



毕 业 论 文

题 目 基于区块链的多方合同

签署协议的设计与实现

姓 名 王啸

学 号 16041808

指导教师 周艺华

日 期 2020 年 5 月

北京工业大学

毕业设计（论文）任务书

题目 基于区块链的多方合同签署协议的设计与实现

专业 信息安全 学号 1604108 姓名 王啸

主要内容、基本要求、主要参考资料等：

主要内容：

- （1）深入学习区块链技术的基本原理、核心技术、共识算法、完整性、认证性、保密性等加密算法；
- （2）深入学习以太坊的系统架构；
- （3）深入学习智能合约、solidity 等编程技术；
- （4）设计并实现一种基于区块链技术的多方合同签署协议系统,利用聚合签名,结合可验证加密算法,签署方在区块链上公平的交换各自的秘密值,最后通过计算,完成合同的签署。聚合签名结果记录在以太坊区块链中,可以实现签署协议的客观公正。

基本要求：

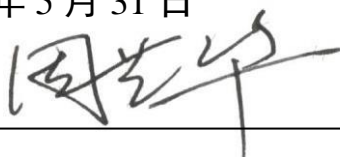
基于以太坊,设计并实现一个基于区块链技术的多方合同签署协议系统。

参考文献：

- [1]基于区块链的多方隐私保护公平合同签署协议[J]. 吴进喜,高莹,张宗洋,殷大鹏. 信息安全学报. 2018(03)
- [2]基于公开区块链的隐私保护公平合同签署协议[J]. 田海博,何杰杰,付利青. 密码学报. 2017(02)
- [3]A new certificateless aggregate signature scheme[J].Lei Zhang,Futai Zhang.Computer Communications.2009 (6)
- [4] 基于区块链的高效公平多方合同签署协议[J]. 高莹,吴进喜. 密码学报. 2018(05)
- [5]Electronic contract signing without using trusted third party. WAN Z,DENG R H, LEE D. International Conference

完成期限：2020 年 5 月 31 日

指导教师签章：



专业负责人签章：

2020 年 5 月 31 日

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京工业大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签名：



日期：2020.5

关于论文使用授权的说明

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：



导师签名：



日期：2020.5

摘要

随着互联网技术的发展与人们生活模式的改变，信息化、高效率的办公方式越来越受到青睐。传统的合同签署总是需要一个可信第三方提供监督，各签署方无法避免第三方为牟利而泄露合同信息带来的危害；并且最终保存者不得不保存合同以及所有签署方的签名，这又容易造成存储资源浪费。

本论文中提出并设计的区块链多方合同签署系统，利用了区块链本身的特性：区块链拥有可以令分散节点建立起信任的共识算法机制，因此其在功能上就相当于一个可信第三方，这样就消除了第三方带来的潜在不公平；同时，该系统又可以基于离散对数生成聚合签名，这样最终保存者仅需存储合约及一个聚合签名即可，这减少了对于存储空间巨大需求。此外，由于区块链写入后几乎不可能修改的特性，也杜绝了签署方反悔的可能。

另一方面，本设计除了完成原有要求，针对高隐私性应用需求，提出了一种带有盲化的多方合同签署系统。该系统（隐私性签署系统）相比初版系统（公平性签署系统），更加侧重用户的隐私保护。隐私性签署系统基于混密钥的方式对合同消息与用户身份进行了盲化，这样可以满足一部分签署方需要匿名的合同签署需求。相比初版系统强调的公平性、可追溯性，该版本更加强调用户信息的隐私性、保密性，满足了不同用户对于合同签署系统的要求。

关键词： 区块链；以太坊；聚合签名；智能合约；去中心化应用

Abstract

With the development of Internet technology and the change of people's life mode, the office working way which is information-based and high-efficient is getting more and more popular. The traditional contract signing process always needs a trusted third party to provide supervision. The signatory parties cannot avoid the harm caused by the third party disclosing the contract information for profit. Meanwhile, ultimately the depositor has to save the contract and the signatures of all the signatories, which can easily lead to a waste of storage resources.

