ELLIPTIC CURVE CRYPTOSYSTEMS

Dr. Kunwar Singh
CSE Department
NIT Trichy

Group

Def (Group): A set *G* with a binary operation * is called a *Group* if

1. ∀ a,b∈G, a*b∈G (Closure)
2. ∀ a,b,c∈G, (a*b)*c=a*(b*c)
3. ∃ identity element e∈G, ∀ a∈G, a*e=a=e*a
4. ∀ a∈G, ∃ a' (inverse of a)∈G, a*(a')=e

Example: Set G = {1,2,3,4}, Binary operation . (multiple) mod 5 is Group.

Example: For Prime p, ZP = {1,2,3,4,...,P-1}, Binary operation . (multiPle) mod p is Group.

Cyclic Group

- Group which can be Generated using the Power of an element, the Group is called the cyclic Group.

  $G = \{ x^i : 1 \leq i \leq P\text{-}1 \}$

- Example of Z_7

x	x^2	x^3	x^4	x^5	x^6
3	2	6	4	5	1
3	3^2	3^3	3^4	3^5	3^6

Generators

x	x^2	x^3	x^4	x^5	x^6
5	2	6	4	5	1
5	5^2	5^3	5^4	5^5	5^6

$|Z_P^*| = (p-1)$
By Fermat's little theorem: $a^{(p-1)} = 1 \pmod p$
For all p the Group is cyclic.

Fields

Def (field): A set F with two binary operations + (addition) and · (multiplication) is called a *field* if

1. $\forall\ a,b \in F,\ a+b \in F$
2. $\forall\ a,b,c \in F,\ (a+b)+c=a+(b+c)$
3. $\forall\ a,b \in F,\ a+b=b+a$
4. $\exists\ 0 \in F,\ \forall\ a \in F,\ a+0=a$
5. $\forall\ a \in F,\ \exists\ -a \in F,\ a+(-a)=0$
6. $\forall\ a,b \in F,\ a \cdot b \in F$
7. $\forall\ a,b,c \in F,\ (a \cdot b) \cdot c=a \cdot (b \cdot c)$
8. $\forall\ a,b \in F,\ a \cdot b=b \cdot a$
9. $\exists\ 1 \in F,\ \forall\ a \in F,\ a \cdot 1=a$
10. $\forall\ a \neq 0 \in F,\ \exists\ a^{-1} \in F,\ a \cdot a^{-1}=1$

$\forall\ a,b,c \in F,\ \ a \cdot (b+c)=a \cdot b+a \cdot c$

- $(R,+,\times)$ if field where R is set of real number.
- GF(p)

$Z_p = \{0,1,2,...,p-1\}$ where p is prime

$(Z_p, +, *)$ is finite field.

+ and $*$ defined as

$x+y = x+y \bmod p$,

$x*y = xy \bmod p$

Example: $(Z_2, +, *)$ is field.

$Z_2 = \{0,1\}$

+ and $*$ defined as

$x+y = x+y \bmod 2$,

$x*y = xy \bmod 2$

Characteristic of a Field

- Definition: Let F be a field with multiplicative identity e. The characteristic of F is the smallest integer p such that

$$e + e + e + \ldots + e = 0 \text{ (additive identity)}$$

Example: R has characteristic 0.

Example: Z_2 has characteristic 2

Quadratic Residues

- **Definition 4.1**. If m is a positive integer, we say that the integer a is a *quadratic residue (square)* of *mod m* if gcd(a,m) = 1 and the congruence $x^2 \equiv a \pmod{m}$ has a solution.

If the congruence $x^2 \equiv a \pmod{m}$ has no solution, we say a is *quadratic non-residue* of m.

Example. Let P = 11. Then

- Is 4 a square modulo p? YES, because 2^2 ≡ 4 (mod 11)
- Is 5 a square modulo p? YES because 4^2 ≡ 5 (mod 11)
- Is 2 a square modulo p? No

Quadratic Residues

Let $p = 11$

a	1	2	3	4	5	6	7	8	9	10
a^2 mod 11	1	4	9	5	3	3	5	9	4	1

$QR(Z_p) = \{1, 3, 4, 5, 9\}$

Observe

- There are 5 squares and 5 non-squares.
- Every square has exactly 2 square roots.

Quadratic Residues

- **Lemma.** *Let p be odd prime and a an integer not divisible by p. Then the congruence $x^2 \equiv a \pmod{p}$ has either no solutions or exactly two incongruent solutions modulo p.*

- **Theorem.** *If p is an odd prime , then there are exactly $(p-1)/2$ quadratic residues of p and $(p-1)/2$ quadratic nonresidues of p among the integer $1, 2, \ldots, p-1$.*

  Proof. To find all the quadratic residues of p among the integers $1, 2, \ldots, p-1$ we compute the least positive residues modulo p of the squares of the integers $1, 2, \ldots, p-1$.

ECC as Light – weight Encryption

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Non singular Elliptic Curves over GF(p)

Elliptic Curve equation: (characteristic is not equal to 2)

$$y^2 \bmod p = x^3 + ax + b \bmod p$$

$$\text{where } 4a^3 + 27b^2 \bmod p \neq 0.$$

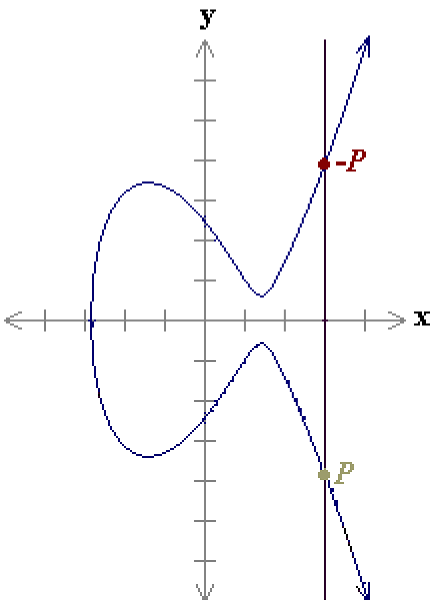
$E_{a,b}$ = points on non singular elliptic curve + Point at infinity

Claim: ($E_{a,b}$, addition (+) abelian Group)

Addition (+) is defined as:

Case (1) : $P = O + P$ for all P

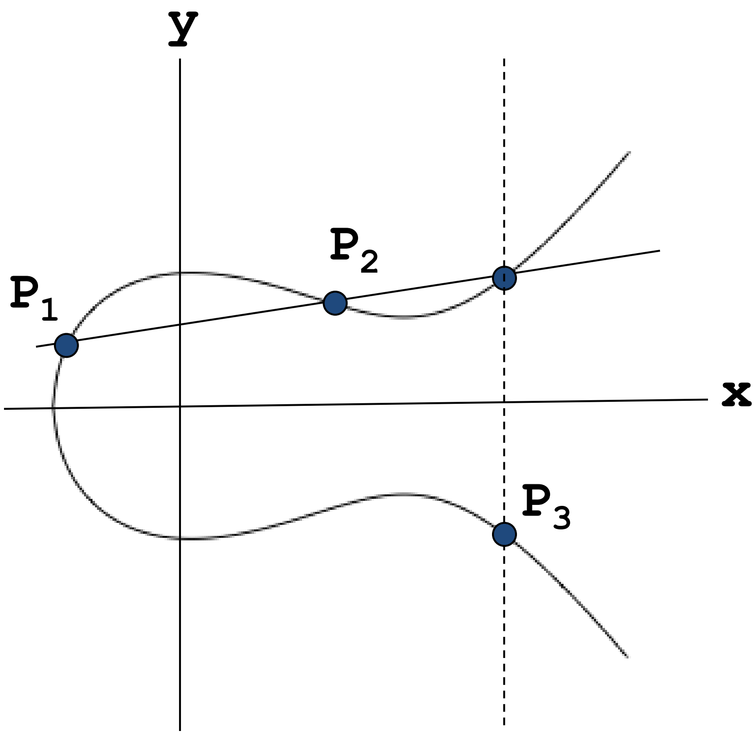
O is called as additive identity



$y^2 = x^3 - 6x + 6$

Case (2) : $P + (-P) = O$

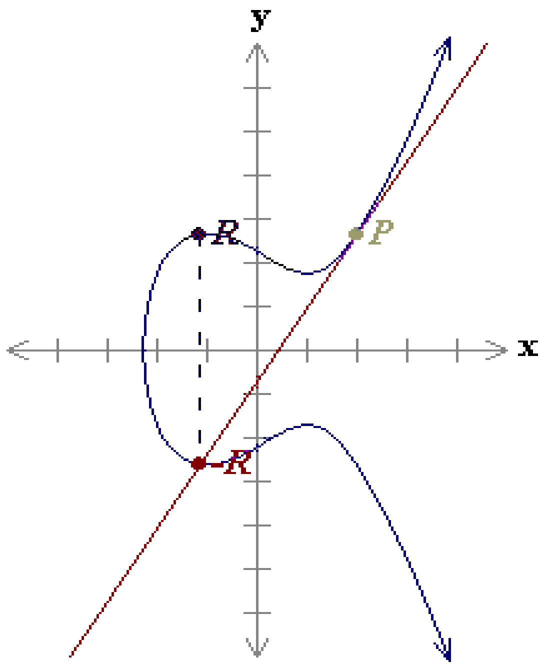
Case 3: $P_1(x_1,y_1) + P_2(x_2,y_2) = P_3(x_3,y_3)$



$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$

$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

Case 4: P +P = R



$y^2 = x^3 - 3x + 5$

$P\ (2, 2.65)$

$-R\ (-1.11, -2.64)$

$R\ (-1.11, 2.64)$

$2P = R = (-1.11, 2.64).$

$$2y\frac{dy}{dx} = 3x^2 + A$$

$$\Rightarrow m = \frac{dy}{dx} = \frac{3x_1^2 + A}{2y_1}$$

$$If, y_1 \neq 0 \text{ (since then } \mathrm{P}_1 + \mathrm{P}_2 = \infty\text{):}$$

$$\therefore 0 = x^3 - m^2x^2 + \ldots$$

$$\Rightarrow x_3 = m^2 - 2x_1, y_3 = m(x_1 - x_3) - y_1$$

Elliptic Curves over GF(p)

Algorithm 10.12 *Pseudocode for finding points on an elliptic curve*

```
ellipticCurve_points (p, a, b)                         // p is the modulus
{
    x ← 0
    while (x < p)
    {
        w ← (x^3 + ax + b) mod p                           // w is y^2
        if (w is a perfect square in Z_p) output (x, √w) (x, − √w)
        x ← x + 1
    {
}
```

- Consider $y^2 = x^3 + 2x + 3$ (**mod 5**)

  $x = 0 \Rightarrow y^2 = 3 \Rightarrow$ no solution (mod 5)

  $x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1,4$ (mod 5)

  $x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0$ (mod 5)

  $x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1,4$ (mod 5)

  $x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0$ (mod 5)

- Then points on the elliptic curve are

  (1,1) (1,4) (2,0) (3,1) (3,4) (4,0)

  and the point at infinity: O

Example 2: elliptic curve over G*F(*23*)*

- p = 23
- $E = 29 : y^2 = x^3 + x + 4$

  $\# E = 29 \Rightarrow$ group is cyclic

Example 2: elliptic curve over G*F(*23*)*

- *p* = 23
- $E = 29 : y^2 = x^3 + x + 4$

  $\# E = 29 \Rightarrow$ group is cyclic
- The points in *E* are O and the following:

G = (0, 2)	2G = (13, 12)	3G = (11, 9)	4G = (1, 12)
5G = (7, 20)	6G = (9, 11)	7G = (15, 9)	8G = (14, 5)
9G = (4, 7)	10G = (22, 5)	11G = (10, 5)	12G = (17, 9)
13G = (8, 15)	14G = (18, 9)	15G = (18, 14)	16G = (8, 8)
17G = (17, 14)	18G = (10, 18)	19G = (22, 18)	20G = (4, 16)
21G = (14, 18)	22G = (15, 17)	23G = (9, 12)	24G = (7, 3)
25G = (1, 11)	26G = (11, 14)	27G = (13, 11)	28G = (0, 21)
29G = *O*	30G = G		

29 Points

Elliptic Curve Cryptosystem

Key Generation: Alice creates Public/Private keys.

- Alice
  - Private Key = a   ($1<a < p-1$)
  - Public Key = $P_A = a * G$

Encryption: Alice takes Plaintext message, M, and encodes it on to a point, $P_M(x_m, y_m)$ from the elliptic Group

– Alice chooses another random integer, k from the interval [1, p-1]

– The ciphertext **$P_C = [\ (kB), (P_M + kP_B)\ ]$**

Decryption:

- To decrypt, Bob computes the Product of the first Point from P_C and his Private key, b
  - **$b * (kG)$**
- Bob then takes this Product and subtracts it from the second Point from P_C
  - **$(P_M + kP_B) - [b(kG)] = P_M + k(bG) - b(kG) = P_M$**
- Bob then decodes P_M to Pet the message, M.

Digital Signature

- It is similar to normal signature except that it depends on the message.
- It provides data integrity, data authentication and non repudiation.
- One party penerates signature, anyone who knows the Public key can verify it.

Signature

Alice

Alice's Unique Hand writing

Message M

Signature

Signature is indePendent of the message

Everyone else

Message M

Alice' s grevious signature

Signature

Verify function

Indeed signed by Alice

Digital Signatures

Alice

Message M

Alice's private Key

Sign function

Signature for M

Everyone else

Message M

Alice's Public Key

Signature for M

Verify function

Message M was indeed signed by Alice

Definition

- A signature scheme consists of three algorithms:
  - gen: randomized *algorithm* that on input 1^n outputs (Pk, sk)
  - Sig: randomized *algorithm* that on input sk and a message m outputs a signature σ.
  - Veri: determinstic *algorithm* that on input Pk and a signature σ outputs a message m or an error $\perp$.

