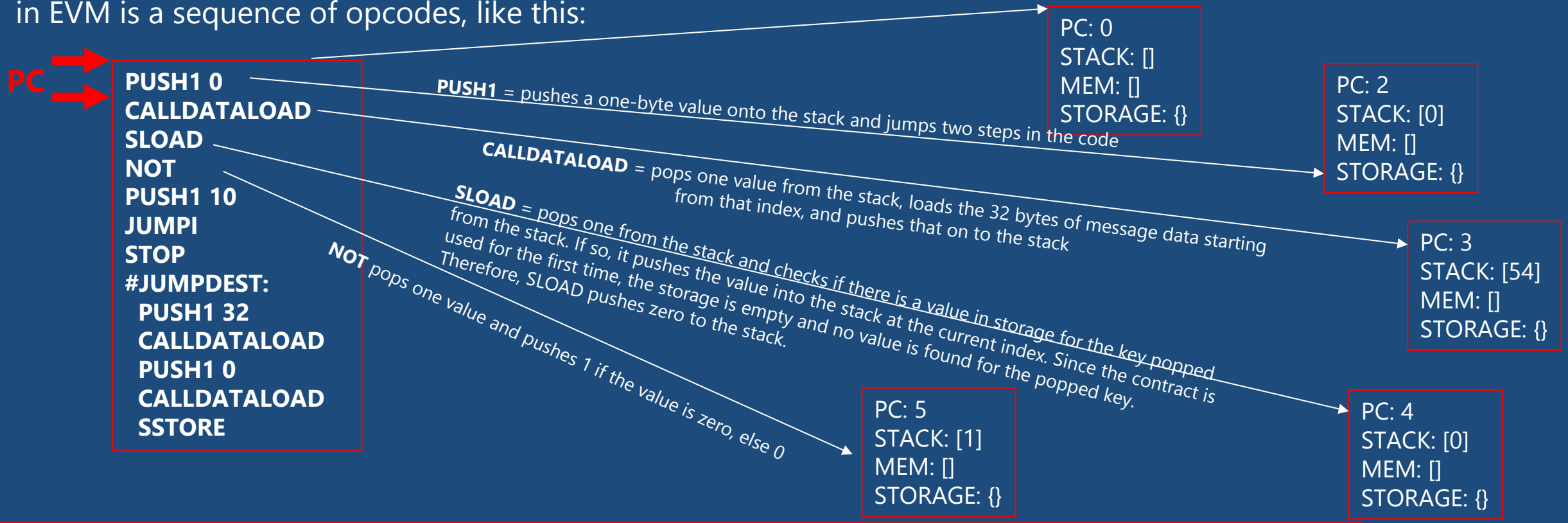
* Note that the GavCoin is all "stored" as entries in the GavCoin contract's database; the word "account" in the context of step 6 simply means that there is a data entry in the GavCoin contract storage with a key for the bet contract's address and a value for its balance. After receiving this message,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



 This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

#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}

 This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

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

{K = 54, V = 2020202020}

```
PC →
PUSH1 0
CALLDATALOAD
SLOAD
NOT
PUSH1 10
JUMPI
STOP
#JUMPDEST:
  PUSH1 32
  CALLDATALOAD
  PUSH1 0
  CALLDATALOAD
  SSTORE
```

PC: 7
STACK: [1, 10]
MEM: []
STORAGE: {}

PC: 9
STACK: []
MEM: []
STORAGE: {}

JUMPI = instruction pops 2 values and jumps to the instruction designated by the first only if the second is nonzero. Here, the second is nonzero, so we jump.

PC: 11
STACK: [32]
MEM: []
STORAGE: {}

PC: 13
STACK: [2020202020]
MEM: []
STORAGE: {}

CALLDATALOAD = again, popping 32 and pushing the bytes in message data starting from byte 32 until byte 63

PC: 15
STACK: [2020202020, 0]
MEM: []
STORAGE: {}

Load message data bytes 0-31 again (loading message data is just as cheap as loading memory, so we don't bother to save it in memory)

PC: 16
STACK: [2020202020, 54]
MEM: []
STORAGE: {}

SSTORE = save the value 2020202020 in storage at index 54

PC: 17
STACK: []
MEM: []
STORAGE: {54: 2020202020}

After index 17, there is no instruction, so we stop. If there was anything left in the stack or memory, it would be deleted, but the storage will stay and be available next time someone sends a message. Thus, if the sender of this message sends the same message again (or perhaps someone else tries to reregister 54 to 3030303030), the next time the JUMPI at position 8 would not process, and execution would STOP early at position 9.

This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

#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.

 This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

#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.

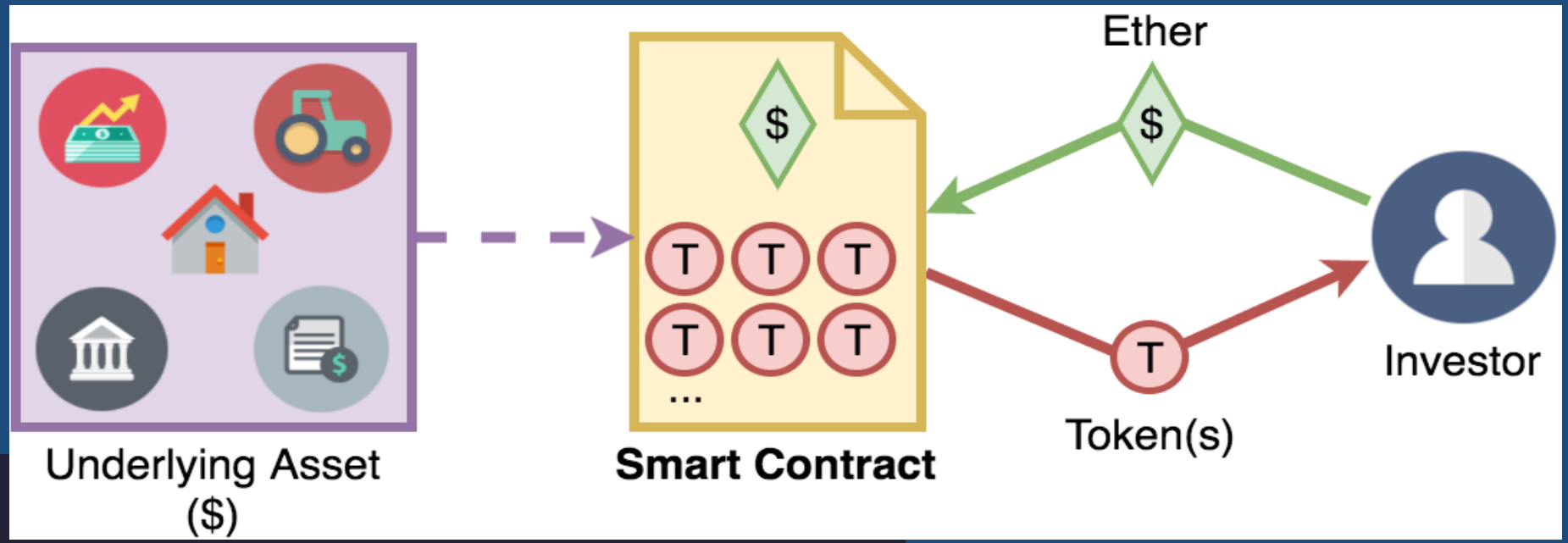
The way the fee works is as follows. Every transaction must contain, alongside its other data, a GASPRICE and STARTGAS value. STARTGAS is the amount of "gas" that the transaction assigns itself, and GASPRICE is the fee that the transaction pays per unit of gas; thus, when a transaction is sent, the first 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.