This paper is mainly about designing and completing a multi-party contract signing system based on Blockchain. This design takes advantage of the features of Blockchain itself: Blockchain has the Consensus Mechanism that can make decentralized nodes establish trust, so it is equivalent to a trusted third party in function, thus eliminating the potential unfairness brought by the third party; At the same time, the system can generate aggregate signatures based on discrete logarithms so that the ultimate saver only needs to keep the original contract and the aggregated signature, which reduces the huge demand for storage space. In addition, due to the fact that it is almost impossible to modify the Blockchain after it was written, the signatory cannot go back on its word.

On the other hand, in addition to the completion of the original requirements, this design also created an additional function: a blind multi-party contract signing system. Compared with the first version of the system (fairness based signing system), this system (privacy based signing system) focuses more on the privacy protection of users. The private based signature system blinks the contract message and the user's identity using the mixed key method, which can meet the requirement of some signatory parties to sign the contract anonymously. Compared with the fairness and traceability of the first version system, this version emphasizes more on the privacy and confidentiality of user information, which meets the requirements of different users for the contract signing system.

Keywords: Blockchain; Ethereum; Aggregate Signature; Smart Contract; Dapp

目录

摘要.....	1
Abstract	11
1. 绪论	1
1.1 研究背景	1
1.2 研究目的及意义	1
1.2.1 研究目的	1
1.2.2 研究意义	1
1.3 国内外研究现状	2
1.3.1 国内研究现状	2
1.3.2 国外研究现状	3
1.4 主要内容介绍	3
2. 理论与技术基础	5
2.1 区块链基础知识	5
2.1.1 基本性质	5
2.1.2 智能合约	6
2.1.3 去中心化应用 (Dapp)	8
2.1.4 以太坊介绍	10
2.2 密码学算法	12
2.2.1 哈希算法: SHA-256	12
2.2.2 哈尔 (Harn) 算法	16
2.2.3 双线性对	16
2.3 平台介绍	17
2.3.1 Eclipse 平台	17
2.3.2 Remix IDE 平台	17
2.3.3 Code Runner 平台	17
2.4 框架与工具介绍	18
2.4.1 Truffle 框架	18
2.4.2 Webpack 框架	18
2.4.3 Ganache	18
2.4.4 Java-Swing 框架	19
2.4.5 JPBC 库	19
2.5 本章小结	19
3. 需求分析及总体设计	20
3.1 角色需求分析	20
3.1.1 签署者 (U_i) 需求分析	20
3.1.2 聚合者 (U_c) 需求分析	21
3.1.3 验证者 (U_v) 需求分析	22
3.2 模块需求分析	22
3.2.1 前端模块需求分析	22
3.2.2 后端模块 (智能合约) 需求分析	22
3.2.3 交互模块需求分析	22
3.3 非功能性需求分析	23

北京工业大学毕业设计（论文）

3.4 整体流程设计	24
3.4.1 功能流程	24
3.4.2 架构流程	26
3.5 本章小结	27
4. 详细设计及实现效果	28
4.1 环境准备	28
4.1.1 以太坊私链搭建	28
4.1.2 智能合约部署	28
4.1.3 交互连接配置	30
4.1.4 Eclipse 环境搭建	31
4.2 前端设计与实现	33
4.2.1 样式与排版	33
4.2.2 交互实现	34
4.2.3 隐私性系统前端的设计与实现	36
4.3 后端（智能合约）设计与实现	38
4.3.1 单用户签名生成	38
4.3.2 单用户签名验证	40
4.3.3 聚合签名生成	42
4.3.4 聚合签名验证	43
4.4 隐私性多方聚合签署系统的设计与实现	44
4.4.1 经典聚合签名算法的安全性分析	44
4.4.2 隐私性多方聚合签署系统	45
4.4.3 隐私性多方聚合签署系统实现	46
4.5 系统测试	47
4.5.1 公平性系统测试	47
4.5.2 隐私性系统测试	49
4.6 本章小结	50
5. 成果总结与展望	51
5.1 成果总结	51
5.2 研究展望	51
结论	52
参考文献	53
致谢	54