Signature scheme is broken:

- *Total Break*: Private key is compromised
- *Selective forgery*: adversary can create a valid signature on a message of his choice
- *Existential forgery*: adversary can create a valid signature on any message

ECDSA - Elliptic Curve Digital Signature Algorithm

- Key generation:
  - Private Key = a   ($1<a < p-1$)
  - Public Key = $P_A = a * G$ = Point on a curve

- Signature Generation:

For signing a message m by sender A, using A's Private key d_A and Public key $Q_A = d_A * G$

1. Calculate e = SHA-256(SHA-256(m)),
2. Select a random integer k from $[1,p - 1]$
3. Calculate $r = x_1 \pmod p$, where $(x_1, y_1) = k * G$. If $r = 0$, go to step 2
4. Calculate $s = k^{-1}(e + d_A r) \pmod p$. If $s = 0$, go to step 2
5. The signature is the pair (r, s)

Signature Verification:

For B to authenticate A's signature, B must have A's Public key Q_A

1. Verify that r and s are integers in $[1, n - 1]$. If not, the signature is invalid
2. Calculate $e = \text{SHA-256}(\text{SHA-256}(m))$,
3. Calculate $w = s^{-1} \pmod{n}$
4. Calculate $u_1 = ew \pmod{n}$ and $u_2 = rw \pmod{n}$
5. Calculate $(x_1, y_1) = u_1P + u_2Q_A$
6. The signature is valid if $x_1 = r \pmod{n}$, invalid otherwise

- $u_1P + u_2Q_A = e\, s^{-1} P + r\, s^{-1} aP$
- $= (e + ar)\, s^{-1} \quad \text{mod } n$
- $= k P$
- $= (x, y)$

- **malleable signature:** Given signature on message m, third party can use it to compute another valid **signature on same message** without knowing the private key.

ECDSA Signature is malleable:

- Lemma: $(n-s)^{-1} = \mathrm{n} - s^{-1}$ in prime field.

- $(n-s)^{-1} \times (\mathrm{n} - s^{-1}) = 1$
- $(\mathrm{n} - s^{-1}) \times (n-s)^{-1} = 1$

- Theorem: (r, s) is a valid ECDSA signature for a message m whenever (r, n-s) is a valid ECDSA signature for the same message.

Signature Verification:

For B to authenticate A's signature, B must have A's Public key Q_A

1. Verify that r and s are integers in $[1, n - 1]$. If not, the signature is invalid
2. Calculate e = SHA-256(SHA-256(m)) ,
3. Calculate $w = (n\text{-}s)^{-1} \pmod n$
4. Calculate $u_1 = e\,(n\text{-}s)^{-1} \pmod n$ and $u_2 = r\,(n\text{-}s)^{-1} \pmod n$
5. Calculate $(x_1, y_1) = u_1P + u_2Q_A$
6. The signature is valid if $x_1 = r \pmod n$, invalid otherwise

- $u_1P + u_2Q_A = e\,(n\text{-}s)^{-1} P + r\,(n\text{-}s)^{-1} aP$
- $= (n\text{-}s)^{-1} (e + ar) P \mod n$
- $= (n\text{-}s^{-1})(e + ar) P \mod n$
- $= n\,(e + ar) P \mod n - s^{-1} (e + ar) P \mod n$
- $= - s^{-1} (e + ar) P \mod n$
- $= - k P$
- $= (x, - y)$

THANK YOU

Blockchain Technology

1st Application
Cryptocurrency Bitcoin

Dr. Kunwar Singh
CSE Department
NIT Trichy

Bitcoin Cryptocurrency

Bitcoin

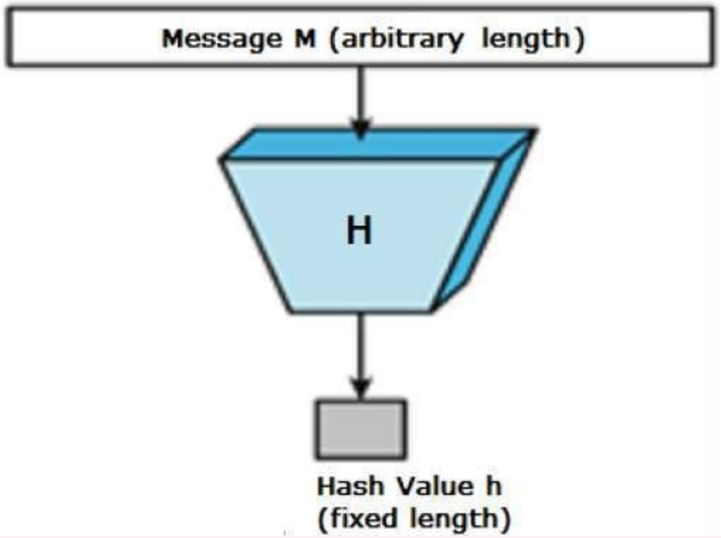
- No trusted party.
- Decentralized ledgers on Internet.
- It is peer to peer network: Distribute and record transactions.

- Very Low Transaction fees
- Not Reversible

- All transactions are known to everyone.

Users can only see the transactions. Actual senders and receivers cannot be identified.

Cryptographic Hash Functions



Properties

- Pre-image Resistant: Given h, hard to find m such that h = hash(m).
- Second Pre-image Resistant: Given m1, hard to find m2 such that hash(m1) = hash(m2)
- Collision Resistant: Hard to find m1 and m2 such that

  hash(m1) = hash(m2).

Public-Key Cryptography

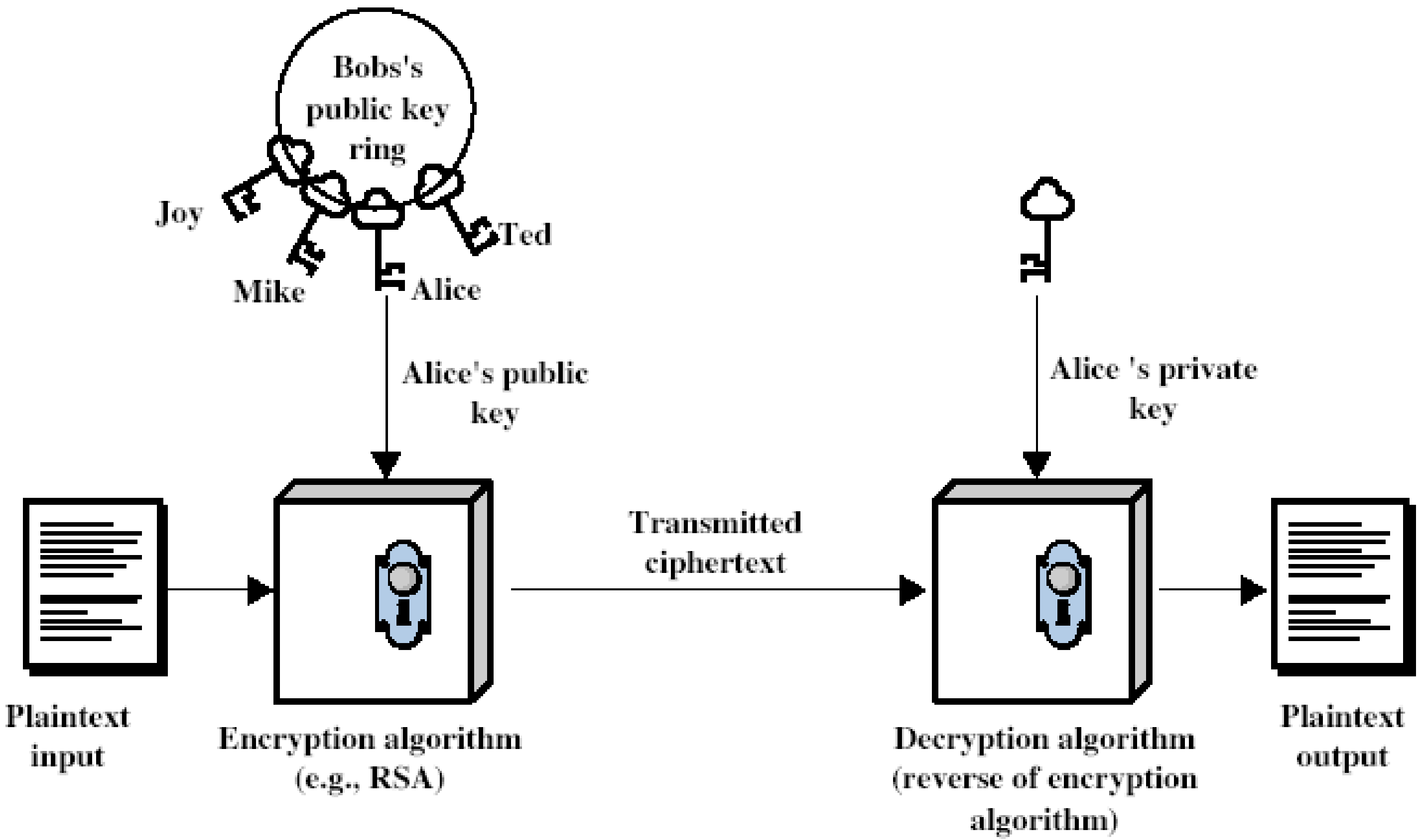
by Diffi & Helleman

- No need to share the secret key before communication (Unlike symmetric cryptosystem),
- Every user has two keys:

  Public key: is made public

  Private key: is private to the user
- One can encrypt (lock) with both keys but decrypt (open) can be done by the key which is different from encryption key.
- Can be used in shared session key set up.

Public Key Encryption



Public-key encryption

- A public-key encryption scheme consists of three algorithms:
  - Gen: randomized *algorithm* that on input 1^n outputs pk, sk
  - Enc: randomized *algorithm* that on input pk and a message m outputs a ciphertext c.
  - Dec: determinstic *algorithm* that on input sk and a ciphertext c outputs a message m or an error $\perp$

Correctness: $Dec_{sk}(Enc_{pk}(m)) = m$

ELLIPTIC CURVE CRYPTOSYSTEMS

Although RSA and ElGamal are secure asymmetric-key cryptosystems, their security comes with a price, their large keys.

ECC provide same level of security with smaller key sizes.