 This post was originally written by **Vitalik Buterin** in 2014, so is more for historical record than something to follow now.

#Ethereum How Smart Contract is working?

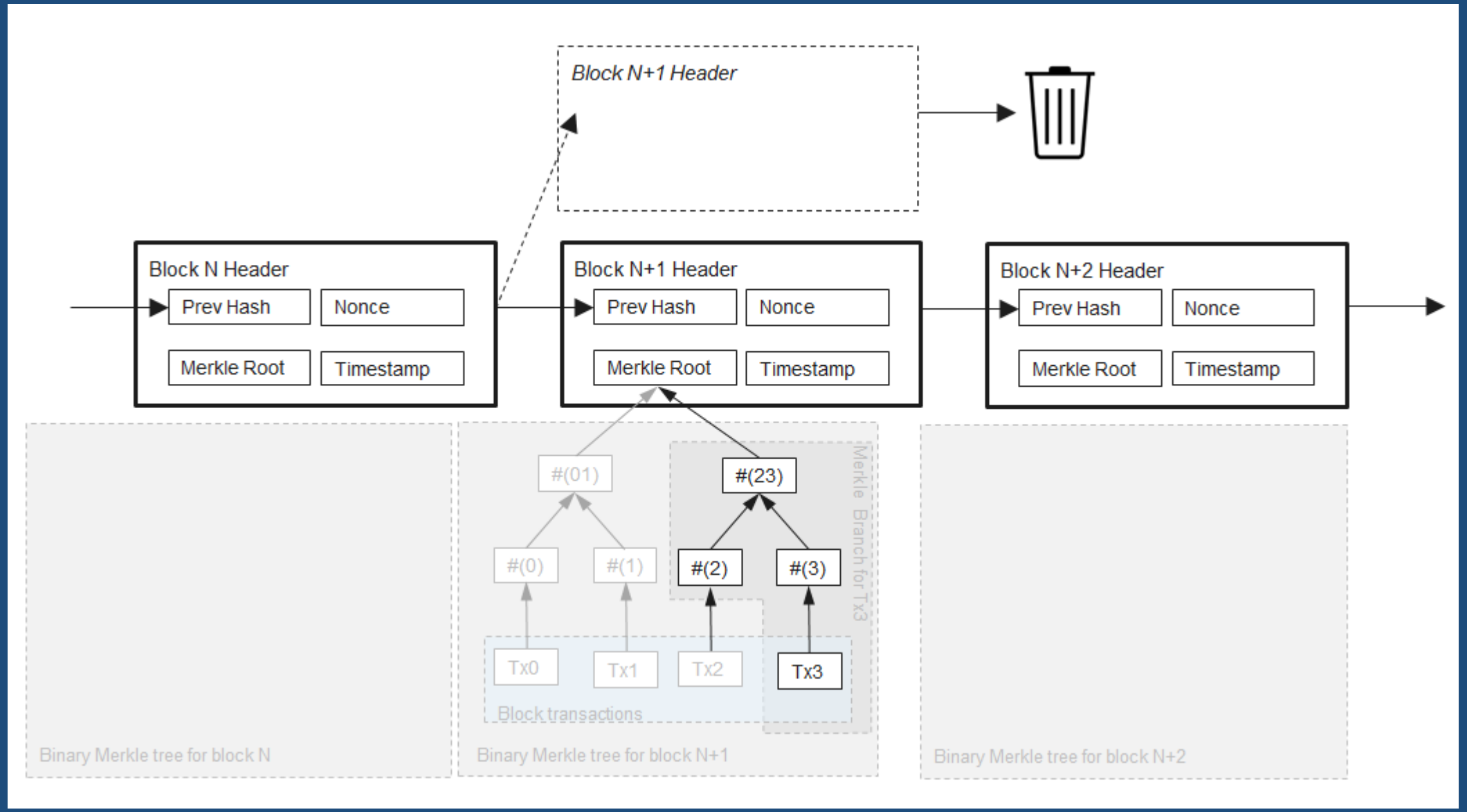
The basic architecture of the **EVM** (Ethereum Virtual Machine) that runs smart contracts is that all calls to the contract are executed as a transaction where the ether required for a contract method executed is transferred from the calling account address to the contract account address.
The contract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

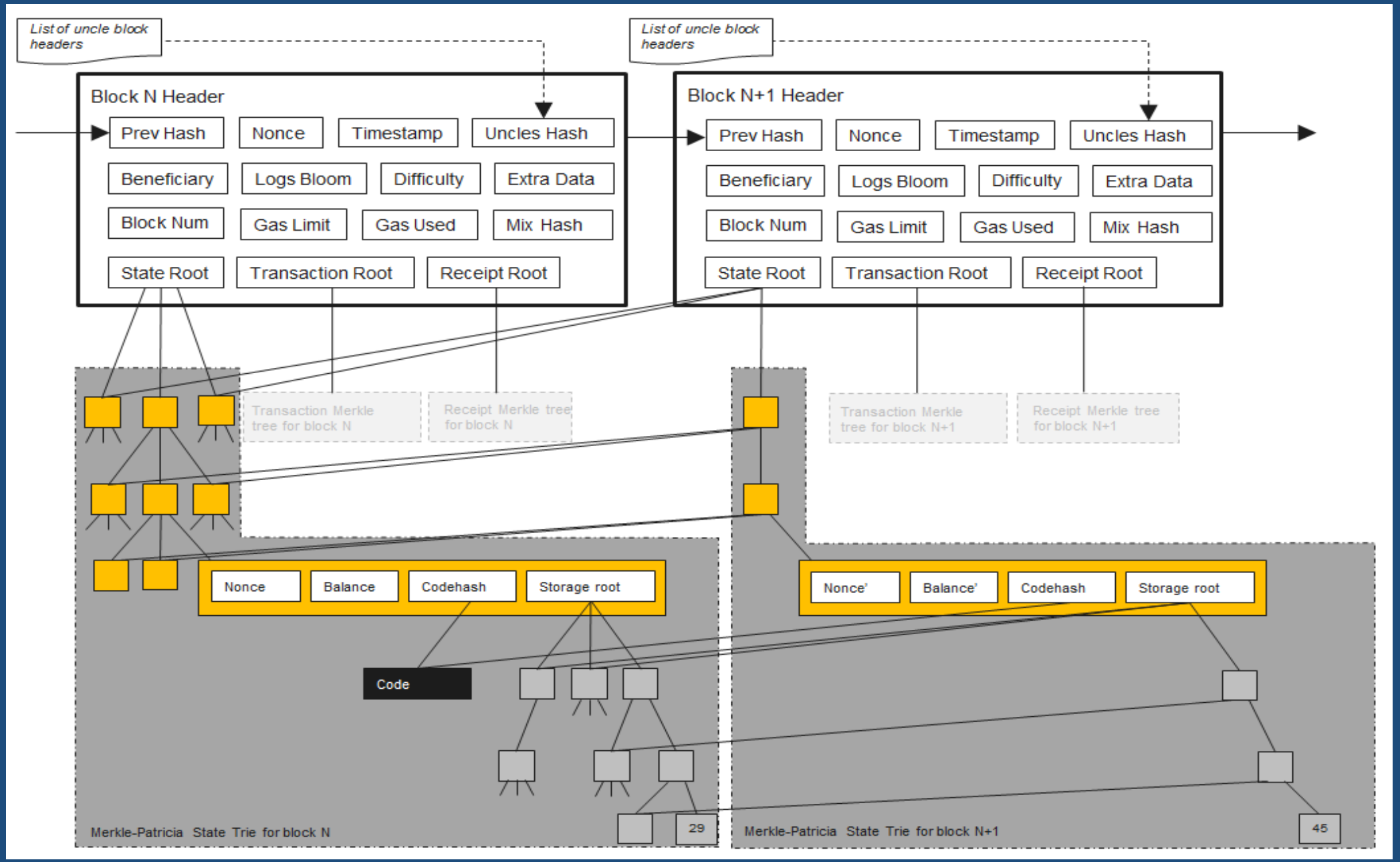



```
// in HTML page:
function start(){
    var Web3 = require('web3');
    var web3 = new Web3();
    web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'));
    var abi = [...];
    var corecontractContract = web3.eth.contract(abi);

    var corecontractContractInst = corecontractContract.at('0x0e22a4f27c2fc3b47e66b70fada85e1c4ca33681');
    console.log(corecontractContract.createCustomer(287187,"custName",13243244,1213334));
}
```