1. 绪论

1.1 研究背景

自 2008 年 Satoshi Nakamoto 提出的“比特币”概念至今，其所依托的区块链技术吸引了计算机领域越来越多的目光。相比于比特币诱人的金融产品属性，其去中心化、分布式账本等先进特性更加值得人们的关注。人们逐渐意识到，除了数字货币，区块链还可以支撑商务谈判、供应链以及溯源系统等许多方面，区块链未来有望在经济、管理、社会等多个层面改变人们的生活。

合同签署是遍布社会各界的一个活动，不论是大宗交易的供货、运输、收货方，还是签署公司新协议的甲乙丙丁员工，目前都面临着合同可能被泄漏，或自己身份被暴露的情况。据此，我设计并实现了一个基于以太坊区块链的多方合同签署系统，我相信区块链的特性将会良好的解决以上问题。

1.2 研究目的及意义

目前我国各领域都在推动信息化进程，将信息化技术应用到不同领域并解决实际问题一直都是重中之重，利用区块链等新兴技术来解决合同签署问题是十分必要的。本文旨在将区块链技术与聚合签名理论相结合，以解决合同签署相关的问题。

1.2.1 研究目的

随着区块链“3.0”时代的到来，Dapp（基于区块链的中心化应用）已经逐渐走向人们的视野。当下的 Dapp 大多都是与挖矿工具或游戏有关的应用，如同初代的 IOS app 大多都是播放软件或游戏软件。但我坚信区块链技术的潜力不仅限于此，人们可以利用区块链去中心化、分布式等特性，设计出功能更强大的 Dapp。

本设计的灵感源于合同签署类问题，这也是信息安全领域中许多学者曾研究过的经典问题之一。我的研究目的就是利用区块链的特性，结合前人的研究成果，设计并实现一个基于区块链的多方合同签署系统。该系统不仅是一个能为合同签署方提供更可靠的、更加抗抵赖的平台的 Dapp，也是对于传统密码学研究与最新技术结合的一次尝试。

1.2.2 研究意义

传统的合同签署过程，所需角色除了各签署方，还需要包括一个可信第三方：该第三方通常需要始终保持在线状态，这就导致了系统开销大并且效率偏低；若该第三方是离线的、仅在争端时出现的，这种情况虽然会降低系统开销，但仍然无法避免第三方可能为了谋取私利而透露合同信息的情况。

与此同时，合同签署过程中，虽然大多数情况下需要明确声明各自的身份，但也存在在某些合同签署过程中，各签署方不愿透露自己的身份，这就需要合同签署系统的开发者思考，针对不同的需求，公平性与保密性之间如何侧重。

本设计的研究意义就旨在从一定程度上解决上述问题：

1. 由于区块链具有中心化、分布式账本等特性，因此基于区块链设计的应用可以避免引入可信第三方。
2. 选择区块链技术的原因之一是区块链数据基于共识算法生成，链上数据一旦写入就不可抹去，这帮助系统实现了合同签署的重要需求：不可抵赖。
3. 设计并实现聚合签名算法，降低系统存储消耗。
4. 根据用户对公平性和保密性的不同侧重需求，设计并实现了两套合同签署系统。
5. 创新的实现了以合同签署为功能的 Dapp，这方面的 Dapp 目前尚在探索阶段，具有未来发展的潜力。

1.3 国内外研究现状

以太坊是第一个用图灵完备语言实现在区块链上部署智能合约的平台，因此以太坊很快成为了最受欢迎的 Dapp 开发平台。迄今为止已经有超过 1000 款 Dapp 是基于以太坊开发的，可以说以太坊区块链是当之无愧的 Dapp 领军平台。

聚合签名是具有聚合性质的数字签名技术， n 个用户对 n 个消息分别签署的 n 个签名，能够合成一个短的签名，而验证者只需对聚合后的短的签名进行验证，便可以确信 n 个消息是否被 n 个用户分别进行了签名。在 2003 年欧密会上，由 Boneh 等人首次提出。此外，聚合签名也可用于设计可验证加密签名、环签名以及公平交易协议等安全技术中。Boneh 等人最初提出的聚合签名限制于不同用户签名不同的消息。对于消息内容相同时，他们建议签名者附加各自的公钥到消息中。