- **Elliptic Curves over GF(*p*)**

ECC as Light – weight Encryption

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Signature

Alice

Everyone else

Alice's Unique Hand writing

Message M

Signature

Signature is independent of the message

Message M

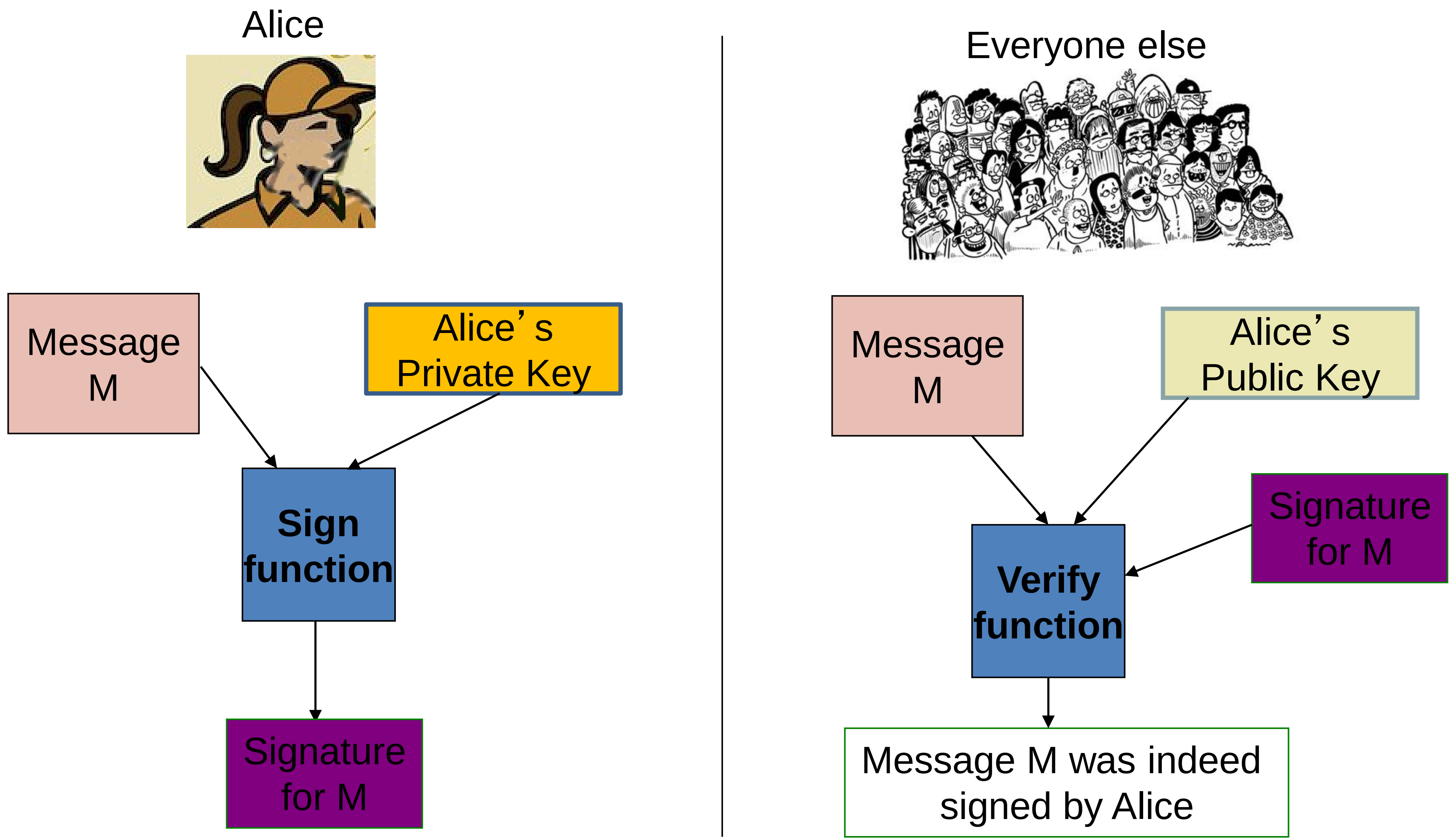
Alice's previous signature

Signature

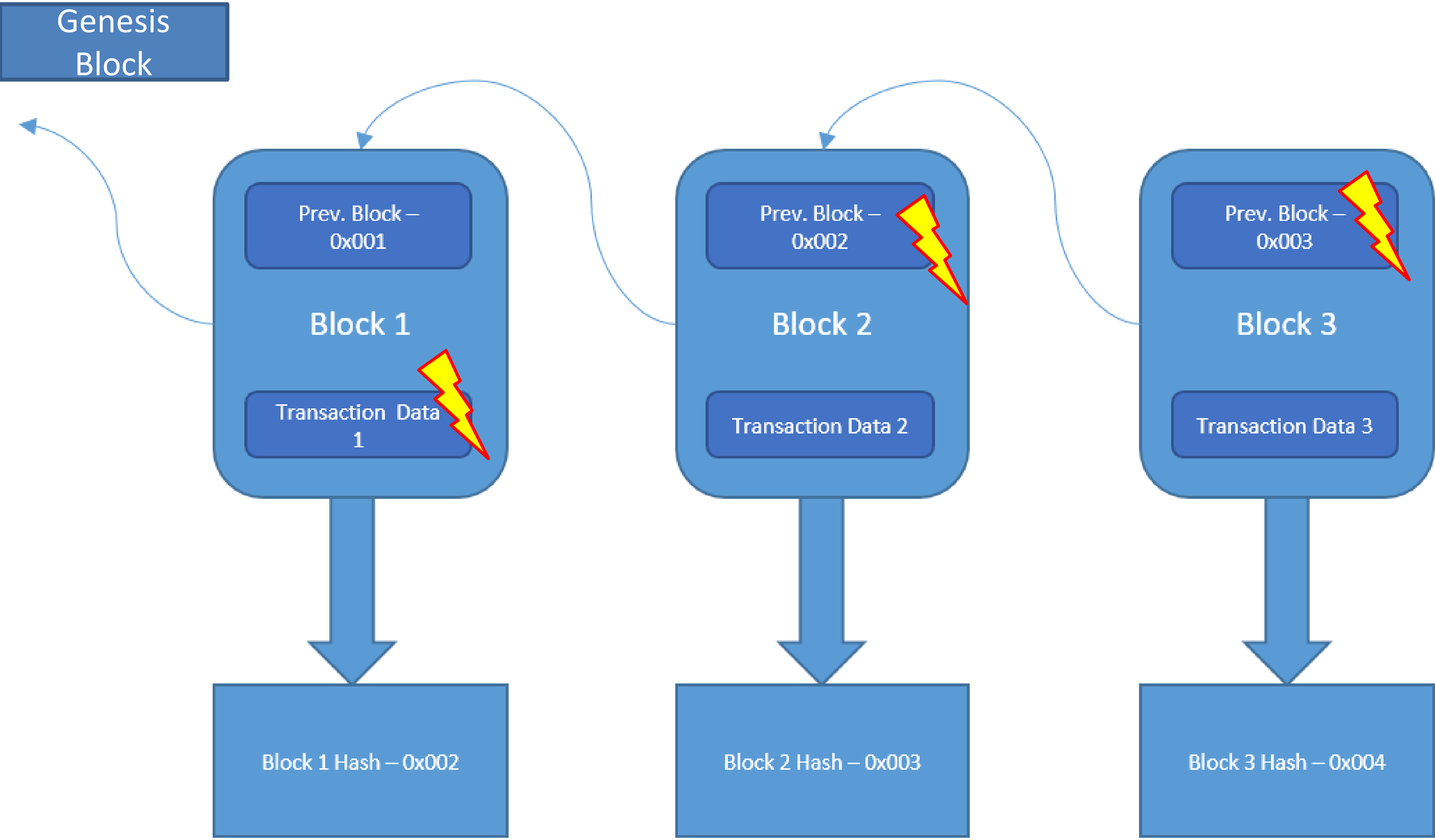
Verify function

Indeed signed by Alice

Digital Signatures

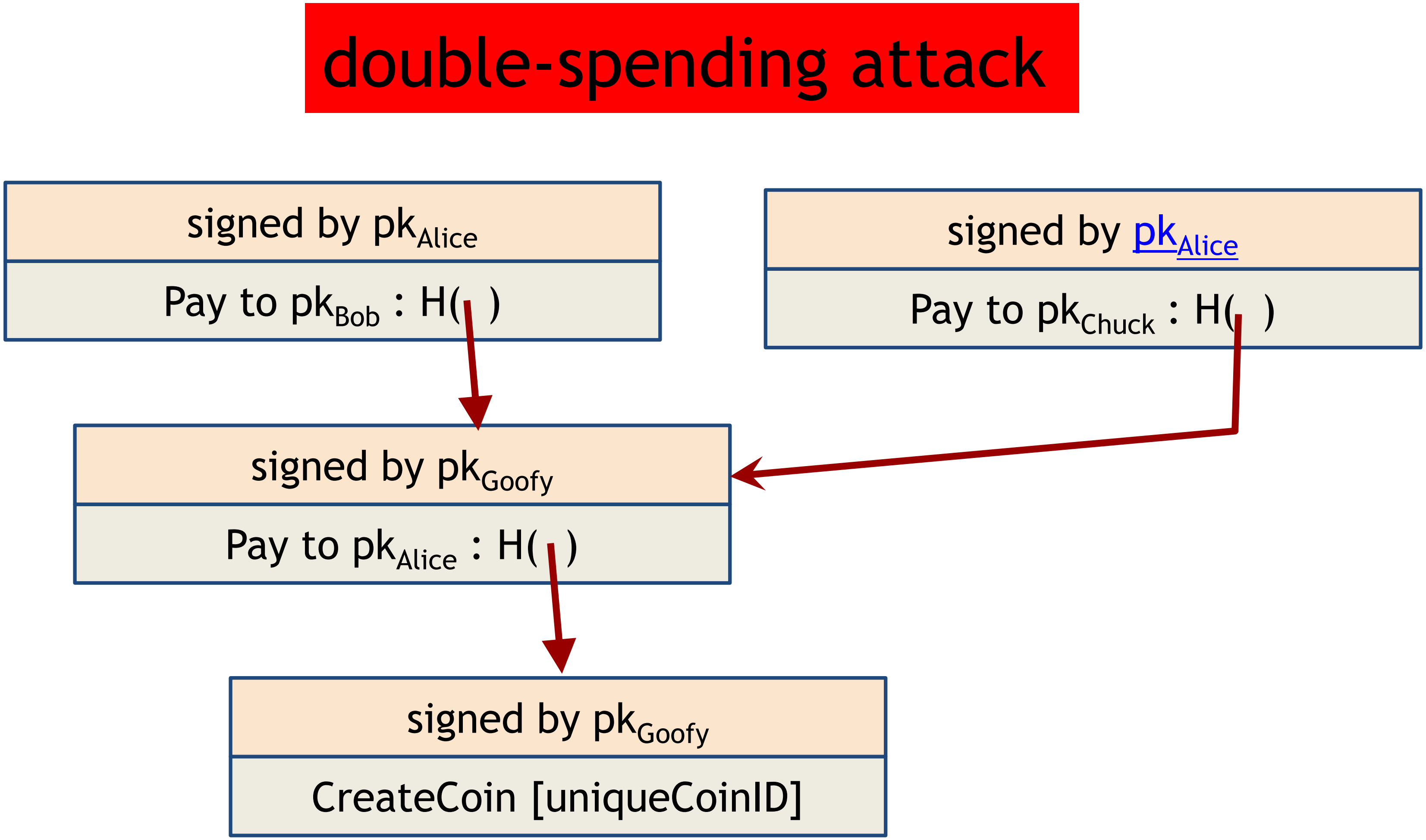


Blockchain

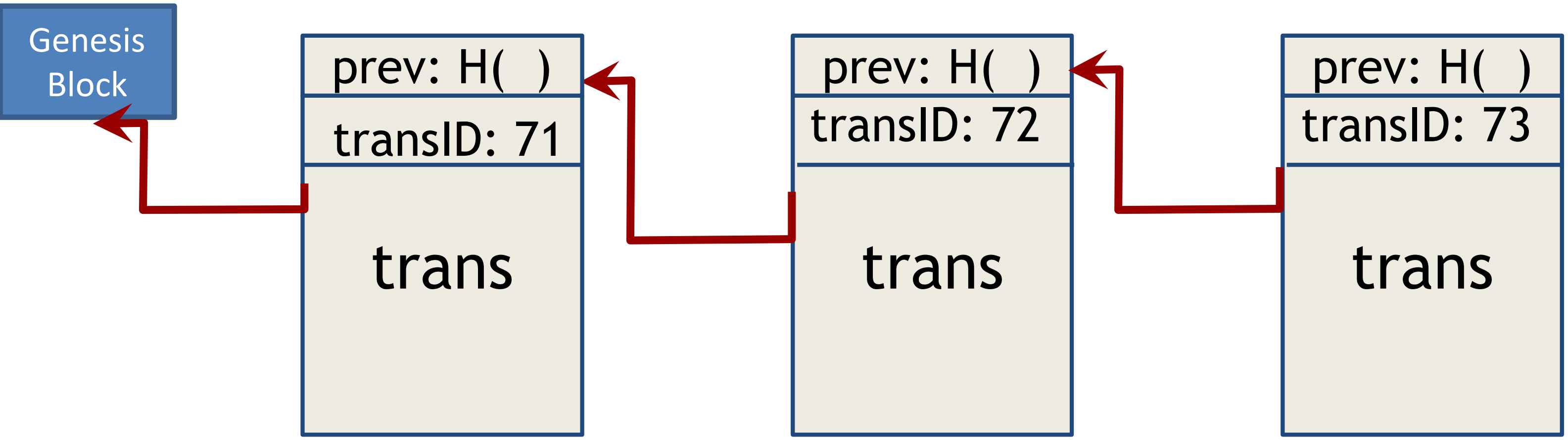


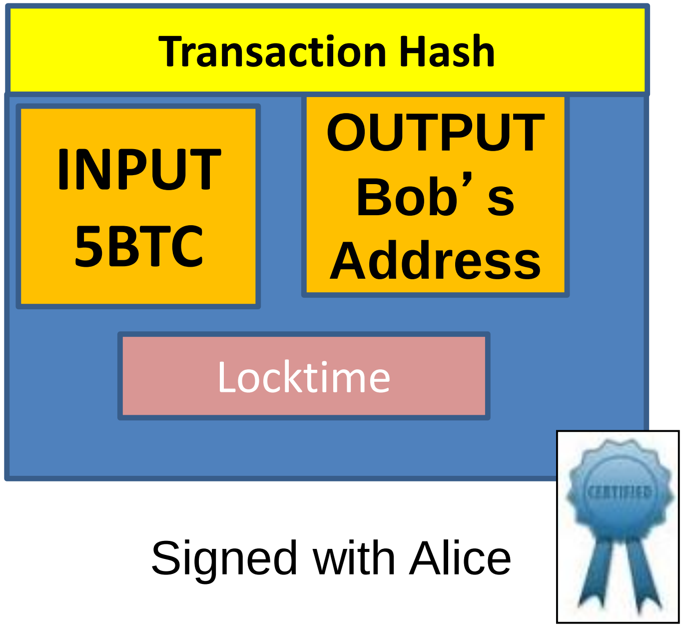
Hash pointer : Hash and address of previous block

Hypothetical Cryptocurrencies (Goofy Coin)