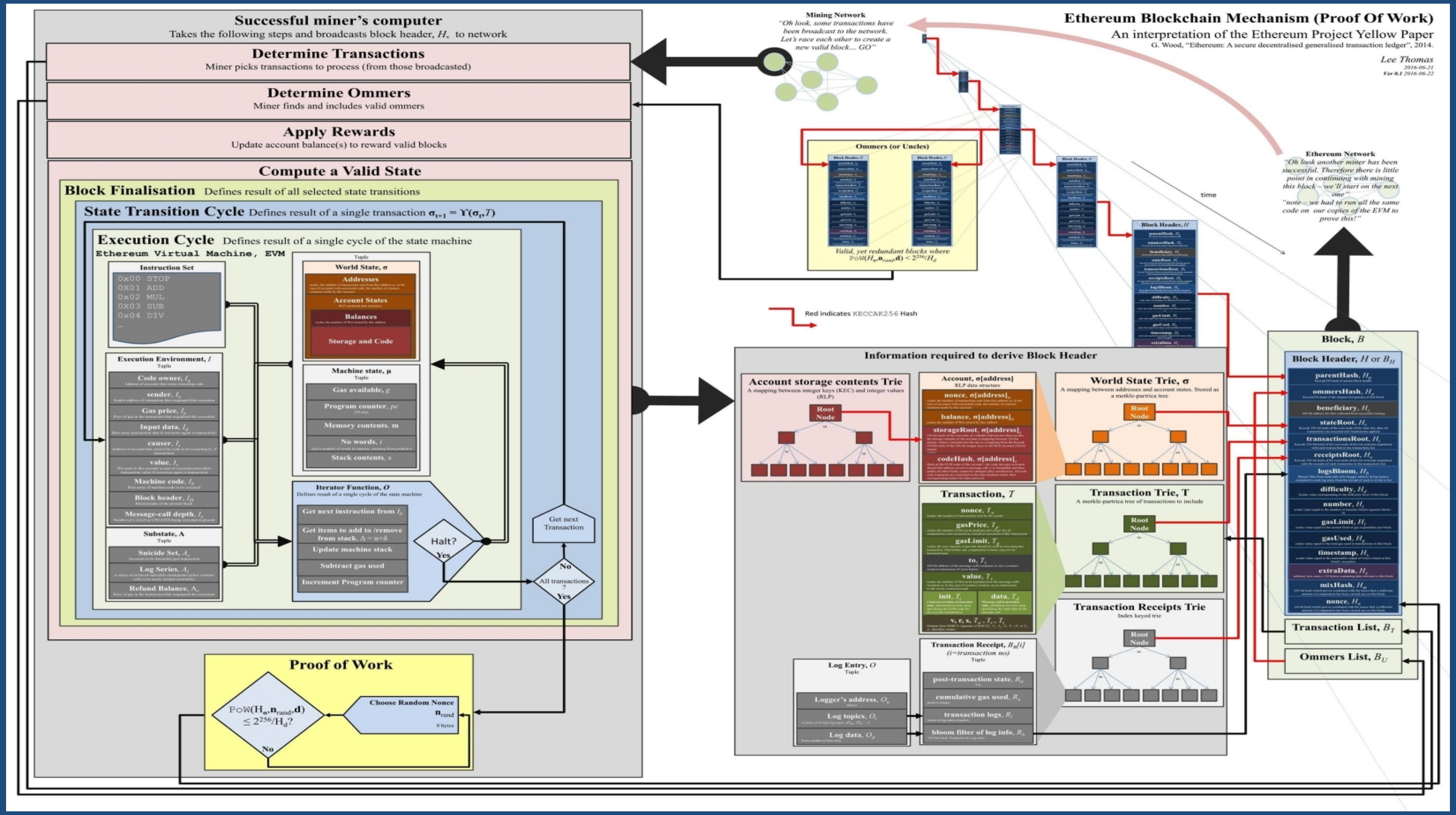
2. Ethereum Blockchain Block – Bitcoin Schematic



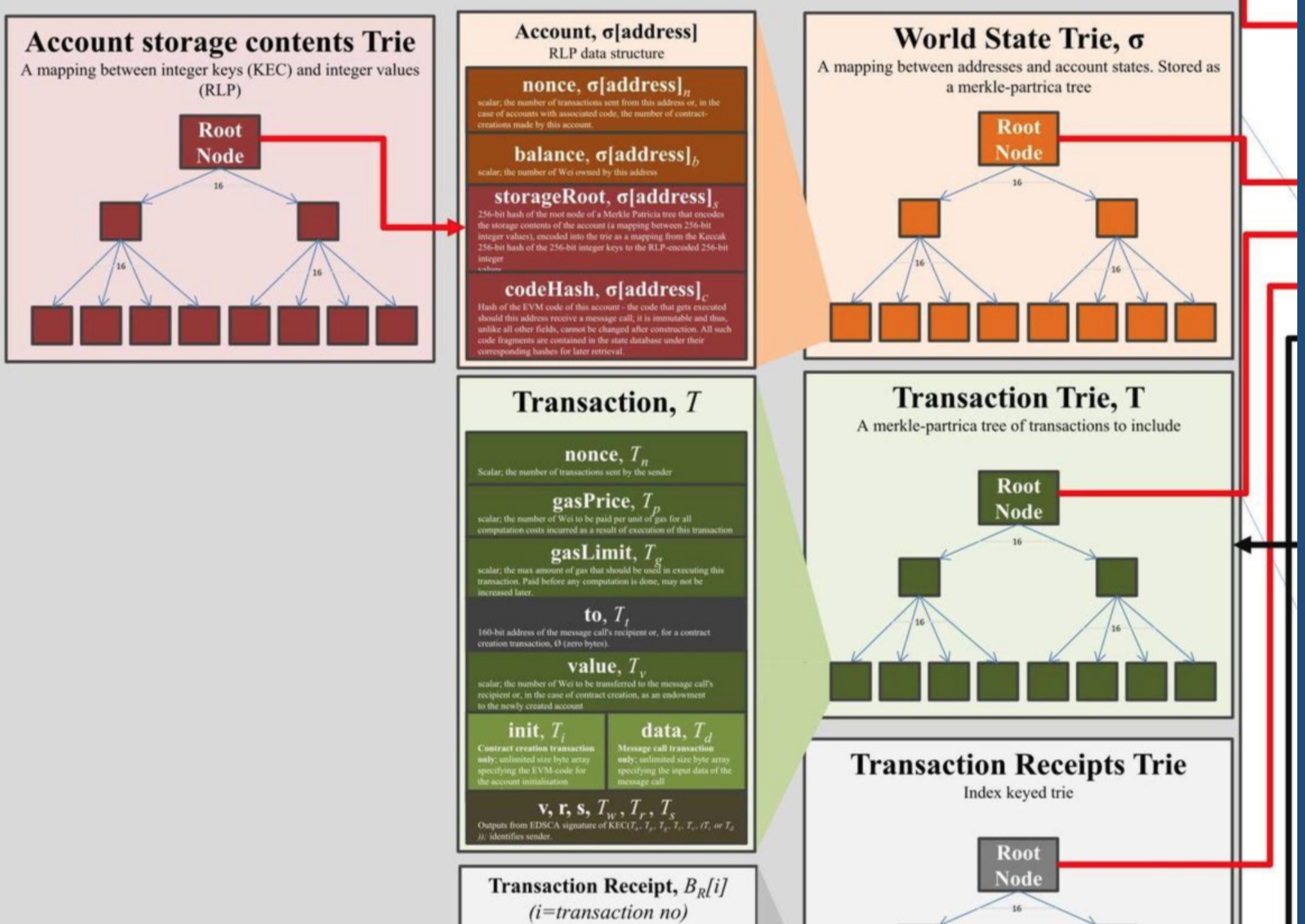
2. Ethereum Blockchain Block – ETH Schematic



