高性能：提供多种安全、高效共识算法，按需选择

共识算法	SOLO	kafka（CFT）	FBFT
节点数	1	2f+1	3f+1
错误节点容忍	不容忍	最多1/2个crash节点	最多1/3个拜占庭错误节点
交易性能	一般	10000+TPS	2000+TPS

– 快速拜占庭容错算法（Fast Byzantine Fault Tolerance ）

关键技术

用户可以根据不同的使用场景以及安全和性能等不同需求选择共识算法

- **SOLO模式**：只需要一个共识节点，简单、快速，建议在开发测试环节使用
- **kafka/Zookeeper**：高速共识算法，1:n个节点，能容忍crash节点；
- **FBFT**：快速拜占庭容错算法，使用3f+1个节点，能容忍最多1/3拜占庭错误节点。

华为云区块链服务关键能力

◆ 简单易用

- **5分钟**完成区块链配置、部署，相对于自建节约**80%**开发部署时间和成本
- 开发、测试、部署、管理和监控全覆盖
- 同时支持联盟链和私有链
- 和华为云其他服务打通

◆ 高可用

- 动态邀请联盟成员，快速组建联盟链
- 按需付费、节点弹性伸缩，故障自动恢复
- 可扩展、海量弹性文件存储共享账本。
- 支持SQL访问和存储共享账本
- 支持Go，Java等多语言智能合约

