点 (Orderer).

其中客户端的主要作用是和 Fabric 系统进行交互, 实现对区块链系统的操作. 客户端主要有命令行客户端 (CLI) 以及用 Fabric SDK 开发的应用客户端. 网络节点是区块链去中心化 P2P 网络中的对等节点, 按照功能可以分为背书节点 (Endorser) 和确认节点 (Committer). 背书节点主要对交易提案进行模拟执行以及背书验证. 确认节点主要负责验证交易的合法性, 并更新和维护区块链数据与账本状态.

CA 节点主要负责为 Fabric 网络中的成员提供基于数字证书的身份信息, 可以生成或取消成员的身份证书. Fabric 可以通过 CA 节点实现节点权限控制的管理.

排序服务节点主要负责对各个节点发送的交易提供排序服务, 并使所有记账节点达成共识. 之后排序节点按照一定规则确定交易顺序之后, 将数据发送给各个节点, 把交易数据持久化到区块链账本中.

各个组件相互关系如图 2 所示.

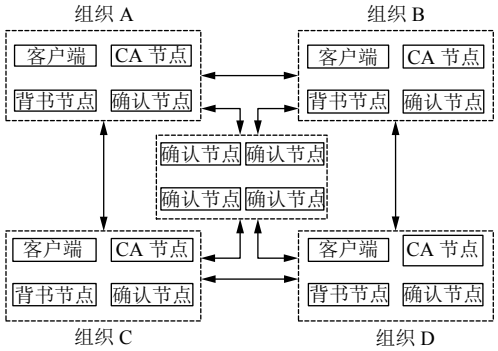

图 2 Fabric 组件关系示意图

1.4 其他开源区块链平台

以太坊与超级账本是两个在区块链技术发展过程中具有划时代意义的项目, 以太坊的出现标志着区块链进入到 2.0 时代, 超级账本项目则代表区块链 3.0 时代. 除此之外, 还有许多区块链平台, 如比特股 (BitShares)[12]、瑞波币 (Ripple)[13]、以及中国的趣链 (Hyperchain)[14]活跃于世界比特币舞台上. 表 2 是几种区块链平台的对比.

要将区块链技术应用于部队导弹业务数字化登记工作中, 首先需要选择合适的开发平台. 根据部队实际情况, 选用联盟链开发平台 Hyperledger Fabric 更符合部队实际, 因为联盟链相比公有链和私有链具有用户证书颁发、用户身份识别的功能, 具有 PKI 体系的认证机制, 能有效控制入网用户的各种权限.

表 2 部分区块链平台对比

区块链平台	共识机制	类别	开发语言	智能合约	TPS (每秒事务处理量)
比特币	PoW	公有链	C++	不支持	<7
以太坊	PoW	公有链	Go	支持	约 100
超级账本	PBFT	联盟链	Go	支持	约 3000
比特股	DPoS	公有链	C++	不支持	>500
瑞波	RPCA	公有链	C++	不支持	<1000
趣链	RBFT	联盟链	Go	支持	>10 000

2 智能合约

2.1 智能合约简介

1994 年, Szabo N 首次提出了智能合约 (smart contract) 的概念[15]. 智能合约的本质是一段运行于网络中的模块化、可重用、自动执行的脚本代码, 同时它还具有两个重要的特点: 图灵完备与沙箱隔离. 也正是由于智能合约的特点, 使其在被提出的初期并没有得到广泛的关注与应用. 随着比特币、区块链以及各种区块链开源平台的出现, 才使得智能合约由概念逐步落地实现, 如今智能合约已经成为区块链应用中的重要组成部分.

智能合约在区块链应用中的地位相当于现实生活中的合同, 加入区块链的用户都要遵照智能合约来完成特定的行为, 开发者可以通过编写智能合约规定用户的行为. 智能合约开发完成后需要部署于区块链网络中, 部署后的智能合约对用户是不可见的, 保证了智能合约的隐私性, 开发者可以在使用过程中对智能合约进行修改、升级.

