CQUPT

# 证安区块链白皮书

## ZA Blockchain White Paper

2019 年 05 月 21 日

V1.2

# 前言

2012 年“电子数据”被《民事诉讼法》纳入证据的法定种类之一，标志着电子证据在诉讼中取得了合法地位。

2015 年 2 月最高人民法院公布了《最高人民法院关于适用<中华人民共和国民事诉讼法>的解释》，进一步对电子数据囊括的范围进行阐述，为电子证据在司法实践中的认定和审查做出了指引。

2016 年 9 月，为规范电子数据的收集提取流程和提高审查判断能力，提高电子证据相关案件办理质量，最高人民法院、最高人民检察院、公安部结合司法实际，根据《中华人民共和国刑事诉讼法》等有关法律规定，制定《关于办理刑事案件收集提取和审查判断电子数据若干问题的规定》。

证安区块链旨在为电子证据保全提供安全可靠的平台，助力国家法制社会的建成，为推动国家法治社会的进程贡献微小力量。

白皮书不定期更新，欢迎留下宝贵意见。

目录

# 01

# 第一章

# 综述

# 1.1 区块链技术

2008 年，Nakamoto S.第一次提出区块链的概念，它是组成电子货币比特币的 核心部分，能够提供去中心化的分布式记账服务。区块链类似于一种分布式数据库，通过维护区块的链式存储结构，来保障区块的稳定增长，后区块对前区块的依赖维护了区块链的不可篡改性，基于时间戳的链式结构提供了可溯源的特性。区块链的主要特征有去中心化、不可篡改性、匿名性、自治性等。

2016 年 10 月，在工信部发布的《中国区块链技术和应用发展白皮书（2016）》中，区块链技术被确定重点发展的前沿性技术。2016 年 12 月区块链被写入国务院 发布的《国务院关于印发“十三五”国家信息化规划的通知》。同时文件总结了当前国内外区块链的发展现状和部分应用场景。并介绍了国内区块链发展路线图，以及未来区块链发展方向和进程。

## 1.2 电子证据

2012 年“电子数据”被《民事诉讼法》纳入证据的法定种类之一，标志着电子证据在诉讼中取得了合法地位。

2015 年 2 月最高人民法院公布了《最高人民法院关于适用<中华人民共和国民事诉讼法>的解释》，进一步对电子数据囊括的范围进行阐述，为电子证据在司法实践中的认定和审查做出了指引。

2016 年 9 月，为规范电子数据的收集提取流程和提高审查判断能力，提高电子证据相关案件办理质量，最高人民法院、最高人民检察院、公安部结合司法实际，根据《中华人民共和国刑事诉讼法》等有关法律规定，制定《关于办理刑事案件收集提取和审查判断电子数据若干问题的规定》。

## 1.3 电子证据的保全

电子证据是证据的一种，首先必须依照一般证据规则予以保存。同时，由于电子证据与传统证据相比更加脆弱易失，容易被篡改和销毁，因此深入研究电 子证据的保全对指导司法工作具有重要意义。

当前，由于缺乏有效的电子证据保全规则，而相关司法实践的应用已经出现，这就导致相关问题层出。电子证据的保全与一般证据保全的目的和价值是相同的。但是，由于电子证据本身是脆弱的，隐蔽的和分散的，因此电子证据保全必须遵循环境安全封闭，多副本保管，对原始数据的提取校验必须进行严格记录。目前电子证据面临的最重要问题就是“信任”与“安全”，加强电子证据的固定就是有效解决这一问题的方案。在互联网、大数据时代，电子证据已经成为解决社会纠纷、利益分配、权益归属的有效手段。本文所研究的电子数据存证保全系统则是通过 B/S 模式提供证据保全和证据查验的系统，为维持电子证据的有效性，合法性提供支持。

证安链将结合电子证据保全的需求与区块链本身具备的去中心化和不可篡改特性，结合区块链技术分布式的结构，解决当前电子证据保全过程中面临的取证，存证困难问题，通过系统保证电子证据的原始性、客观性、有效性，进而推动电子数据的存证保全和司法落地。

# 02

# 第二章

# 项目说明

## 2.1 设计思路

目前，电子证据保全面临着严重中心化的问题和潜在的合谋攻击、数据篡改的风险。数据的安全性和可信性需要参考存储或者认证这些电子证据的第三方机构的信誉。考虑到区块链技术的安全性，将区块链技术与电子证据保全进行结合可以补足传统电子证据保全的不足之处。

