YAPE 区块链隐私计算框架

发行版本 *0.1*

YAPE

2023 年 08 月 24 日

入门

YAPE 是面向区块链的数据安全共享和隐私计算的密码学框架，通过 YAPE 框架，区块链系统和应用的开发者可以实现基于区块链的流通过程中“数据可用不可见”的安全解决方案。YAPE 框架采用安全多方计算、秘密分享、同态加密、零知识证明等密码学方案，基于 YAPE 框架的解决方案相对于基于联邦学习、可信执行环境等技术的方案，具有可证明安全的更高安全性，也具有不依赖特定型号的 TEE 硬件的广泛的适用性。

YAPE 是一个进行安全数据共享、隐私信息数据交易和计算的通用平台，大幅降低区块链数据安全应用开发的商业门槛和技术门槛，使掌握 Solidity 智能合约和主流通用编程语言的开发者可以快速开发“数据可用不可见”的安全解决方案。

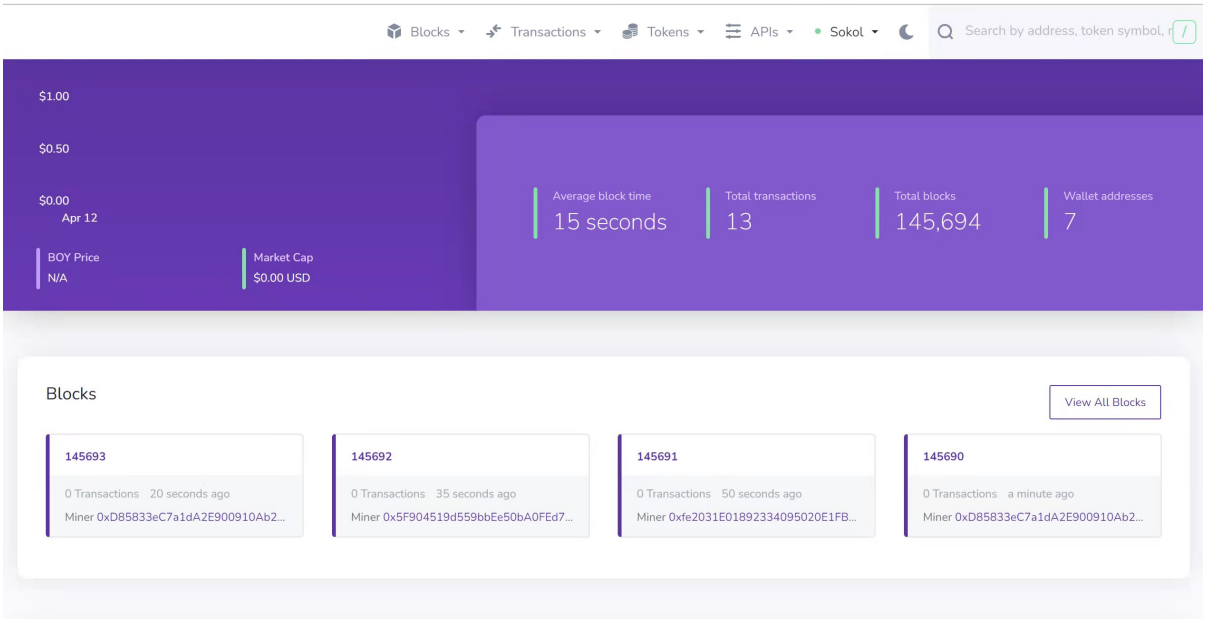
YAPE 隐私执行框架主要包括 EVM 预编译合约、面向典型安全场景的可扩展密码组件合约、隐私数据交易市场合约、链下通用隐私执行环境和应用端 SDK 等组件构成。框架目前已经集成在博雅链测试网络中。

备注： YAPE 框架中的部分关键功能依赖于博雅链 EVM 虚拟机执行环境，开发者和用户需要检查 YAPE 框架的版本和 EVM 虚拟机的版本兼容性，具体参考 [YAPE 版本号]

CHAPTER 1

区块链浏览器

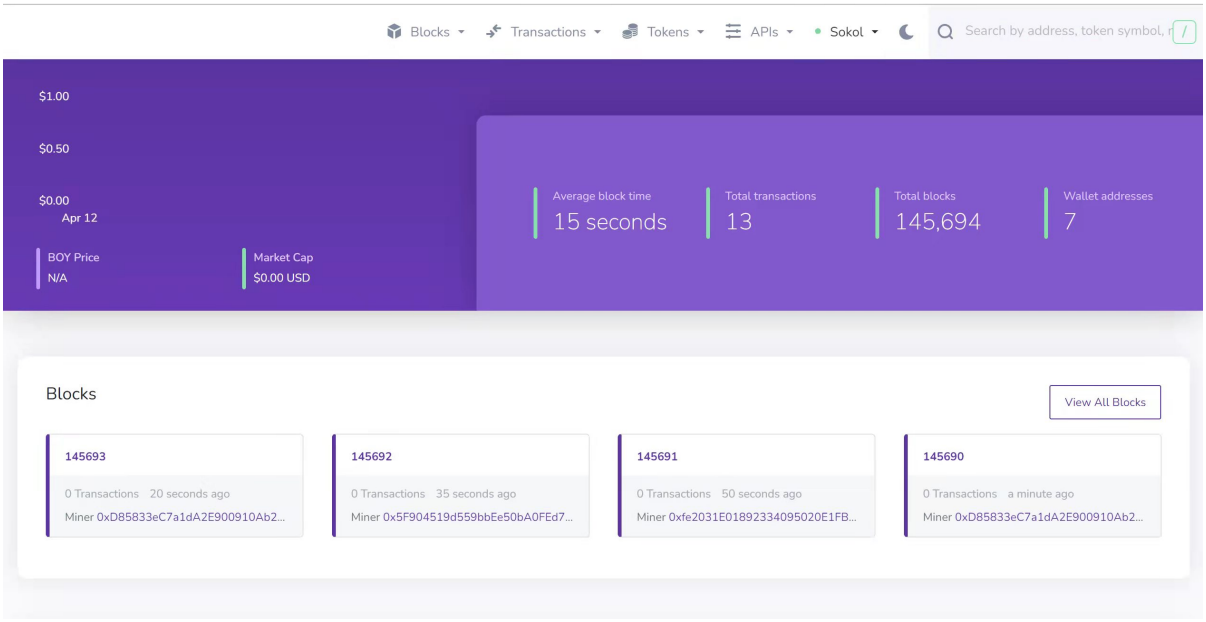
博雅链 BON 测试网络提供了区块链浏览器，即供用户浏览与查询区块链所有信息的 Web 应用。获得许可的用户也可以通过连接 Yallet 客户端向区块链中写入数据。