◆ 高安全

- 华为云安全立体安全防护
- 多级加密：签名、通道、内容
- 支持国密、加法同态保护数据隐私
- 零知识证明保护用户隐私

◆ 高性能

- 秒级共识。
- 多种高效共识算法可选 （kafka，FBFT）

交易性能**2000-10000** TPS

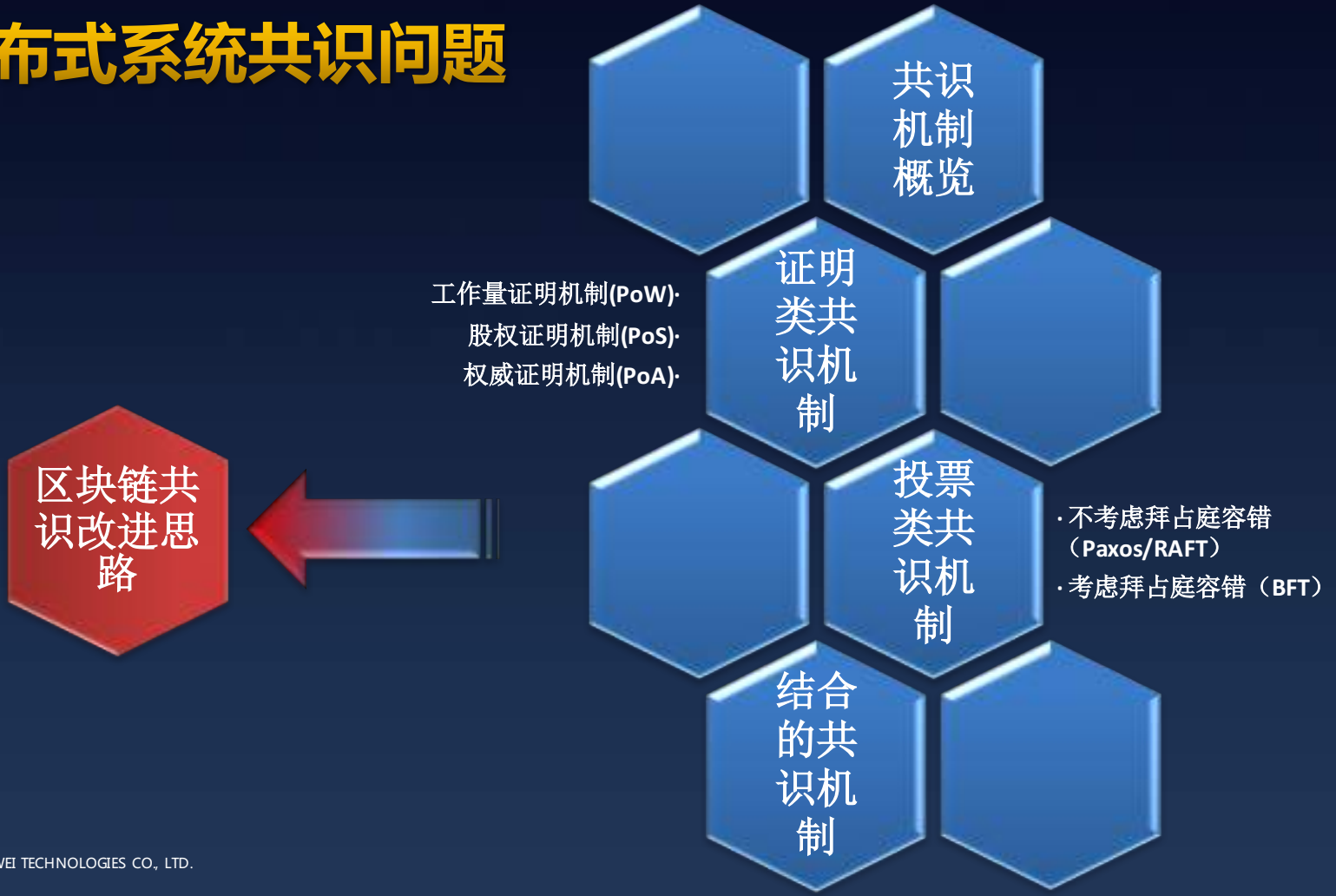
- 电信级网络，系统延时小于300毫秒

◆ 全球部署

- 支持华为云大陆和香港站点部署
- 规划和合营云全球公有云部署区块链（德电、法电等）
- 和私有云结合形成混合部署

目 录

分布式系统共识问题



共识机制概览

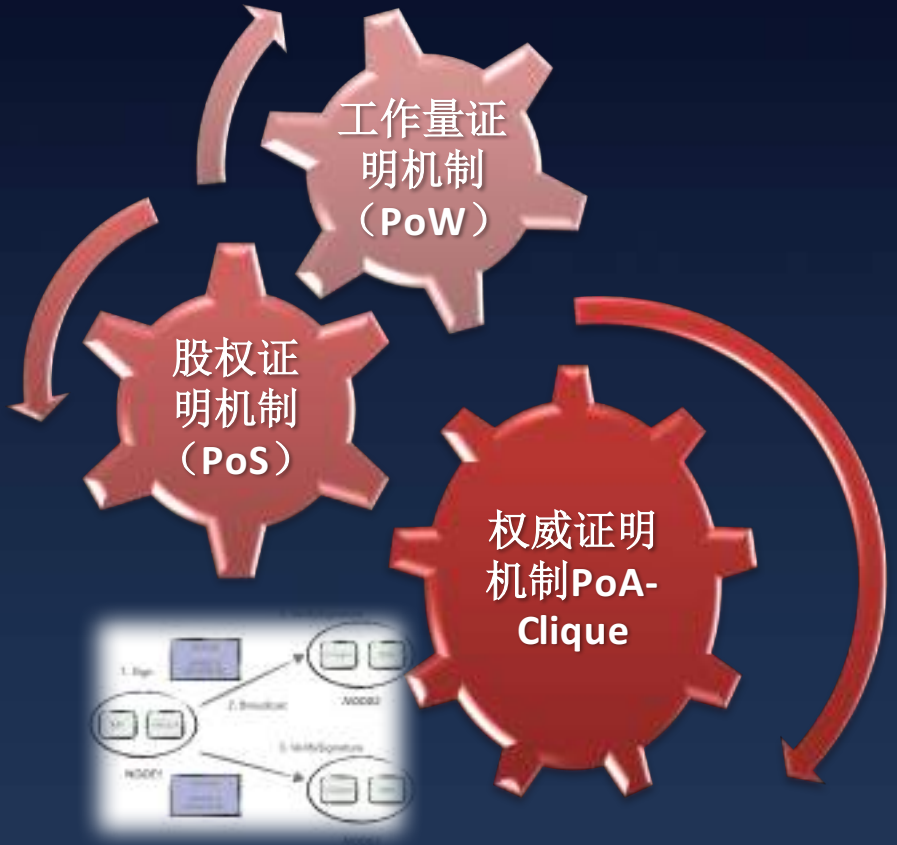
一致性问题

什么是拜占庭问题？ N > 3f+1

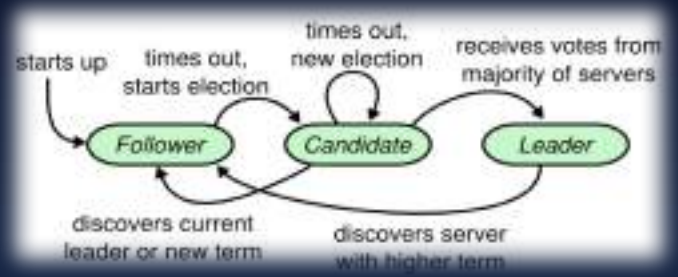
共识算法

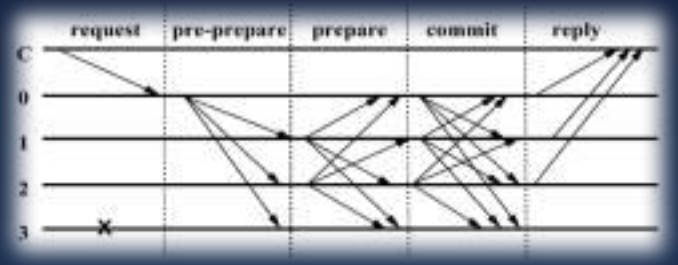
FLP不可能原理

证明类共识机制



投票类共识机制



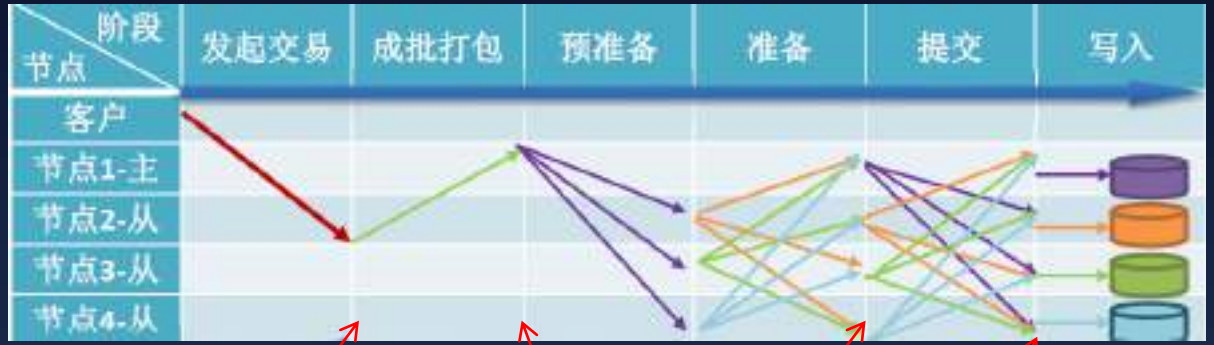


两类共识机制结合（Algorand）

共识算法总结

共识算法名称	中心化	延时/吞吐量	一致性	适用区块链场景	备注