一个需要指出的问题是，在对项目开发的过程中，如果从区块链底层进行项目开发，是一个繁琐的过程，区块链技术本身并不复杂，但考虑到目前已经拥有成熟的区块链平台和较为完备的区块链底层结构，再对区块链底层进行搭建就会将项目的工作变得繁琐冗余。所以本项目的区块链技术部分决定采用以太坊的技术。

项目使用 Spring Boot +Spring MVC+Spring Security 作为基础架构，Bootstrap 作为前端响应式框架。在以太坊平台的基础上采用 solidity 编写项目所需智能合约，实现电子证据的保全。

证安链致力于提供适用于电子证据保全业务的服务，并不再上添加除存、取、分析等必要动作之外的功能，保证代码的简洁清晰。

## 2.2 项目模型

项目采用 EEPM 系统架构为电子证据提供”信任“与”安全“。实现电子证据保全服务器的去中心化，提供数据安全保障没信息不可篡改、可溯源的机制。该模型的底层采用通过以太坊平台调用智能合约与区块链数据交互，达到数据存储，完成证据保全相关逻辑的目的，项目整体架构如图 1 所示。

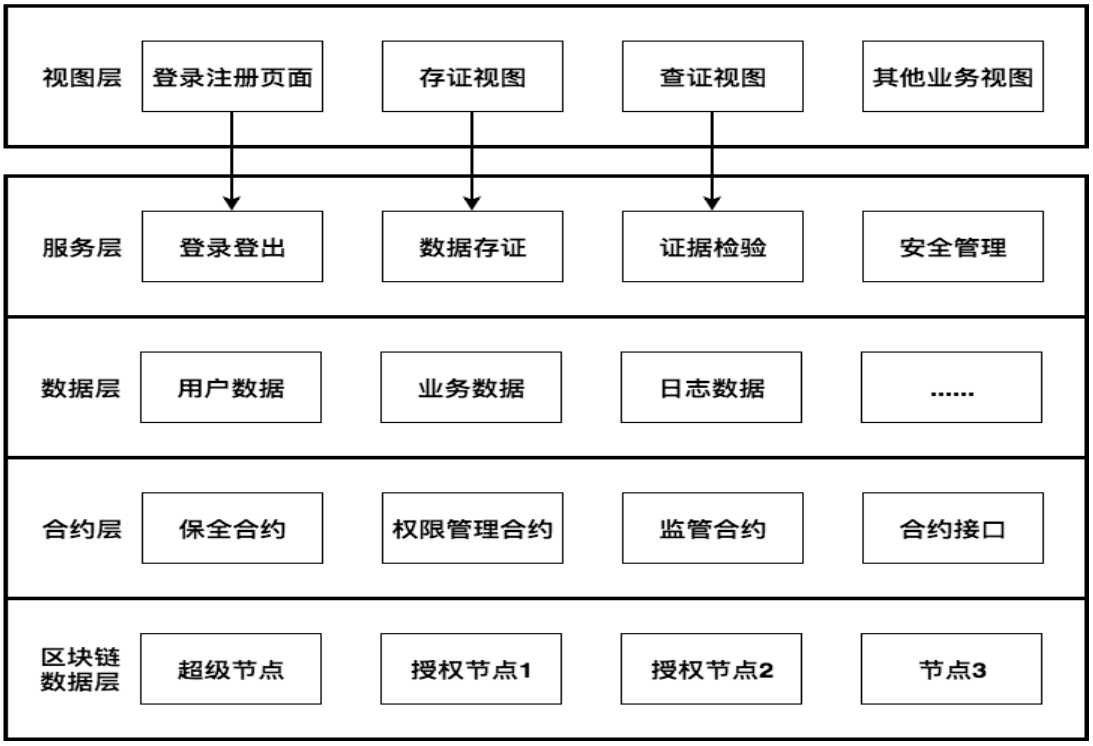


图 1EEPM 电子证据保全模型架构图

整个模型依赖与以太坊技术，但是依然需要在节点管理中做出个性化配置。在理想环境下，EEPM 模型结构中不同的节点可以由不同计算能力的服务

器构成，各节点应具有以下特征：

1. 节点间采用联盟链设计，加入节点需要获取授权

2. 节点角色应当由国家权力机关、高校、律师协会、网络安全协会等具有高可信度，且相互制约的权利机关，团体组织构成

3. 节点需要具有备份与宕机授权机制，保证系统同步进程，维持数据完整性

节点间通信需要解决拜占庭容错问题，EEPM 系统则采用联盟链为底层结构，内部指定多个预选的节点为记账人，每个块的生成由所有的预选节点共同决定。而预选节点间采用基于以太坊的更高效率的 PoS 共识算法，保障数据一致性。宏观如图 2 所示：