CHAPTER 2

Yallet 轻节点客户端

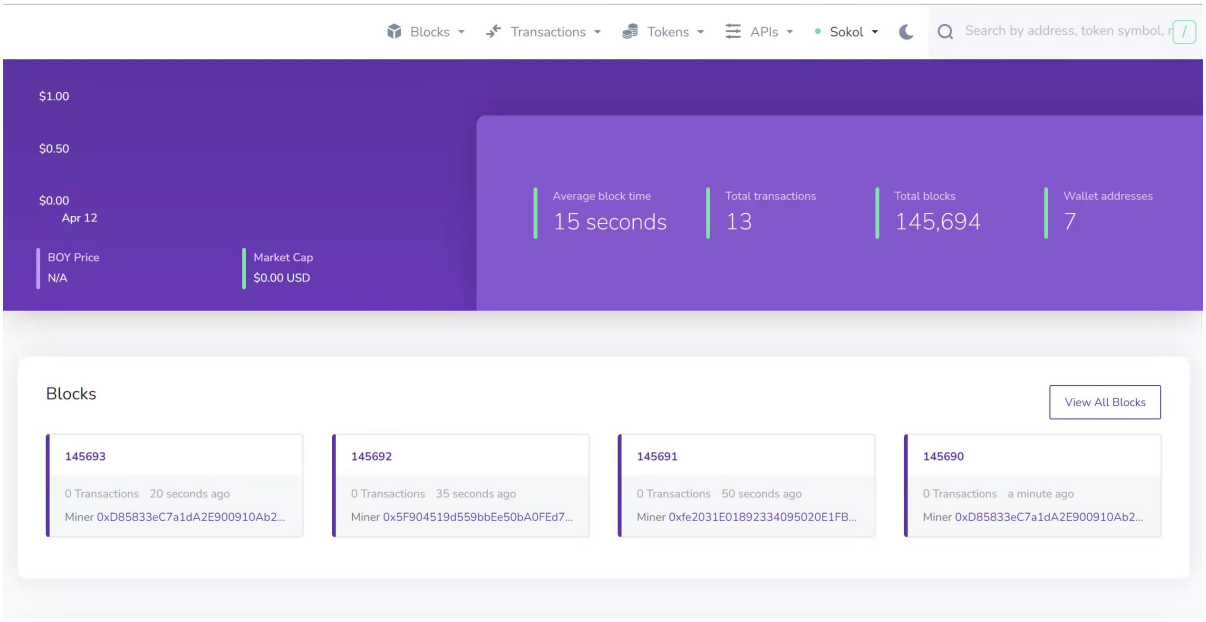
Yallet 轻节点客户端



CHAPTER 3

全节点客户端

全节点客户端



3.1 博雅链测试网络 (BON)

博雅链公开测试网络 (Boya Open testing Network, BON) 是基于博雅链区块链平台构建的的区块链网络，其中区块链基础平台软件由北京大学开发，网络和共识节点由网络联盟成员共同维护，并且以联盟链的架构运营，网络对公众提供公开的智能合约运行环境，获得许可的个人或组织可以在区块链网络上部署智能合约或发起交易，全部区块链数据以只读的方式向不具有许可权限的公众公开的。

BON 网络提供公开的并完全兼容以太坊 EVM 的智能合约运行环境，去中心化应用 (dApp) 的应用开发商无需独立部署区块链系统，甚至无需部署区块链全节点，就可以在 BON 网络上部署合约及其他必须的组件，可以满足区块链小微企业甚至个人开发者的区块链应用诉求。

BON 网络通过系统合约、预部署合约等方式在网络中引入去中心化身份管理、数据交易市场、存证溯源等大量智能合约框架，以及通过大量引入成熟的第三方合约和 dApp，为 dApp 合约开发者提供可组合的智能合约开发环境，简化典型 dApp 的开发工作量，打造丰富的 dApp 生态。

BON 采用可扩展的区块链共识方案，在保持去中心化和安全性的前提具有更高的可扩展性，通过将计算与共识分离，仅在选举出的代理节点之间进行共识，避免全节点共识带来的资源浪费，并通过可验证计算简化计算的验证过程，提高计算和共识的效率，实现亚秒级的交易确认时间和高并发的交易吞吐量。

BON 网络具有更好的安全性和合规性，在数据加密和认证中采用国家商用密码标准和网络安全标准，满足金融信息系统信息安全等级的保护要求。通过引入 YAPE 隐私执行环境，基于安全多方计算、同态加密、零知识证明等密码学技术保证数据安全和隐私保护。通过支持具有内生安全属性的 RegLang 智能合约编程语言及形式化验证技术，保证智能合约的安全性、功能正确性和可审计性。

目前，BON 网络已经在公有云和教育网中部署了多个共识节点和一个全节点云服务，完成测试网上线的里程碑节点，并开始向测试用户提供服务。

3.1.1 网络部署

获得许可的用户可以通过 BaaS 服务平台 (https://bonbaas.com) 查看当前网络部署情况及共识节点运行状态。

3.1.2 区块链浏览器

公众可以通过 https://bonexplo.io 区块链浏览器查看区块数据和合约状态。

3.1.3 加入共识节点

联盟成员有权限维护共识节点，并通过联盟投票确认共识节点的数量和运行维护机制。

3.2 开发第一个区块链应用

这个实例主要针对没有区块链去中心化应用经验的开发者，这里我们以一个简化的存证应用为例，介绍如何在区块链上开发、部署一个完整的应用。注意，本实例不依赖 EVM 虚拟机中的 YAPE 扩展功能，因此这个例子应用也可以运行在其他区块链的 EVM 环境中。

典型的数据库 Web 应用主要包括前端界面、后端应用逻辑和数据库构成，其中前端界面主要由 HTML/JavaScript 实现，后端应用以 Java/PHP/Python/Node 等语言实现，数据库如 MySQL 通常运行在一个专用的服务器中，通过网络和后端应用所在的服务器连接。用户在访问一个数据库应用时，首先需要通过用户名和口令登录到应用系统中，用户的操作请求被后端应用逻辑转化为 SQL 查询请求，发送给数据库，数据库运行 SQL 查询后给应用端返回查询结果，最终通过浏览器将结果呈现给用户。基于区块链的去中心化应用 (dApp) 和数据库应用大体类似，区别在于，用户在登录应用时不使用口令登录，而是通过浏览器插件客户端以数字签名的方式进行高安全等级的认证，后端数据库被区块链取代，后端应用对数据库的 SQL 查询请求也被智能合约查询请求所取代。

下图显示了一个 dApp 的主要组成部分：浏览器插件客户端、前端和应用逻辑、智能合约。

下面给出具体的代码示例和开发过程，这个例子只依赖 Remix 开发环境、JavaScript/HTML 代码编辑器和 Yallet 客户端。

3.3 开发一个 YAPE 应用

身份管理服务的 Identity 合约：

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

import "@openzeppelin/contracts/access/Ownable.sol";

contract Identity is Ownable {

    struct User{
        uint256 s;
        uint256 r;
        int n;
        string proof1;
        string proof2;
        uint256 register_status;//0:not registered, 1:deleted, 2:in
        uint256 register_time;
        uint256 approval_num;
        mapping(address => bool) approval_from_committee;
    }
```

(续下页)

(接上页)

```
    struct CA{
        address addr;
        uint256 pubkey_x;
        uint256 pubkey_y;
    }

    //struct for user_info
    struct sub_Info{
        address addr;
        uint256 tx_hash;
        string[] sec_list;
        mapping(address => bool) approval_from_committee;
        string info;
    }

    CA public ca;
    mapping(address => User) public users;
    mapping(address => bool) public user_list;//the list for users in status: 1,2
    mapping(address => bool) public committee;

    uint256 public min_approval_num = 2;

    uint256 apply_reveal_num = 0;
    mapping(uint256 => sub_Info) public user_info;

    uint256[] public root_list;

    event reveal(uint256 apply_id);

    function set_ca(address ca_addr, uint256 pubkey_x, uint256 pubkey_y) public␣
↪onlyOwner {
        ca.addr = ca_addr;
        ca.pubkey_x = pubkey_x;
        ca.pubkey_y = pubkey_y;
    }

    function add_committee(address addr) public onlyOwner {
        committee[addr] = true;
    }

    function remove_committee(address addr) public onlyOwner {
        committee[addr] = false;
    }
```

(续下页)

(接上页)

```
    function set_num(uint256 n) public onlyOwner {
        min_approval_num = n;
    }

    function update_root(uint256 root) public onlyOwner {
        root_list.push(root);
    }

    function register(uint256 s, uint256 r, int n, string  calldata proof1, string␣
↪calldata proof2) public{
        User storage u = users[msg.sender];
        u.s = s;
        u.r = r;
        u.n = n;
        u.proof1 = proof1;
        u.proof2 = proof2;
    }

    function approve_register(address addr) public{
        require(committee[msg.sender], "Only member of committee can call this");
        require(!users[addr].approval_from_committee[msg.sender], "Each member of␣
↪committee can only approve of each user_register once.");

        users[addr].approval_num += 1;
        users[addr].approval_from_committee[msg.sender] = true;

        if(users[addr].approval_num >= min_approval_num){
            users[addr].register_status = 2;
            users[addr].register_time = block.timestamp;
            user_list[addr] = true;
        }
    }

    function check(address addr) public view returns (uint){
        if(!user_list[addr]){
            return 0;
        }
        else if(users[addr].register_status == 2){
            return users[addr].register_time;
        }
        else{
            return 1;
```

(续下页)

(接上页)

