GCD(a,b)

```
func gcd(a, b int) int {
    for b != 0 {
        a, b = b, a%b
    }
    return a
}
```

Extended Euclidean Algorithm

$ax + by = \gcd(a, b)$

Def: A is the modular inverse of B mod n if AB mod n = 1.

There exists a modular inverse for B mod n iff B is relatively prime to n. ($\gcb(b, n) = 1$)

Euler's Function

$\phi(n) = n(1 - 1/p_1)(1 - 1/p_2) \cdots (1 - 1/p_k)$

1. The Euler phi function is multiplicative. I.e. if $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m) \times \phi(n)$.
2. For a prime p and an integer $e \geq 1$, $\phi(p^e) = p^{e-1}(p - 1)$.

- if a and n are co-prime, then $a^k \bmod n = a^{k \bmod \phi(n)} \bmod n$

Fermat's Little Theorem

Let p be a prime. Any integer a not divisible by p satisfies $a^{p-1} \equiv 1 (\bmod p)$.

- Compute $11^{1,073,741,823} \bmod 13$.

let $p = 13, \quad 13 \bmod 11 \neq 0$, So:

$$11^{13-1=12} \equiv 1 (\bmod 13)$$

Raise both sides to a large power. First, find that power: $\frac{1,073,741,823}{12} = 89478485 \cdots\cdots 3$

$$(11^{12})^{89478485} \equiv 1^{89478485} (\bmod 13)$$

$$11^{1073741820} \equiv 1 (\bmod 13)$$

$$11^3 \times 11^{1073741820} \equiv 11^3 (\bmod 13) \equiv 1331 (\bmod 13) \equiv 5 (\bmod 13)$$

$$11^{1,073,741,823} \bmod 13 = 5$$

Modular exponentiation

1. Repeated Modular Multiplication (as defined)

$a^e \bmod n$ is defined as repeated multiplications of a for e times modulo n; Complexity = $O(e)$

2. Square-and-Multiply Algorithm

$11^{15} \bmod 13 = 11^8 \times 11^4 \times 11^2 \times 11 \bmod 13$, Complexity = $O(|e|)$ or $O(\lg(e))$

```
func modularExp(a, e, n int) int {
    z := 1
    for i := e; i > 0; i /= 2 {
        if i%2 != 0 {
            z = (z * a) % n
        }
        a = (a * a) % n
    }
    return z
}
```

Determine whether 227 and 79 are relatively prime.

gcd(227,79)=gcd(79,69)=gcd(69,10)=gcd(10,9)=1 So they are relatively prime.

Find the multiplicative inverse of 79 mod 229.

Like Q1, gcd(79,229)=1.

229=79 x 2+71

79=71+8

71=8 x 8 +7

8=7+1

reverse 1 into the form: ax + by

1=8-7=8-(71-8x8)=9x8-71=9x(79-71)-71=9x79-10x71=9x79-10x(229-79x2)=29x79-10x229

$1 \bmod 229 = 1 = 29 \times 79 \bmod 229$

29 is the modular inverse of 79 modulo 229.

Without calculating anything, by simply looking at the numbers, can you tell whether 7932 has a multiplicative inverse mod 11958? Explain.

No. because they are not relativity prime since they can all divided by 2.

Show the steps of how to calculate $\phi(730)$.

$730 = 2 \times 5 \times 73$

$\phi(730) = 1 \times 4 \times 72 = 228$

Calculate $227^{54996213} \bmod 21$ as efficient as possible.

227 and 21 are co-prime

$a^k \bmod n = a^{k \bmod \phi(n)} \bmod n$

$227^{54996213} \mod 21 = 227^{54996213 \mod \phi(21)} \mod 21 = 227^9 \equiv 17^9 \mod 21 \equiv 20$

Abelian group

- Abelian Group Definition

1. **Closure**: For all elements a, b in the group, the result of $a * b$ must also be in the group.
2. **Associativity**: For all elements a, b, and c in the group, $(a * b) * c = a * (b * c)$.
3. **Identity Element**: There exists an element e in the group such that for every element a in the group, $a * e = e * a = a$.

4. **Inverse Element**: For each element a in the group, there exists an element b in the group such that $a * b = b * a = e$, where e is the identity element.
5. **Commutativity**: For all elements a, b in the group, $a * b = b * a$.

e.g: {1,2,3,4} with operator * (mod 5) is an abelian group. However, {0,1,2,3} is not

- Multiplicative Groups $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n | \gcd(a, n) = 1\}$
- Cyclic Groups

If there is an element $g \in G$, for each $a \in G$, there is an integer i with $a = g^i$, that is g operates on it self for i times. -> genrator: g

e.g: (Z_7^*, x) is a cyclic multiplicative group with g=3

i	1	2	3	4	5	6
$g^i \bmod 7$	3	2	6	4	5	1

$$\mathbb{Z}_n^* \text{ has a (at least one) generator if and only if } n = 2, 4, p^k, \text{ or } 2p^k,$$

where (p) is an odd prime and ($k \geq 1$).

is $(\mathbb{Z}_5, +)$ cyclic? give a genrator of the group

The element 1 generates the group as $1 + 1 = 2, 2 + 1 = 3, 3 + 1 = 4, 4 + 1 = 0$ (all modulo 5).

Similarly, the element 2 generates the group as $2 + 2 = 4, 4 + 2 = 1, 1 + 2 = 3, 3 + 2 = 0$ (all modulo 5).

Hence, the generators for $(\mathbb{Z}_5, +)$ are $\{1, 2, 3, 4\}$.

is $(\mathbb{Z}_8^*, \times)$ cyclic? give a genrator of the group

The elements of $(\mathbb{Z}_8^*)$ are $\{1, 3, 5, 7\}$. To check for cyclicity:

- The element 1 is not a generator since $1 \times 1 = 1$.
- The element 3 is not a generator because $3^2 = 9 \equiv 1 \mod 8$, and it does not generate all elements.
- The element 5 is not a generator because $5^2 = 25 \equiv 1 \mod 8$, and it also does not generate all elements.
- The element 7 is not a generator because $7^2 = 49 \equiv 1 \mod 8$, and it fails to generate all elements.

Since none of the elements generate the entire group, ((\mathbb{Z}_8^*, \times)) is not cyclic.

is $\mathbb{Z}_6^*$ forms a cyclic group?

$\mathbb{Z}_6^* = \{a \in \mathbb{Z}_6 | \gcd(a, 6) = 1\} = \{1, 5\}$

- $5^1 \equiv 5 \mod 6$
- $5^2 \equiv 25 \equiv 1 \mod 6$

$\mathbb{Z}_6^*$ is cyclic with 5 as a generator.

Generate Primes

- Security Parameter: Security parameter is an integer value (denoted by n) that parameterizes both cryptographic schemes as well as all involved parties
- Efficient Algorithms: There is some polynomial such that the algorithm runs for time at most $p(n)$ when the security parameter is n.
- Negligible Probability: **"small probabilities" with probabilities smaller than any inverse polynomial in n. Such probabilities are called negligible.**

$f : \mathbb{Z}^+ \to \{0\} \cup \mathbb{R}^+$ is negligible , if for every positive polynomial p and all sufficiently large values of n it holds that $f(n) < \frac{1}{p(n)}$.