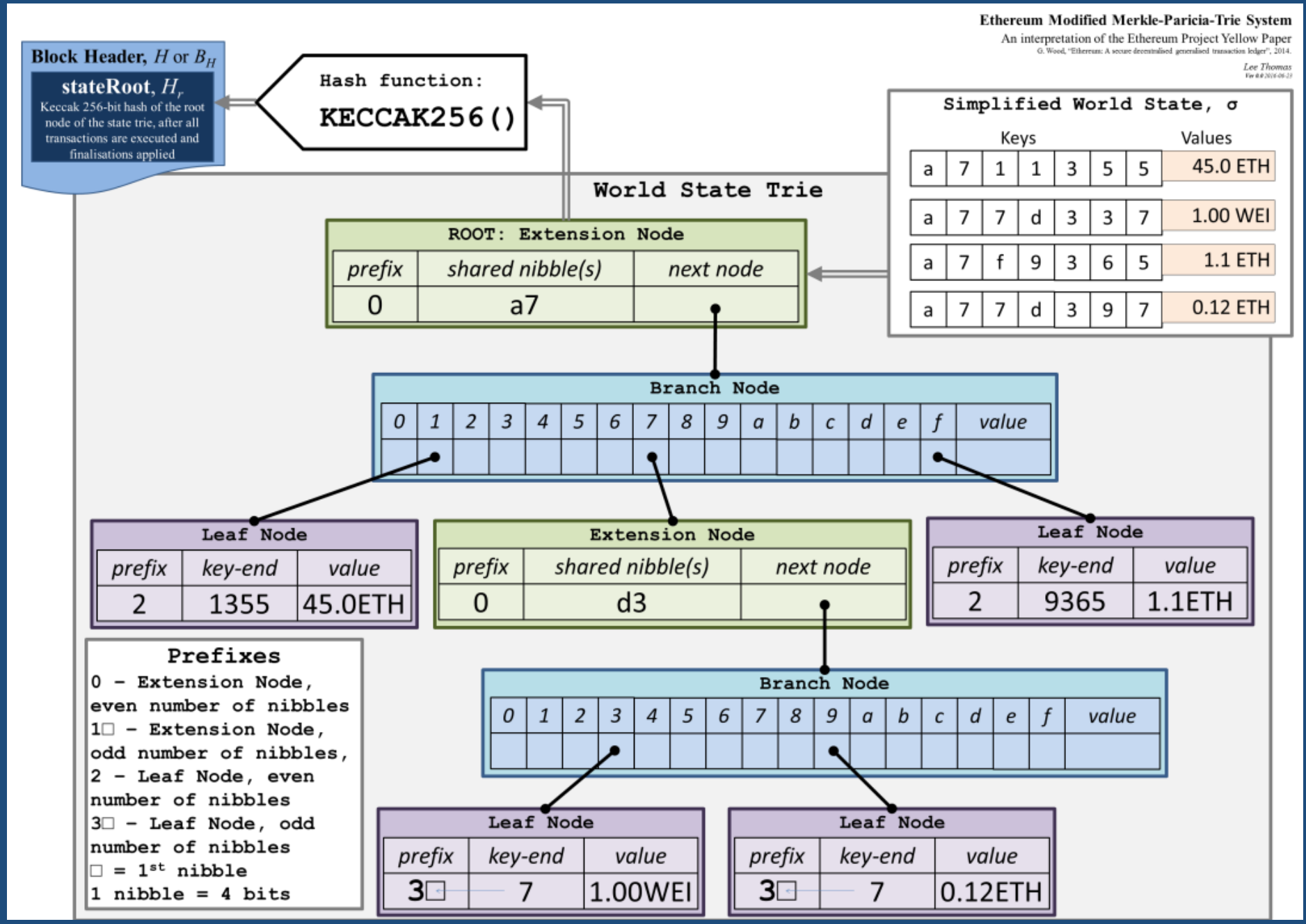
2. Ethereum Blockchain Block – ETH Schematic



2. Ethereu


Account storage contents Trie
A mapping between integer keys (KEC) and integer values (RLP)
Root Node
16
16
16
16
Account, σ[address]
RLP data structure
nonce, σ[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, σ[address]b
scalar; the number of Wei owned by this address
storageRoot, σ[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
codeHash, σ[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.
World State Trie, σ
A mapping between addresses and account states. Stored as a merkle-partrica tree
Root Node
16
16
16
16
Transaction, T
nonce, Tn
Scalar; the number of transactions sent by the sender
gasPrice, Tp
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, Tg
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, Tt
160-bit address of the message call's recipient or, for a contract creation transaction, 0 (zero bytes).
value, Tv
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, Ti
Contract creation transaction only; unlimited size byte array specifying the EVM-code for the account initialisation
data, Td
Message call transaction only; unlimited size byte array specifying the input data of the message call
v, r, s, Tw, Tr, Ts
Outputs from EDSCA signature of KEC(Tn, Tp, Tg, Tt, Tv, Ti or Td); identifies sender.
Transaction Trie, T
A merkle-partrica tree of transactions to include
Root Node
16
16
16
16
Transaction Receipts Trie
Index keyed trie
Root Node
16
Transaction Receipt, BR[i]
(i=transaction no)