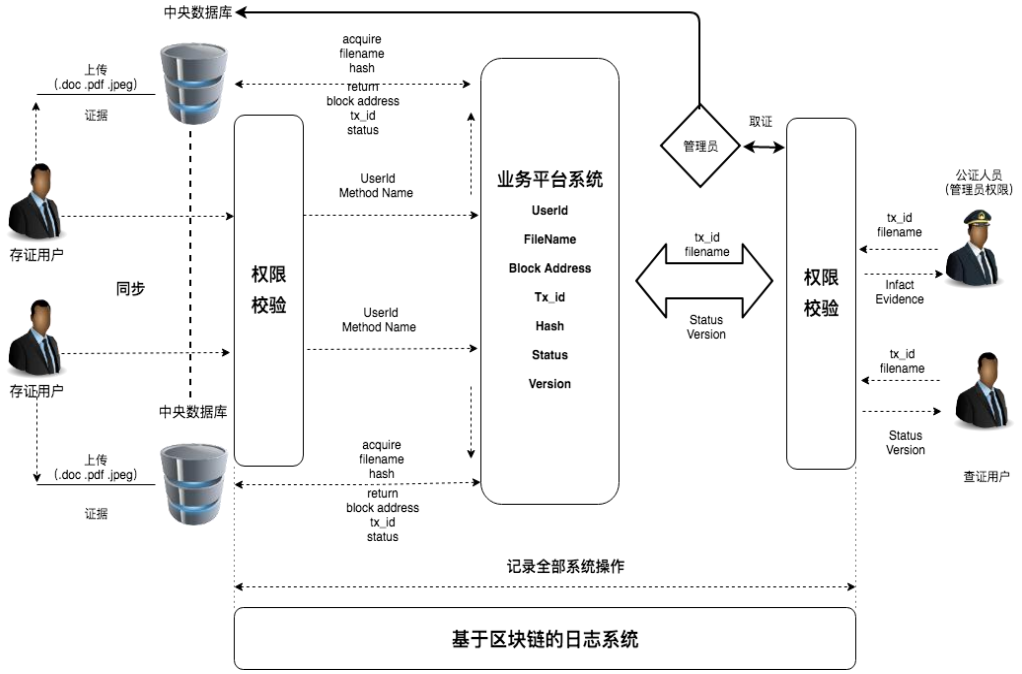


图 2 EEPM 模型图

## 2.3 数据库架构

项目数据由四部分数据组成，分别为**完整的证据信息**、**用户数据信息**、**平台业务数据**、**日志数据**。

模型中仅保存证据信息的 Hash 值，采用独立式数据库架构，将完整的证据信息保存在中央数据库或用户认为可信任的第三方数据库中，同时要求数据库采用多备份的原则，保障数据的安全性，提高信息的抗灾能力。完整证据信息与中心化业务系统的隔离，有助于保障申请电子证据保全用户的数据隐私与数据安全。日志信息的存储独立于业务系统，同时以区块链技术背书，让所有日志信息上链，保障日志信息的准确性，可靠性，不可篡改性。分离式数据库架构可以保障当业务系统故障、宕机时，依然能够独立地获取完整日志信息。整体数据划分逻辑如图 3 所示。