E.g: $f(n) = 2^{-n}$ is a negligible function.

```
func generateRandomPrime(n int) (int, error) {
    if n >= 31 { // 31-bit is the practical limit for a signed int
        return 0, fmt.Errorf("bit length too large for int type")
    }

    var p int
    max := 1 << n       // 2^n
    min := 1 << (n - 1) // Ensure n-bit length

    rand.Seed(time.Now().UnixNano())

    for i := 0; i < 3*n*n; i++ {
        p = rand.Intn(max-min) + min
        p |= 1 // Ensure p is odd

        if isProbablyPrime(p, n) {
            return p, nil
        }
    }

    return 0, fmt.Errorf("failed to generate a prime number")
}
```

Cryptographic Hardness Assuptions

- **Gengroup:** A generic, polynomial-time, group-generation algorithm that, on input 1^n, outputs a description of a cyclic G, its order q, and a generator $g \in G$.
- **Discrete Logarithm:** Let G be a cyclic group, for every $h \in G$, there is a unique $x \in Z_q$ such that $h = g^x$ and this x is called the discrete logarithm of h with respect to g, written as $x = \log_g h$.

$\Pr[\mathrm{DLog}_{\mathrm{GenGroup,A}}(n) = 1] \leq negl(n)$

- **Discrete-logarithm (DL) Problem:** Given the $(G, q, g) \leftarrow \mathrm{GenGroup}(1^n)$ and a uniform $h \in G$, find $x \in Z_q$ such that $h = g^x$.
- **Computational Diffie-Hellman Problem:** Given the $(G, q, g) \leftarrow \mathrm{GenGroup}(1^n)$ and uniform elements $h_1, h_2 \in G$, find $h \in G$ such that $h = g^{x_1 x_2}$ where $x_1 = \log_g h_1, x_2 = \log_g h_2$.
- **Decisional Diffie-Hellman Problem:** Given the $(G, q, g) \leftarrow \mathrm{GenGroup}(1^n)$ and uniform elements $h_1, h_2 \in G$, and an element $T \in G$. Determine whether $T = g^{x_1 x_2}$ holds, where $x_1 = \log_g h_1, x_2 = \log_g h_2$.

Public Key Encryption

- Syntax: $\mathrm{KeyGen}(\lambda) \to (pk, sk),\ \mathrm{Enc}(pk, m) \to c, \mathrm{Dec}(sk, c) \to m$
- Kerckhoffs' Principle: Cryptographic designs should be made completely public; Only the key is secret.
- Security guarantee: **defines what the scheme is intended to prevent the attacker from doing**

❌ It should be impossible for an attacker to recover the key.

❌ It should be impossible for an attacker to recover the **entire plaintext** from the ciphertext.

❌ It should be impossible for an attacker to recover **any character** of the plaintext from the ciphertext.

✅ regardless of any information an attacker already has, a ciphertext should leak no additional information about the underling plaintext.

e.g: Let's say an attacker knows that a message being sent every day at a specific time is either "The stock price is up" or "The stock price is down", and nothing else. If the encryption scheme is semantically secure, then the attacker, upon intercepting the encrypted message, would gain no additional information about which of the two possible messages was sent on any given day.

- The threat model: **describes the power of the adversary, i.e., what actions the attacker is assumed able to carry out.**

1. Ciphertext-only attack
2. Known-plaintext attack
3. Chosen-plaintext attack **(CPA)**
4. Chosen-Ciphertext attack **(CCA, strongest)**

Strong: CCA>CPA>KPA>COA

- El Gammal Encryption:

$KeyGen(1^\lambda) \to (pk, sk)$, choose a random $x \in Z_q$ and computes $h := g^x$.

$pk := (G, q, g, h)$, $sk := (x)$

$Enc(pk, m) \to c$; ciphertext $(c_1, c_2) := (g^y, m \times h^y)$.

$Dec(sk, c) \to m$; $m := c_2 / c_1^x = \frac{m \times h^y}{g^{xy}} = \frac{m \times h^y}{h^y} = m$

If the DDH problem is hard relative to $GenGroup$, then the El Gamal Encryption scheme is CPA-secure. However, it is not CCA-Secure.

Digital Signature

- Correctness: It is required that except with negligible probability over (pk, sk) output by $KeyGen()$, it holds that $Verify(pk, m, Sign(sk, m)) = 1$ for every message $m \in M$.
- Features: Public Verifiability; Transferability; Non-repudiation

Hash Function

Syntax: polynomial-time algorithm $H : \{0,1\}^* \to \{0,1\}^{l(n)}$ that takes as input a string $x \in \{0,1\}^*$ and outputs a string $H(x) \in \{0,1\}^{l(n)}$

- Collision-Resistance: no efficient adversary can find a collision except with non-negligible probability.

$Pr[\text{HashColl}_{A,H}(n) = 1] \leq \text{negl}(n)$

For Adversay A: 1. A is given H and n. 2. A outputs two strings $x, x' \in \{0,1\}^*$.

The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H(x) = H(x')$.

- Second-preimage or target-collision resistance: Given a uniform $x \in \{0,1\}^*$ it is **infeasible** for a PPT adversary to find $x' \neq x$ and $H(x') = H(x)$
- Preimage resistance: given a uniform $y \in \{0,1\}^{l(n)}$ it is infeasible for a PPT adversary to find a value $x \in \{0,1\}^*$ such that $H(x) = y$.

Hardness: Preimage > Second-preimage > Collision

> MD5: 128 bit output, MD5 collisions found (easily)

> SHA-1: 160 bit output, collision found in 2017, new progress was made in 2019
>
> SHA-2 (SHA 256): longer hash value

Hybrid Encryption

Symmetric Key Encryption (SKE) **(Secret key encryption scheme)**	**Public key Encryption (PKE)** **Public key encryption scheme**
Two parties MUST trust each other	**Two parties DO NOT need to trust each other**
Both share the same key (or one key is computable from the other)	Two separate keys: a public and a private key
Attack approach: bruteforce	Attack approach: solving mathematical problems (e.g. factorization, discrete log problem)
Faster	Slower (100-1000 times slower)
Smaller key size	Larger key size
Has major problem in key distribution.	No key distribution problem.

- Hybrid encryption scheme uses: Public key encryption to avoid key distribution problem. Secret key encryption to do bulk encrypting for efficiency.

1. choosing a uniform value k.
2. encrypting k using a public-key encryption scheme.