```
        }
    }

    function apply_reveal(address addr, uint256 tx_hash) public returns (uint){
        apply_reveal_num += 1;
        sub_Info storage si = user_info[apply_reveal_num];
        si.addr = addr;
        si.tx_hash = tx_hash;
        return apply_reveal_num;
    }

    function agree_reveal(uint256 apply_id, string calldata sec) public{
        require(committee[msg.sender], "Only member of committee can call this");
        require(!user_info[apply_id].approval_from_committee[msg.sender], "Each␣
↪member of committee can only approve of each user_info once.");

        user_info[apply_id].sec_list.push(sec);
        user_info[apply_id].approval_from_committee[msg.sender] = true;

        if(user_info[apply_id].sec_list.length >= min_approval_num){
            emit reveal(apply_id);
        }
    }

    function do_reveal(uint256 apply_id, string calldata info) public{
        require(msg.sender == ca.addr, "Only CA can call this.");

        user_info[apply_id].info = info;
    }

}
```

下面我们基于身份管理服务实现一个身份验证的例子：

- 注册 user 地址：0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB，注册时间：1648530833。
- 未注册 user 地址：0x617F2E2fD72FD9D5503197092aC168c91465E7f2。

Identity_test 合约：

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Identity_test{
```

(续下页)

(接上页)

```
    address registered_user = 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB;
    address unregistered_user = 0x617F2E2fD72FD9D5503197092aC168c91465E7f2;

    function test(address Identity_contract_addr) public returns (uint256, uint256){
        (bool success1, bytes memory bytes_result_registered_user) = Identity_
↪contract_addr.call(abi.encodeWithSignature("check(address)", registered_user));
        require(success1, "failed");
        (bool success2, bytes memory bytes_result_unregistered_user) = Identity_
↪contract_addr.call(abi.encodeWithSignature("check(address)", unregistered_user));
        require(success2, "failed");

        uint256 result_registered_user = bytesToUint(bytes_result_registered_user);
        uint256 result_unregistered_user = bytesToUint(bytes_result_unregistered_
↪user);

        return (result_registered_user, result_unregistered_user);
    }

    function bytesToUint(bytes memory b) public pure returns (uint256){

        uint256 number;
        for(uint i = 0; i < b.length; i++){
            number = number + uint8(b[i])*(2**(8*(b.length - (i + 1))));
        }
        return number;
    }

}
```

- 部署 Identity_test 合约，并调用 test 函数，得到返回结果：
- 返回结果与用户实际注册状态一致。

3.4 浏览器插件客户端 (Yallet)

Yallet 是博雅链的浏览器插件客户端，主要作为个人用户访问区块链的客户端入口。用户无需安装独立的桌面程序，只需要在 Web 浏览器中安装一个 Yallet 插件，就可以方便地管理区块链账户和私钥，并且通过 Yallet 客户端在博雅链上签发交易及部署合约。通过装有 Yallet 钱包的 Web 浏览器，用户可以访问博雅链上的去中心化身份服务和隐私数据交易服务，体验各种第三方开发的去中心化应用 (DApp)。

Yallet 钱包属于轻节点客户端，仅保存用户个人的账户信息和私钥数据，不同步全部区块链数据，因此不需要在本地维护区块链全节点并保存大量的区块数据，对用户的计算环境和存储空间要求很低。Yallet 客户单作为轻节点客户端需要通过连接一个全节点才能发送交易并访问区块链，Yallet 钱包默认配置采用博雅链官方的区块链全节点服务（https://bon-mainnet.boyanet.io/api），用户也可以自行在本地部署全节点为 Yallet 提供 RPC 服务。

Yallet 主要面向个人用户使用场景，因此在隐私性和安全性上低于全节点钱包，如果应用需要实现高吞吐量的区块链交互，应该选择博雅链的全节点客户端。

3.4.1 下载和安装

Yallet 当前版本仅支持 Google Chrome 浏览器，并可以通过两种方式在 Chrome 浏览器中安装 Yallet 插件。

通过 Chrome 插件市场安装，在 Chrome Web Store 中查找 Yallet，如下图所示可以找到 Yallet 插件，点击安装并完成安装后，可以在浏览器工具栏看到 Yallet 图标。

通过 Chrome 插件市场只能够安装当前主线版本的 Yallet 插件，如果需要安装特定版本或者开发版插件，需要用户在官网下载 Yallet 插件的安装包手动安装，并修改 Chrome 的权限以启用该插件。

3.4.2 功能与使用说明

和常规的区块链客户端类似，Yallet 客户端主要包括私钥管理功能、交易签名功能、身份管理功能等。

下面通过界面截图的方式介绍 Yallet 的具体使用过程。

3.5 命令行全节点客户端（boyanetool）

博雅正链测试网络使用 boyanetool 作为命令行工具，来启动区块链网络、与其他客户建立 p2p 通信信道、签署和广播交易、挖掘，部署和与智能合约交互等。该工具由 golang 语言编写，需要完整的 go 语言编译与执行环境。

boyanetool 主要由初始化、账号管理、区块数据文件管理、交互终端（JavaScript）以及关键系统参数命令组成。初始化主要完成创世节点的配置工作，在指定创世文件后，该命令能够在指定的文件夹内创建创世区块；账号管理负责生成、导入、更新和展示区块链网络中的账户；区块数据文件管理主要对存储与本地的区块数据进行读取、导入、导出等操作；交互终端是 boyanetool 的重要命令，通过 javescript 环境的 API 接口来与区

块链节点进行通信，完成区块信息查询与交易的签名、部署等；关键系统参数主要用来规制区块链网络工作节点的具体特性和行为。

要加入 boyanetool 网络，首先要初始化创世节点。在创世节点配置文件中配置好链 ID，gaslimit, 初始代币分配等，就可以进行初始化了。

```
boyanetool init --datadir data genesis.json
```

完成初始化后，我们就可以加入网络了。为了更顺利的加入网络，我们可以指定启动节点。节点运行的方式有 3 种：全节点、轻节点和快照模式。

```
boyanetool --datadir data --networkid 15 --gcmode full --bootnodes enode://
↪6c213a1332c861e9b651c52f42da79b2efbc0b99b90101a15c1aec53afa6492a6df87494fe540702687f960b91a93cfb7a2
↪142.139.142:0?discport=30301
```

为了能使节点对外提供服务，我们可以指定 API 地址、端口和具体功能。提供服务的方式有两种：http 和 ws。

```
boyanetool --http --http.port 8545 --http.addr 172.25.101.59 --http.api debug,net,eth,
↪shh,web3,txpool,personal --ws --ws.port 8546 --ws.addr 172.25.101.59 --ws.api eth,
↪net,web3,network,debug,txpool,personal
```

当然，为了方便交易，或者挖矿，我们可以在节点启动的时候就对账户进行解锁。

```
boyanetool --allow-insecure-unlock --unlock␣
↪0x13b21840D8f478b18aA337D5248a115Ce065E1AE --password password --mine
```

下面是 boyanetool 的详细帮助信息：

```
NAME:
   boyanetool - the boyanet command line interface

USAGE:
   boyanetool [options] [command] [command options] [arguments...]

COMMANDS:
   account                          Manage accounts
   attach                           Start an interactive JavaScript environment␣
↪(connect to node)
   console                          Start an interactive JavaScript environment
   db                               Low level database operations
   dump                             Dump a specific block from storage
   dumpconfig                       Show configuration values
   dumpgenesis                      Dumps genesis block JSON configuration to stdout
   export                           Export blockchain into file
   export-preimages                 Export the preimage database into an RLP stream
   import                           Import a blockchain file
```

(续下页)