实用拜占庭容错（PBFT）	部分去中心化，预设了决策角色（无Leader）	低/较高	最终收敛	N/A	后面所有协议本质上都是拜占庭容错的改进。几位将军如何在有一定数量个叛徒的条件下达成一致决策，即分布式系统如何在部分信任环境下如果取得状态一致
Paxos/Raft	Paxos允许有多个Leader，Raft有单一Leader	低/高	最终收敛	私有链、信任度高的联盟链	简化了拜占庭将军问题，排除拜占庭节点的存在，仅考虑节点之间由于网络中断、系统崩溃等原因导致无法正常沟通时的状态一致性问题
工作量证明 Proof-of-Work(PoW)	完全去中心化	高/低	依概率收敛	公有链	首个在完全不信任的网络环境中可以达成状态一致的共识算法；需要消耗大量的计算资源，安全性依赖网络中诚实节点的计算资源总和
股权持有证明 Proof-of-Stake(PoS)	持有更多股份者有较大权利，可能有中心化倾向	中/依赖实现	依赖实现	公有链、联盟链	克服了PoW浪费大量算力的缺点。PoS的基本原理是将区块链系统内用户的一部分资源作为其信用的凭证，让其在一致性决策时拥有较大的话语权。例如和PoW结合可以依据资产降低挖矿难度；或者由资产最多的用户选择100位其它用户作为代表决策投票；或每若干块由持股人投票持久化分支；设计时考虑需短程攻击、长程攻击、无成本作恶等问题