2.2 Hyperledger 中的智能合约

在 Fabric 中, 智能合约也被称为 Chaincode (链码), 即链上代码, 主要分为系统链码和用户链码, 在区块链应用开发中设计并编写的链码指的是用户链码. 在 Fabirc 中, 访问、修改、查询账本都是通过调用部署的链码完成的, 一般使用 Go 语言编写, 通过 Fabric 官方提供的 API, 实现规定接口的代码, 这些 API 位于 Fabric 源码中的 Shim 包下. 同时, 链码之间也可以相互调用. 与以太坊中的智能合约一样, Fabric 中链码也运行于一个沙盒环境——Docker 容器中, 与背书节点的运行互相隔离. 目前 Fabric 支持使用 Go、Java、Node.js 三种编程语言进行链码开发, 其中 Go 语言是支持最好、最稳定的, 因此本文选择 Go 语言进行

Fabric 链码的开发. Fabric 中智能合约的运行情况如图 3 所示.

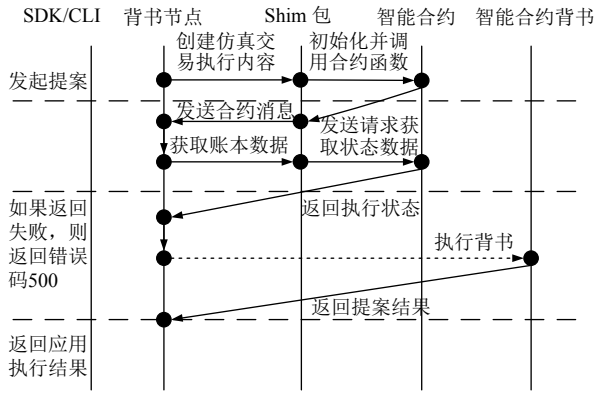

图 3 智能合约执行流程图

首先, 用户通过客户端 (SDK/CLI) 发起一个提案, 背书节点通过调用 Shim 包的方法创建账本仿真交易执行内容, 并通过 Shim 包对智能合约进行初始化和参数调用; 背书节点模拟提案执行, 分别执行读账本和写账本操作, 模拟更新状态数据; 若返回执行成功, 则开始执行背书操作, 若返回失败, 返回错误码 500; 背书节点对交易结果进行背书签名, 并将带有背书结果和背书签名的交易提案返回客户端进行背书认证.

3 某型航空导弹业务数字化登记智能合约设计

目前, 部队航空导弹业务登记工作形式主要是手工登记. 手工登记方式存在着诸多问题及安全隐患, 如: 存储过于集中、数据溯源困难、笔迹不清、书写习惯或格式不统一、重复填写、易被篡改等. 区块链技术有着去中心化、数据可追溯、防篡改的特点, 因此基于区块链技术, 对航空导弹业务登记工作进行数字化, 能够很大程度上解决目前手工登记方式存在的“痛点”问题. 本文以航空导弹业务为应用场景, 进行区块链架构设计、智能合约设计与开发.

根据区块链的基本架构, 合约层是部署于数据层、网络层和共识层之上的. 因此在对智能合约进行设计之前, 需要首先对区块链架构进行设计.

3.1 区块链架构设计

与传统区块链架构相比, 本文研究的某型航空导弹业务数字化登记中不需要出现类似于比特币系统中货币的概念, 因此不需要激励层, 合约层之上即为应用层, 二者通过客户端 (CLI/SDK) 关联起来. 在传统区块链架构的基础上进行具体设计与改进, 如图 4 所示.

在 Fabric 中, 有两种支持 KVS 存储的数据库可提供选择使用, 一种是默认的 LevelDB, 另一种是可以配置选择的 CouchDB. 这里选用的是 CouchDB, 因为 Fabric 支持对 CouchDB 的富查询 (Rich Query), 同时 CouchDB 还有图形化管理界面, 为开发人员提供了极大的便利, 且 CouchDB 支持 JSON 数据格式, 更适用于表格的数据存储.

图 4 某型航空导弹业务数字化登记区块链架构

Fabric 的网络层中, 节点间的 P2P 网络由 Gossip 协议实现, 在通道中的各个节点会持续广播和接受 Gossip 消息. 通道 (Channel) 是 Fabric 中重要的概念, 多个节点之间可以组成一个通道, 同一个节点也可以加入不同的通道. 一个通道内部的所有节点共同维护同一个账本, 通道与通道之间互相隔离. 在本文中的研究中, 预设计一个 orderer 节点, 两个组织 Org1 和 Org2, 每个组织都有两个节点 Peer0 和 Peer1, 锚节点都是 Peer0 节点. 两个组织中的所有节点都加入到同一个通道 channel1 中. P2P 网络拓扑结构如图 5 所示.