2. Ethereum Blockchain Block – ETH Schematic



2. Ethereum Blockchain Block – ETH Schematic

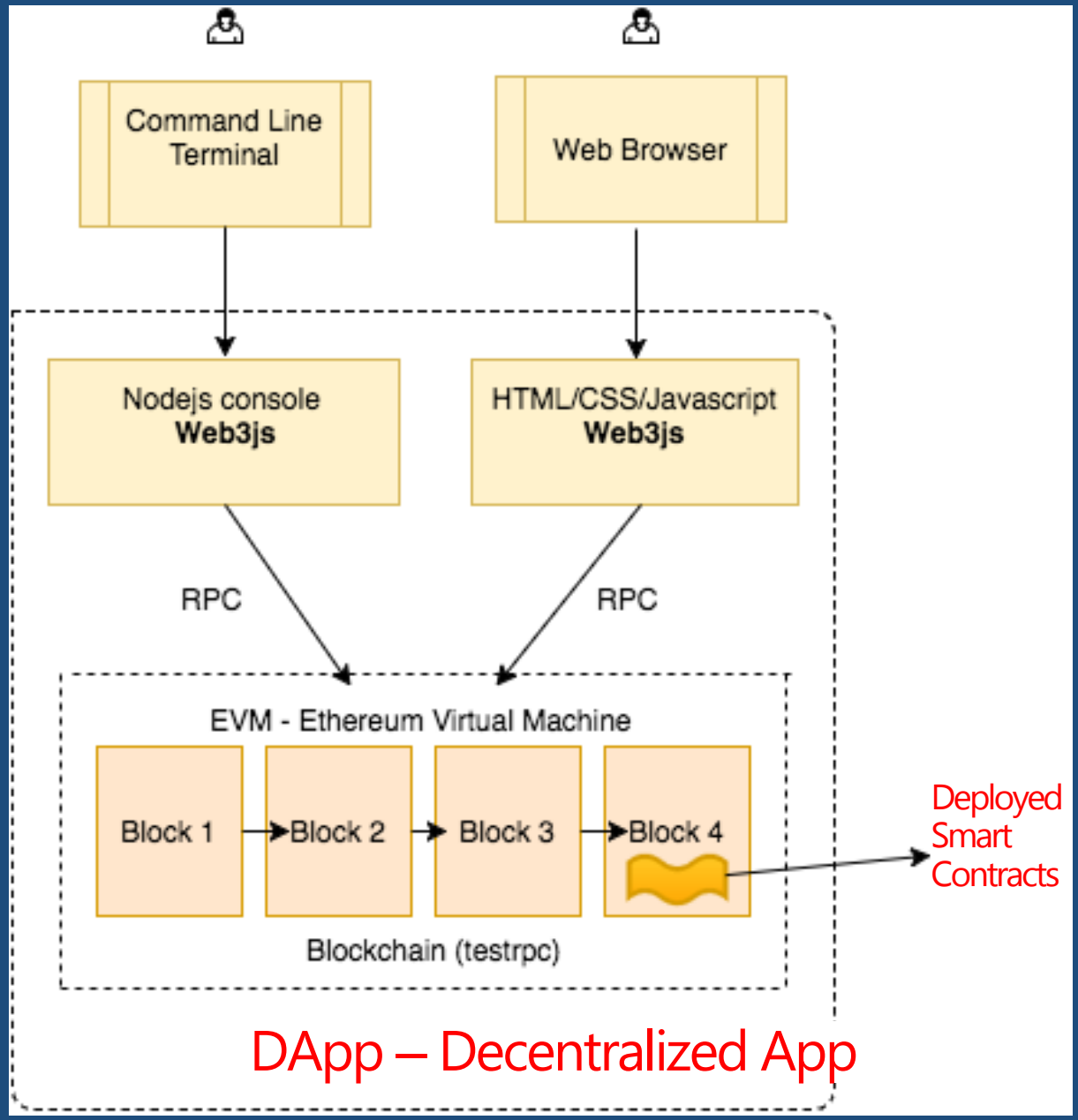
```
type BlockChain struct {
    config *params.ChainConfig // chain & network configuration

    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{}
}
```