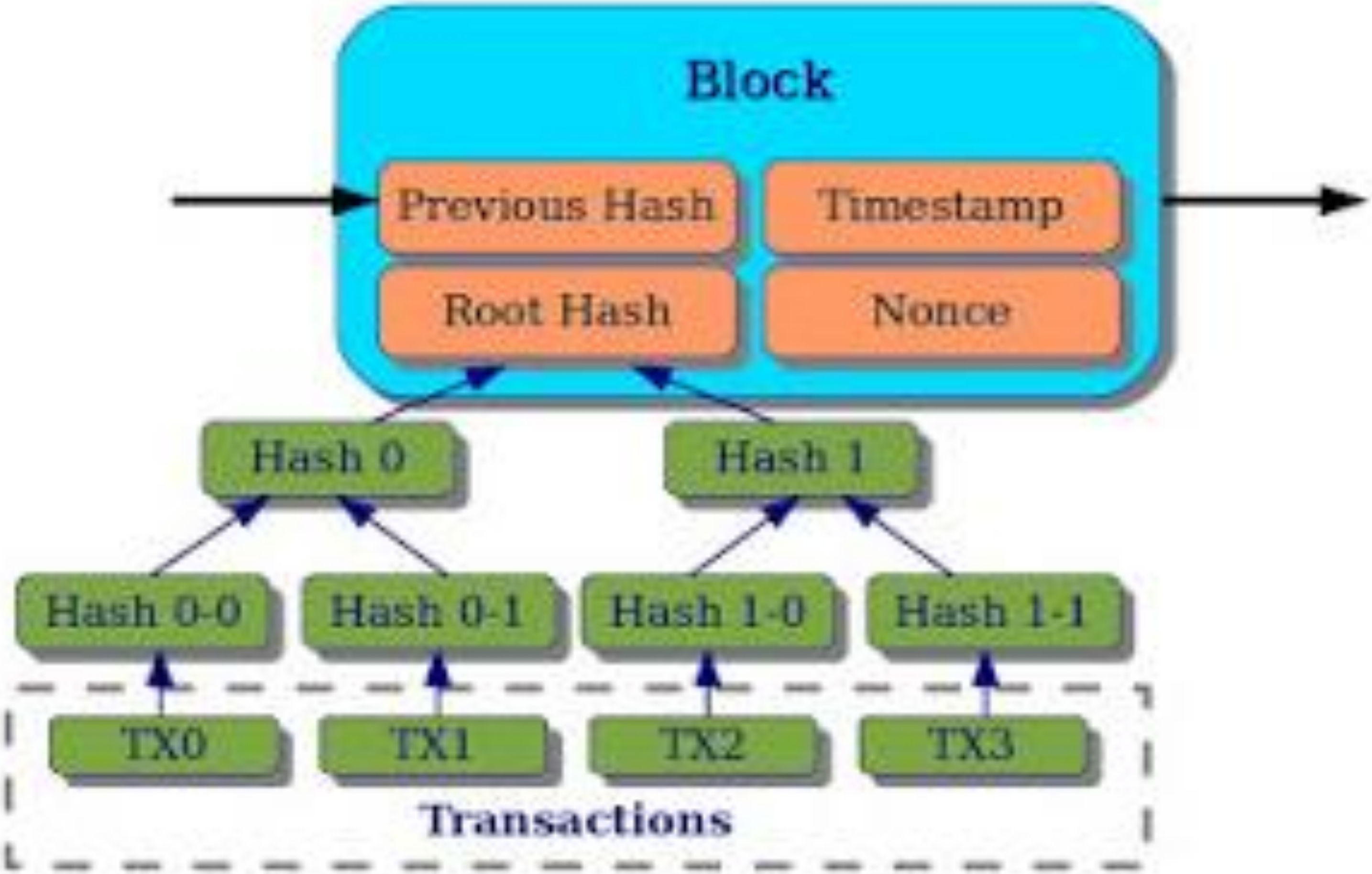
Centralised Goofy Coin



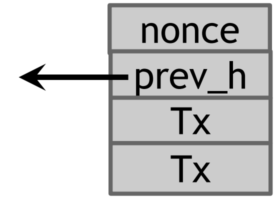


Signed with Alice

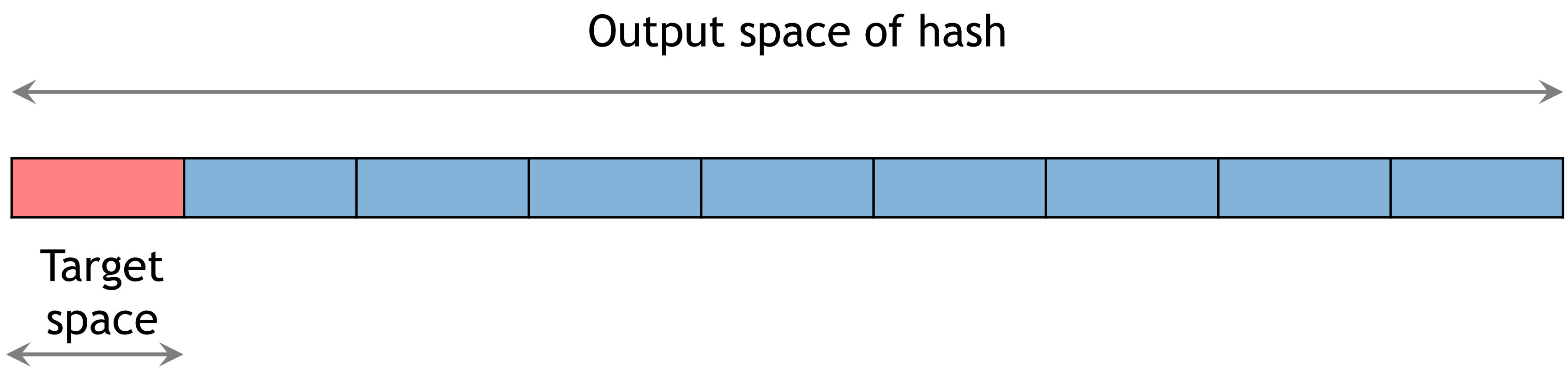
Blockchain


Block
Previous Hash
Timestamp
Root Hash
Nonce
Hash 0
Hash 1
Hash 0-0
Hash 0-1
Hash 1-0
Hash 1-1
TX0
TX1
TX2
TX3
Transactions

Hash puzzles:



To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small
SHA 256



The target originally started out at:
00000000ffff0000000000000000000000000000000000000000000000000000

The current target is:

0000000000000000001f0d9b0000000000000000000000000000000000000000

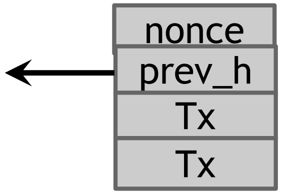
Decentralized Goofy Coin (Bitcoin)

Consensus Algorithm (Proof of Work):

1. New transactions are broadcast to all nodes.
2. Special nodes(miner) track transactions and add them to “candidate block”. Due to transaction ordering issues, candidate blocks in each miner may be different.
3. Special node which solves hash puzzle will be able to add new block into Blockchain. This node (called miner) will be rewarded 6.25 BTC(1bitcoin = 53,751$).

Hash puzzles:

To create block, find nonce s.t. Output space of hash
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small



4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)

5. Nodes express their acceptance of the block by working on creating the next block

- Disadvantage: requires huge amounts of energy. 54 TWe per year. Which is electricity requirement of New Zealand or Hungary.

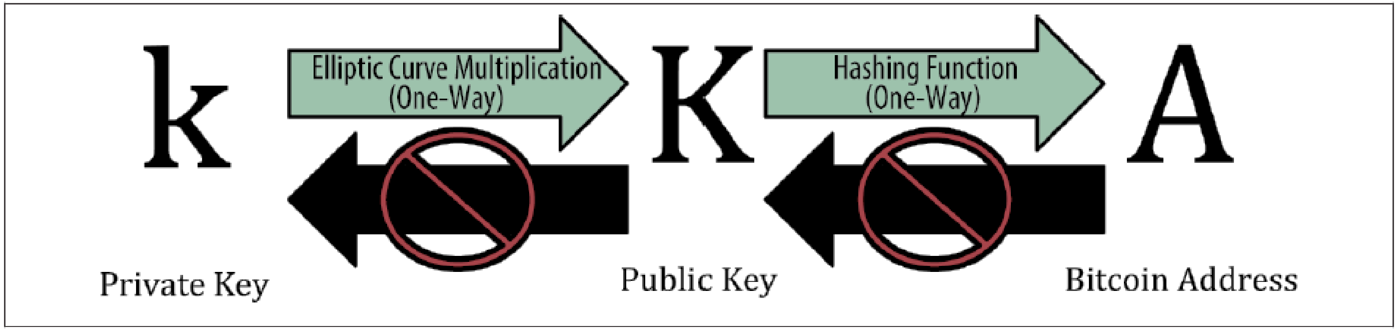
Proof of Stake (PoS)

- Attributing mining power to the proportion of coins held by a miner.
- Suppose Alice has 20% of the total coins and Bob has 15% of the total coins. Alice will be given 20% of total nodes and so Bob.
- One node is randomly chosen as minor.

- assumption that a majority of the wealth in the system is controlled by honest participants.
- The rationale behind PoS is that users who have significant stakes in the system have an economic incentive in keeping the system running according to the protocol specification, as they risk that their stakes will become worthless if trust in the cryptocurrency vanishes.

Bitcoin Wallet

- Uses Public Key Cryptography



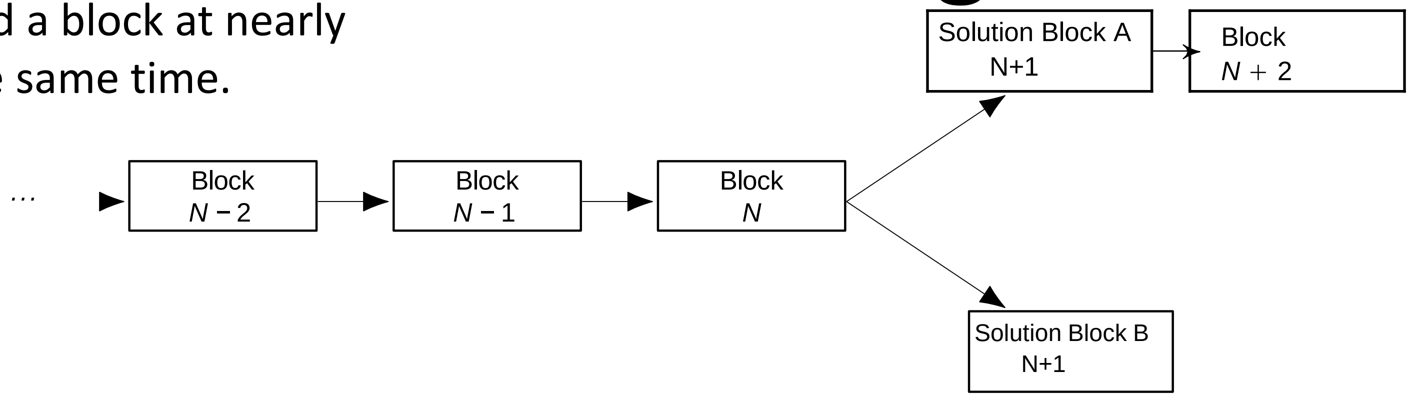
- To spend Bitcoin assigned to the address owner has to produce Signature corresponding to the address.

Threshold ECDSA for Securing Bitcoin Wallet

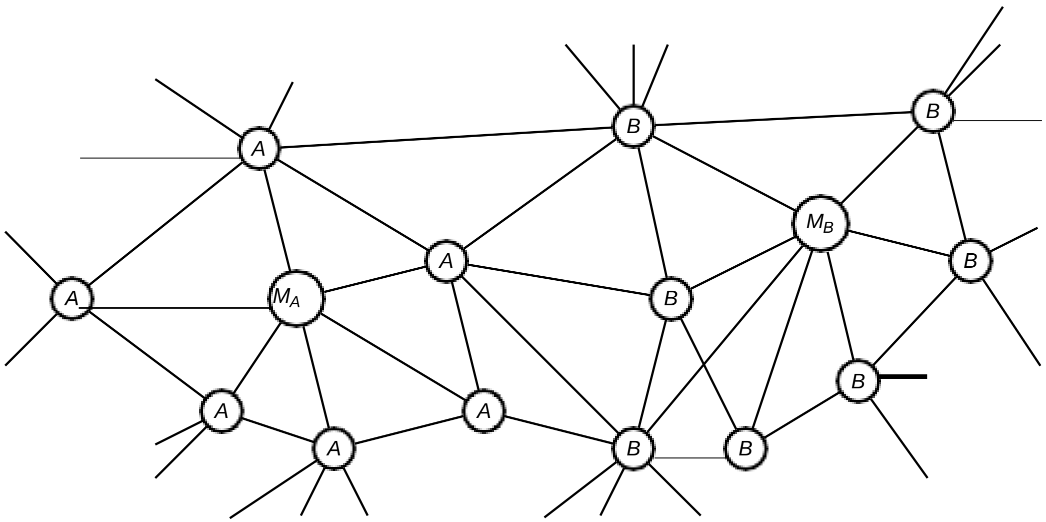
S Goldfeder etc al. presented (t, n)- threshold signature scheme which distributes signing power to n players. Any set of at least t players can generate a signature, whereas a group of less than t can not.

Forking

- *fork* happens when two or more miners find a block at nearly the same time.



- Both miners will broadcast their solution on the network
- Nodes will accept the first solution they hear and reject others



- Nodes always switch to the longest chain they hear
- The network abandons the blocks that are not in the longest chain (they are called *orphaned* blocks).

- Accidental bifurcation is therefore rare, and occurs on average once about every 60 blocks

- Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: IEEE P2P. (2013)

Total Bitcoin

- The rate at which the new Bitcoins are generated is designed to slowly decrease towards zero, and will reach zero when almost 21 million Bitcoins are created.
- Then, the miners’ revenue will be only from transaction fees

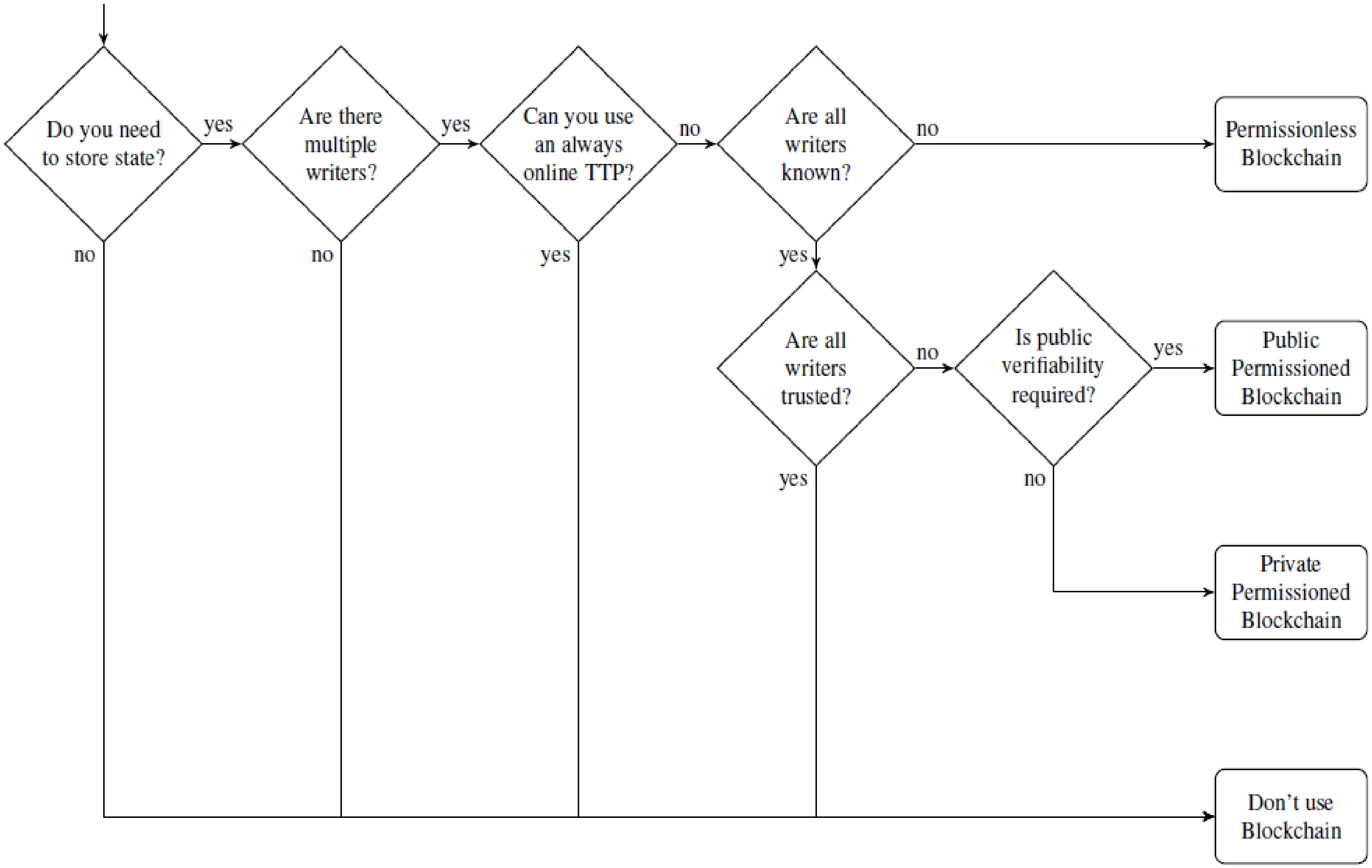
- Coins are divisible and transactions are multi-input and multi-output.