Certification Authority (CA)

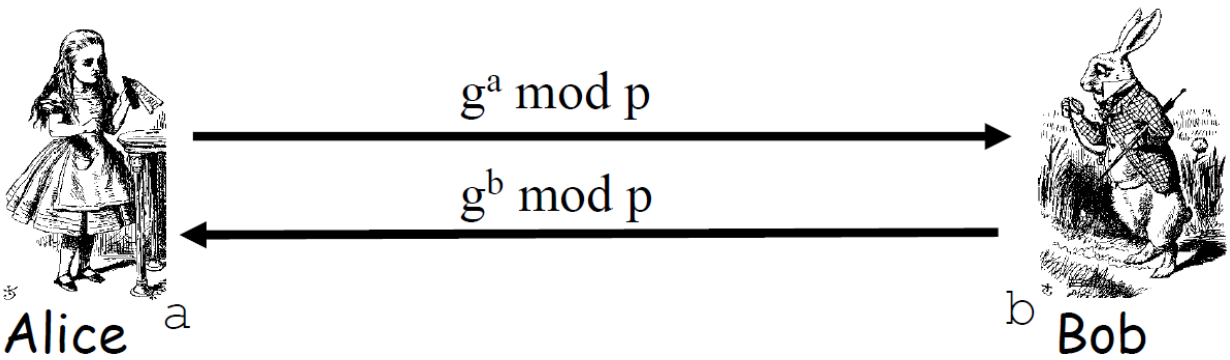
> Sender has to be sure that the public key used for encryption is indeed the receiver's public key -> CA
>
> Verifier has to be sure that the public key used for signature verification is indeed the signer's public key

CA has a public key which is ASSUMED to be well known. The certification bears:

1.The public key owner's identity. 2. the public key. 3. A validity period of the certificate and 4. the CA's signature.

$$\text{Cert}_A = (\text{ID}_A, \text{PK}_A, \text{expiry-date}, \text{Sign}_{CA}(\text{ID}_A, \text{PK}_A, \text{expiry-date}))$$

Diffie-Hellman Key Exchange



Alice computes $(g^b)^a = g^{ba} = g^{ab} \bmod p$. Bob computes $(g^a)^b = g^{ab} \bmod p$.

Could use $K = g^{ab} \bmod p$ as symmetric key

could agasint **eavesdroppers**, insecure in **man-in-the-middle attack**.

考点总结：第一章

比特币的介绍，笼统，没有细节。

区块链应用，都是宏观，考概念。不考综合。 基础题占50%-60% 两个大题一半一半

第二章(大块头)

考一些基础性的题目，指数取模（可能出计算）。ppt仔细去看，练习要会做

- 哈希函数基本定义（什么样的函数是？满足什么样的性质-才能在密码学里使用？安全模型？攻击者模型？定义安全性？满足什么情况的时候是安全的？抗碰撞性是怎么定义的？ 不止要自然语言，也要数学语言(PKE)
- 可忽略函数的含义？某个概率可忽略？

比特币

账本-怎么设定、交易-结构、基本原理、基本构造、基本安全性的性质、比特币脚本、挖矿的原理-难度-参数、攻击、安全性分析-多长是安全的-双挖、**比特币钱包**、确定性的分层的算法(**算法要重点学，密码学基础之上，与金融密切相关，考核很重要的点**)

PBFT 宏观脉络，协议是干什么的

比特币的隐私概念、Coin-mix-coin shuffle基本的概念(知道基本概念、原理就行，不详细)

不要知道细节，知道干什么的就行了 ，基本语法

development、匿名性、怎么定义的 攻击模型的脉络

门罗币必考：

比较复杂的协议，不需要背所有细节。宏观的粗粒度的就可以了

解释一下门罗币的接收者的隐藏策略

解释下基本原理

协议构造

Bitcoin Basic

- TXO: Transaction-Output, each TXO represents a coin, containing the information of (owner, value) for the coin. Each TXO is a **(public key, value)** pair.
- UTXO: (unspent transaction outputs): A transaction can consumes only the Unspent TXOs.

The secret key is the only thing that can be used to own and spend a TXO/coin.

> Each transaction consumes existing TXOs/coins and generates new TXOs/coins. The value in the consumed/old coins are transferred to the generated/new coins. Thus, each TXO/coin can be consumed/spent only one time.

> A user can generate (public key, secret key) pair as he needs, without needing to ask anyone (CA) to certificate the binding between his public key and his real identity. (Anonymity, decentralization)

- Privacy in Bitcoin: Each user can generate an arbitrary number of (public key, secret key) pair as he needs; Ideally, each (public key, secret key) pair is used only one time. 3 ways to enhance the privacy:

Hide the source of the transfer; Hide the destination of the transfer; Hide the amount

- Using Hash as the Address: Each TXO is an **(address, value)** pair, where address is the hash of the payee's public key / a hash value of something.

For a transfer transaction, the payee does not need to send his public key to the payer, instead, only needs to send a hash value to the payer.

The hash value can be the hash of anything, e.g., a public key, a script describing some conditions, or just a meaningless string

Transaction of Bitcoin

```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b86b6",
    // Unique identifier of the transaction, obtained by hashing the transaction data
  "ver": 1,
    // Version of the transaction format (1 is the original version)
  "vin_sz": 2,
    // Number of input transactions (vin stands for "value in")
  "vout_sz": 1,
    // Number of output transactions (vout stands for "value out")
  "lock_time": 0,
    // Block number or timestamp until which the transaction is locked (0 means no lock)
  "size": 404,
    // Size of the transaction in bytes
  "in": [ // List of transaction inputs
    {
      "prev_out": { // Details of the previous transaction output being spent
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0db2aa503a91d307b42ba76117d79280260",
          // Hash of the previous transaction
        "n": 0
          // Index of the specific output in the previous transaction to be spent
      },
      "scriptSig": "304402...AC"
```

```
          // Signature script that provides the necessary signatures and public keys to
spend the input
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0d5147babc0948d1473db4518f81afc5c3f52f91ff6b34e",
          // Another previous transaction hash
        "n": 0
          // Another output index from a different previous transaction
      },
      "scriptSig": "3f34ace81...BC"
        // Another signature script for the second input
    }
  ],
  "out": [ // List of transaction outputs
    {
      "value": "10.12278097", // Amount of bitcoins being sent in this output
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5750a5d6db28ed51771c6894b3d42e
OP_EQUALVERIFY OP_CHECKSIG"
        // Locking script that specifies the condition to spend this output, typically a
hashed public key or a Bitcoin address
    }
  ]
}
```

- Each input includes :

(1) a reference to existing TXO (i.e., some output of some previous transaction);
(2) A proof that the spending (of that TXO) is authorized

- Each output includes :

(1) a value; (2) an address (script)