(接上页)

```
   import-preimages                  Import the preimage database from an RLP stream
   init                              Bootstrap and initialize a new genesis block
   js                                Execute the specified JavaScript files
   license                           Display license information
   removedb                          Remove blockchain and state databases
   snapshot                          A set of commands based on the snapshot
   version                           Print version numbers
   wallet                            Manage presale wallets
   help, h                           Shows a list of commands or help for one command

BOYANET OPTIONS:
  --config value                     TOML configuration file
  --datadir value                    Data directory for the databases and keystore
  --datadir.ancient value            Data directory for ancient chain segments␣
↪(default = inside chaindata)
  --datadir.minfreedisk value        Minimum free disk space in MB, once reached␣
↪triggers auto shut down (default = --cache.gc converted to MB, 0 = disabled)
  --keystore value                   Directory for the keystore (default = inside␣
↪the datadir)
  --usb                              Enable monitoring and management of USB␣
↪hardware wallets
  --pcscdpath value                  Path to the smartcard daemon (pcscd) socket␣
↪file (default: "/run/pcscd/pcscd.comm")
  --networkid value                  Explicitly set network id (integer)(For␣
↪testnets: use --ropsten, --rinkeby, --goerli instead) (default: 1)
  --syncmode value                   Blockchain sync mode ("snap", "full" or "light
↪") (default: snap)
  --exitwhensynced                   Exits after block synchronisation completes
  --gcmode value                     Blockchain garbage collection mode ("full",
↪"archive") (default: "full")
  --txlookuplimit value              Number of recent blocks to maintain␣
↪transactions index for (default = about one year, 0 = entire chain) (default:␣
↪2350000)
  --identity value                   Custom node name
  --lightkdf                         Reduce key-derivation RAM & CPU usage at some␣
↪expense of KDF strength


ACCOUNT OPTIONS:
  --unlock value                     Comma separated list of accounts to unlock
  --password value                   Password file to use for non-interactive␣
↪password input
  --signer value                     External signer (url or path to ipc file)
```

(续下页)

(接上页)

```
  --allow-insecure-unlock              Allow insecure account unlocking when account-
↪related RPCs are exposed by http

API AND CONSOLE OPTIONS:
  --ipcdisable                         Disable the IPC-RPC server
  --ipcpath value                      Filename for IPC socket/pipe within the datadir␣
↪(explicit paths escape it)
  --http                               Enable the HTTP-RPC server
  --http.addr value                    HTTP-RPC server listening interface (default:
↪"localhost")
  --http.port value                    HTTP-RPC server listening port (default: 8545)
  --http.api value                     API's offered over the HTTP-RPC interface
  --http.rpcprefix value               HTTP path path prefix on which JSON-RPC is␣
↪served. Use '/' to serve on all paths.
  --http.corsdomain value              Comma separated list of domains from which to␣
↪accept cross origin requests (browser enforced)
  --http.vhosts value                  Comma separated list of virtual hostnames from␣
↪which to accept requests (server enforced). Accepts '*' wildcard. (default:
↪"localhost")
  --ws                                 Enable the WS-RPC server
  --ws.addr value                      WS-RPC server listening interface (default:
↪"localhost")
  --ws.port value                      WS-RPC server listening port (default: 8546)
  --ws.api value                       API's offered over the WS-RPC interface
  --ws.rpcprefix value                 HTTP path prefix on which JSON-RPC is served.␣
↪Use '/' to serve on all paths.
  --ws.origins value                   Origins from which to accept websockets requests
  --authrpc.jwtsecret value            Path to a JWT secret to use for authenticated␣
↪RPC endpoints
  --authrpc.addr value                 Listening address for authenticated APIs␣
↪(default: "localhost")
  --authrpc.port value                 Listening port for authenticated APIs (default:␣
↪8551)
  --authrpc.vhosts value               Comma separated list of virtual hostnames from␣
↪which to accept requests (server enforced). Accepts '*' wildcard. (default:
↪"localhost")
  --graphql                            Enable GraphQL on the HTTP-RPC server. Note␣
↪that GraphQL can only be started if an HTTP server is started as well.
  --graphql.corsdomain value           Comma separated list of domains from which to␣
↪accept cross origin requests (browser enforced)
  --graphql.vhosts value               Comma separated list of virtual hostnames from␣
↪which to accept requests (server enforced). Accepts '*' wildcard. (default:
↪"localhost")
```

(续下页)

(接上页)

```
  --rpc.allow-unprotected-txs         Allow for unprotected (non EIP155 signed)␣
↪transactions to be submitted via RPC
  --jspath loadScript                 JavaScript root path for loadScript (default: ".
↪")
  --exec value                        Execute JavaScript statement
  --preload value                     Comma separated list of JavaScript files to␣
↪preload into the console

NETWORKING OPTIONS:
  --bootnodes value                   Comma separated enode URLs for P2P discovery␣
↪bootstrap
  --discovery.dns value               Sets DNS discovery entry points (use "" to␣
↪disable DNS)
  --port value                        Network listening port (default: 30303)
  --maxpeers value                    Maximum number of network peers (network␣
↪disabled if set to 0) (default: 50)
  --maxpendpeers value                Maximum number of pending connection attempts␣
↪(defaults used if set to 0) (default: 0)
  --nat value                         NAT port mapping mechanism␣
↪(any|none|upnp|pmp|extip:<IP>) (default: "any")
  --nodiscover                        Disables the peer discovery mechanism (manual␣
↪peer addition)
  --v5disc                            Enables the experimental RLPx V5 (Topic␣
↪Discovery) mechanism
  --netrestrict value                 Restricts network communication to the given IP␣
↪networks (CIDR masks)
  --nodekey value                     P2P node key file
  --nodekeyhex value                  P2P node key as hex (for testing)

MINER OPTIONS:
  --mine                              Enable mining
  --miner.threads value               Number of CPU threads to use for mining␣
↪(default: 0)
  --miner.notify value                Comma separated HTTP URL list to notify of new␣
↪work packages
  --miner.notify.full                 Notify with pending block headers instead of␣
↪work packages
  --miner.gasprice value              Minimum gas price for mining a transaction␣
↪(default: 1000000000)
  --miner.gaslimit value              Target gas ceiling for mined blocks (default:␣
↪30000000)
  --miner.base value             Public address for block mining rewards (default =␣
↪first account) (default: "0")
```

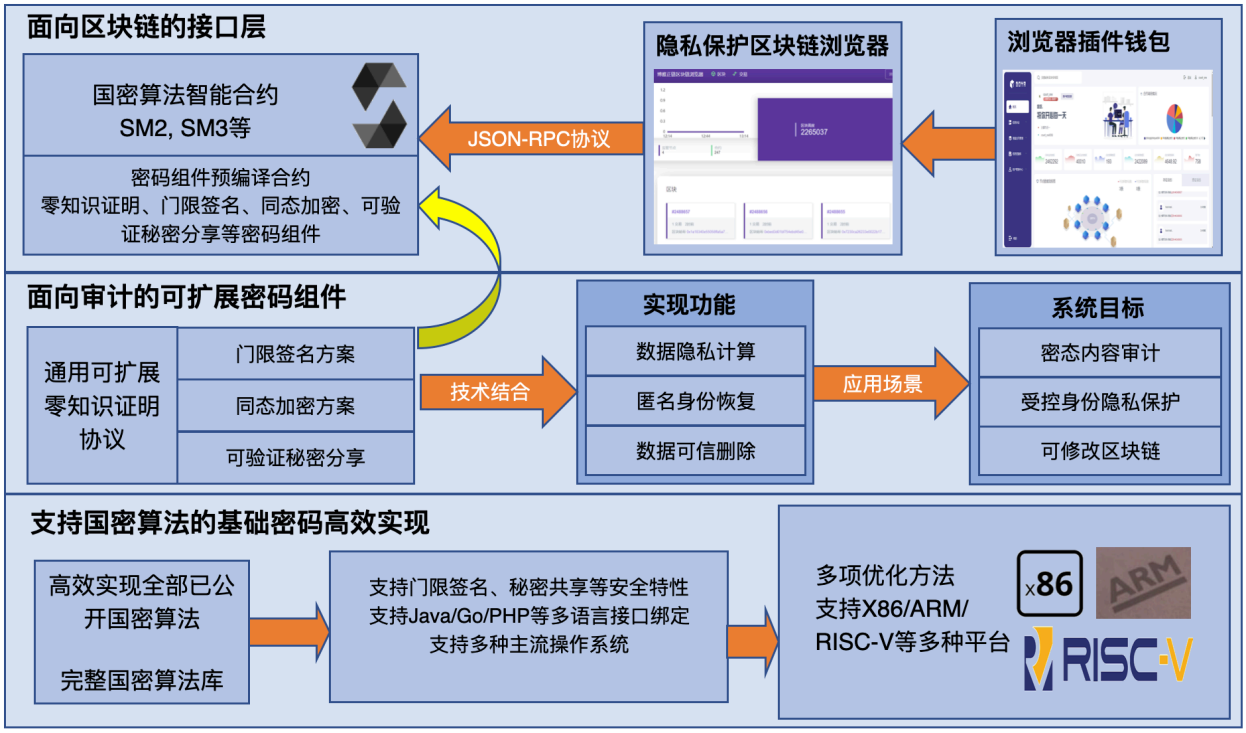
(续下页)