THANK YOU

Blockchain 2.0
Ethereum blockchain

Dr. Kunwar Singh
CSE Department
NIT Trichy

Blockchain: When to use



	Permissionless Blockchain	Permissioned Blockchain	Central Database
Throughput	Low	High	Very High
Latency	Slow	Medium	Fast
Number of readers	High	High	High
Number of writers	High	Low	High
Number of untrusted writers	High	Low	0
Consensus mechanism	Mainly PoW, some PoS	BFT protocols (e.g. PBFT [6])	None
Centrally managed	No	Yes	Yes

Blockchain 2.0

ethereum

BLOCKCHAIN APP PLATFORM

It’s more than cryptocurrency

Build unstoppable applications

Ethereum is a **decentralized platform that runs smart contracts** : applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference.

These apps run on a custom built **blockchain, an enormously powerful shared global infrastructure that can move value around and represen the ownership of property.**

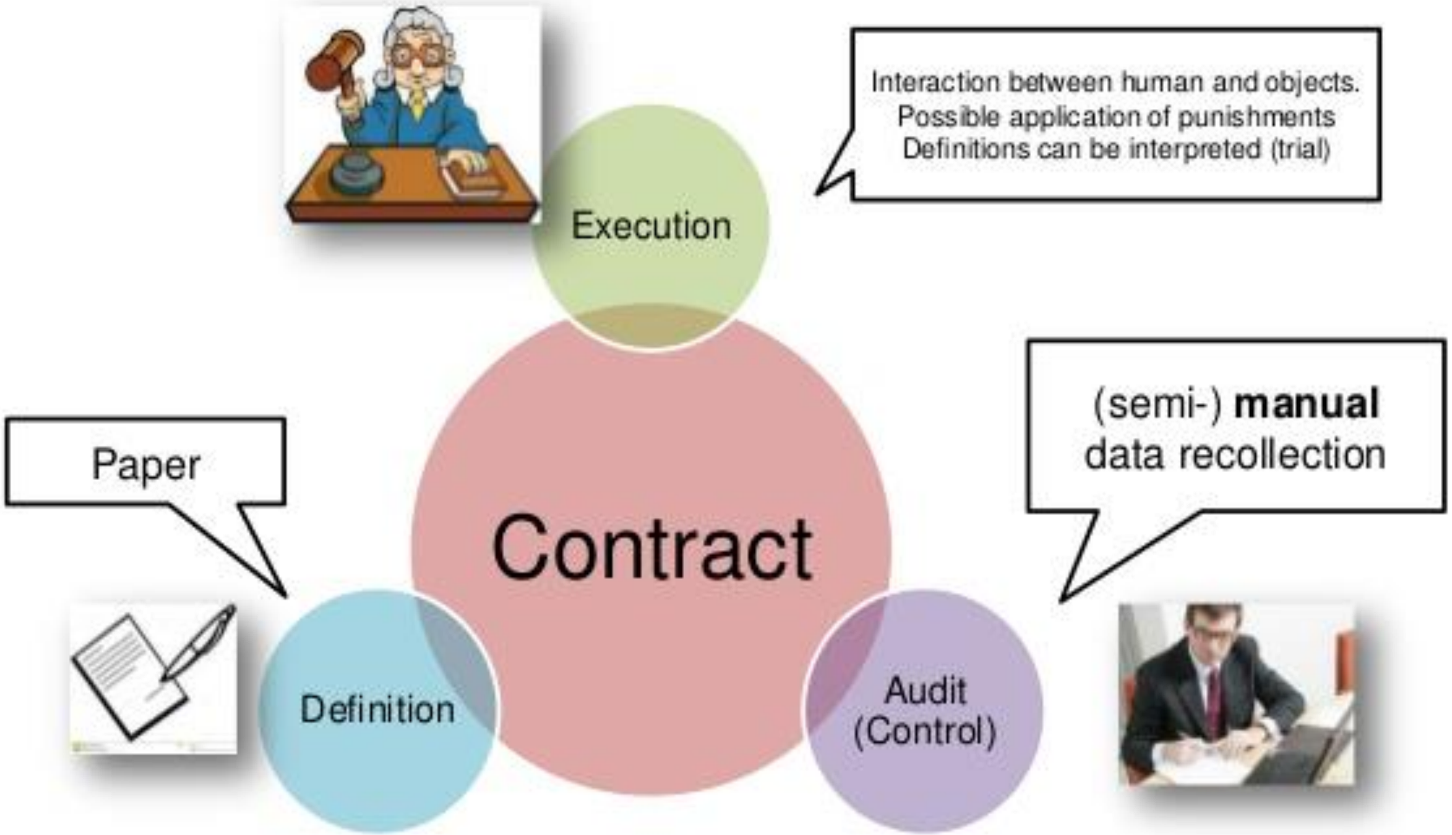
This enables developers to create markets, store registries of debts or promises, move funds in accordance with instructions given long in the pa (like a will or a futures contract) and many other things that have not been invented yet, all without a middleman or counterparty risk.

The project was bootstrapped via an ether presale in August 2014 by fans a around the world. It is developed by the Ethereum Foundation, a Swiss no profit, with contributions from great minds across the globe.

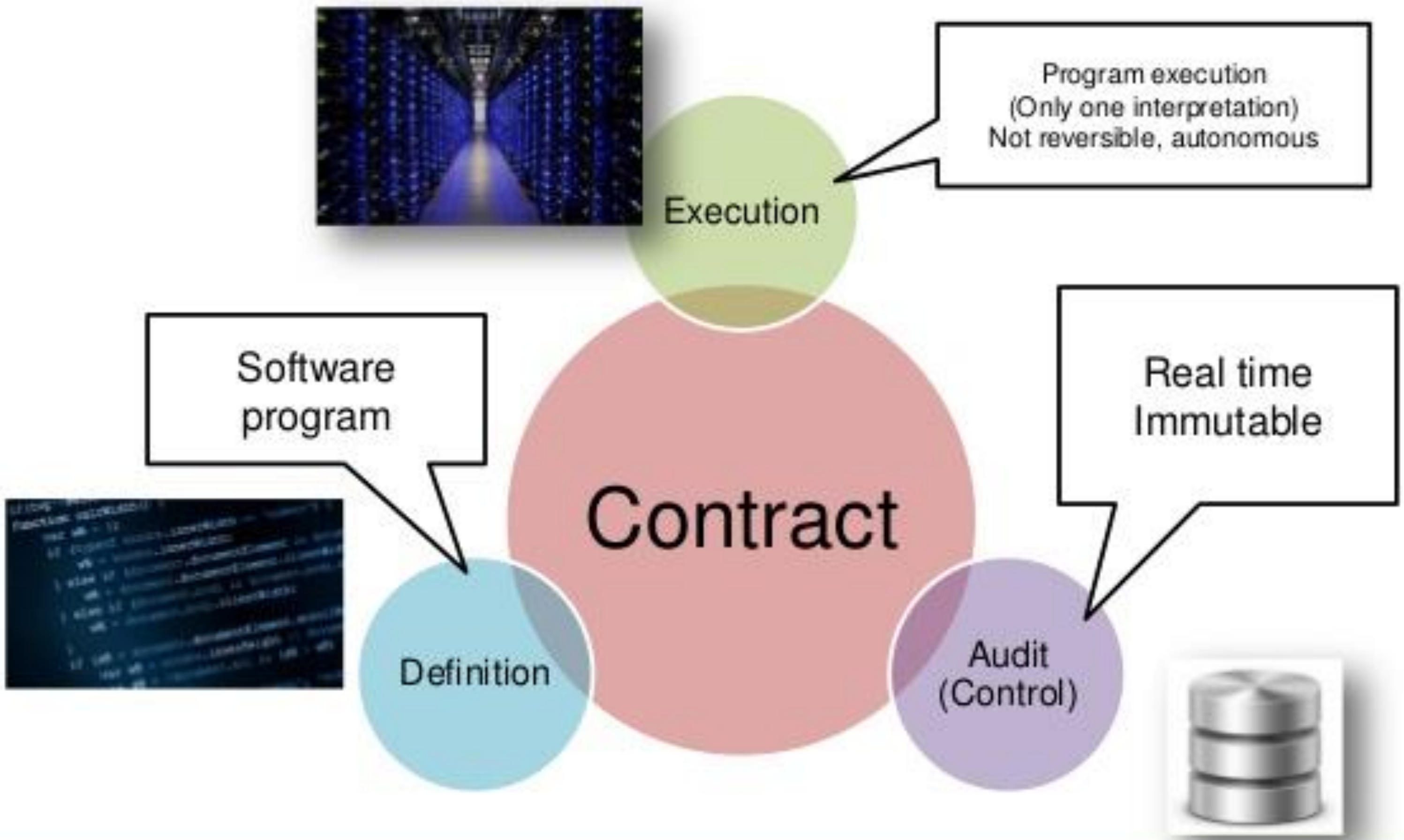
Ethereum blockchain

- Ethereum Blockchain is the most popular Blockchain for developing smart contracts.
- Ethereum is a public or private Blockchain with a built-in Turing-complete language to allow writing any smart contract.
- Ether is a cryptocurrency in Ethereum network.

«Traditional» contract

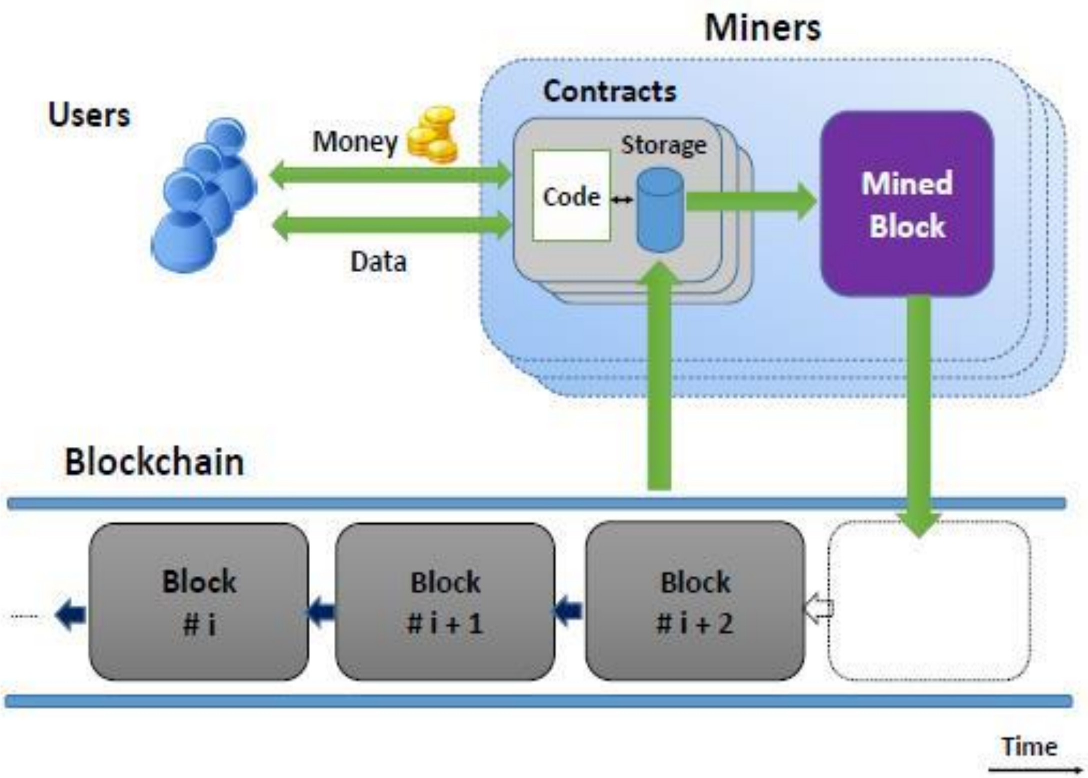


Smart contract



Smart Contracts (Nick Szabo in 1994)

- A smart contract is an executable code that runs on the Blockchain to facilitate, execute and enforce agreement between untrusted parties without the involvement of a trusted third party.
- Smart contracts can be called through transactions. Smart contracts are irreversible and immutable.



B. Smart Contract Programming

- Solidity is an object-oriented, contract oriented, high-level language for implementing smart contracts.
- Smart contracts are programs which govern the behavior of accounts within the Ethereum state.
- Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

contract Counter

- pragma solidity ^0.4.19;
- contract Counter

```
{    int private count = 0;
     function incrementCounter() public {   count += 1;    }
     function decrementCounter() public {    count -= 1;    }
     function getCount() public returns (int) {
     return count;    }
}
```

Pragma

- It is a directive that specifies the compiler version to be used for current solidity file.
- The version number comprises of two number- major build and minor build number

ex:

```
pragma Solidity ^0.4.19;
```

where 4 as major build and 19 as minor build number.

Functions

- When a function in a contract is called,it results in the creation of a transaction.
- The visibility of a function can be any one of the following :

public: function access can be directly from outside

internal: It means this function can be used within the current contract and any other contract that inherits from it.

Cont>>

private: private function can only be used in contracts declaring them. they can not be used even within derived contracts.

external: This makes function access directly from externally but not internally.

Default function is Public

Default variable is internal

Mapping

- Mapping is similar to hash table or dictionary in other languages storing key-value pair.

Constructor

- In other programming languages, a constructor is executed whenever a new object instance is created.