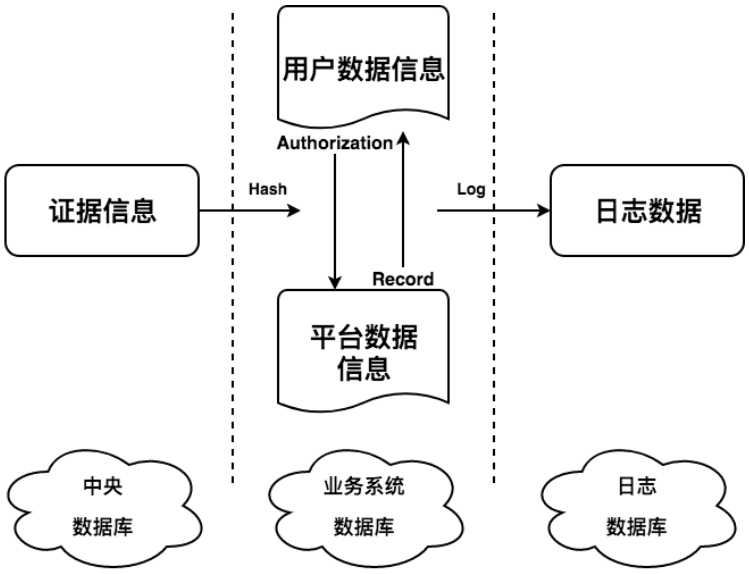


图 3 数据库架构图

## 2.4 系统角色特征

项目中主要存在以下角色：

**存证用户**：主要面向在离线环境提取电子证据的专业证据提取人员以及对文件、合同等电子数据有保全需求的普通证据保全人员 。

**普通查证人员**：根据证据编号，利用节点查询区块并验证区块链系统中保全的 Hash 摘要值是否与目标证据匹配 。

**高级调查员**：在电子证据作为合法证据呈现时，根据证据编号，获取调用智能合约权限并对证据关键信息进行提取，同时向证据中心数据库请求读取完整证据信息，并分析电子证据得出结论 。

**系统管理员**：管理员具有分配电子证据调查权限的能力，同时可以对系统中的账户信息进行管理 。

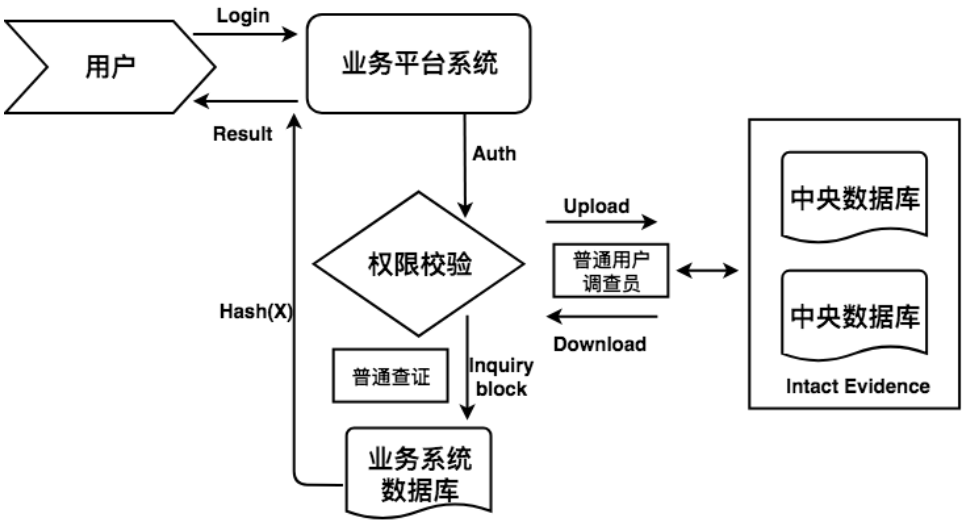


图 4 用户业务模型

# 2.5 系统安全性

本章所介绍的系统模型构建于区块链开发平台以太坊之上，关于以太坊的安全论证已经有很多文献进行论证，所以在此对以太坊的安全性论证不再进行相关追溯。

EEPM 模型系统要求全站采用 HTTPS 协议。这样的优势是不仅能够建立一个安全的信息传输通道，保证数据传输的安全。还能使用户能够及时确认网站的真实性。凡是采用 HTTPS 协议搭建的 B/S 系统，都可以通过浏览器地址栏的锁头标志来验证网站的真实信息，也可以通过 CA 机构查询。对于整个模型而言，HTTPS 协议能有效防范中间人攻击并保障数据在传输过程中的安全。基于 B/S 模式的模型系统 HTTPS 信息如图所示。