- A transaction is valid, only if

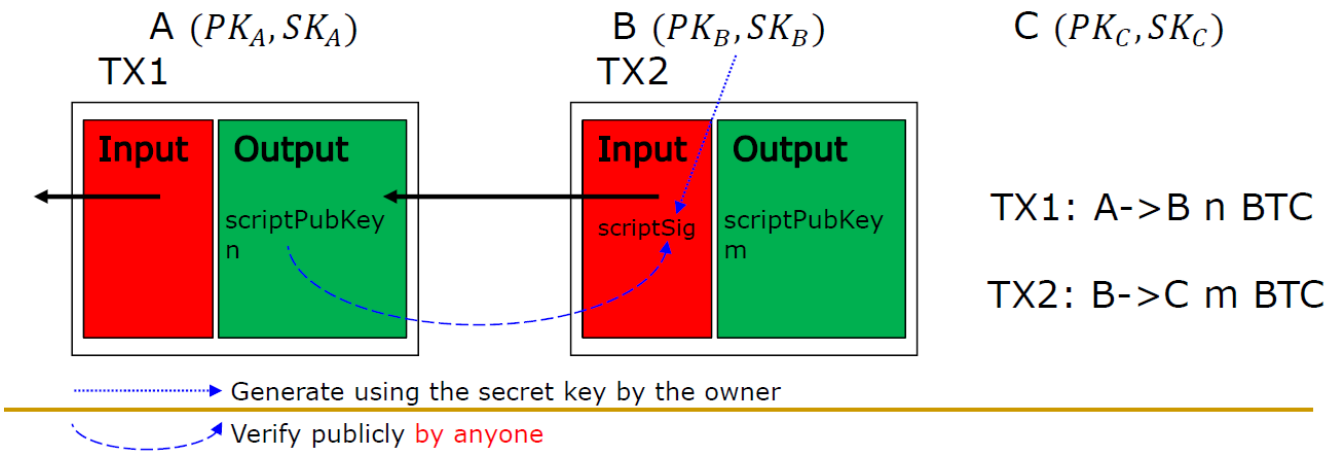
1. The transaction output (TXO) referenced by each input is not referenced by other previous transactions (i.e. not double spending)
2. The public key and signature provided in each input is verified valid (with respect to the referenced TXO's address) (i.e. authorized and authenticated)
3. The total value of outputs is not greater than that of the input coins (the difference will be the transaction fee) (i.e. not over spending)

Script of Bitcoin

"in"->scriptSig: The **signature** and **public key** in each input

"out"->ScriptPubKey: The address in TXO contains a script source code containing opcodes and operands

The **scriptSig** of an input will be put together with the **scriptPubKey** of the TXO it references, running the opcodes on the operands to check the validify of the spending.



Later, when the receiver wants redeem these coins, he needs to reveal the script that has the given hash, and provides data that will make the script evaluate to true.

Block of Bitcoin

- Merkle Tree
  1. A binary tree built with **Hash pointers**
  2. Allows a concise proof of membership
  3. Allows a quick check on whether the data in the set are tampered

  Tamper-resistance: Any modification of a data block will incurs the change of the hash values on the path from the leaf node to the root node.

  Proof of membership: 1) a data block. 2) the nodes on the path from the data block to the root.

  Time/Space Complexity: $O(\log n)$

Transactions are organized into a Merkle Tree, the hash of the Merkle Tree Root is contained in Block Header, and the block headers are chained by hash pointers --- forming Block-Chain.

- Hash Pointer: Given a hash value $hval$, use $hval$ as the identifier for records and fetch the data DATA from the database, then check whether $hval = H(DATA)$ holds. Only if it holds, the fetched DATA is correct.
- Hash Chain: Contains **only hash of the block header.** $prev_i = H(Block_{i-1}) = H(prev_{i-1}||Data_{i-1})$
- Block header: containing the hash of previous block header, and the hash of Merkle tree root node.

FIeld	size	Description
previousblockhash	32Byte	Hash of the previous block
merkleroot	32Byte	Hash of Merkle Tree for the transactions
difficulty	4Byte	Difficulty for mining
nonce	4Byte	Nonce

> Q: The hash of the block is not included in the block itself, neither transmitting on network or storing as a part of the block in storage. why?
>
> A: 1.Since the hash is a result of the block's data, including the hash within the block itself would be a logical impossibility. 2. By referencing the previous block's hash in each new block, a secure link is formed.

- Coinbase transaction: One input, and one output; Does not consume any coins;

Output value = current block reward + transaction fee

"coinbase" field: the miner can put any information in this parameter.

> Q: **Note that there is not signature to guarantee that the coinbase transaction's integrity. Could an attacker modify other's coinbase transaction to get the block reward and transaction fee?**
>
> A: **The integrity of the coinbase transaction, like all transactions in a block, is protected by the block's hash and the Proof of Work system.**

Mining

1. Set the value of the nonce in the block header, then compute the hash value of the block header. If it is smaller than the target, a valid block candidate is found, otherwise, change the nonce, and try again.
2. There are 2^{32} values for Nonce. If no valid block candidate is found even after all 2^{32} values for Nonce have been tried, a miner can modify the coinbase parameter in the coinbase transaction, so that the hash value of Merkle tree root changes, then search again in the scope of nonce.

- Mining is to repeatedly run the hash algorithm on block header, until a hash value smaller than the target is found.
- The difficulty will be adjusted each 2016 blocks are generated.
- More (computing) resource to mine, more possible to be a winner.

Bitcoin Network

A peer-to-peer network; all nodes are equal; runs over TCP and has a random topology; The network changes over time and is quite dynamic due to nodes entering and leaving.

1. When a user/node issues a transaction, it broadcasts the transaction to its peers.
2. When a node receives a transaction, it executes the flooding algorithm.

Check the validity of the transaction(using local blockchain):{

Check whether there is double-spending **By maintain a local UTXO list**

Check the balance of input and out coin value

Check whether the transaction is authorized **By checking whether each scriptSig and the corresponding scriptPubKey returns TRUE**

}

Check whether this transaction have been received and in the pool

3. Mining: Miner uses the latest block in the current local blockchain as the previous block node, selects a set of transactions from its transaction pool, constructs the corresponding coinbase transaction, and constructs Merkle Tree of these transactions, then, begin to mine: **compute hash by combining previousblockhash, merkle tree root, and different nonce. If necessary, change the coinbase parameter in the coinbase transaction.**
4. Some miner finds a valid block candidate, then broadcasts the block candidate to the Bitcoin Network.
5. When a node receives a new candidate block, it runs the flooding algorithm

- race condition: the nodes will temporarily disagree on which transactions should be put into the next block. This race condition is solved when miners finds new block.
- Default policy of Bitcoin protocol:

1. When a node receives multiple conflicting transactions, it will keep the one received first, and 'discard' others.
2. When the received new blocks make the local blockchain fork, if the blockchain forks have the same length, the node will select the one received first. Otherwise, the node will select the longest chain fork.

Bitcoin Consensus

- **Data stored in fully validating nodes(mine):**

1. Complete Blockchain (headers, transactions)
2. List of UTXO (To proceed the transactions quickly)
3. Transaction Pool (select transactions and mine)

- **Data stored in lightweight nodes (Simplified Payment Verification (SPV) Clients, do not mine):**

1. Complete chain of the block headers.
2. Pieces that they need to verify specific transactions that they care about

- **Target of consensus:** At any moment, all nodes in the Bitcoin Network agree on a series of blocks and the transactions in these blocks.
- Why does a node behave honestly? Once the proposed block is accepted by whole network, the miner will obtain the block reward and transaction fee.
- **Why follow/choose the longest chain?** Each block implies corresponding computation power has been invested on the block. Longer chain implies more computation power accepting the chain, and implies bigger probability these blocks are accepted by the consensus blockchain. And the blocks not on the longest chain will be "discarded" (not accepted by the nodes). So even the blocks on chain may be discarded later.
- when do the blocks achieve finality/stability and become one part of the consensus blockchain? 6

Mining Pools

Pool Manager prepares the block to mine, does very thing except computing hash. **Pool Members** only computes hash to find the right nonce.