  However in Solidity, whenever smart contract is deployed into blockchain constructor function is instantiated (called).
- There can be atmost one constructor (unlike other languages)
- It does not return any data explicitly
- Has the same name as contract
- Can take parameters

contract Counter

- pragma solidity ^0.4.19;
- contract Counter

```
{    int private count;
     function Counter() public { count = 0;}
     function incrementCounter() public {   count += 1;    }
     function decrementCounter() public {    count -= 1;    }
     function getCount() public returns (int) {
     return count;    }
}
```

Minning

Consensus Algorithm (Proof of Work):

1. New transactions are broadcast to all nodes.
2. Special nodes(miner) track transactions and add them to "candidate block". Due to transaction ordering issues, candidate blocks in each miner may be different.
3. Special node which solves hash puzzle will be able to add new block into Blockchain. This node (called miner) will be rewarded 5 ether.
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)
5. Nodes express their acceptance of the block by working on creating the next block

Disadvantage: requires huge amounts of energy. 54 TWe per year. Which is electricity requirement of New Zealand or Hungary.

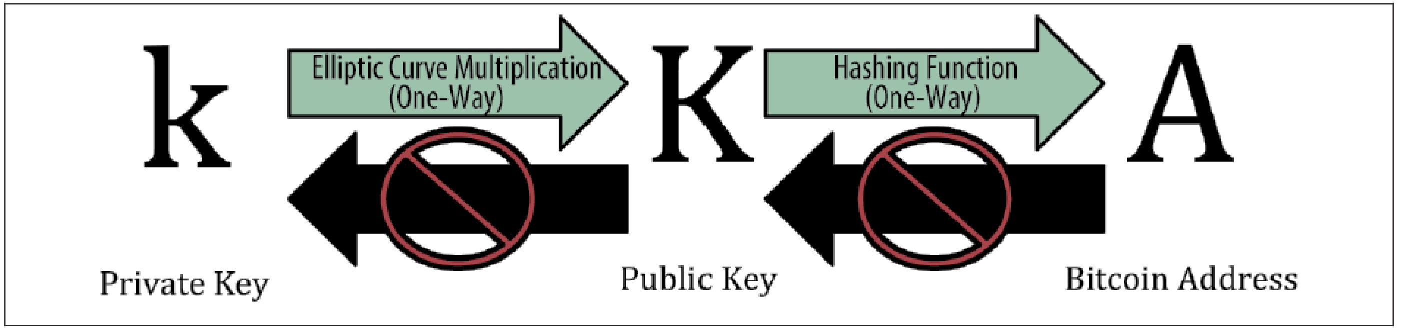
Proof of Stake (PoS)

- Attributing mining power to the proportion of coins held by a miner.
- Suppose Alice has 20% of the total coins and Bob has 15% of the total coins. Alice will be given 20% of total nodes and so Bob.
- One node is randomly chosen as minor.

- assumption that a majority of the wealth in the system is controlled by honest participants.
- The rationale behind PoS is that users who have significant stakes in the system have an economic incentive in keeping the system running according to the protocol specification, as they risk that their stakes will become worthless if trust in the cryptocurrency vanishes.

Wallet

- Uses Public Key Cryptography



- To spend ether assigned to the address owner has to produce Signature corresponding to the address.

Accounts and Wallets

- • Accounts:
  - Two Kinds:
    - **External Owned Accounts** - (**EOA**): owned by person
    - **Contract Accounts: owned by code**
  - Allow for interaction with the blockchain
- • Wallets:
  - A set of one or more external accounts
  - Used to store/transfer Ether

Gas and Gas Cost

- Problem: Cannot tell whether or not a program will run infinitely or program has flawed code. DoS attack
- Solution: charge fee per computational step to limit infinite loops and stop flawed code from executing
- **Gas Price**: current market price of a unit of Gas (in Wei). Depend on market price of ether.
- **Gas Limit**: maximum amount of Gas user is willing to spend
- **Gas Cost** (used when sending transactions) is calculated by **gas used*gasPrice**
- **Gas used**
    - normal transaction - 21,000
    - smart contracts: depends on resources consumed - instructions executed and storage used
- **What if gas limit < gas used in that transaction?**

Unit	Wei
Wei	1
Kwei / ada / femtotether	1,000
Mwei / babbage / picoether	1,000,000
Gwei / shannon / nanoether / nano	1,000,000,000
Szabo / microether / micro	1,000,000,000,000
Finney / milliether / milli	1,000,000,000,000,000
Ether	1,000,000,000,000,000,000

The Address Functions

Address provides single property and five functions.

Property: It’s possible to get the Ether balance in Wei (the smallest Ether denomination) associated with an address by querying the balance property.

```
contract AddressExamples
{
    address ownerAddress =
    0x10abb5EfEcdC09581f8b7cb95791FE2936790b4E;

    function getOwnerBalance() public returns (uint)  {
        uint balance = ownerAddress.balance;
        return balance;
    }
}
```

transfer()	*To transfer Ether in Wei*—If the transaction fails on the receiving side, an exception is thrown to the sender and the payment is automatically reverted (although any spent gas isn't refunded), so no error handling code is necessary; transfer() can spend only up to a maximum of 2300 gas.
send()	*To send Ether in Wei*—If the transaction fails on the receiving side, a value of false is returned to the sender but the payment isn't reverted, so it must be handled correctly and reverted with further instructions; send() can spend only up to a maximum of 2,300 gas.
call()	*To invoke a function on the target contract associated with the address (the target account is assumed to be a contract)*—An amount of Ether can be sent together with the function call by specifying it in this way: call.value(10)("contractName", "functionName"); call() transfers the entire gas budget from the sender to the called function. If call() fails, it returns false, so failure must be handled in a similar way to send().

- destinationAddress.transfer(10);

- if (!destinationAddress.send(10))

  revert();

- destinationContractAddress.call.value(10)(

  "contractName", "functionName");

Transactions

- A request to modify the state of the blockchain
- Launched by an EOA (external transaction) or Contract account

- Types

  1. Fund transfer between EOAs:
     1. EOA to EOA (ether)
     2. EOA to Contract account
     3. Contract account to Contract account
     4. Contract account to EOA

From	Fund sender, an EOA (20-byte address)
To	Fund recipient, another EOA (20-byte address)
Value	Amount, in weis
Data / Input	Empty
Gas Limit	Larger enough for an ether transfer transaction
Gas Price	To be determined by transaction initiator

2. Execute a function on a deployed contract:

    Executing a function in contract that changes state is considered a transaction

3. Deploy a contract on Ethereum network

A simple wallet contract.

```
contract Awallet
   {
      address owner;
     mapping (address => uint) public outflow;
     function AWallet(){ owner = msg.sender; }
     function pay(uint amount, address recipient) returns (bool)
      {
        if (msg.sender != owner || msg.value != 0) throw;
        if (amount > this.balance) return false;
        outflow[recipient] += amount;
        if (!recipient.send(amount)) throw;
         return true;
        }
  }
```

Smart contract that rewards users who solve a computational puzzle

```
contract Puzzle{
  address public owner;
  bool public locked;
  uint public reward;
  bytes32 public diff;
  bytes public solution;

  function Puzzle() //constructor{
    owner = msg.sender;
    reward = msg.value;
    locked = false;
    diff = bytes32(11111); //pre-defined difficulty
  }

  function(){ //main code, runs at every invocation
    if (msg.sender == owner){ //update reward
      if (locked)
        throw;
      owner.send(reward);
      reward = msg.value;
    }
    else
      if (msg.data.length > 0){ //submit a solution
        if (locked) throw;
        if (sha256(msg.data) < diff){
          msg.sender.send(reward); //send reward
          solution = msg.data;
          locked = true;
        }}}}
```

contract Election

- pragma solidity 0.4.20;
- contract Election {   // Model a Candidate
  struct Candidate { uint id;   string name;   uint voteCount; }

// Store accounts that have voted
mapping(address => bool) public voters;

// Store Candidates
// Fetch Candidate
mapping(uint => Candidate) public candidates;

// Store Candidates Count
uint public candidatesCount;

// voted event
event votedEvent (uint indexed _candidateId);

```
function Election () public
  {  addCandidate("Candidate 1");
     addCandidate("Candidate 2");  }

function addCandidate (string _name) private {
 candidatesCount ++;
 candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);    }

function vote (uint _candidateId) public
  {   // require that they haven't voted before
   require(!voters[msg.sender]);

   // require a valid candidate
   require(_candidateId > 0 && _candidateId <= candidatesCount);

  // record that voter has voted
   voters[msg.sender] = true;

 // update candidate vote Count
   candidates[_candidateId].voteCount ++;

  // trigger voted event
   votedEvent(_candidateId);
 }
```

Fallback Function

- In solidity, fallback function is declared without any name. This function cannot have arguments and cannot return anything.
- A contract can have exactly one unnamed function. It is executed on a call to the contract if none of the other functions match the given function identifier (or if no data was supplied at all).
- Furthermore, this function is executed whenever the contract receives plain Ether (without data). Additionally, in order to receive Ether, the fallback function must be marked payable. If no such function exists, the contract cannot receive Ether through regular transactions.
- In the worst case, the fallback function can only rely on 2300 gas being available (for example when send or transfer is used), leaving not much room to perform other operations except basic logging.

- Like any function, the fallback function can execute complex operations as long as there is enough gas passed on to it.

Gasless send

- 2300 units of gas only allow to execute a limited set of byte- code instructions, e.g. those which do not alter the state of the contract. To send the ether 2300 units of gas is enough. In any other case, the call will end up in an out-of-gas exception.

```
contract C {
    function pay(uint n, address d){
      d.send(n);  } }

contract D1 {
   uint public count = 0;                    // fails with an out-of-gas exception
   function() public payable { count++; }
 }

contract D2 { function() public payable {} }              //   succeds
```

DAO

- DAO stands for decentralized autonomous organization and created by the Slock.it. It was a virtual venture capital fund that is governed by the investors of the DAO.
- idea: Funds raised from the investors, the token holders, are pooled. These money can be used to fund their startup and other startups of their choice (voting).
- The DAO launched on 30th April, 2016
- The DAO contract raised about $150M before being attacked
- However, on 16 June 2016, the DAO got hacked and siphon off $60 Million.

Reentrancy

- Because the fallback function an attacker was able to re-enter the caller function.

- ```
  contract Bob {
  bool sent = false;
  function ping(address c) {
  if (!sent) {
  c.call.value(2)();
  sent = true;
  }}}
  ```

- ```
  contract Mallory {
  function() {
  Bob(msg.sender).ping(this);
  } }
  ```

The DAO Code (simplified)

```
contract SimpleDAO {
mapping (address => uint) public credit;
    function donate(address to){credit[to] += msg.value;}
    function queryCredit(address to) returns (uint){
        return credit[to];
    }
    function withdraw(uint amount) {
        if (credit[msg.sender]>= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender]-=amount;
}}}
```

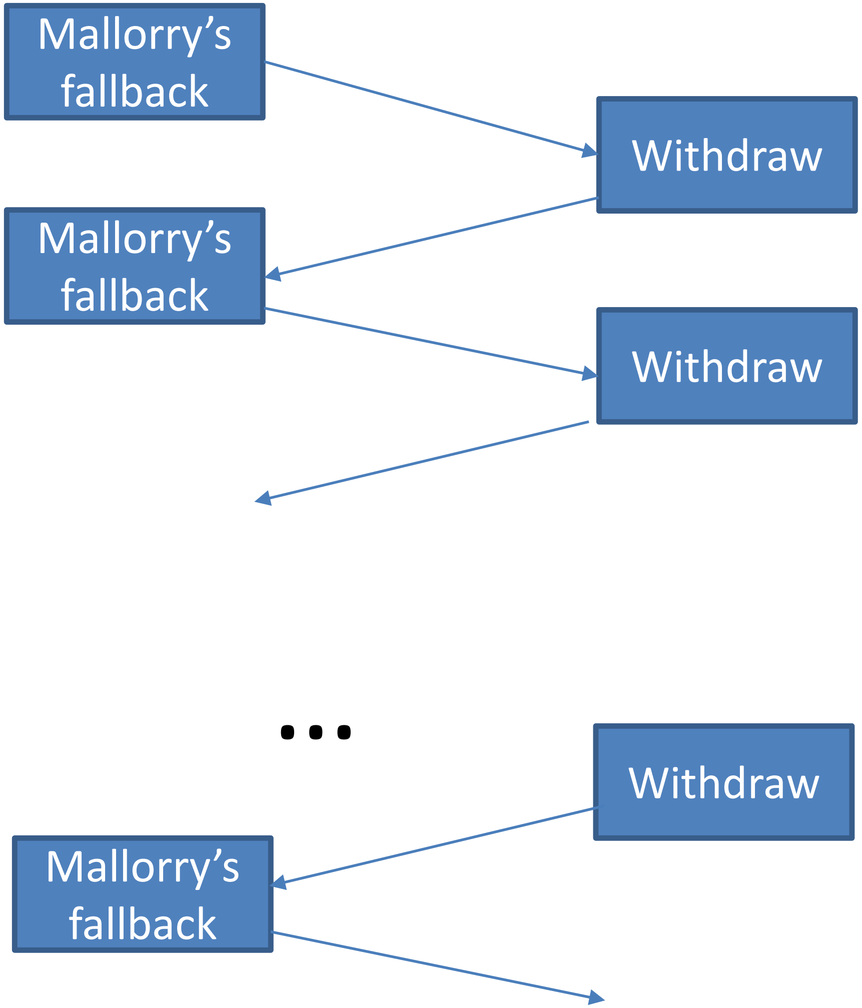
The DAO Attack (Simplified Version)

- To perform the attack:
  - Deploy a contract Mallory shown right
  - Donate some Ether for Mallory and invoke the withdraw() function

```
contract Mallory {
    SimpleDAO public dao = SimpleDAO(0x354...);
    address owner;
    function Mallory(){owner = msg.sender; }
    function() { dao.withdraw(dao.queryCredit(this)); }
    function getJackpot(){ owner.send(this.balance); }
}
```

Looping until:

- Out of gas
- Stack limit is reached
- the balance of DAO becomes zero



The Hard-Fork Proposal

- to ask the miners to completely unwind the theft and return all ether to The DAO, through forking. This way they managed save some money

THANK YOU

Bitcoin Wallet

Dr. Kunwar Singh
CSE Department
NIT Trichy

Hot storage

online

Cold storage

offline

hot secret key(s)

address(es)

payments

cold secret key(s)

address(es)