典型改进协议（Ripple、Stellar、Tendermint等）	依赖实现	低/高	最终收敛	联盟链	Ripple是拜占庭容错的一个实现，每个节点可以按照规则信任其它节点，每个共识周期中对尚未确认的Tx进行多轮投票来决定是否将其永久记录；Tendermint可以看作是PoS和PBFT的结合；Stellar恒星协议是提供了更灵活信任容忍度的经典PBFT改进方案
Intel PoET（Elapse Time）	去中心化	低/高	最终收敛	联盟链	使用Intel的可信执行环境（TEE），根据芯片产生的等待时间随机产生Leader，生成新的Block；所有节点必须使用Intel的芯片

– 快速拜占庭容错算法（Fast Byzantine Fault Tolerance ）



从节点将消息发送给主节点

主节点对交易进行验证

每个节点收到2f个准备消息后对交易进行验证

从节点收到2f+1个commit后进行写区块

目 录

区块链中关键概念：密码学

哈希算法（Hash）

- 一段数字内容的Hash可以用于验证数据的完整性
- 数字内容的微小修改都会引起Hash值的巨大变化
- 合格的Hash算法很容易由数字内得到Hash值，却几乎不可能通过Hash值反算出原数字内容

公私钥体系（PKI）

- 公私钥体系是现在加密通讯的基石，通过加密算法随机生成公私钥对，一般私钥需要在用户手中绝对安全保存，保证只有用户才能接触到，公钥可以对外公开。可靠的加密算法可以保证任何人无法通过公钥计算出其对应的私钥