1.3.1 国内研究现状

目前我国 Dapp 领域的开发趋势基本与国际保持一致：主要以挖矿工具和游戏为主。较为出名的有海南布洛克城出品的“公信宝”，该 Dapp 可以令用户更方便的利用手机进行“mining”——也就是区块链领域大名鼎鼎的“挖矿”操作，同时用户可以在虚拟的“布洛克城”进行一系列模拟真实生活的活动。值得一提的是，于 2019 年 8 月走入人们视野的“链签宝”，就是一个借助区块链特性的、解决传统合同签署问题的 Dapp。目前“链签宝”发展尚不足一年，平均用户也只有 8000 多人，可以说该类 Dapp 的潜力尚未被完全开发，目前尚处在不成熟阶段，有许多方面值得各方研究人员继续商讨、改进，这也正是本研究开展的原因。

近年来，国内许多学者都对聚合签名进行了深入研究。如 2014 年，明洋等学者提出一个新的高效的无证书聚合签名方案。在随机预言机模型中计算性 Diffie-Hellman 假设下，所提方案能够抵抗适应性选择消息攻击下的存在性伪造攻击，同时所提方案签名长度独立于签名者的数量仅为 2 个群元素，签名验证中仅需要 4 个对和 n 个标量乘运算；2015 年，汤小超等学者根据已有的聚合签名方案的部分密钥提取过程中存在被敌手伪造的问题，基于双线性映射提出了一种新的无证书的顺序聚合签名方案，并将自认证方案与聚合签名方案相结合，从而保证了部分密钥的安全，同时对聚合签名方案过程中的签名算法进行改进以提高性能。

1.3.2 国外研究现状

目前国外的 Dapp 领域中比较出名的应用有 AxiomZen 工作室开发的 CryptoKitties 应用，该应用是一款虚拟饲养电子猫的游戏类 Dapp。据悉，继这款“加密猫”游戏的走红，又有很多厂商接连推出了以太坊“加密狗”、“加密猪”、“加密兔”等应用，导致了 2019 年第一季度，三大主链的 DAPP 交易额高达 36 亿美元。不过这些“跟风”行为，实际上限制了以太坊 Dapp 的创新；这些 Dapp 虽然为大众提供了娱乐，却没有完全发挥出区块链应用应该承担的安全功能和社会功能。因此，本设计希望尝试使用区块链技术解决现实问题，拓展、尝试 Dapp 的新领域。

在聚合签名研究方面，2004 年，Mykletun 等人提出一种具有不变性 (Immutability) 的 iBGLS 方案，专门防止增量式聚合，提出聚合者应将其对所有待聚合的消息的连接为一个消息，并对该消息进行签名，再将这个签名聚合到原聚合签名中；2011 年，Neven 提出一个更高效的有序聚合签名 ED-SAS 方案。并通过安全分析，一个合法的签名者的安全不会受到同组签名者密钥对选择的影响；同前面两方案相比，密钥对选取相同不会对聚合签名产生影响或者出错，具有更好的通用性。

1.4 主要内容介绍

本文分为五个章节，内容概括如下：

1. 绪论。

表述写作目的和意义以及研究背景，提出基于以太坊区块链，设计一个基于聚合签名的合同签署 Dapp 应用；同时简述了国内外研究现状，先是总结了目前国内外 Dapp 的主要类型、形式，又努力寻找到了与本研究相关的落地产品。最后又总体论述了本文的写作意义和主要研究内容、方向。