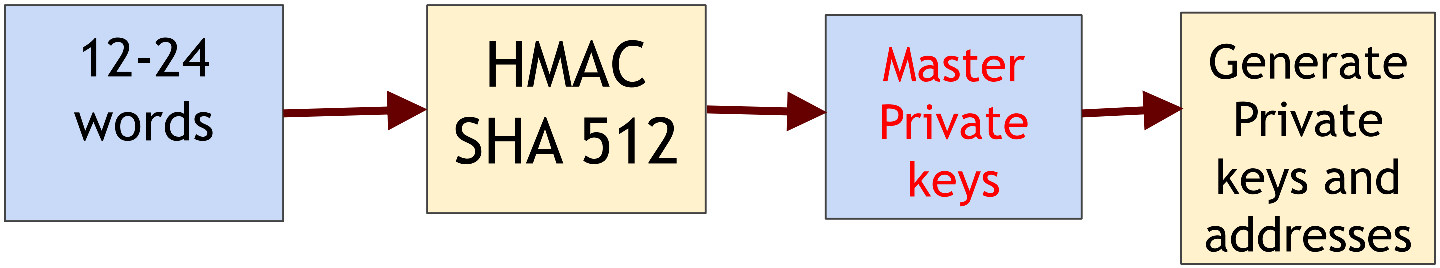
Non-Detrministic Wallet

- Generates private key 1…address1

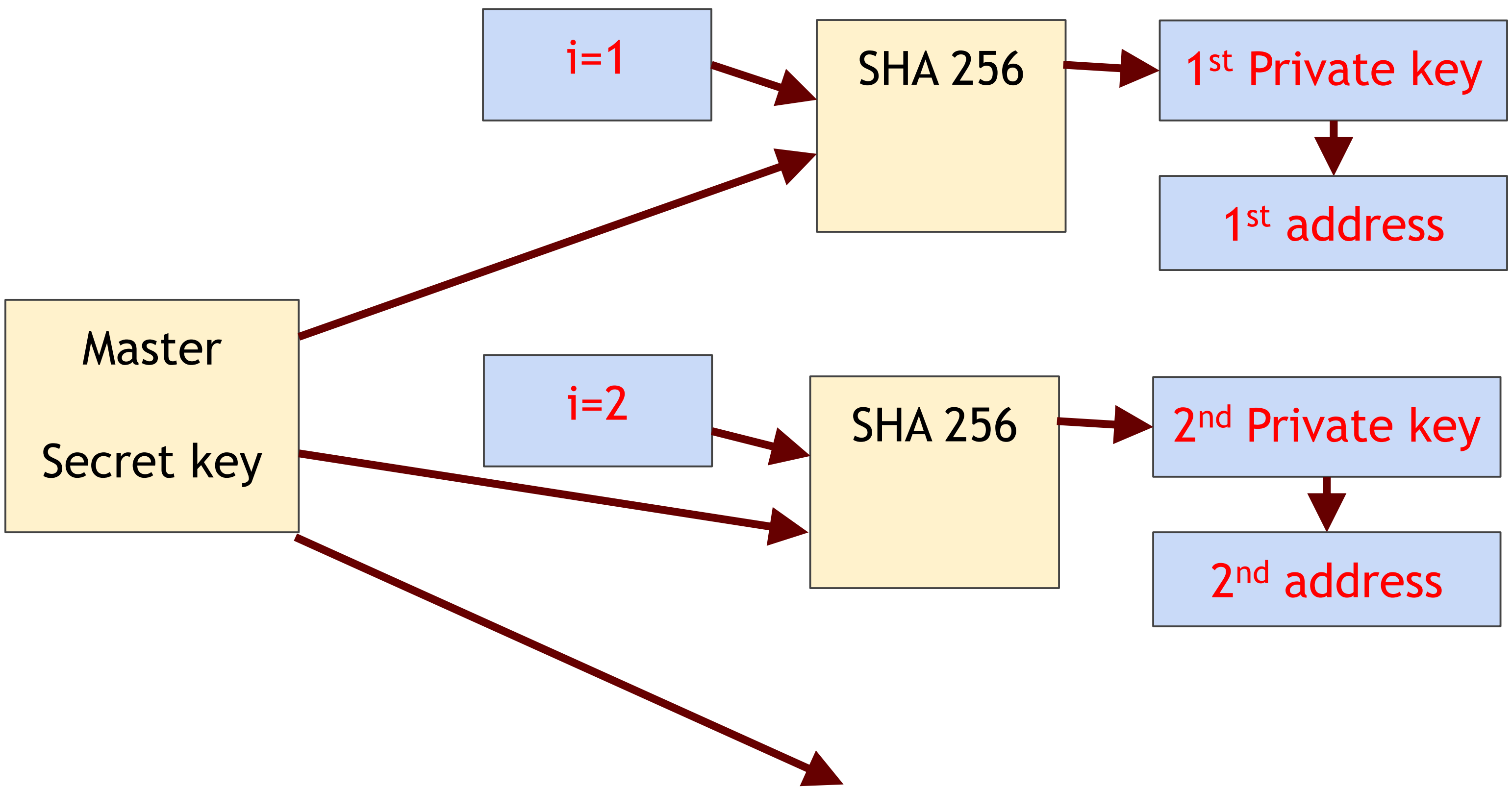
  Generates private key 2…address2

- Private keys are not related to each other.

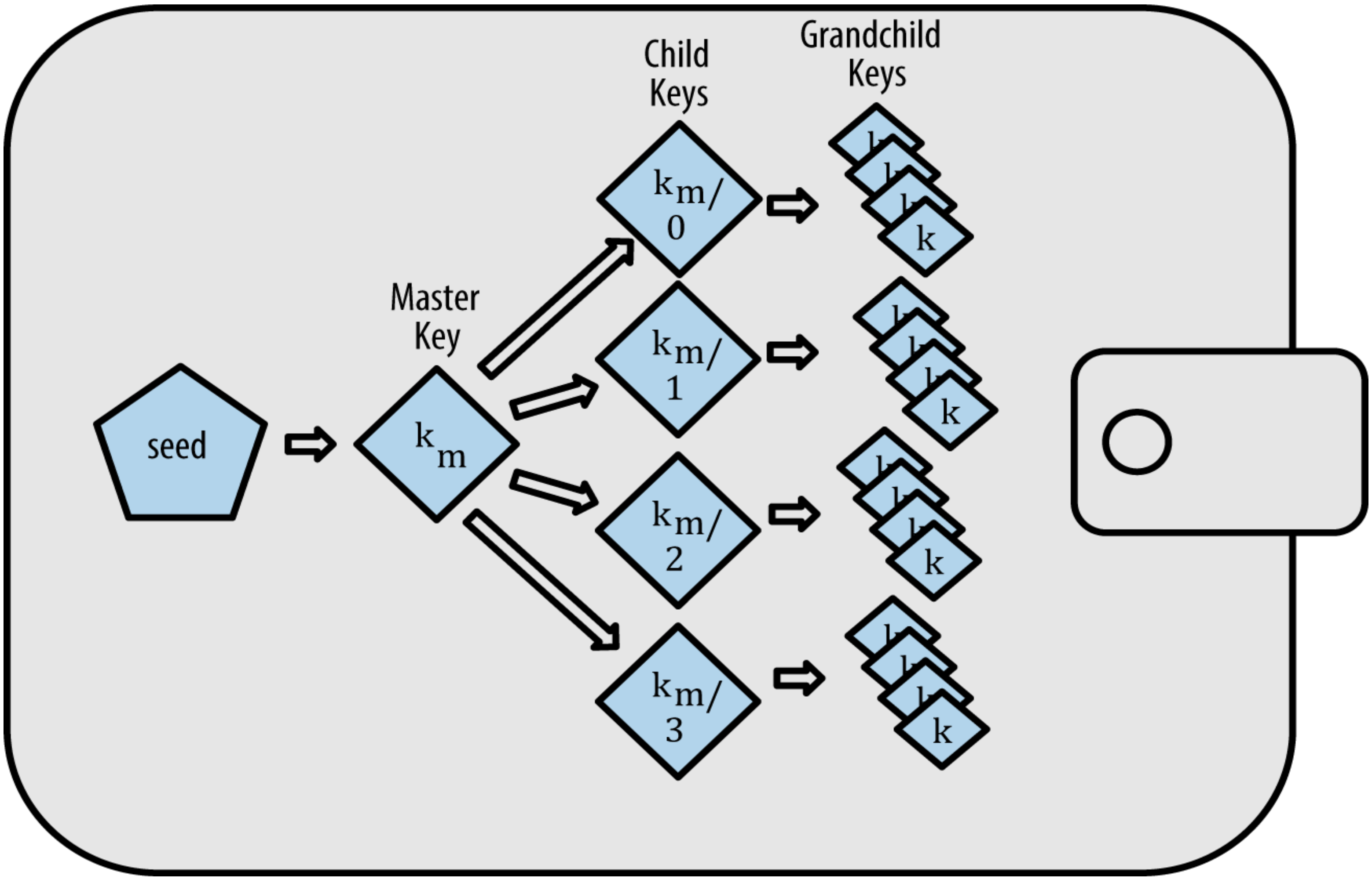
Deterministic Wallet



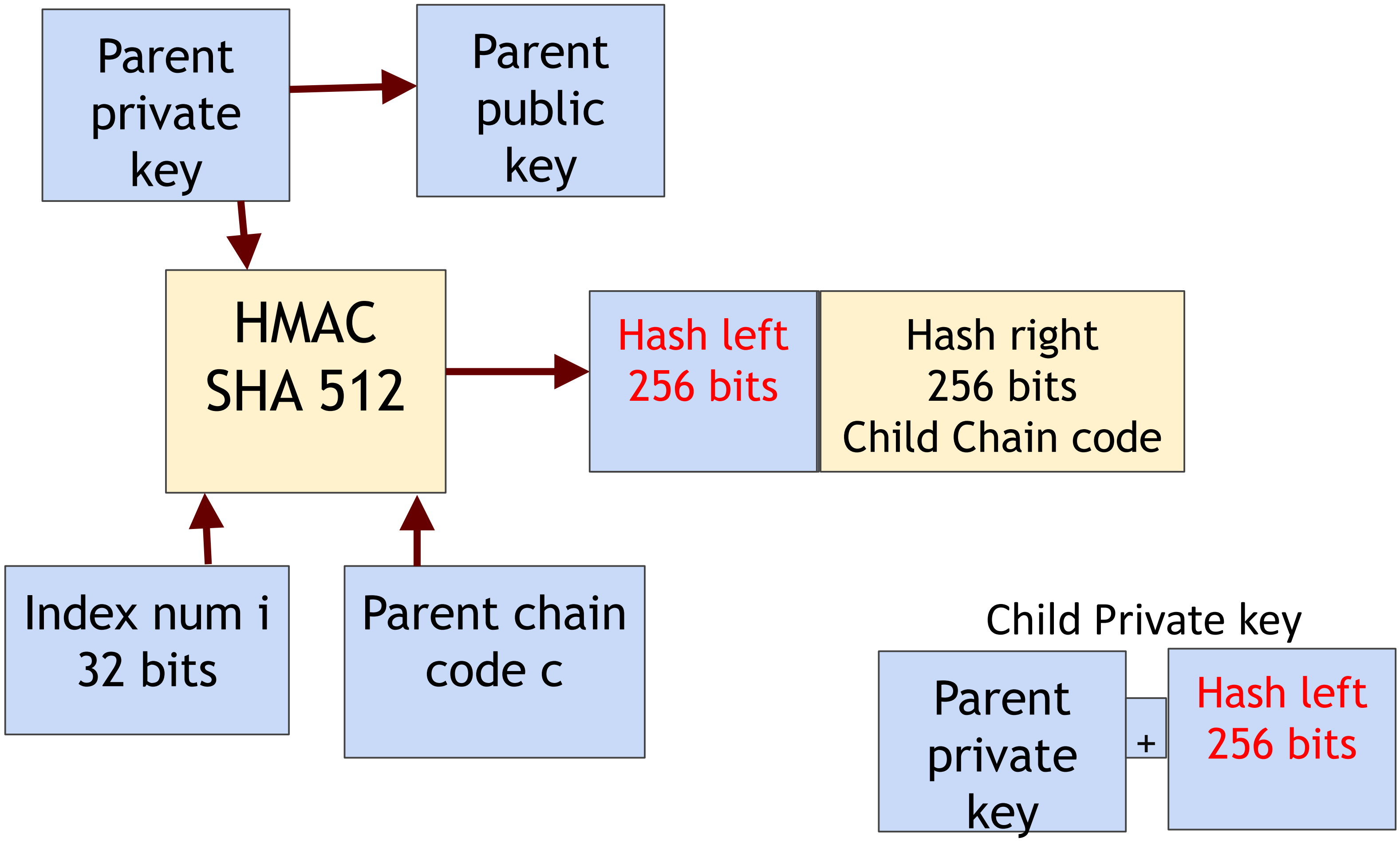
Sequential Deterministic Wallet



Hierarchical Deterministic Wallet



Hierarchical Deterministic Wallet



Back up Schemes for Secret Key

- Shamir Secret Sharing:

The main idea is:

2 points are sufficient to define a unique line,

3 points are sufficient to define a unique parabola,

4 points to define a unique cubic curve

…..

Theorem: Given k points, there is one and only one polynomial of degree $k-1$.

Shamir Secret Sharing (n,k)

- Goal is to create n- secret shares of the secret S such that at least k shares are required to compute secret S.