The pool manager will distribute the fees to all the pool members based on how much work each member actually performed.

- **Pay models:**

1. Pay-per-share: Pool manager pays a flat fee for every share, without waiting for the pool to find a block. The pool manager absorbs all of the risk.
2. Proportional: The amount of payment depends on whether or not the pool actually found a valid block.

- **A weakness of Mining Pool:** there is nothing to enforce that the member miners actually submit valid blocks to the pool manager in the event that they find them.
- Pros and Cons of Mining Pools:

1. Pros: Make mining much more predictable for the participants and make it easier for small miners to get involved in the game; Make it easier to update the network
2. Cons: Mining pools are a form of centralization; Lower the number of fully validating nodes; Make some attacks easier

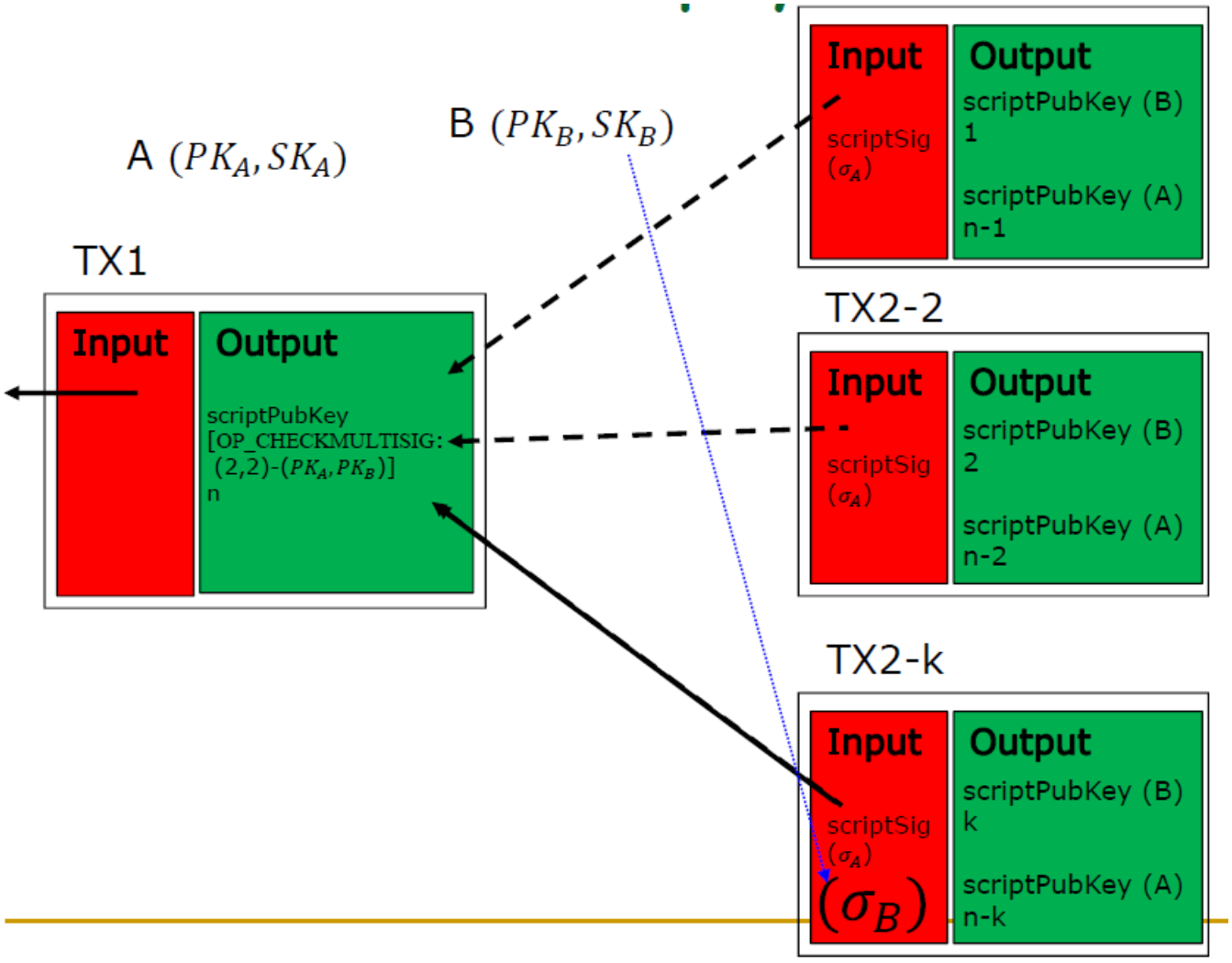
- **Non-outsourceable mining:** make the miners who search the nonce have to know the secret key for the coinbase transaction.

Attacks(Both result Double-Spending)

- Forking Attacks(51%) : A malicious miner sends coins by a transaction to Bob and receives some goods or service in exchange for it. The miner then forks the block chain to create a longer branch containing conflicting transaction. The payment to Bob will be invalid in this new consensus chain. (control 51% computation power or creating a new mining pool)
- Selfish Mining: Say that a miner just found a block, it does not announce it right away as the default/honest protocol requires. Instead, it tries to get ahead by doing some more mining on top of this block in hopes of finding two blocks in a row before the rest of network finds even one, keeping its blocks secret the whole time; Once a new block by other miners is announced, the attacker immediately announces its two blocks in a row, so that its blocks make a longer chain.