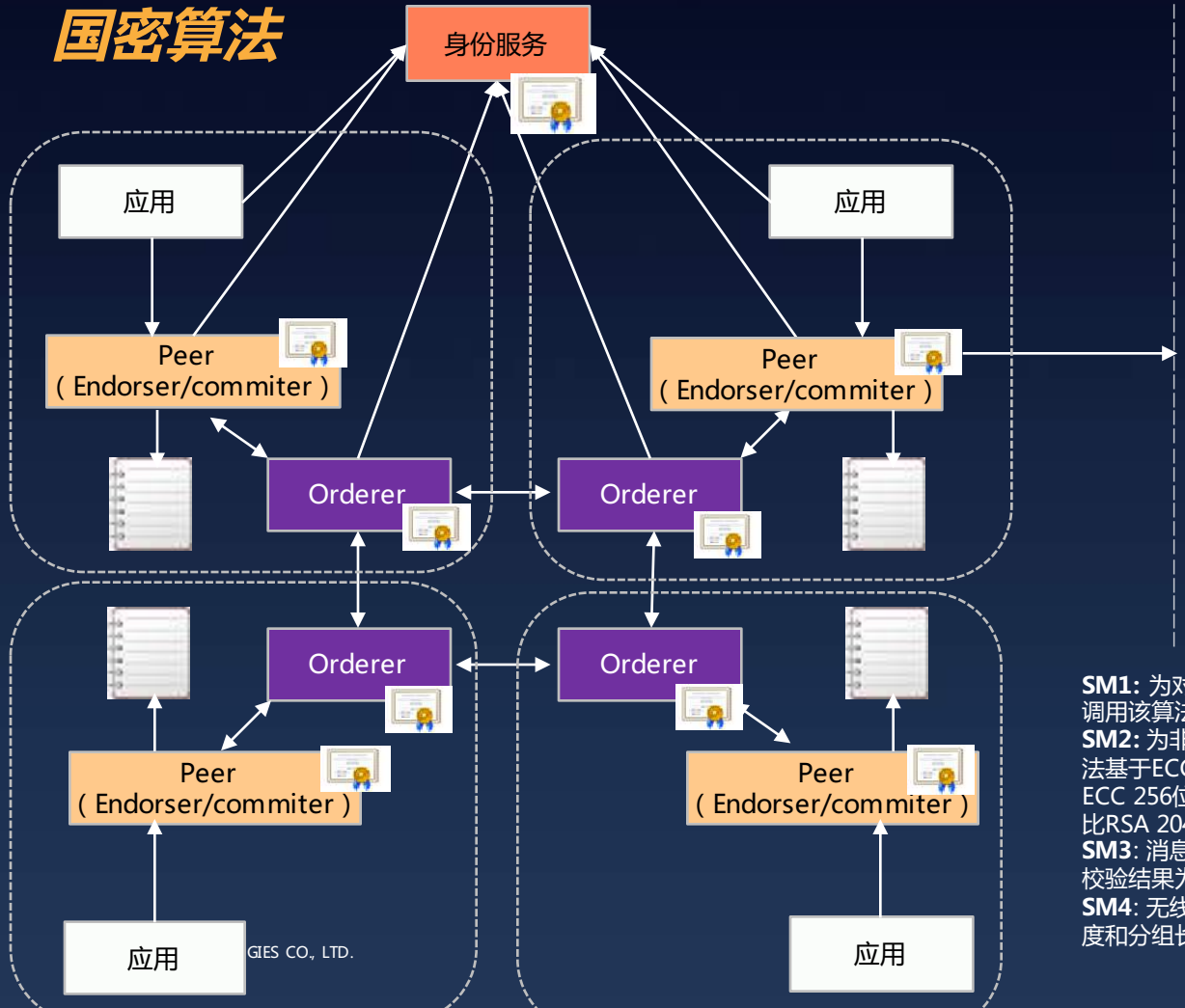
基于PKI的加密（Encryption）

- 加密一般是指向特定用户发送加密内容，保证只有接收方才能解密原内容。具体地，发送方用接收方的公钥加密原内容，得到密文，将密文发送给接收方，接收方用自己的私钥便可由密文解密（Decryption）出原内容

基于PKI的签名（Signature）

- 签名用于他人验证消息内容确实来自声明的内容发送者。具体地，内容发送者发送一段明文，并将明文的Hash用自己的私钥加密，生成签名。任何接收方收到明文后，同样对明文进行Hash，然后用发送者的公钥解密签名，将得到的数据与自己对明文Hash的结果对比如果一致，则可以证明消息确实是该公钥对应的私钥持有者发出的