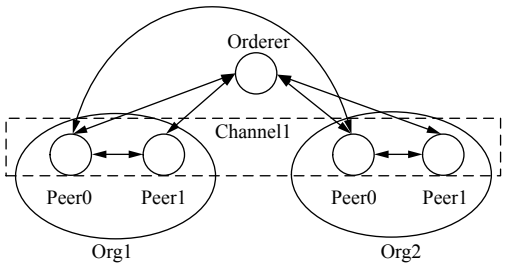

图 5 网络拓扑结构

共识层使用的是 PBFT 共识算法, PBFT 是一种考虑了拜占庭容错的共识算法, 能够容忍网络中存在最多三分之一的作恶节点. 相比 PoW、PoS 等共识算法, PBFT 的效率更高. 同时由于联盟链规模相比公有链小很多, 因此恶意节点对账本同步的影响更大, 使用

PBFT 能更好地保证区块链中各个节点有效达成共识.

3.2 智能合约设计

区块链应用于传统应用最主要的区别是使用智能合约来实现主体业务逻辑, 智能合约既是程序逻辑的主体, 也是数据存储的主体. 航空导弹业务分为许多方面, 现以某型航空导弹技术准备中的气密性检查为例进行智能合约设计.

气密性检查即在某个时刻记录下内部压力, 然后进行充气, 充到一定压力值后保持一段时间, 再记录下压力值. 如果两个压力值之差小于某个阈值, 说明该弹气密性检查结论为合格, 否则为不合格.

智能合约必须要实现 Init 接口和 Invoke 接口, 其中 Init 是对智能合约进行初始化, 执行的结果就是在状态数据库中创建一个数据库, 用来存放通过该智能合约写入账本的数据, 本文的智能合约设计中没有考虑预先在数据库中录入数据, 因此 Init 接口在实现过程中不需要进行操作, 只需要返回一个 Success 消息即可; Invoke 接口是执行交易的接口, 在本文中的作用就是调用已编写的函数.

Invoke 接口的调用操作过程如图 6 所示.

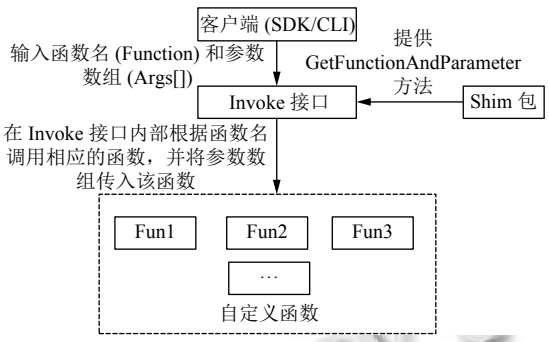

图 6 Invoke 接口调用操作过程示意图

Invoke 接口需要客户端输入两个参数; 函数名 (Function) 和函数所需参数 (Parameter), 智能合约中的 Invoke 只需要通过 Shim 包中的 GetFunctionAndParameter 方法就能接收到客户端传入的函数名和函数所需参数, 再通过编写 Invoke 内部的判别方法, 就可以根据需要, 通过输入相应的函数名来指定调用编写好的函数. Invoke 接口内可提供调用的函数信息如表 3 所示.

根据某型航空导弹气密性检查操作过程以及实际业务需求, Invoke 接口中需要调用的自定义函数有: 创建检查记录函数 (Create)、根据导弹编号查询检查记录 (Query)、查询某个编号导弹的时间戳 (queryTime)、查询某个编号导弹的不合格检查记录 (queryJl)、查询某个编号导弹气密性检测记录的创建者 (queryCreator) 等.

表 3 Invoke 接口提供的函数信息

函数名	调用函数所需的 Function 字段	实现的功能
Create	Create	创建检查记录
Query	Query	根据导弹编号查询检查记录
queryTime	queryTime	查询某个编号导弹的时间戳
queryJl	queryJl	查询某个编号导弹的不合格检查记录
queryCreator	queryCreator	查询某个编号导弹气密性检测记录的创建者

4 智能合约的开发、测试与安全性分析

4.1 智能合约开发

选择 Ubuntu16.04 作为开发、运行环境, Fabric 版本为最稳定的 v1.2, Go 语言使用稳定版本 1.11.4. 根据业务实际情况以及对智能合约的设计结果, 开发中需要用到的 Shim 包 API 如表 4 所示.