2. 区块链相关理论基础。

介绍了关于本设计的理论知识与所需工具，通过区块链基础知识、密码学算法、所需平台、所需框架及工具四个模块进行阐述。区块链基础知识以及密

码学算法模块表面了本设计的理论依据；平台及框架模块则提供了完成可运行系统的工具。

3. 需求分析及总体设计

由签署方、聚合方以及验证方这些抽象角色入手，分析他们在合同签署过程中的行为，从而得出设计中必须包含的元素以及各角色可能面临的风险。同时分析出了这样一个大型 Dapp 想要成功运行，需要的前端、后端以及连接模块的需求。

4. 详细设计及实现效果。

本章详细介绍了本设计从系统配置到运行效果的全过程，介绍遵循“理论基础-代码设计-实现效果”这一经典说明方式。首先讲明理论基础与算法，证明了该设计的可靠性；之后详细讲述代码设计，有助于更加深刻理解设计的含义；最后再向读者展示作品的完成效果，验证本作品确实符合预期。

5. 研究总结与展望。

本章是本论文正式章节中的最后一章，意在表达出本人在设计、实现本作品时的总体感受，并对研究成果进行总结；同时指出本作品存在不足的地方，为后期的改正与进一步研究提供铺垫。

2. 理论与技术基础

2.1 区块链基础知识

2.1.1 基本性质

区块链的名字，来源于它的工作方式以及存储数据的方式。打包成块的信息与其他类似信息的块进行链接，最终形成一个链。当以下重要性质结合在一起时，就形成了一个不容置疑的信息存储空间，一个不容争议的、不允许声明不真实的信息存储空间：

1. 去中心化

区块链上所产生的数据，都是通过密码学算法和一些数学算法进行建立信任关系，不需要传统意义上的中心化机构来构建。基于分布式的点对点的系统，使几方之间可以依托区块链提供的信任机制进行交易，这样就免去了对可信第三方的需求。

2. 不可篡改

每当一组数据从节点产生，都需要经过“验证”过程。只有通过大部分节点验证认可后，可以将该数据写入区块链，并且每一个节点都会复制数据并保存。每个块不仅包含它正在记录的数据、时间戳等，还包括记录在区块头里面的“父区块哈希值”字段。由于这一特性，整个链的任何信息都几乎不可能被改变：这是因为一旦父区块发生变化，子区块也会跟着变化；并且随着区块在区块链中的位置加深，其因为“分叉”而被重新计算的可能性越来越低，因此要修改稳定的区块链几乎是不可能的。

3. 分布式账本

区块链中每个节点都拥有一本“账本”，用以记录数据。分布式账本指每个网络节点都可以记录整个账本的交易记录，以此来维护交易的公平性与可追溯性。“账本”具有如下三点重要性质：

- 1) 记录性:存储的信息具有时间戳。
- 2) 透明性:区块链上的任何用户，都可以看到交易的分类帐。
- 3) 分散性:账本存在于多台不同的计算机上，通常称为节点。

4. 共识信任机制

区块链的共识机制决定了区块链的交易速度、不可篡改性、业务吞吐量等等，是最重要的因素之一。因为区块链过密码学算法和数学算法，建立起了智能合约的执行系统，通过这种去中心化的、分布式的、含有密码学的记账的技术，交易不再需要外请的第三方来进行判断、认可以及证明。共识机制的存在，使数以万计的独立节点能遵守复杂的规则，使异步交互自发的形成，促成交易。

5. 开放性

在区块链上，除了参与交易的用户的身份信息被加密，其他的信息都是开放的，任何一个区块链中的参与者，都可以公开进行查询相关记录。

6. 匿名性

因为区块链的许多特性，所以交易节点之间可以保证相互信任，因此节点之间在一般情况下不需要公开自己的身份。

7. 跨平台

区块链未来的发展，必将更加深入经济、人文、社会等多个领域，同时与云计算技术、人工智能技术等高新技术相结合，实现更大的价值。

2.1.2 智能合约

智能合约的定义是：存储在区块链上的代码行，一旦当前条件满足预定的要求时，自动执行。简单来讲，它们就是一个由开发人员创建并运行的程序。智能合约的好处体现在方方面面，尤其是在业务协作中较为明显：在业务协作中，智能合约通常用于强制执行某种类型的协议，以便所有参与者都可以确定结果，而不需要中介的参与。