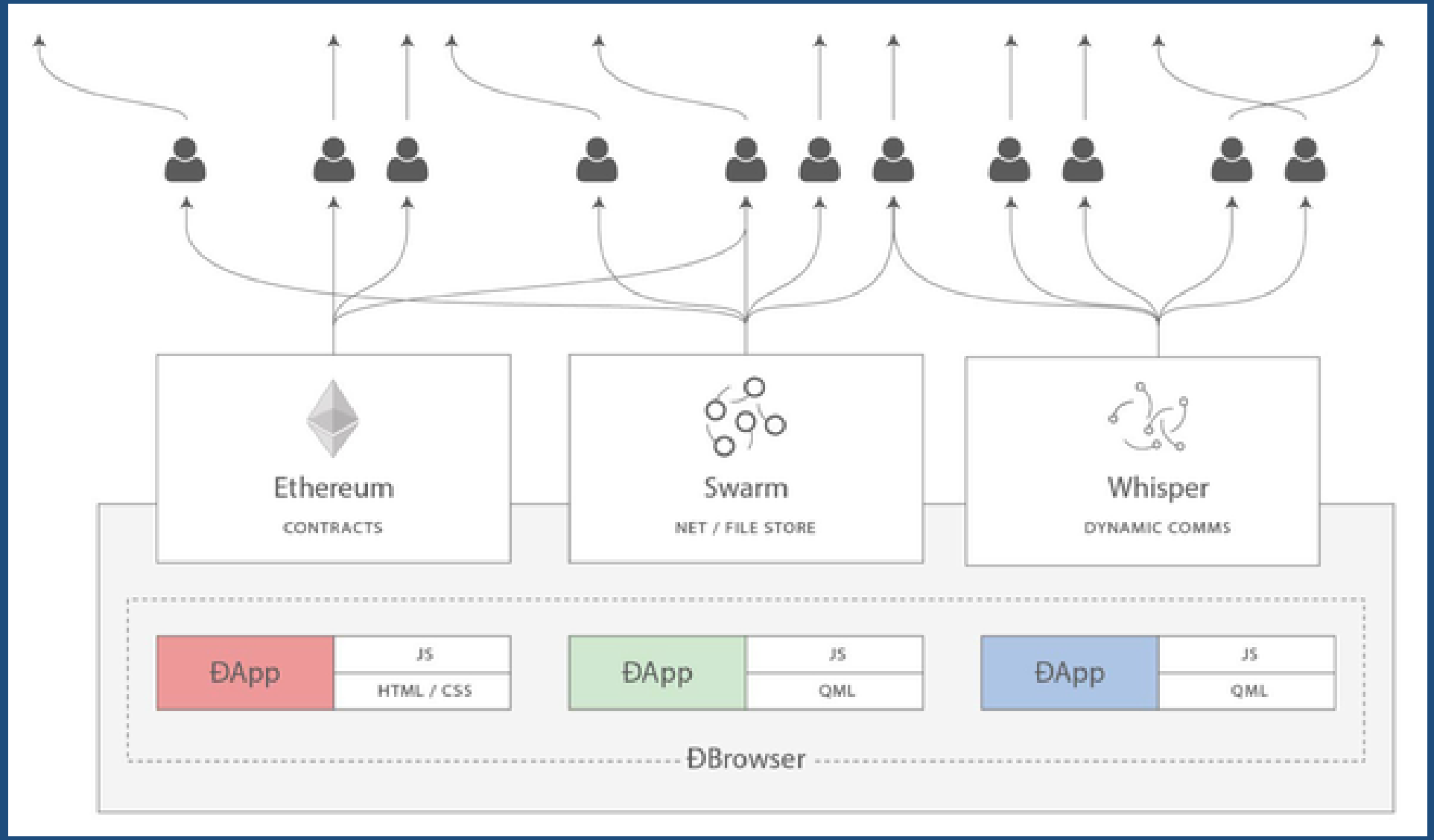
```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
    ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
    UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
    Coinbase    common.Address `json:"miner"            gencodec:"required"`
    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

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

	Bitcoin (BTC)	Ether (ETH)
What is it?	A currency	A token
Inventor	Satoshi Nakamoto	Vitalik Buterin, Joseph Lubin, Gavin Wood, etc.
Went alive	January 2009	July 2015
Supply Style	Deflationary (a finite # of bitcoin will be made)	Inflationary (much like fiat currency, where more tokens can be made over time)
Supply Cap	21 million in total	18 million every year
Smallest Unit	1 Satoshi = 0.00000001 BTC	1 Wei = 0.000000000000000001 ETH
New token issuance time	Every 10 minutes approximately	Every 10 to 20 seconds
Amount of new token at issuance	12.5 at the moment. Half at every 210,000 blocks	5 per every new block
Utility	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.
Price	Around $18,000 - $55,000 (in 2020 - 2021)	Around $610 - $2,000 (in 2020 - 2021)
Purpose	Bitcoin is a new currency created to compete against the gold standard and fiat currencies	Ethereum is a token capable of facilitating Smart Contracts (For example a lawyer's contract, an exchange of ownership of property, and voting)

03. Blockchain Use-cases in Fin-Tech

- Introduction
- **Use case 1** – Reductions of Fraud
- **Use case 2** – KYC – Know Your Customer
- **Use case 3** – Trading platforms
- **Use case 4** – Payments
- **Use case 5** – Simplifying Cross border payments
- **Use case 6** – The rise of smart assets
- **Use case 7** – Loyalty & Rewards Program
- 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 – Reductions of Fraud

- Traditionally, bank ledgers have been created within a centralized database. Hackers and cyber-criminals are well aware of evolving digital technology and have been able to bypass these security systems to commit data breaches and fraud.

- In contrast, as the blockchain is decentralized it is less prone to this type of fraud. By using blockchain there would not only be real-time execution of payments but also complete transparency which would enable real-time fraud analysis and prevention.