表 4 开发所需部分 API

API	作用
GetFunctionAndParameter	获取由客户端传入的函数名 及参数
PutState	将数据写入账本
GetState	从账本读取数据
GetCreator	获取记录创建人
GetTxTimestamp	获取记录的时间戳
GetStateByRange	从账本中读取某个 Key 值范围的数据
GetQueryResutl	获取自定义查询的结果

某型航空导弹技术准备中的气密性检查的过程中, 在气密性检查中, 需要记录的信息有: 导弹编号、检查日期、检查人、开始时间、开始时压力 (0.1±0.005 Mpa)、结束时间、结束时压力 (Mpa)、压力差 (Mpa) 以及结论 (压力差≤0.01 Mpa 为合格), 共计 9 个数据项. 链码中定义数据格式代码段如下:

```
type QMX struct {
Ddbh string `json:"ddbh"`      //导弹编号
Jcrq `json:"jcrq"`             //检测日期
Jcr string `json:"jcr"`        //检测人
Kssj string `json:"kssj"`      //开始时间
Kssyl float64 `json:"kssyl"`   //开始时压力
Jssj string `json:"jssj"`      //结束时间
```

```
    Jssyl float64 `json:"jssyl"`    //结束时压力
    Ylc float64 `json:"ylc"`        //压力差
    Jl string `json:"jl"`           //结论
    }
```

压力差与结论需要智能合约内部的 Create 函数进行判断的, 首先计算压力差, 再根据压力差与阈值的关系进行结论的判断.

根据前文的设计, 必须实现的 Init 接口代码段如下:

```
Func (s *SmartContract) Init (stub shim.ChaincodeStubInterface) pb.Response {
    return shim.Success(nil)
    }
```

根据智能合约设计结果, Invoke 接口内部的具体实现代码段如下:

```
Func (s *SmartContract) Invoke(stub shim.Chaincode StubInterface) pb.Response {
    function, args:= stub.GetFunctionAndParameters()
    if function == "Create" {
    // 按顺序输入: 导弹编号 检测日期 检测人 开始时间 开始时压力 结束时间 结束时压力
    return s.Create(stub, args)
    }
    if function == "Query" {
    // 输入需要查询的导弹编号
    return s.Query(stub, args)
    }
    if function == "queryJl" {
    // 输入需要查询不合格记录的导弹编号
    return s.queryJl(stub, args)
    }
    if function == "queryCreator" {
    // 输入需要查询记录创建者的导弹编号
    return s.queryCreator(stub, args)
    }
    if function == "queryTime" {
    // 输入需要查询时间戳的导弹编号
    return s.queryTime(stub, args)
    }
    return shim.Error("没有可调用的函数")
    }
```

4.2 智能合约测试

利用 Shim 包所提供的 API, 对每个自定义的功能函数进行编写, 然后进行部署调试. 在 CLI 客户端中输入相应指令, 即可得到运行结果.

首先通过 docker exec –it ci bash 命令进入到客户端 CLI 容器中, 使用 peer chaincode install 安装链码, 安装时需要指定通道. 本文创建的通道名为 channel1; 再利用 peer chaincode init 命令, 对智能合约进行初始化. 由于之前在实现 Init 接口时没有要求 CLI 传入任何参数, 因此输入 Function 参数和 Args 参数数组时为空即可. 初始化完成后, 在 CouchDB 中创建了关于该智能合约的数据库; 最后使用 peer chaincode invoke 命令, 输入通道名称、智能合约名称以及需要传入的 Function 参数以及 Args 参数数组, 之后即可得到查询结果, 如图 7 所示.

图 7 在客户端中运行智能合约

智能合约的设计与开发是 Fabric 平台的核心工作之一, 安装在区块链网络的背书节点上, 类似于社会中的法律合同, 对业务进行了强制性的规范. 通过智能合约的设计、开发与演示可以看出, 相比于纸质手工登记, 使用区块链技术既具有传统数字化登记系统有利于存储、格式规范、登记速度快等优点, 同时又实现了去中心化的分布式存储方式保证了登记数据可追溯、不可篡改. 然而要将区块链技术进一步应用于航空导弹业务登记领域, 还需要进一步研究探索, 后续将要进行的工作主要有: 使用国密 SM2 和 SM3 算法对 Fabric 中的加密模块进行改进, 实现加密算法的“国产化”, 进一步提高区块链网络的安全性与可靠性; 对 PBFT 共识算法进行优化, 降低节点之间达成共识的耗时.