国密算法



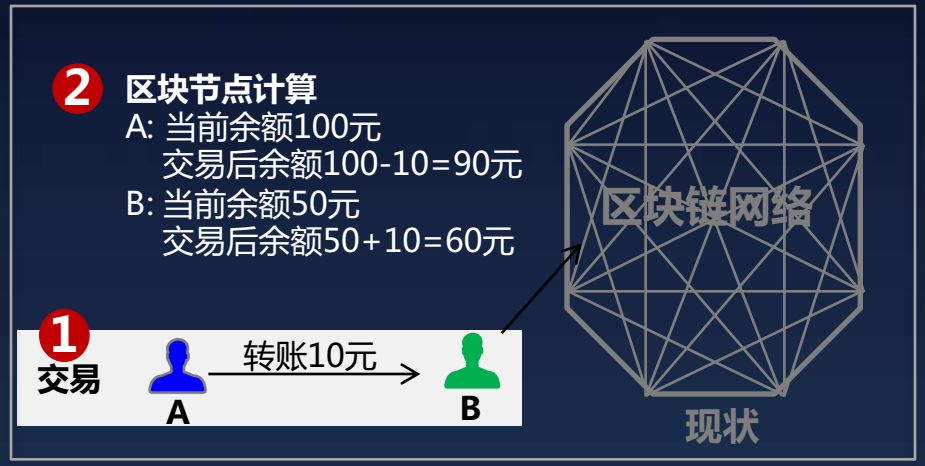

```
├── peers
│   ├── peer0.org1.example.com
│   │   ├── msp
│   │   │   ├── admincerts
│   │   │   │   └── Admin@org1.example.com-cert.pem
│   │   │   ├── cacerts
│   │   │   │   └── ca.org1.example.com-cert.pem
│   │   │   ├── keystore
│   │   │   │   ├── aeddeddd3f2e9cff9df633ebb1f94593c1af5889a0fa3ff2dc18b93754392c06_sk
│   │   │   │   └── e793df0a0c15141f643f7dd28fa53ef4270778f647f2740d52b88958e7b2f2d7_sk
│   │   │   ├── signcerts
│   │   │   │   └── peer0.org1.example.com-cert.pem
│   │   │   └── tlscacerts
│   │   │       └── tlsca.org1.example.com-cert.pem
│   │   └── tls
│   │       ├── ca.crt
│   │       ├── server.crt
│   │       └── server.key
│   └── peer1.org1.example.com
│       ├── msp
│       │   ├── admincerts
│       │   │   └── Admin@org1.example.com-cert.pem
│       │   ├── cacerts
│       │   │   └── ca.org1.example.com-cert.pem
│       │   ├── keystore
│       │   │   ├── 2382e79a6a79843472b1b8c4fd058fcfdcde3fc2aff4453d524fc9e2bd7cf3b3_sk
│       │   │   └── 51c4a538ef160f608af028956843ac7e255181078f14ef4c1c563f8775e181da_sk
│       │   ├── signcerts
│       │   │   └── peer1.org1.example.com-cert.pem
│       │   └── tlscacerts
│       │       └── tlsca.org1.example.com-cert.pem
│       └── tls
│           ├── ca.crt
│           ├── server.crt
│           └── server.key
```

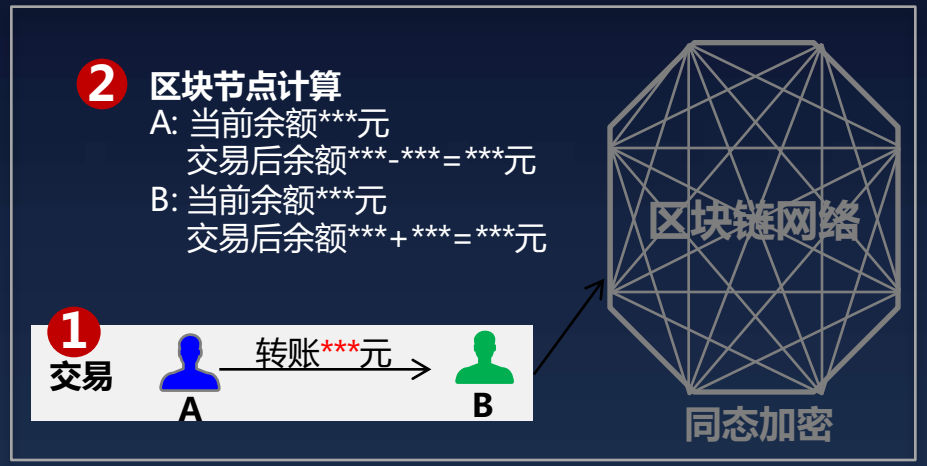
SM1: 为对称加密。其加密强度与AES相当。该算法不公开，调用该算法时，需要通过加密芯片的接口进行调用。
SM2: 为非对称加密，基于ECC。该算法已公开。由于该算法基于ECC，故其签名速度与秘钥生成速度都快于RSA。ECC 256位（SM2采用的就是ECC 256位的一种）安全强度比RSA 2048位高，但运算速度快于RSA。
SM3: 消息摘要。可以用MD5作为对比理解。该算法已公开。校验结果为256位。
SM4: 无线局域网标准的分组数据算法。对称加密，密钥长度和分组长度均为128位。