Use case 1 – Reductions of Fraud

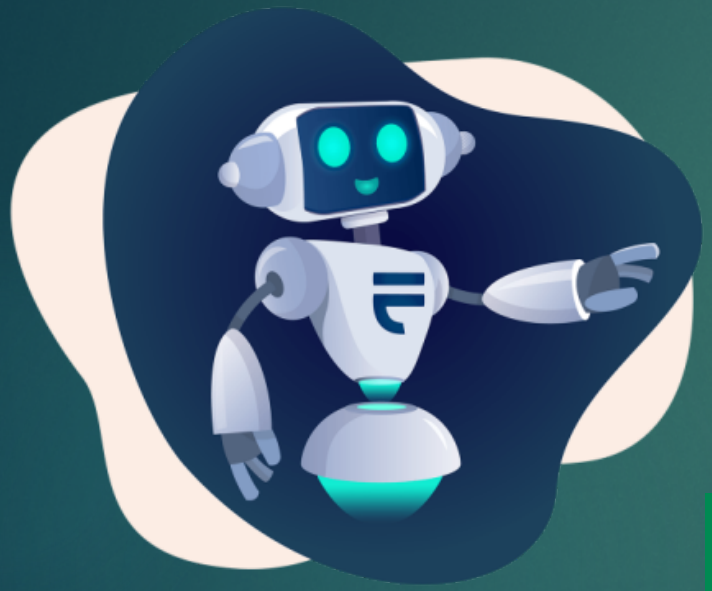
- Blockchain helps to eliminate the possible fraud on all transactions .

Use case 1 – Reductions of Fraud

On 30 December 2015 Nasdaq announced that it had made its first ever share trade using blockchain technology. Nasdaq used its proprietary Linq platform (developed in collaboration with Chain.com and global design firm IDEO) to sell shares.

In August 2016, nearly 120,000 units of digital currency Bitcoin worth about US $72 million was stolen from the exchange platform Bitfinex in Hong Kong. The Bitcoin was stolen from users' segregated wallets and amounted to about 0.75% of all Bitcoin in circulation at that time. Since the hack, Bitfinex has taken steps to reimburse accountholders with "BFX tokens" which are cryptographic tokens on the Omni blockchain that can be exchanged for $1 beneficial interests in iFinex (Bitfinex's parent company).

Use case 1 – Reductions of Fraud



Everledger

Auxesis

IBM Blockchain

vechain

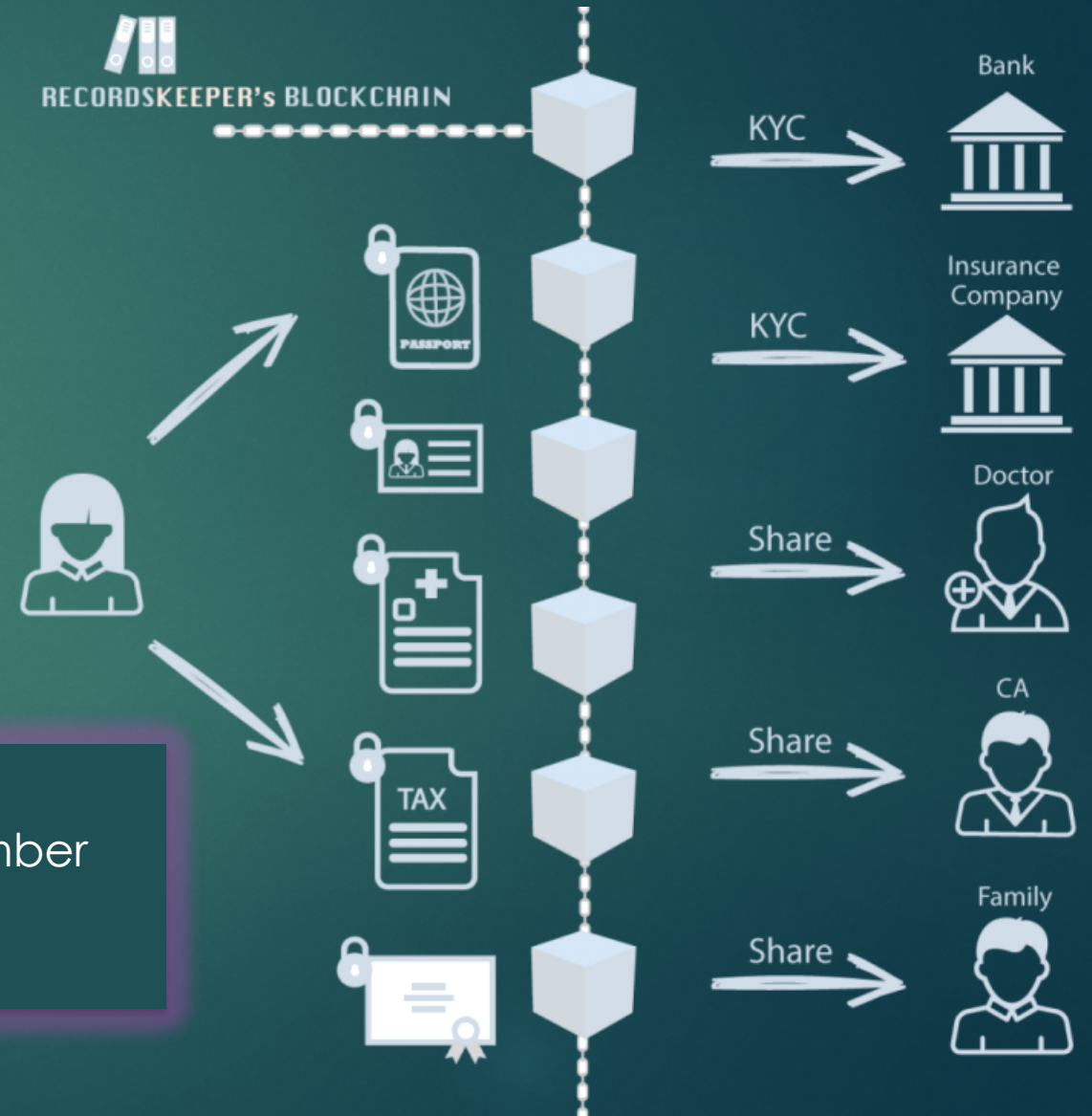
Use case 2 – KYC

- 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

ShoCard

SECURE KEY

Use case 3 – Trading platforms

- The blockchain technology offers a potential new medium to exchange assets without centralized trusts or intermediaries – and without the risk of double spending.
- Blockchain can eliminate the threat or the risk of fraud in all areas of banking, and this could equally apply to a trading platform.
- The blockchain gives the benefit of distributed and verifiable trust that was not present before.

Use case 3 – Trading platforms

SETL

Deloitte.

coinbase

CATEINA
TECHNOLOGIES

BINANCE

NASDAQ®

BITTREX

Use case 4 – Payments

- The main use case of blockchain for banking is that of payments.

Use case 4 – Payments

Use case 5 – Cross border payments

- Mastercard and R3 announced in 2019 a strategic partnership to develop and pilot a blockchain-enabled cross-border payments solution that will initially focus on connecting global faster payments infrastructures.

Use case 5 – Cross border payments

ripple

stellar

Open Source, Distributed Payments Infrastructure

Use case 6 – The rise of smart assets