4.3 安全性分析

在航空导弹业务场景中, 本文设计并实现的区块链系统及智能合约能够实现无人监督的业务数据登记录入工作, 免去了对于第三方监督的依赖. 同时还能够保证数据的可靠性、完整性以及可追溯性, 提高了数据的可信度并提高了防止数据被恶意篡改能力. 引入的共识机制能够保证数据来源的高可信度, 可以有效地防止恶意节点对数据系统进行破坏. 本文设计的系

统使用一种 NoSQL 数据库: CouchDB 作为存储环境, 相比于传统使用 SQL 数据库的数据记录软件, 能够有效防止 SQL 注入攻击.

智能合约是对相关业务或逻辑的代码实现, 其目的在于不依赖于可信任的第三方执行各种操作. 如果被恶意修改会造成业务或逻辑混乱, 甚至会对系统造破坏, 因此对智能合约的保护是一项重要内容. 本文提出的智能合约隐私保护措施主要是将智能合约安装并运行于 Docker 上. Docker 是一种容器, 能够实现虚拟化但又不同于虚拟化. 容器是完全沙箱机制的, 之间不存在任何接口. 同时在 Linux 系统内核中也存在着分别实现对容器资源隔离和资源限制的功能. 智能合约运行在 Docker 容器上时, 能够保证智能合约的独立性, 同时对于已经安装在 Docker 上的智能合约对外界也是不可见的, 即使修改智能合约代码, 已经安装的智能合约也不会受到影响.

5 结语

本文针对部队传统航空导弹业务登记中存在的问题与隐患, 提出了基于区块链技术对传统登记方式进行数字化的方案, 并完成了智能合约的设计, 最后对某型航空导弹气密性检测登记进行了智能合约的开发与测试, 对区块链技术在部队工作中的应用进行了初步探索, 安全性分析证明了该方案具有较好的可行性与有效性.

参考文献

1 Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2018-12-23.

2 袁勇, 王飞跃. 区块链技术发展现状与展望. 自动化学报, 2016, 42(4): 481–494.

3 Ekblaw A, Azaria A, Halamka JD, *et al*. A case study for blockchain in healthcare: “MedRec” prototype for electronic health records and medical research data. Proceedings of the 2nd IEEE of International Conference on Open & Big Data. 2016. 25–30.

4 张俊, 高文忠, 张应晨, 等. 运行于区块链上的智能分布式电力能源系统: 需求、概念、方法以及展望. 自动化学报, 2017, 43(9): 1544–1554.

5 庹小忠. 区块链在身份认证中的应用. 科技经济导刊, 2017, (3): 26–27.

6 Lee K, James JI, Ejeta TG, *et al*. Electronic voting service using block-chain. Journal of Digital Forensics, Security and Law, 2016, 11(2): 8.

7 贺海武, 延安, 陈泽华. 基于区块链的智能合约技术与应用综述. 计算机研究与发展, 2018, 55(11): 2452–2466. [doi: 10.7544/issn1000-1239.2018.20170658]

8 工信部. 中国区块链技术和应用发展白皮书. http://sh.qihoo.com/pc/9a20df8118f58243b?cota=4&tjurl=so_rec&sign360_57c3bbd1&referscene=so_1, [2018-12-25].

9 Ethereum White Paper. A next-generation smart contract and decentralized application platform. ahttps://github.com/ethereum/wiki/wiki/White-Paper, [2018-12-25].

10 王晓光. 区块链技术共识算法综述. 信息与电脑, 2017, (9): 72–74. [doi: 10.3969/j.issn.1003-9767.2017.09.027]

11 Hyperledger project charter. https://www.hyperledger.org/about/charter, 2018-12-25.

12 The BitShare blockchain. https://github.com/bitshares-foundation/bitshares.foundation/blob/master/download/articles/BitSharesBlockchain.pdf, [2018-12-26].

13 Distributed ledger technology mature enough to go commercial: Ripple-accenture white paper. http://www.econotimes.com/Distributed-ledger-technology-mature-enough-to-go-commercial-Ripple-Accenture-white-paper-236052, [2018-12-26].

14 趣链科技. 趣链科技公司与技术介绍. https://upload.hyperchain.cn/%E8%B6%A3%E9%93%BE%E7%A7%91%E6%8A%80%E5%85%AC%E5%8F%B8%E4%B8%8E%E6%8A%80%E6%9C%AF%E4%BB%8B%E7%BB%8D.pdf. [2018-12-28].

15 Szabo N. A formal language for analyzing contracts. http://nakamotoinstitute.org/contract-language/.[2018-12-28].