(接上页)

```
  --miner.extradata value             Block extra data set by the miner (default =␣
↪client version)
  --miner.recommit value              Time interval to recreate the block being mined␣
↪(default: 3s)
  --miner.noverify                    Disable remote sealing verification

MISC OPTIONS:
  --help, -h                                show help
```

3.6 隐私执行环境简介



3.7 多方安全计算

3.8 隐私集合求交

3.8.1 隐私求交技术简述

隐私数据保护最早源于安全多方计算 (secure multiparty computation,MPC)，由姚期智借百万富翁问题提出，指各计算参与方无法得到除计算结果外的任何其他信息，解决互不信任的数据持有者如何对隐私数据进行计算的问题. 隐私求交 (Private Set Intersection ,PSI)，是安全多方计算中的一个典型问题，指参与双方在不泄露任何额外信息的情况下，得到双方持有数据的交集。在这里，额外的信息指的是除了双方的数据交集以外的任何信息。在需要隐私保护的场景中，PSI 技术具有重要意义，包括实名认证（多因素核验）、联合风控、数据对齐、数据发现等。对隐私求交协议的简单分类可以从两个方面入手，从安全假设角度分类，隐私求交协议在发展中从诚实模型、半诚实模型逐步发展到恶意模型下的 PSI 协议；而从参与方数量分类，隐私求交协议也从最初的两方隐私求交协议逐步扩展到目前的各种多方隐私求交协议。Meadows 基于公钥加密和利用 Diffie—Hellman 密钥交换的乘法同态性质提出了第 1 个 PSI 协议。随后，由 Huberman 等人对 Meadows 的方案做出了完整描述。以下便是对基于 Diffie-Hellman 密钥交换的 PSI 协议的简要流程介绍：

- A 拥有数据 x_1,x_2,⋯,x_m，B 拥有数据 y_1,y_2,⋯,y_n

$$U_A = \left(H(x_1)\right)^{\alpha},\ \left(H(x_2)\right)^{\alpha},.$$

- A 对数据哈希并使用加密得到 U_A,并发送给 *B*

$$U_B = \left(H(y_1)\right)^{\beta},\ \left(H(y_2)\right)^{\beta},$$

- B 对数据哈希并使用加密得到 U_B,并发送给 *A*。

$$U_{AB} = ((H(x_1)^{\alpha})^{\beta}, (H(x_2)^{\alpha})^{\beta},$$

- 同时 B 对收到的 U_A 加密得到 U_AB,并发送给 A。

$$U_{BA} = \left(\left(H(y_1)^{\beta}\right)^{\alpha}, \left(H(y_2)^{\beta}\right)^{\alpha}, \ldots \left(H(y\right.$$

- A 对收到的 U_B 加密得到 U_BA
- 求取 U_AB 和 U_BA 的交集，就是两方所共同持有的数据。
- 该算法的核心是需要找到一个满足连续两次加密操作（两方先后加密）可以交换顺序的加密算法，实际应用中，此加密算法需要事先约定一个大素数，加密操作为用各自选定的 α,β 值进行幂次运算并针对大素数取模。

之后涌现了大量 PSI 的研究成果，一大批新技术手段和构造框架被提出。除了传统的安全多方计算理论中的混淆电路 (garbled circuit，GC)、不经意传输 (oblivious transfer，OT)、秘密共享 (secret sharing，SS)、同态加密 (homomorphic encryption，HE) 等技术外，不经意伪随机函数 (oblivious pseudo random function，OPRF)、不经意多项式求值 (oblivious polynomial evaluation，OPE)、布隆过滤器 (Bloom filter，BF) 等集合元素比较技术的应用，使得 PSI 的效率得到了很大的提高。

随着研究人员对隐私集合交集协议的深入研究，除了传统两方 PSI 协议之外，已衍生出了云辅助 PSI、阈值 PSI(threshold PSI，TPSI)、不平衡 PSI(unbalanced PSI，UPSI) 和多方 PSI 新型应用场景。其中云辅助 PSI 是随着云技术的发展和成熟逐渐成为热点，云服务器具有高计算能力和存储容量，为现有的 PSI 协议提供了成熟的优化方法，云辅助 PSI 协议可分为数据加密和数据盲化 2 种。阈值 PSI 指当交集的基数大于或等于门限值时，接收方才能获得隐私集合交集，如网约顺风车场景中，在不泄露陌生人路径的情况下如何共享双方的公共路径是该场景的重点问题。不平衡 PSI 是在某些实际应用——如隐私联系人发现——中，客户拥有几百到几千个联系人集合，而服务方拥有百万甚至千万级的用户集合，它们的集合大小不平衡，技术能力和存储空间也相差甚远，使用传统 PSI 协议，它们的通信开销和计算开销均与较大集合成线性关系，会导致客户端需承受巨大的存储与计算开销，这也就激发了对不平衡 PSI 的研究。隐私集合交集目前存在的高效协议大多只针对两方设置，多方高效的 PSI 协议仍有长足的发展空间，这可能与各方之间不可避免的通信而导致极大的通信成本有关。目前关于多方设置的 PSI 协议大多采用 2 种网络模型：1) 星型拓扑网络结构减少双方之间的中间交流，但给指定方带来了很高的工作量；2) 星型——路径网络结构其使每一方 (除指定方) 的通信量和计算复杂度仅取决于自身输入集大小。目前，PSI 技术未来的主要研究方向主要在 4 个方面：一是考虑威胁性更高的场景；二是考虑更多的参与方；三是面向通过 PSI 协议得到交集数据仍然是敏感数据的应用场景构建定制化协议，允许参与方在交集保密的情况下对交集中的元素进行进一步的处理与运算操作；四是提升现有方案效率，构建面向海量数据的高效 PSI 协议，提升大数据场景下的运算速度。

3.9 零知识证明

零知识证明通常是指一种方法，其中的一个参与方（证明者）可以向另一方（验证者）证明某一论断为真（例如：拥有某一数学问题的一组解），而不泄露关于“此论断为真”以外的任何信息。注意到在此例子中，一个朴素的证明方法就是直接公开拥有的解，因此难点是在不泄露任何关于解的信息的同时实现证明。

目前已有的零知识证明协议大多具有如下形式：对于某个问题 P 和值 y，证明者拥有 x 使得 $P(x)=y$。证明者利用 x 计算出一组数据（证明）并交给验证者验证，如果验证者证实这些数据确实满足协议中给出的特定关系（通常是一组等式），则验证者相信证明者确实拥有 x 使得 $P(x)=y$。

零知识证明协议必须满足以下性质：

- 完整性：如果 $P(x)=y$，则证明者使用 x 生成的证明总应该被接受；
- 可靠性：对于任何不满足 $P(x)=y$ 的 x，证明者使用 x 生成的证明最多以一个小概率 ρ 被接受（这使得验证者可以通过多次要求证明的方式鉴别虚假的证明者）；
- 零知识：验证者不能从证明中获得关于 x 的任何信息。