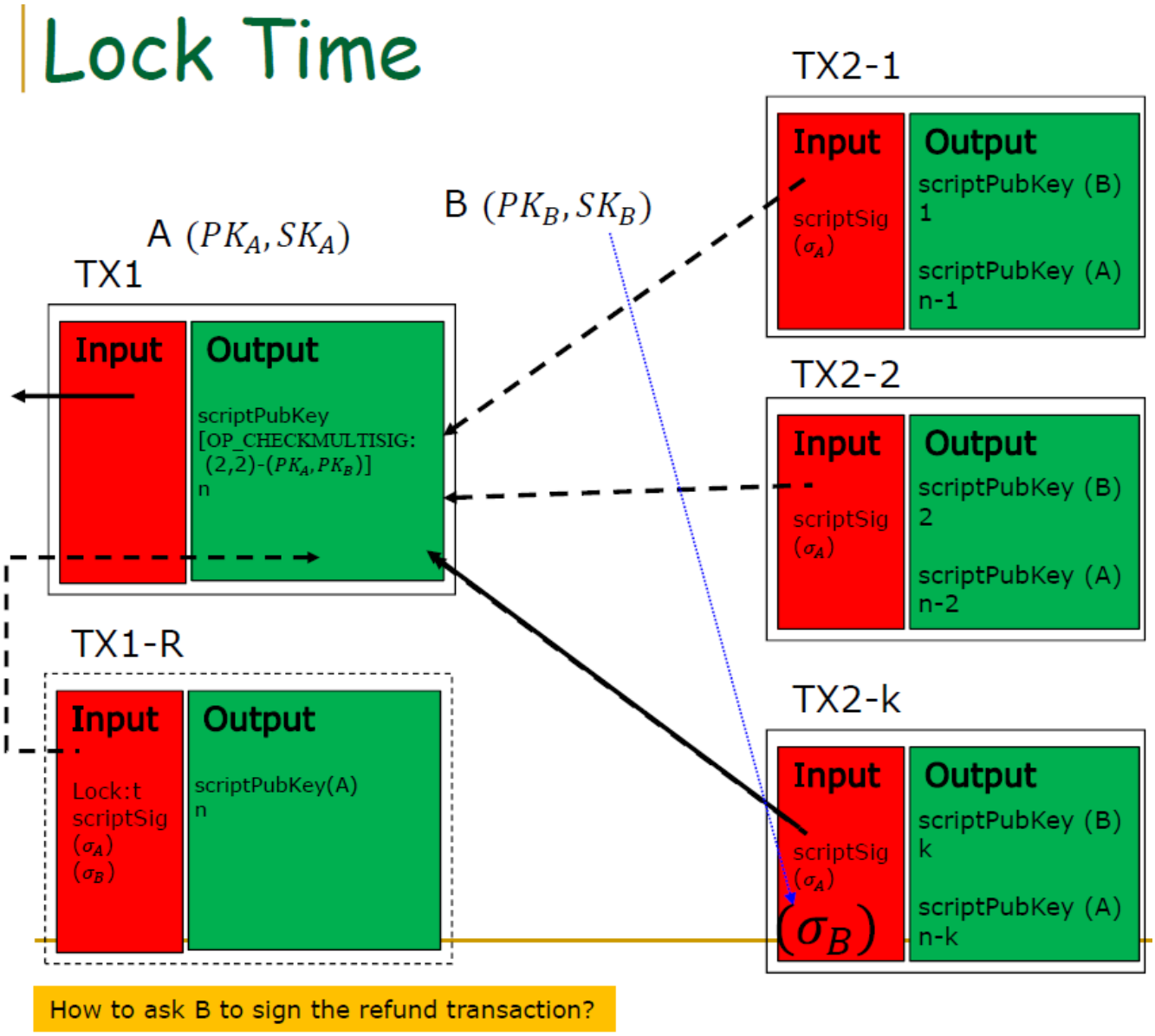
Bitcoin Scripts

- **MULTISIG transaction:** requires two of three people to sign in order to redeem the coins.
- **Efficient micro-payment:** Alice sets up a MULTISIG escrow transaction on the blockchain, prepaying the maximum amount she might need for Bob's service. For each minute of service used, Alice signs a new transaction allocating a small payment to Bob and returning the rest to herself. These transactions are not yet signed by Bob or broadcasted to the blockchain. Once the service period ends, Alice stops creating new transactions. Bob then signs and publishes the final transaction Alice sent, which pays him for the total service provided and refunds any remaining amount to Alice. The intermediate transactions Alice signed are discarded as they are never completed and published on the blockchain.



Problem: what if Bob never signs the last transaction? He may just say, “I’m happy to let the coins sit there in escrow forever,” in which case, maybe the coins won’t move, but Alice will lose the full value that she paid at the beginning. Introduce Lock Time.

- Lock Time: Before publishing this ESCROW transaction, Alice asks Bob to sign a REFUND transaction which refunds all of Alice's money back to her, but the refund is "locked" until some time in the future. Alice also signs this refund transaction and hold on to it. Then Alice publish the ESCROW transaction to the blockchain.



If Bob does not sign a transaction to redeem pays for his service before the refund transaction becomes valid, all the escrowed amount will be sent back to Alice.
If Bob honestly signs a transaction to redeem pays to him, the refund transaction will become a double-spending transaction, and miners will discarded it.

Wallet

A software or hardware that stores and manages the keys for the owner

- Hot Storage: On-line and Internet-connected storage, Convenient, but may be insecure, like cash in wallet

The addresses and secret keys in hot storage are used frequently to receive coins and spend coins.

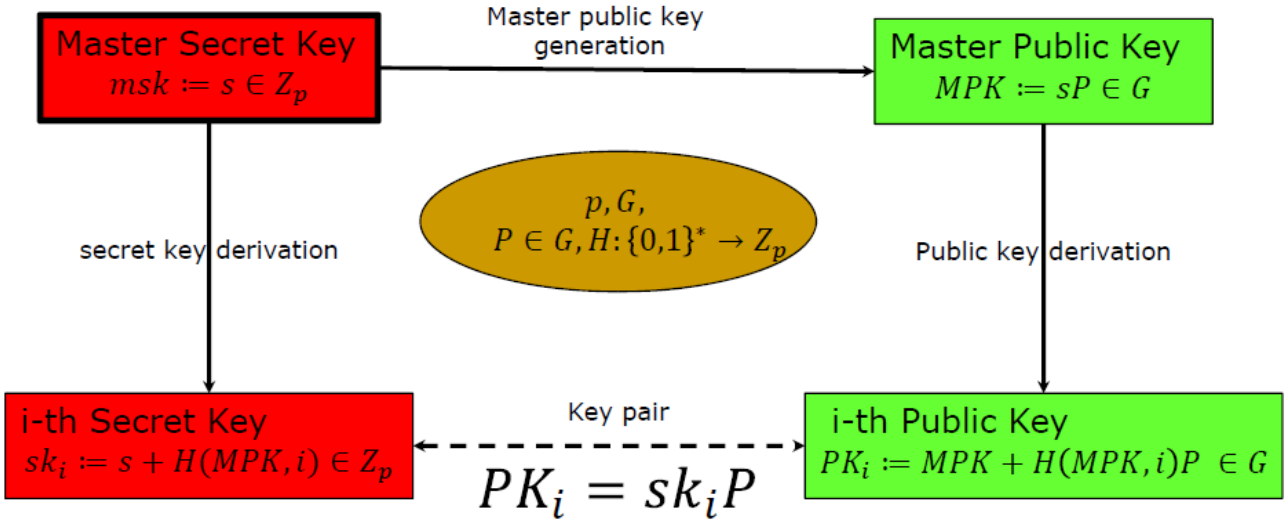
When the number of coins on a hot address is big, you can spend the coins to some cold address.

- Cold Storage: Off-line storage, locked somewhere. Safe, but not convenient

To have better privacy, each cold address is used only once, i.e. appearing in only one transaction as receiving address

Hierarchical Deterministic Wallet

- Master Public Key Property: the public keys can be derived from a master public key, without needing the (master) secret key(s)
- Hierarchy Property: Each (public key, secret key) pair can act as the master key for its sub-organization.