HYPERLEDGER

DIGIX

BANKEX

KrypC

Modex

Use case 7 – Loyalty & Rewards Program

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.

https://www.omfif.org/wp-content/uploads/2020/05/The-role-of-blockchain-in-banking.pdf

XRP Ripple as SWIFT replacement

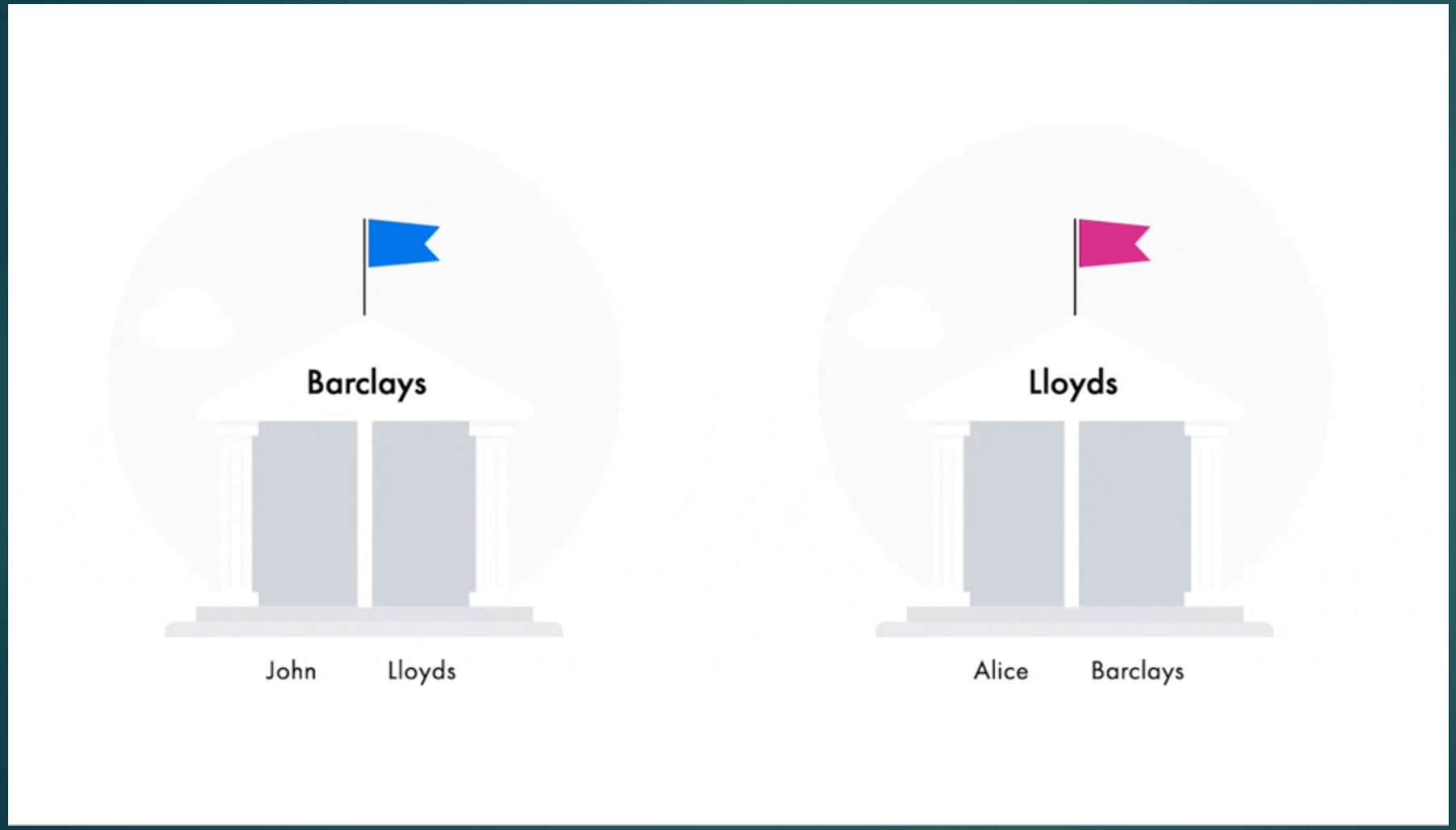
- Ripple comes up in 2012 with the name OpenCoin. Their objective has been to provide secure payments via a global network.
- In 2013 their system started to be used in banks for payment systems.
- In 2014 Ripple helped to create a new cryptocurrency: XRP. This cryptocurrency helps to make transfer money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

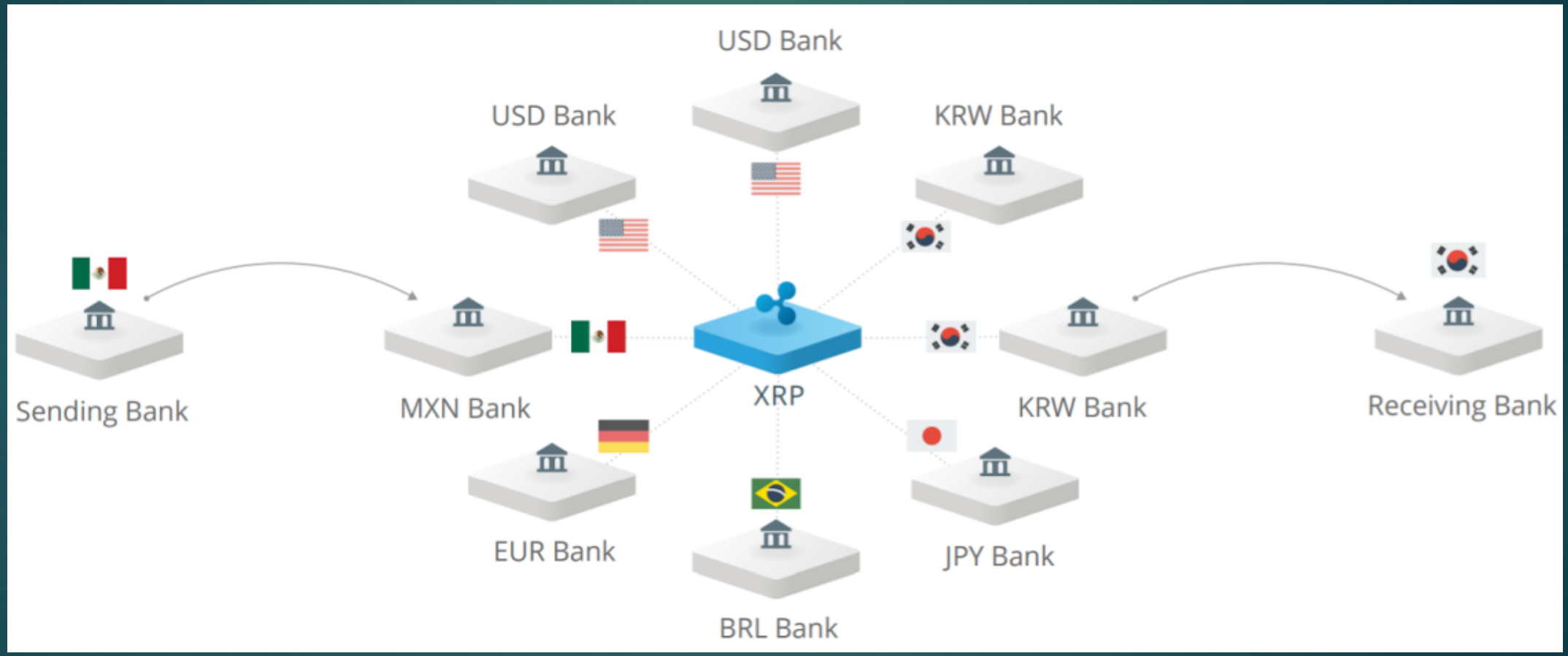
- Ripple is attempting to change this way of conducting cross-border payments, by using XRP as a ‘bridge’ between fiat currencies.

XRP Ripple as SWIFT replacement



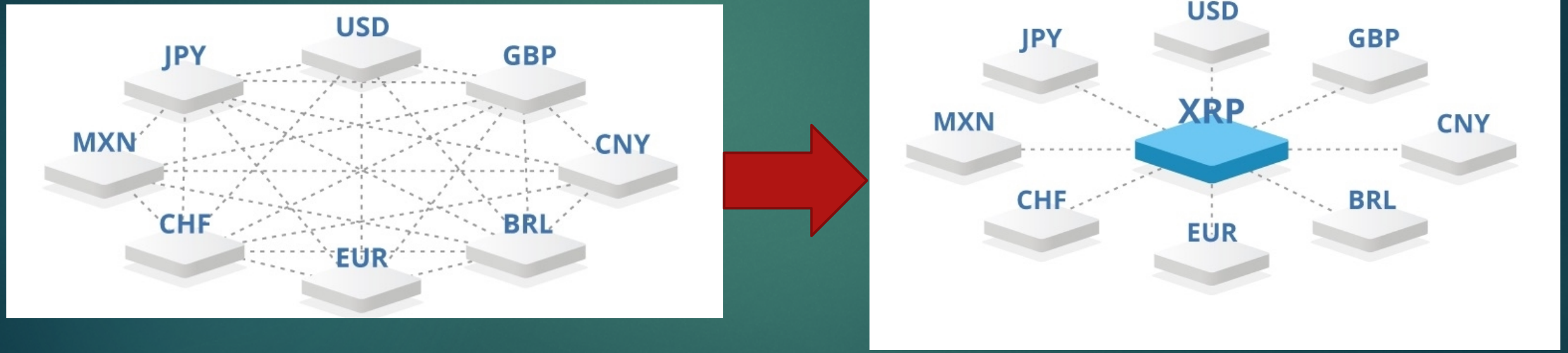
https://blog.revolut.com/swift-sepa-how-international-money-transfers-actually-work/

XRP Ripple as SWIFT replacement



https://www.coingecko.com/buzz/what-is-ripple-how-to-buy

XRP Ripple as SWIFT replacement



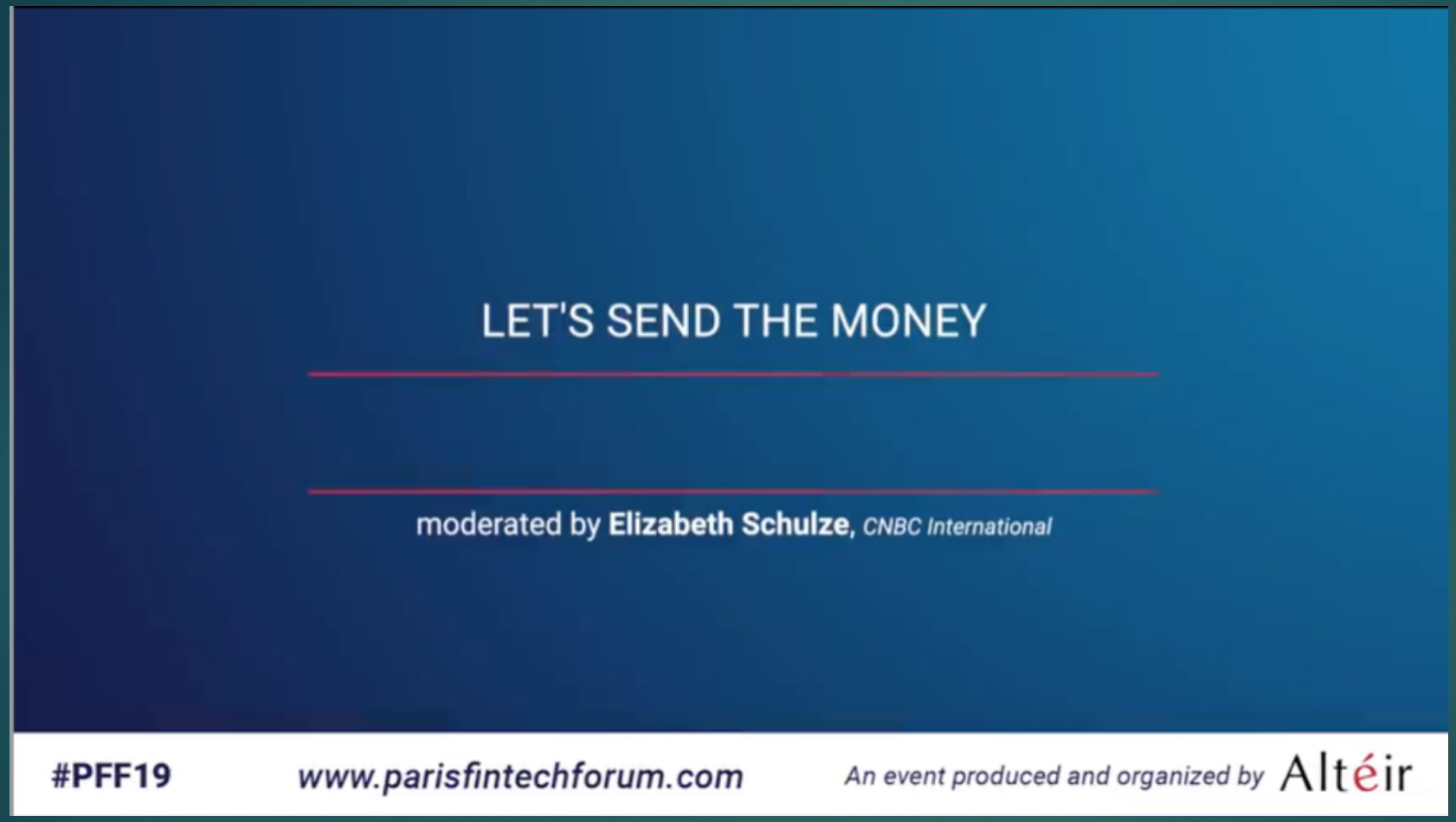
https://www.coindesk.com/global-banks-test-ripples-digital-currency-new-blockchain-trial

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.

- WIFT 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.

XRP Ripple as SWIFT replacement



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

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

Q & A


danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
vinaka
спасибі
blagodaram
kiitos
dankie
dhanyavad
nanni
nandri
bayarlalaa
gràcie
hvala
mauruuru
köszönöm
akun
dankon
ačiū
thank you
gracias
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
enkosi
bedankt
djiere dieuf
tau
mochchakkeram
дякую
mamnun
dziękuję
chnorakaloutioun
gratias agu
gràcies
sulpáy
go raibh maith agat
sobodi
dēkuji
sukriya
kop khun krap
takk
arigatô
takk
dakujem
trugarez
obrigado
mesi
sagolun
ありがとう
tanemirt
grazie
najis tuke
rahmet
terima kasih
diolch
dhanyavadagalu
shukriya
merce
мерси
didi madloba
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci

www: ism.ase.ro | acs.ase.ro | dice.ase.ro

Scan the Tag
to get the web
Mobile Address