目前常用的零知识证明协议包括 Groth16、Sonic、Plonk 等，这些协议均能够实现对常规计算问题的证明。

零知识证明在区块链系统中有非常重要的应用。例如，交易发起方可以为交易节点生成正确性证明，验证节点可以在不知道交易内容的情况下对交易的有效性进行确认，从而增强区块链的隐私性。同时，零知识证明可以将大部分计算转移到链下，区块仅对计算结果的正确性进行校验，通过减少区块链的计算和存储开销提高其可扩展性。

3.10 可验证计算

可验证计算是指：数据拥有者将数据的计算任务外包给不完全可信、但与数据持有者相比拥有更强算力的第三方。同时，第三方在完成计算任务的同时，需要提交一份关于计算结果的正确性证明，以使数据拥有者确认计算被正确地完成。

可验证计算的形式化描述为：

1. 验证者 V 把程序 f 和输入变量 x 发送给证明者 P，P 计算 $f(x)$ 并返回一个值 y；
2. 如果 $y=f(x)$，那么 P 有能力向 V 证明 y 的正确性（一般以提供证明数据的方式），即使得 V 接受 y；
3. 如果 $y\neq f(x)$，V 能以很高的概率拒绝接受 y。

在实际应用中，可验证计算协议通常包含如下步骤：

1. 编译系统将需要计算的程序转换为数学模型，程序正确执行等价于数学模型可满足；
2. 程序正确执行后，计算协议通过数学模型生成一组满足特定关系的代数数据，作为正确执行的证明。

可验证计算的一个基本原则是：在使用的协议中，验证者对证明者提供的证明进行验证的开销应小于验证者自己完成计算任务的开销。因此，可验证计算可以和区块链系统结合，证明者收到数据后，将开销较大的计算任务在链下完成，仅将证明公布上链，以区块链透明且不可修改的性质为其提供担保，并通过智能合约等手段快速完成证明数据的验证工作。

3.11 同态加密

同态加密是一种公钥加密算法，有别于常规的公钥加密算法，同态加密支持密文上的计算。YAPE 通过预编译合约支持两种同态加密算法：SM2 同态加密和 Paillier 同态加密。这两种算法在功能上相同，当整型数值加密并通过交易上链后，智能合约可以调用对应的预编译合约对密文进行加法计算并得到和的密文。注意，预编译合约的调用方应保证计算不会溢出。

同态加密的预编译合约仅完成密文加法计算功能，数值的加密和解密算法需要通过调用 YAPE SDK 完成。

下面给出调用 SM2 密文加法的预编译合约的合约代码例子：

```
pragma solidity >=0.4.21;

contract Precompiles {
    function callBn256Pairing(bytes memory input) public returns (bytes32 result) {
```

(续下页)

(接上页)

```
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

下面给出调用 Paillier 密文加法的预编译合约的合约代码例子:

```
pragma solidity >=0.4.21;

contract Precompiles {
    function callBn256Pairing(bytes memory input) public returns (bytes32 result) {
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

3.12 函数加密

3.13 环签名验证

环签名是一种支持隐私保护的数字签名方案。常规的签名方案，如 SM2 签名和 ECDSA 签名，签名方的地址可以通过签名值导出，无法实现签名的匿名性。在环签名方案中，验证方仅能够验证一个有效的签名来自一个群组中的某个成员，但是无法获得确切的签名方身份信息。环签名用于需要身份匿名和隐私保护的场景，如投票、拍卖等。

YAPE 支持 SM2 环签名算法，其中预编译合约 (SM2RingVerify) 仅完成环签名验证功能，环签名的生成需要通过调用 YAPE SDK 完成。

```
pragma solidity >=0.8;

contract Precompiles {
    function callSM9Pairing(bytes memory input) public returns (bytes32 result) {
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

3.14 盲签名验证

盲签名是一种特殊的具有隐私保护功能的签名，签名者可以在不暴露消息内容的同时获得有效的签名值。博雅链 EVM 支持盲签名验签预编译合约。下面给出调用盲签名验签预编译合约的合约代码例子：

```
pragma solidity >=0.8;

contract Precompiles {
    function callSM9Pairing(bytes memory input) public returns (bytes32 result) {
```

(续下页)

(接上页)

```
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

3.15 代理重加密和代理重签名

代理重加密是一种基于公钥加密体系的密码学方案。在代理重加密中，基于授权人公钥加密的密文可以被转换为另一种密文，且保持对应明文不变，被转换后的密文可以由被授权人的私钥进行解密。该密文转换过程由一个半可信的第三方代理执行，在执行该过程前，代理需要持有一个由授权人到被授权人的转换密钥，该转换密钥一般由授权人事先生成并交给代理。同时在密文转换的整个过程中，代理者无法获取关于该密文对应明文的任何信息。这一方案使得数据拥有着可以将加密后的数据存放在云计算服务商等中间机构中，并在需要时允许其他用户下载并解密。

类似地，代理重签名允许第三方半可信代理使用一个重签名转换密钥，将授权人的签名转换为被授权人在同一消息上的签名，但代理并不能伪装为任何一方对任何消息产生合法的签名。

代理重加密功能通过基于预编译合约的 Solidity Library 实现。

```
pragma solidity >=0.8;

contract Precompiles {
    function callSM9Pairing(bytes memory input) public returns (bytes32 result) {
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
```

(续下页)

(接上页)

```
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

3.16 密文检索

密文检索功能通过基于预编译合约的 Solidity Library 实现。

```
pragma solidity >=0.8;

contract Precompiles {
    function callSM9Pairing(bytes memory input) public returns (bytes32 result) {
        // input is a serialized bytes stream of (a1, b1, a2, b2, ..., ak, bk) from␣
↪(G_1 x G_2)^k
        uint256 len = input.length;
        require(len % 192 == 0);
        assembly {
            let memPtr := mload(0x40)
            let success := call(gas, 0x08, 0, add(input, 0x20), len, memPtr, 0x20)
            switch success
            case 0 {
                revert(0,0)
            } default {
                result := mload(memPtr)
            }
        }
    }
}
```

3.17 SM2 数字签名算法验签

GMSSL 预编译合约的地址为：0x1000 - 0x10FF，总共预留了 256 个，sm2-verify 地址为 0x1002。0x0000 - 0x00FF reserved for ETH precompiles.

SM2 是国密椭圆曲线密码标准，其中包括加密、签名和密钥交换协议，SM2 标准默认使用一个 256 比特的素域曲线参数。SM2RECOVER 在功能上类似于以太坊的 ECRECOVER 预编译合约，SM2RECOVER 预编译可以用于验证标准的 SM2 签名或以太坊 EIP 风格的 SM2 签名。本文写作时 Solidity 编译器还不支持 SM2RECOVER 预编译合约，因此在编写合约时需要通过内联汇编 (Inline Assembly) 来调用 SM2RECOVER 合约。下面给出调用 SM2RECOVER 预编译合约的合约代码例子：

```
pragma solidity >=0.8;

contract SysContracts {

    function sm2verify(string memory spk, bytes memory sig, string memory sdata)␣
↪public view returns (uint8){
                //0-2, public key type, default pem
        //2, public key len
        //3, sig len
        //4-?, public key
        //?-?, sig
        //?-?, data

        //gas estimatie 156543

        //example
        /*
        ins.sm2verify("-----BEGIN PUBLIC KEY-----\
↪nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAESqZ8l4sMcyjURbwsPpyw6cFKaMGw\
↪nmIEBwVpMJkr+PB6C7ADq5NBERXQKdjzeNSweTXmVM5J4JZmZFC6MhuPrIg==\n-----END PUBLIC KEY--
↪---",
↪"0x3045022063CA1B61FE5CA093D11297722AC9555BF7CDED24B17C66FF567C9B9F7A94364C022100BBF90EED502DB7A9B1
↪","The Go appserver can access an array created by the C++ library")
        */


        bytes memory pk = bytes(spk);
        bytes memory data = bytes(sdata);

        uint8 pklen = uint8(pk.length);
        uint8 siglen = uint8(sig.length);
        uint32 datalen = uint32(data.length);
```

(续下页)

(接上页)