- Applications of Deterministic Wallet

1. Low-maintenance wallets with easy backup and recovery: Only need to back up the master secret key
2. Freshly generated cold addresses: Store the master public key on hot storage, then can easily and conveniently generate cold addresses
3. Trustless audit: Reveal the master public key to the auditors, then the auditors can view all the transactions related to the wallet
4. Hierarchical Wallet allowing a treasurer to allocate funds to departments

- How to store and protect the master secret key? Brain Wallet; Paper Wallet
- Online Wallet: the site delivers the code that runs on your browser or the app, and it also stores your keys.
- Exchanges: Buying or selling coins at the exchange will not have transactions on the blockchain.
  When a user withdraws coins from an exchange, a corresponding transaction appears on the blockchain.

Consensus

- Definition of Consensus:
  1. Termination: All normal nodes will end up in a finite number of steps and make decisions.
  2. Agreement: All nodes must reach to a common decision (Agreement); If all nodes proposed the same initial decision value, then all normal nodes should choose that value (Integrity).
  3. Validity: The final agreement must be one of the values submitted by the nodes.
- FLP Impossibility Theorem: in asynchronous communication scenarios, even if only one node fails, no algorithm can guarantee the consistency among the other normal nodes.
- CAP Theorem: A distributed system can deliver only two of the three desired characteristics.

1. Consistency: Consistency means that all clients see the same data at the same time, no matter which node they connect to.
2. Availability: Availability means that any client making a request for data gets a response, even if one or more nodes are broken down.
3. Partition tolerance: Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

- BASE Theory by Dan Pritchett: Achieving high availability, at the cost of weakening the strong consistency; Even if strong consistency cannot be achieved, final consistency (Eventual Consistency) can be achieved in a suitable way.

Bitcoin PoW Consensus

The number of successful mining in time t is subject to $B(bt, p)$, where v is the number of hash operations in unit time.

The probability of an attacker catching up from a given deficit is analogous to a **Gambler's Ruin problem**.

$p = \text{probability an honest node finds the next block}$

$q = \text{probability the attacker finds the next block}$

$q_z = \text{probability the attacker will ever catch up from z blocks behind}$

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

the number of blocks k on the forking chain generated by the attacker after block B_{i-1} satisfies the Poisson distribution with parameter $\lambda = z\frac{q}{p}$.

The probability that the forking chain can catch up $\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \times \begin{cases} (q/p)^{(z-k)} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases}$

Bitcoin system 6-confirmation rule is based on the assumption that the attacker's computer power is less than 10%.

Practical Byzantine fault tolerance algorithm(PBFT)

- **Byzantine failures:** Components may fail and there is imperfect information on whether a component has failed.
- **PBFT:** For synchronous (weak synchronous) networks, solved the problem of low efficiency that the original Byzantine fault-tolerant algorithm suffers, reducing the complexity from exponential to polynomial level.

Provide better privacy

- Enhance user's privacy (based on the original protocols)
- Build new built-in privacy-preserving cryptocurrencies/blockchain, using privacy-preserving cryptographic primitives

1. Hide the receiver/source
2. Hide the sender/destination
3. Hide the amount

Coin-Mix

Definition: Users send their coins to the address of the service provider; After collecting sufficient coins from DIFFERENT users, the SP issues transactions to send the coins to the addresses of corresponding users, maybe from time to time; SP charges for the service.

Coin-Shuffle

A decentralized protocol where users collectively create a single transaction that shuffles their coins. Each user receives coins back to a fresh address, ensuring that there is no link between the incoming and outgoing coins. No trusted third party is required, and no single user has enough information to trace the coins.

Monero(XMR)

1. Hide the receiver using **Stealth Address**
2. Hide the sender using (Linkable) **Ring Signature**
3. Hide the amount using **Commitment**

Stealth Address

- The Payer: $r \leftarrow_R Z_p$; $R = rG$; calculate $PK_R = H(rA)G + B$
- The Public: $A = aG, B = bG$; $addr := (R, PK_R)$; A-> public view key; B-> public spend key
- The Payee: $(a, b) \in Z_p$

Monitor the blockchain to check whether a TXO has an address $addr := (R, PK_R)$ such that $PK_R = H(aR)G + B = sk_RG$. If it holds, compute $sk_R = H(aR) + b$.

Summary: $PK_R = H(rA)G + B = H(aR)G + B = sk_RG = (H(aR) + b)G$

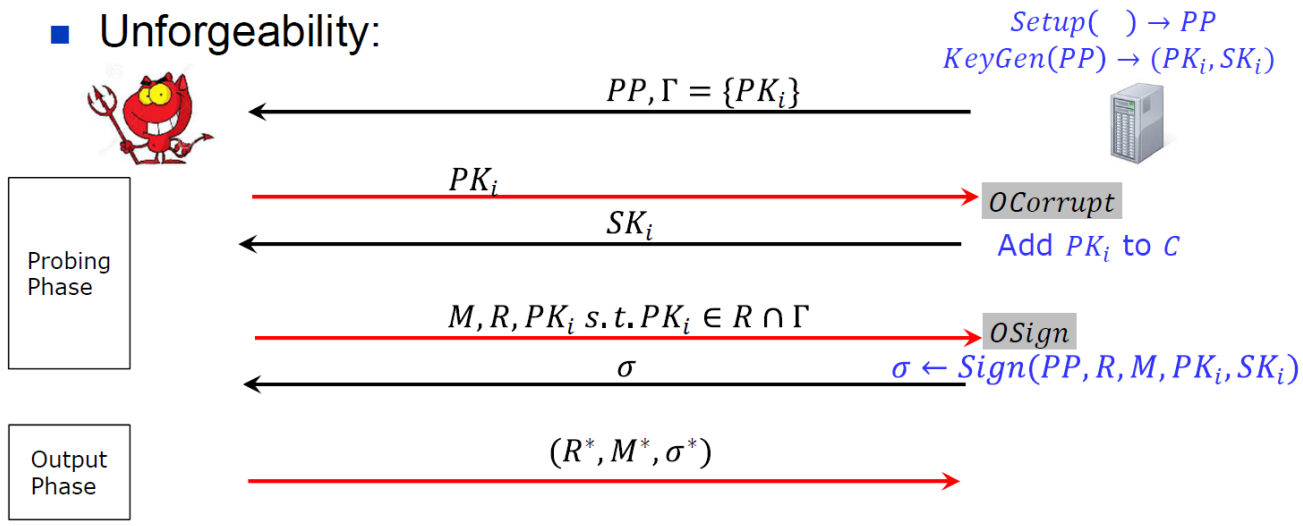
- **Privacy:** Each coin receiving address is freshly generated, with random r.
- **Security:** Only the payee knows the value of b, thus only the payee can spend the coin.
- **Convenience:** From the view of B, for each transaction output, he needs to run the check one time.
- **Enhance Security and convenience:** Even an adversary compromises the value of a, he is not able to spend the coins.