- Dealer D pick a random polynomial of degree k-1.

$$P(x) = S + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

Where $a_1, a_2, \ldots, a_{k-1}$ random numbers

Shares of the Secrete

The polynomial $P(x) = S + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$ over $\mathbb{Z}_p$

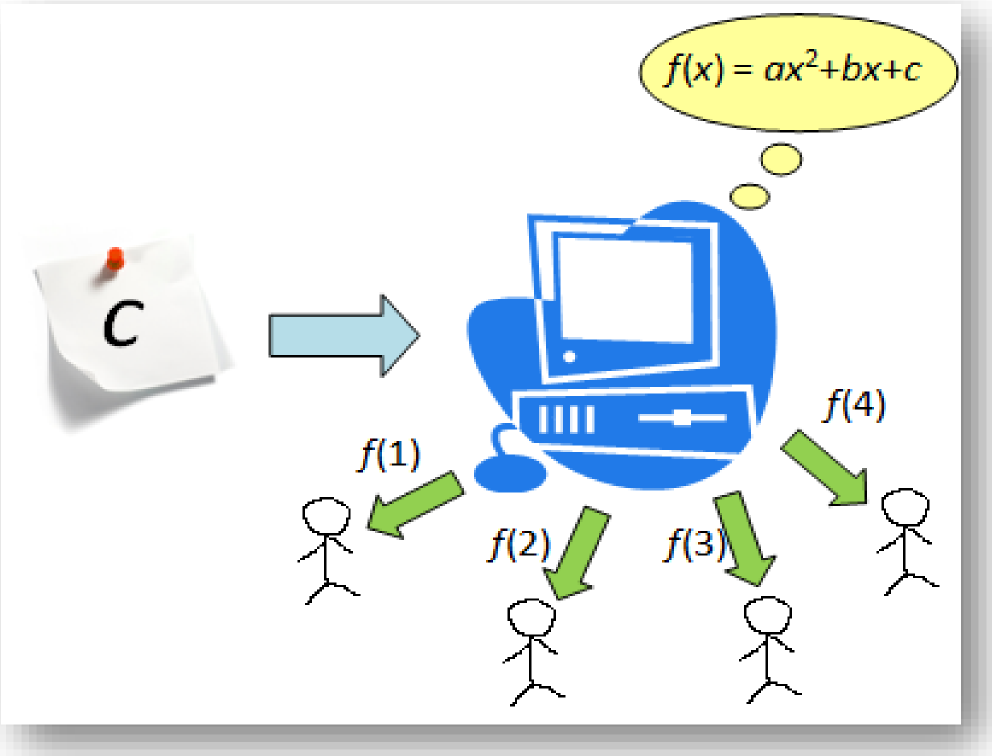
Then, compute

$$P_i = P(i)(\text{mod } p) \text{ for } i = 1, 2, 3, \dots, n$$

n shares are n ordered pair of points (i, P_i).

These n shares are distributed to n participants or shareholders.

(4,3)



Reconstructing the Secret from k Shares

Let $G \subset \{1, 2, 3, \dots, n\}$ contain exactly k elements

Lagrange interpolation:

Let $H = (i, P_i)$ for $i \in G$

$$P(x) = \sum_{i \in H} P_i \prod_{j \in H, j \neq i} \frac{x - j}{i - j}$$

Thus

$$S = P(0) = \sum_{i \in H} P_i \prod_{j \in H, j \neq i} \frac{-j}{i - j}$$

is the original secret.

Example:

Let $S = 9, p = 41$ and the threshold value be $k = 3$

Choose at random a_1 and a_2 in $\mathbb{Z}_{41}$. For example $a_1 = 2$ and $a_2 = 31$

Now we have $P(x) = 9 + 2x + 31x^2$ over $\mathbb{Z}_{41}$

Then generate as many share as we wish. For example if $n = 7$ we have

$(1, P_1) = (1, 1)$, $(2, P_2) = (2, 14)$, $(3, P_3) = (3, 7)$,
$(4, P_4) = (4, 21)$ $(5, P_5) = (5, 15)$, $(6, P_6) = (6, 30)$,
$(7, P_7) = (7, 25)$

Reconstruction:

Suppose we have 3 shares $H = \{(1,1),(3,7),(7,25)\}$ then we can reconstruct the secret from

$$S = \sum_{i \in H} P_i \prod_{j \in H, j \neq i} \frac{-j}{i-j}$$

over $\mathbb{Z}_{41}$. Hence,

$$S = 1\frac{(-3)(-7)}{(1-3)(1-7)} + 7\frac{(-1)(-7)}{(3-1)(3-7)} + 25\frac{(-1)(-3)}{(7-1)(7-3)}$$

$$S = \frac{7}{4} + \frac{49}{-8} + \frac{25}{8} = \frac{48}{4} - \frac{8}{8} - \frac{16}{8} = 9$$

Weighted Threshold ECDSA for Securing Bitcoin Wallet

- Goldfeder et al. (2014) threshold ECDSA for Bitcoin wallet. In Goldfeder et al., all the players get equal shares of the secret key.

- We have proposed two provably secure weighted threshold ECDSA for bitcoin security (ISEA Asia Security and Privacy (ISEASP 2017)

1. In our scheme, each player is given one or more shares of the secret key according to his weightage/priority. There are total n shares distributed among m players.

$$w_1 + w_2 + \cdots + w_m = n$$

t (≤ k) players can compute the secret if

$$w_1 + w_2 + \cdots + w_m \geq k$$

Weighted Threshold ECDSA for Securing Bitcoin Wallet

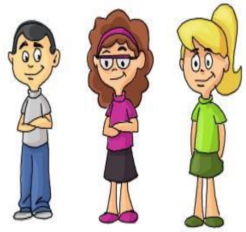
2. There are many players having same weightage. Hence grouped together.

(n,6)

w = 1

w = 2

w = 3



Signature Vulnerability of ECDSA

Signature Generation (d,m)

Keys = (d,Q)

1. A random number k, $1 \le k \le n-1$ is chosen
2. $kG = (x_1,y_1)$ is computed. x_1 is converted to its corresponding integer x_1'
3. Next, $r = x_1 \bmod n$ is computed
4. We then compute $k^{-1} \bmod q$
5. $e = HASH(m)$ where m is the message to be signed
6. $s = k^{-1}(e + dr) \bmod n$

We have the signature on m as (r,s)

Verification (m,r,s,Q)

1. Verify whether r and s belong to the interval [1, n-1] for the signature to be valid.

2. Compute e = HASH(m).

3. Compute $w = s^{-1} \bmod n$.

4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.

5. Compute $(x_1,y_1) = u_1G + u_2Q$.

6. The signature is valid if $r = x_1 \bmod n$, otherwise invalid.

Signature Vulnerability

- Suppose user signs different messages m_1 and m_2 with the same nonce k. User signs m_1 as (r_1 s_1) and m_2 as (r_2,s_2) with $r_2 = r_1 = r$
- $s_1 = k^{-1}(e_1 + dr)$ $\qquad$ $s_2 = k^{-1}(e_2 + dr)$

  $s_1 - s_2 = k^{-1}(e_1 - e_2)$

  Compute k.

  Can compute d from $\qquad$ $s_1 = k^{-1}(e_1 + dr)$

Back up Scheme

- Back up generation:

  Get signature on M and store back up as (M,s). Nonce k used in this signature also be used in one of signed transactions.

Recovery key

- Collect transactions' signature denoted as $(r_{1,}\ s_1),\ldots,(r_{t,}\ s_t)$ and their corresponding message $(m_{1,\ldots,}m_t)$.
- For each i from 1 to t do

  $k_i = (H(m_i) – H(M) / (s_i - s)$

  Compute $k_iG = (x_i, y_i)$, and convert x_i to an integer x_i ,

  If $r_i = = x_i$ mod n then

  Compute $d = (s_i\ k_i - H(M))r_i^{-1}$ mod n.

THANK YOU

Definitions

Let G_1 and G_2 be abelian groups, written additively.

Let n be a prime number such that $[n]P$ for all P in G_1 and G_2 .

Let G_3 be a cyclic group of order n, written multiplicatively.

Then a pairing is a map:

$$e: G_1 \times G_2 \to G_3$$

(*Bilinearity*)

$$e(P+P',Q) = e(P,Q)e(P',Q) \quad \text{for all } P,P' \in G_1, Q \in G_2$$
$$e(P,Q+Q') = e(P,Q)e(P,Q') \quad \text{for all } P \in G_1,\ Q,Q' \in G_2$$

(*Non-Degeneracy*)

$$\text{For all non-zero } P \in G_1, \text{ there is a } Q \in G_2 \text{ such that } e(P,Q) \neq 1.$$
$$\text{For all non-zero } Q \in G_2, \text{ there is a } P \in G_1 \text{ such that } e(P,Q) \neq 1.$$

Properties of Bilinear Pairings

1) $e(P,0) = e(0,Q) = 1$

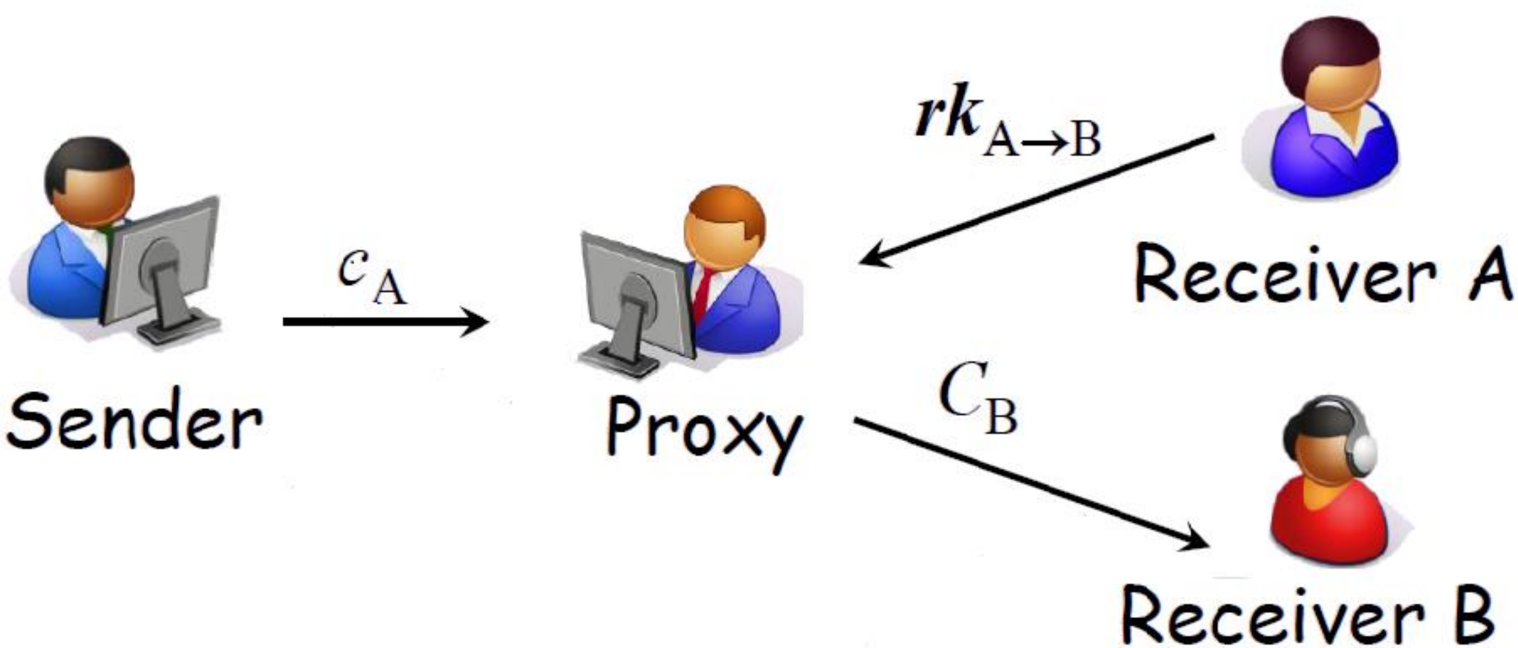
2) $e(-P,Q) = e(P,Q)^{-1} = e(P,-Q)$

3) $e([a]P,Q) = e(P,Q)^{a} = e(P,[a]Q)$ for all $a \in \mathbf{Z}$

4) $e([a]P,[b]Q) = e(P,Q)^{ab}$ for all $a,b \in \mathbf{Z}$

Proxy re-encryption:

Blaze, Bleumer and Strauss(BBS) presented a new primitive called proxy re-encryption in1998. PRE allows semi trusted proxy to convert a ciphertext for Alice into a ciphertext for Bob without knowing the message.



Uni-Directional Proxy Re-Encryption

- **Key Generation:**
- $< g > =$ G$_1$ of prime order q.
- $SK_a = a \in \; Z_q^*$, $SK_b = b \in \; Z_q^*$
- $PK_a = g^a$, $PK_b = g^b$
- **Re-encryption key:**
- $RK_{A \to B} = (g^b)^{1/a} = g^{b/a}$
- **Encryption:**
- $m \in \; G_2$
- $C_a = (Z^r.m, g^{ra})$ compute $Z = e(g, g)$ where e(g,g) is a bilinear pairing
- **Re-Encryption:**
- $C_a = (Z^r.m, g^{ra})$
- $C_b = (Z^r.m, e(g^{ra}, RK_{A \to B})) = (Z^r.m, e(g^{ra}, g^{b/a}) = (Z^r.m, Z^{rb})$
- **Decryption:**
- $m = \frac{Z^r.m}{(Z^{rb})^{1/b}}$.

Secure Distributed Medical Record Storage Using Blockchain and Emergency Sharing using Multiparty Computation (BSC 2021)

- Providing hospitals with quick access to the medical records of patients who are in **critical** condition. This enables doctors to diagnose the problem more efficiently and give the best treatment
- Sharing of medical records needs to be done in a **Secure** manner without leaking data to potential adversaries.
- The sharing should be secure regardless which hospital the patient is admitted to.
- The sharing needs to be done with minimal Patient interaction

IPFS (Inter Planetary File System)

- is a protocol and peer-to-peer network for storing and sharing data in a distributed file system
- The medical files are uploaded to IPFS, and it returns a hash. Using the hash, the file can be identified using the provided IPFS gateway.
- Storing files in blockchain directly is **very expensive**. It takes roughly about **$100** for storing a **120 kB PDF file** into blockchain ,but storing 256bit Hash string returned from IPFS into blockchain takes only **2.7 cents** based on the gas prices.

- Medical report is encrypted using public key of the person.
- Encrypted medical report is kept in IPFS
- Person wants to show the medical report to a hospital.
- Using hospital's public key, person computes the proxy re-encryption key and gives to the hospital.
- Hospital re-encrypts the encrypted medical report.
- Hospital decrypt the re-encrypted medical report using its private key.

- L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,CCS '16, pp. 254-269, ACM, 2016.

- A. Juels, A. Kosba, and E. Shi, “The ring of gyges: Investigating the future of criminal smart contracts," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 283-295, ACM, 2016