HUAWEI

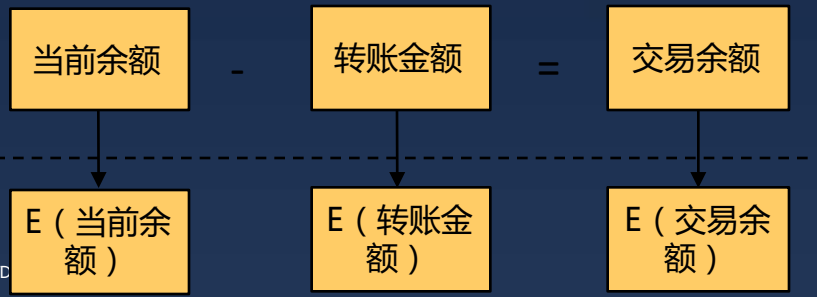
同态加密

问题：A向B转账10元，需要区块链节点记账，**但是不想让区块链节点知道交易金额以及最新余额**





明文操作：



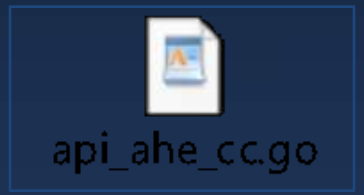
加法同态加密：

挑战：如何解决区块链技术应用的隐私和可用性？

方案：引入同态加密(解决隐私问题)

我们提供客户端 SDK库和Chaincode库，该库主要用于交易类的密文运算服务，达到用户交易的隐私保护。

- *SDK库*：用于在client端提供加法同态功能和生成交易金额的证明信息。
- *同态加密链代码IDChaincode.go*：在同态加密的场景下，用户在部署应用前需要下载安装并且实例化此链代码至区块链服务。
- *Chaincode库*：提供零知识证明功能，用于在密文条件下，校验用户交易的证明，并生成交易后的数据，使背书者无需解密用户交易的数据，达到余额范围的判断。



示例

为用户生成一对同态公私钥

- privKeyStr, pubKeyStr, err := pswapi_sdk.GenerateKey(propwd) check(err) fmt.Println("key is nil") userdata.PubKey = pubKeyStr userdata.PriKey = privKeyStr

注册公钥

- res, err := sdk_client.Invoke(setup, "Register", [][]byte{[]byte(userdata.PubKey), []byte(senderAddr)}) if err != nil { fmt.Println("Fail to register user pk ", err.Error()) } else { addrByte := res[0].ProposalResponse.GetResponse().Payload fmt.Println("Register addr: ", string(addrByte))

注册初始余额

- balanceInfo, err := pswapi_sdk.InitBalance(initbalance, userdata.PubKey) check(err) setup.ChainCodeID = txchaincode _, err = sdk_client.Invoke(setup, "init", [][]byte{[]byte(userdata.PubKey), []byte(balanceInfo)}) if err != nil { fmt.Println("Register error for user: ", senderAddr, err.Error()) } else { fmt.Println("init balance successfully: ", senderAddr) } check(err)

查询代码

- balanceInfo,err:=pswapi_sdk.InitBalance(balance,userdata.PubKey) check(err)
- setup.ChainCodeID = "TxChaincode" _, err = sdk_client.Invoke(setup, "init", [][]byte{[]byte(userdata.PubKey), []byte(balanceInfo)}) if err != nil { fmt.Println("Register error for user: ", senderAddr, err.Error()) } else { fmt.Println("Register success: ", senderAddr, "->", string(userdata.PubKey)) }

未来——区块链发展思路

THANK YOU

Building a better connected world

演讲者 / 张子怡

HUAWEI TECHNOLOGIES CO., LTD.