Ring Signature

- Correctness: For any $Setup(\lambda) \rightarrow PP$, any message M in message space, any $R = (PK_1, \cdots, PK_l)$ such that $(PK_i, SK_i) \leftarrow KeyGen(PP)(\text{for } i = 1, \cdots, l)$, any $s \in \{1, \cdots, l\}$, it holds that

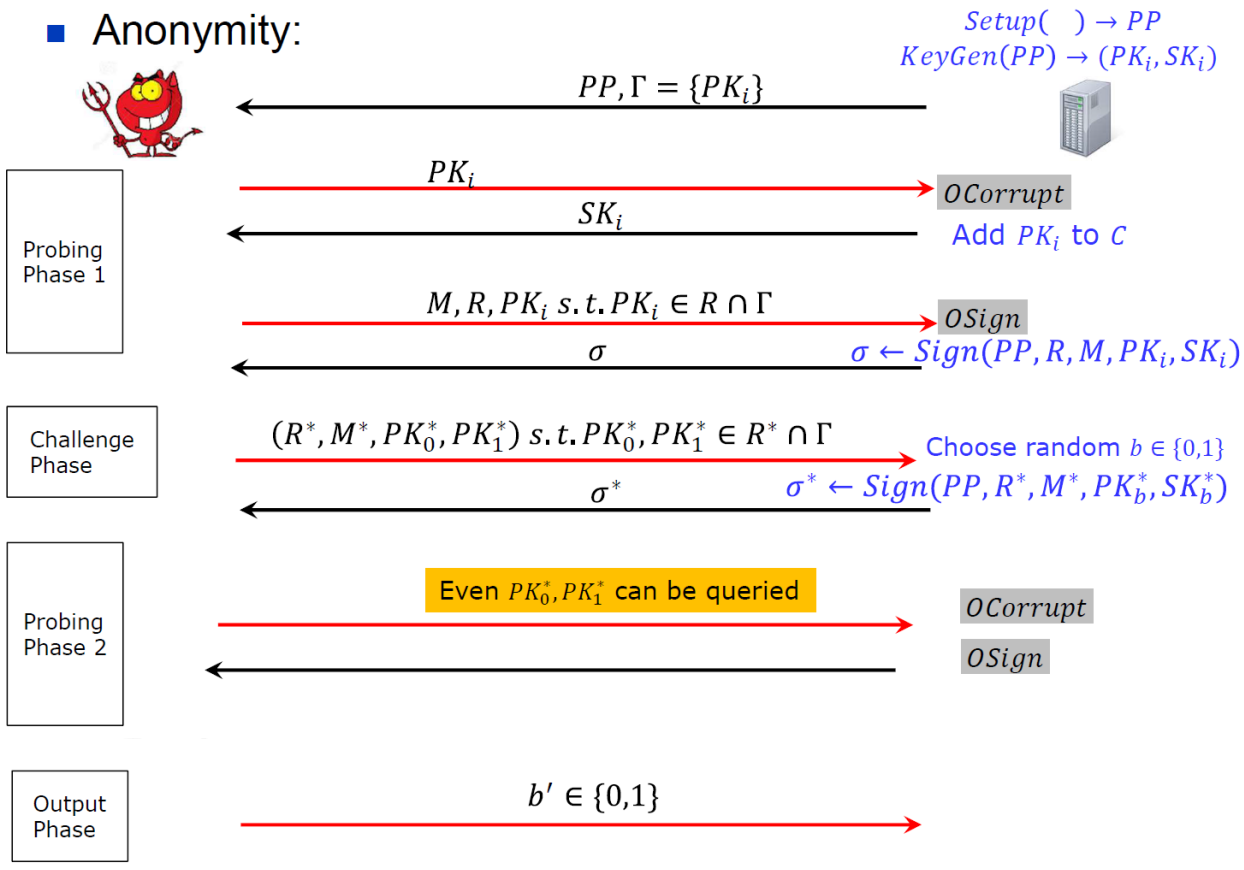
$Verify(PP, R, M, Sign(PP, R, M, PK_s, SK_s)) = 1.$

- Security-Unforgeability: Given a ring $R = (PK_1, \cdots, PK_l)$, only having one of the corresponding private keys, a user can generate a valid signature with respect to R.



- ❑ The adversary wins the game if
  - ➢ $Veriyf(PP, R^*, M^*, \sigma^*) = 1$; AND
  - ➢ (R^*, M^*, σ^*) is not query-answer tuple by $OSign$; AND
  - ➢ $R^* \subseteq \Gamma \setminus C$, where C is the set of public keys queried to $OCorrupt$
- ❑ A signature scheme is unforgeable, if no PPT adversary can win the game with non-negligible probability.

- Security-**Anonymity**: Given a tuple (R, M, σ) such that $Verify(PP, R, M, \sigma) = 1$, it is **infeasible to identify the signer's public key out of** R.



- ❑ The adversary wins the game if $b' = b$.
  - ❑ The advantage of the adversary is $|\Pr[b' = b] - 1/2|$
- ❑ A signature scheme is anonymous, if for any PPT adversary, the advantage of the adversary is at most non-negligible.

Linkable Ring Signature(prevent double-spending)

$Link(PP, R_1, M_1, \sigma_1, R_2, M_2, \sigma_2) \rightarrow 1/0$

Given two valid signature, say $Verify(PP, R_1, M_1, \sigma_1) = 1$ for $i = 1, 2$, the algorithm returns 1, implying the two signatures are generated by the same public key, otherwise returns 0.

- Security-Linkability: No adversary can generate two signatures using the same public key, while the $Link$ algorithm returns that they are unlinked.
- Security-Non-slanderability: Given a public key PK and a corresponding signature σ with respect to (R, M), without the corresponding private key, no adversary can generate a tuple $(R^{'}, M^{'}, \sigma^{'})$ such that (1) $Verify(PP, R^{'}, M^{'}, \sigma^{'}) = 1$ and (2) $Link(PP, R, M, \sigma, R^{'}, M^{'}, \sigma^{'}) = 1$.

Each transaction output generates a unique **key image** that is a cryptographic representation of the sender's private key. When a transaction is broadcast to the network, miners check the **key image** against a list of all key images previously used in the blockchain. If the key image has already been used, the transaction is rejected as a double-spend attempt

Commitment(hide amount in cryptocurrencies)

$Setup(1^{\lambda}) \rightarrow ck$ generates the public commitment key ck, including a message space S_M, a randomness space S_R, and a commitment space S_C.

$Commit_{ck}(m, r) \rightarrow C = rG + mH \qquad Open_{ck}(C, m^{'}, r^{'}) \rightarrow 1/0$

- Security-Hiding: The commitment c gives the adversary no information about the value m.
- Security-Binding: commitment c cannot be opened in two different ways.
- Additively Homomorphic: $Commit(m, r) + Commit(m^{'}, r^{'}) = Commit(m + m^{'}, r + r^{'})$,

  Thus, the prover only needs to prove that $C - C^{'}$ is a commitment of 0.

When the miners check the validity of a transaction, they do not care the amounts, and they only need to make sure that the **value of consumed coins equals to that of generated coins**.

Each commitment should be accompanied by a **zero-knowledge proof** proving $v \in [0, V]$, without leaking the value of v.

Extended Linkable Ring Signature

When Linkable Ring Signature is used to hide the sender, the case of multiple inputs (i.e. a transaction consumes multiple coins) needs to be considered.