图 5　https 协议

# 03

# 第三章

# 智能合约

# 3.1 合约设计

对于智能合约，我们针对电子证据保全设计并部署了专用的智能合约。智能合约的代码本身是公开的，我们将智能合约的代码进行公布，以下是基于 Solidity 语言编写的项目智能合约源代码。

```
pragma solidity >=0.4.22 <0.6.0;

contract Evidence {
    uint public creationTime;
    string public name;
    State public evidenceState;
    address public owner;
    address public admin;
    string evidenceInfo;
    string private key;
    bool private keyState;
    string private analysisResult;
    bool public ownerApproved = false;

    enum State {Created, Analyzing, Analyzed, Locked, Inactive, Finished}
    enum Role {User, Admin, Researcher}
    enum RoleState {Normal, Locked}

    struct User {
        Role role;
        RoleState roleState;
    }

    mapping(address => User) public Users;

    constructor(string memory _name, string memory _info, string memory _key,
address _admin) public {
        name = _name;
        evidenceInfo = _info;
```

```
        key = _key;
        keyState = true;
        admin = _admin;
        //这是一个默认账户
        owner = msg.sender;
        creationTime = now;
        evidenceState = State.Created;
    }

    modifier condition(bool _condition, string memory info) {
        require(_condition, info);
        _;
    }

    modifier onlyOwner() {
        require(
            msg.sender == owner,
            "Only owner can call this."
        );
        _;
    }

    modifier onlyAdmin(){
        require(
            msg.sender == admin,
            "Only admin can call this."
        );
        _;
    }

    modifier inState(State _state) {
        require(
            evidenceState == _state,
            "Invalid state."
        );
        _;
    }

    modifier hasRole(Role role, User memory user){
        require(
            user.role == role,
            "You do not have sufficient permissions to call this."
        );
        _;
```

```
    }

    modifier checkRoleState() {
        require(
            Users[msg.sender].roleState == RoleState.Normal,
            "RoleState is locked."
        );
        _;
    }

    event AccountInfo(address account, RoleState roleState);
    event Authorized(address user, Role role);
    event Analyzing(address block, string key, address user, string name);
    event Analyzed(address user, string name, address block);
    event AcquireResult(string analysisResult, bool approved, address user,
string name, address block, Role role);
    event OwnerApproved(address user, bool approve);

    function giveRightToAdmin(address user) public checkRoleState onlyOwner {
        require(
            Users[user].roleState == RoleState.Normal,
            "RoleState is locked."
        );
        Users[user].role = Role.Admin;
        emit Authorized(user, Users[user].role);

    }

    function giveRightToResearcher(address user) public checkRoleState
onlyAdmin condition(
        Users[user].roleState == RoleState.Normal,
        "RoleState is locked."
    ) {
        Users[user].role = Role.Researcher;
        emit Authorized(user, Users[user].role);

    }

    function lockAccount(address account) public checkRoleState onlyAdmin {
        Users[account].roleState = RoleState.Locked;
        emit AccountInfo(account, RoleState.Locked);
    }

    function unlockAccount(address account) public checkRoleState onlyAdmin {
```

```
        Users[account].roleState = RoleState.Normal;
        emit AccountInfo(account, RoleState.Normal);
    }

    function queryEvidenceInfo() public view returns (string memory) {
        return evidenceInfo;
    }

    function acquireKeyForResearch() public checkRoleState
inState(State.Created) condition(
        keyState == true,
        "keyState is false."
    ) hasRole(
        Role.Researcher,
        Users[msg.sender]
    ) returns (string memory, address account){
        emit Analyzing(address(this), key, msg.sender, name);
        evidenceState = State.Analyzing;
        return (key, address(this));
    }

    function uploadAnalysisResult(string memory _AnalysisResult, address
evidenceID, string memory _key) public checkRoleState inState(State
        .Analyzing) hasRole(
        Role.Researcher,
        Users[msg.sender]
    ) returns (bool success){
        if (evidenceID == address(this) && hashCompare(key, _key)) {
            analysisResult = _AnalysisResult;
            evidenceState = State.Analyzed;
            emit Analyzed(msg.sender, name, address(this));
            return true;
        }
        return false;
    }

    function acquireAnalysisResult() checkRoleState public returns (string
memory, bool){

        require(
            evidenceState == State.Analyzed || evidenceState == State.Finished
        );

        require(
```

```
            Users[msg.sender].role == Role.Admin || Users[msg.sender].role ==
Role.Researcher || msg.sender == owner,
            "You do not have sufficient permissions to call this."
        );
        emit AcquireResult(analysisResult, ownerApproved, msg.sender, name,
address(this), Users[msg.sender].role);
        return (analysisResult, ownerApproved);
    }

    function resetKey(string memory _key) checkRoleState onlyOwner public {
        key = _key;
        keyState = true;
    }

    function approve() checkRoleState onlyOwner public {
        ownerApproved = true;
        evidenceState = State.Finished;
        emit OwnerApproved(msg.sender, true);
    }

    function hashCompare(string memory a, string memory b) internal returns
(bool) {
        return keccak256(abi.encodePacked(a)) ==
keccak256(abi.encodePacked(b));
    }
}
```

# 04

# 第四章

# 总结

# 总结

项目源代码已经开源上传至 Github ， 项目源代码地址：

https://github.com/ybeario/blockchain_ultra

项目展示地址：

https://www.ybear-web.com

欢迎加入本开源项目。

# 开发者团队

团队名称：南山老火锅

指导老师：徐光侠（教授）

项目设计：熊宇、陈云龙、刘文婧、赖恩梅、王有臻、邓思铭

项目开发：熊宇、陈云龙、邓思铭

特别致谢：杜江（教授）