```
        bytes memory input = new bytes(pklen + siglen + datalen + 4);

        uint32 i;
        input[2] = bytes1(pklen);
        input[3] = bytes1(siglen);

        uint32 offset = 4;
        for(i=0;i<pklen;i++) input[offset+i] = pk[i];

        offset+= pklen;
        for(i=0;i<siglen;i++) input[offset+i] = sig[i];

        offset+= siglen;
        for(i=0;i<datalen;i++) input[offset+i] = data[i];

        bytes1[1] memory ret;
        assembly{
            if iszero(
                staticcall(100000, 0x0102, add(input, 32) , mload(input), ret, 1)
            ) {
                invalid()
            }
        }
        //ret = 0: verification successful, ret = 1: verification failed
        return uint8(ret[0]);
    }
}
```

3.17.1 合约调用示例

```
ins.sm2verify(
    "-----BEGIN PUBLIC KEY-----\
↪nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAESqZ8l4sMcyjURbwsPpyw6cFKaMGw\
↪nmIEBwVpMJkr+PB6C7ADq5NBERXQKdjzeNSweTXmVM5J4JZmZFC6MhuPrIg==\n-----END PUBLIC KEY--
↪---",

↪"0x3045022063CA1B61FE5CA093D11297722AC9555BF7CDED24B17C66FF567C9B9F7A94364C022100BBF90EED502DB7A9B1
↪",
"The Go appserver can access an array created by the C++ library")

Result: 0 (success)
```

3.18 SM3 哈希算法

GMSSL 预编译合约的地址为：0x1000 - 0x10FF，总共预留了 256 个，sm3-hash 地址为 0x1003。0x0000 - 0x00FF reserved for ETH precompiles.

SM3 哈希算法是国密密码杂凑算法标准，用于替代由美国 NIST 设计的 SHA-1、SHA-256 算法。SM3 输出的哈希值长度和 SHA-256 一样，均为 256 比特（32 字节或者 64 个十六进制字符），但是设计安全性比 SHA-256 更高。在标准的 EVM 虚拟机支持 SHA-256 预编译合约，博雅链 EVM 中增加了 SM3 算法的预编译合约。本文写作时 Solidity 编译器还不支持 SM3 预编译合约，因此在编写合约时需要通过内联汇编 (Inline Assembly) 来调用 SM3 合约。下面给出调用 SM3 预编译合约的合约代码例子：

```
pragma solidity >=0.8;

contract SysContracts {
        function sm3(string memory data) public view returns (bytes32){
        bytes32[1] memory h;
            assembly{
            if iszero(
                staticcall(1000000, 0x0103, add(data, 32) , mload(data), h, 32)
            ) {
                invalid()
            }
        }
        return h[0];
    }
}
```

3.18.1 合约调用示例

```
ins.sm3("hello world")
Result: '0x44f0061e69fa6fdfc290c494654a05dc0c053da7e5c52b84ef93a9d67d3fff88'
```

3.19 SM9 标识密码算法

GMSSL 预编译合约的地址为：0x1000 - 0x10FF，总共预留了 256 个，sm9 地址为 0x0109 - 0x0111。0x0000 - 0x00FF reserved for ETH precompiles.

SM9 是国密标识密码算法标准，其中包括标识加密、标识签名和标识密钥交换协议，SM9 标准中还包括一个推荐的 BN 曲线参数。博雅链 EVM 中增加了针对 SM9 曲线的椭圆曲线算术运算操作，合约可以通过调用椭圆曲线算法运算实现零知识证明、秘密共享、同态加密等上层密码方案。

3.19.1 SM9ADD - 0x0109

SM9Add 预编译合约实现 SM9 椭圆曲线点的点加操作，其输入是两个椭圆曲线点的仿射坐标，输出是两个结果点的仿射坐标。本文写作时 Solidity 编译器还不支持 SM9Add 预编译合约，因此在编写合约时需要通过内联汇编 (Inline Assembly) 来调用 SM9Add 合约。下面给出调用 SM9Add 预编译合约的合约代码例子：

```
pragma solidity >=0.8.1;

contract Precompiles {
    function sm9add(string memory x1, string memory y1, string memory x2, string␣
↪memory y2) public view returns (string memory x, string memory y){
        bytes memory input = new bytes(256);
        uint32 i;
        uint32 offset =0;

        bytes memory bx1 = bytes(x1);
        bytes memory by1 = bytes(y1);
        bytes memory bx2 = bytes(x2);
        bytes memory by2 = bytes(y2);

        //input
        //0-64 x1
        //64-128 y1
        //128-192 x2
        //192-256 y2
        for(i=0;i<64;i++) input[offset+i] = bx1[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = by1[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = bx2[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = by2[i];

        bytes32[4] memory ret32;

        assembly{
            if iszero(
                staticcall(100000, 0x0109, add(input, 32) , mload(input), ret32, 128)
            ) {
                invalid()
            }
        }

        bytes memory ret1 = abi.encodePacked(ret32);
        return (string(bytes_slice(ret1,0,64)),string(bytes_slice(ret1,64,128)));
    }
}
```

合约调用示例

```
ins.sm9add("917be49d159184fba140f4dfc5d653464e94f718fe195b226b3f715829e6e768",
"288578d9505d462867a50acee40ee143b896e72505be10e8ce4c6b0c945b642b",
"593417680f252445fd0522383e23c77a54b11fe222de4a886eabc26e16bffa3c",
"38e8fc9a8b60f5ba0c6c411f721c117044435a833757d8fee65828511b8b245d")
Result {
  '0': '5e4ff71938d96a2569d289dc13d3129fcf3dbc13ad7ebc53941c6ca331d400cb',
  '1': '10002bca9f0e8aa073ad25456d75c57238f7eb1883d8df0a99af5f46fc45b501'
}
```

3.19.2 SM9MUL - 0x0110

SM9Mul 预编译合约实现 SM9 椭圆曲线点的标量乘法操作，其输入是一个曲线点的仿射坐标和一个 uint256 整数表示的标量，输出是两个点乘操作的结果，点的仿射坐标。本文写作时 Solidity 编译器还不支持 SM9Mul 预编译合约，因此在编写合约时需要通过内联汇编 (Inline Assembly) 来调用 SM9Mul 合约。下面给出调用 SM9Mul 预编译合约的合约代码例子：

```
pragma solidity >=0.8.1;

contract Precompiles {
    function sm9mul(string memory x, string memory y, string memory iv) public view␣
↪returns (string memory , string memory ){
        bytes memory input = new bytes(192);
        uint32 i;
        uint32 offset =0;

        bytes memory bx = bytes(x);
        bytes memory by = bytes(y);
        bytes memory biv = bytes(iv);

        //input
        //0-64 x
        //64-128 y
        //128-192 iv

        for(i=0;i<64;i++) input[offset+i] = bx[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = by[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = biv[i];

        bytes32[4] memory ret32;

        assembly{
```

(续下页)

(接上页)

```
            if iszero(
                staticcall(100000, 0x0110, add(input, 32) , mload(input), ret32, 128)
            ) {
                invalid()
            }
        }

        bytes memory ret1 = abi.encodePacked(ret32);
        return (string(bytes_slice(ret1,0,64)),string(bytes_slice(ret1,64,128)));
    }
}
```

合约调用示例

```
ins.sm9mul("917be49d159184fba140f4dfc5d653464e94f718fe195b226b3f715829e6e768",
"288578d9505d462867a50acee40ee143b896e72505be10e8ce4c6b0c945b642b",
"123456789abcdef00fedcba98765432112345678 9abcdef00fedcba987654321")
Result {
  '0': '997fcff625adbae62566f684f9e89181713f972c5a9cd9ce6764636761ba87d1',
  '1': '8142a28d1bd109501452a649e2d68f012e265460e0c7d3da743fb036eb23b03b'
}
```

3.19.3 SM9PAIRING - 0x0111

SM9Pairing 预编译合约实现 SM9 曲线参数上的双线性对 (Pairing) 操作，可以用于零知识证明等复杂密码方案。本文写作时 Solidity 编译器还不支持 SM9Pairing 预编译合约，因此在编写合约时需要通过内联汇编 (Inline Assembly) 来调用 SM9Pairing 合约。下面给出调用 SM9Pairing 预编译合约的合约代码例子：

```
pragma solidity >=0.8.1;

contract Precompiles {
    function sm9pairing(string memory px, string memory py,string memory twistpx,␣
↪string memory twistpy) public view returns (bytes32[24] memory ){
        bytes memory input = new bytes(384);
        uint32 i;
        uint32 offset =0;

        bytes memory bx1 = bytes(px);
        bytes memory by1 = bytes(py);
        bytes memory bx2 = bytes(twistpx);
        bytes memory by2 = bytes(twistpy);
```

(续下页)

(接上页)

```
        for(i=0;i<64;i++) input[offset+i] = bx1[i]; offset +=64;
        for(i=0;i<64;i++) input[offset+i] = by1[i]; offset +=64;
        for(i=0;i<128;i++) input[offset+i] = bx2[i]; offset +=128;
        for(i=0;i<128;i++) input[offset+i] = by2[i];
        bytes32[24] memory ret;

        assembly{
            if iszero(
                staticcall(100000, 0x0111, add(input, 32) , mload(input), ret, 768)
            ) {
                invalid()
            }
        }

        return ret;
    }
}
```

合约调用示例

```
ins.sm9pairing(
    "7CBA5B19069EE66AA79D490413D11846B9BA76DD22567F809CF23B6D964BB265",
    "A9760C99CB6F706343FED05637085864958D6C90902ABA7D405FBEDF7B781599",

↪"74CCC3AC9C383C60AF083972B96D05C75F12C8907D128A17ADAFBAB8C5A4ACF701092FF4DE89362670C21711B6DBE52DC
↪",

↪"44B0294AA04290E1524FF3E3DA8CFD432BB64DE3A8040B5B88D1B5FC86A4EBC18CFC48FB4FF37F1E27727464F3C34E215
↪")
Result {
4e378fb5561cd0668f906b731ac58fee25738edf09cadc7a29c0abc0177aea6d
28b3404a61908f5d6198815c99af1990c8af38655930058c28c21bb539ce0000
38bffe40a22d529a0c66124b2c308dac9229912656f62b4facfced408e02380f
a01f2c8bee81769609462c69c96aa923fd863e209d3ce26dd889b55e2e3873db
67e0e0c2eed7a6993dce28fe9aa2ef56834307860839677f96685f2b44d0911f
5a1ae172102efd95df7338dbc577c66d8d6c15e0a0158c7507228efb078f42a6
1604a3fcfa9783e667ce9fcb1062c2a5c6685c316dda62de0548baa6ba30038b
93634f44fa13af76169f3cc8fbea880adaff8475d5fd28a75deb83c44362b439
b3129a75d31d17194675a1bc56947920898fbf390a5bf5d931ce6cbb3340f66d
4c744e69c4a2e1c8ed72f796d151a17ce2325b943260fc460b9f73cb57c9014b
84b87422330d7936eaba1109fa5a7a7181ee16f2438b0aeb2f38fd5f7554e57a
```

(续下页)

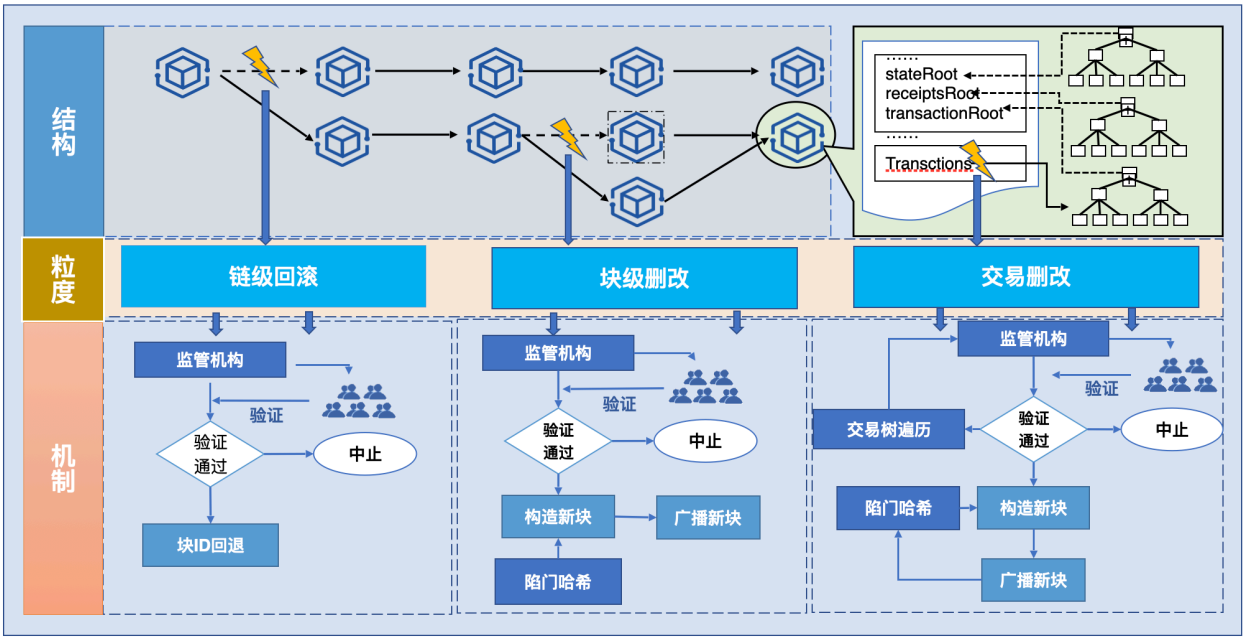
(接上页)

```
aab9f06a4eeba4323a7833db202e4e35639d93fa3305af73f0f071d7d284fcfb
}
```

3.20 隐私身份管理

3.21 数据隐私审计

3.22 可修改区块链



3.23 隐私身份管理

3.23.1 身份认证节点

对外 RPC 接口

register(c: String, user_info: String)

接收用户身份注册请求，如果成功，发送盲签名给用户

get_user_info(c: String)

查询用户的身份信息

内部接口

gen_cert(c: String, user_info: String)

- 记录用户身份信息（c 与 user_info 的键值对）
- 生成盲签名
- 更新 Merkle Tree 并在链上更新 Merkle root

revoke(c: String, user_address: Address)

在链上揭示用户的身份信息

3.23.2 委员会节点

对外 RPC 接口

verify_cert(address: Address, secret: String)

- 接收秘密分享
- 验证秘密分享的正确性

内部接口

revoke(address: Address, tx_hash: H256)

- 监控 event，收到 revoke 请求后进行恢复身份操作
- 验证 tx_hash 对应的交易是否存在违规行为（暂时不需要实现）
- 将秘密分享片段上传到身份合约

3.23.3 用户注册节点

对外 RPC 接口

register(addr: Address, user_info: String)

返回值为注册结果

内部函数

apply_cert(addr: Address, user_info: String)

对 address 进行盲化，向认证节点申请盲签名证书（发送 RPC 请求）

request_verify(addr: Address, cert: String)

- 对盲签名证书进行去盲，得到证书
- 对盲化因子进行秘密分享并生成证明
- 对秘密分享的值做零知识证明
- 跟合约进行交互，发送验证请求（发送证书、address、零知识证明、秘密分享证明）
- 发送秘密分享的值及合约的 request id 给各个委员会节点

revoke(addr: Address)

跟合约交互，撤销掉该身份

is_ok(addr: Address)

查询用户地址是否注册