智能合约通过遵循设计好的代码来工作，这些语句被写入到区块链上的代码中。由计算机组成的区块链网络将执行这些操作，然后在事务完成时更新区块链。

举一个供应链的例子：买家 Bob 想从卖家 Alice 那里买东西，所以她把钱放在一个代管账户里。卖方 Alice 将使用托运人 Charlie 将产品交付给买方 Bob。当买方 Bob 收到货物时，托管的钱将被释放给卖方 Alice 和托运人 Charlie。如果买方 Bob 在 x 月 y 日之前没有收到货物，托管的钱将被退回。执行此交易时，通知制造商创建另一个已出售的产品以增加供应。所有这些流程，如果写入合约，都将是自动完成的。

在智能合约中，可以有许多满足参与者任务将圆满完成的需求的规定。若想要建立一些条款，区块链中的参与者需要明确标书怎样传达事务和数据，并且就这些事务的管理方式以及管理规则达成一致，并探索、明确所有可能的例外情况，定义解决争端的框架。它通常是一个迭代过程，涉及到开发人员和全体业务涉众。

下面举一个实例，来具体讲述智能合约的编译、部署以及调用：

如下图 2.1 所示，合约文件 Helloworld.sol 经过编译器编译生成 Helloworld.abi 和 Helloworld.bin 文件。

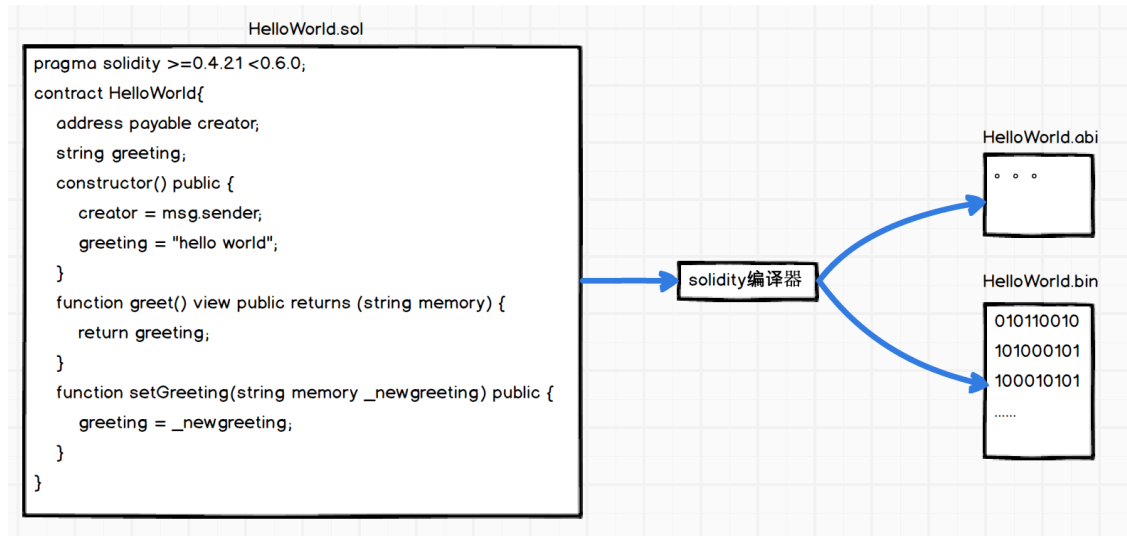


图 2.1 合约编译

合约编译完成后，即可进入部署阶段。合约的部署跟发送一笔交易是一样的操作，调用 `transaction` 函数，`from` 为发布者的地址，`to` 为 0，`data` 为合约的 evm 操作码。在矿工打包的时候会生成智能合约地址。智能合约地址的生成是由创建者的账号和发送的交易数作为随机数输入，通过 Keccak-256 加密算法重新创建一个地址作为账号。也就是说最后合约地址对应合约的代码会保存在区块链数据库。调用者只需要有合约地址和 abi 文件就可以调用合约的代码，如图 2.2 所示：

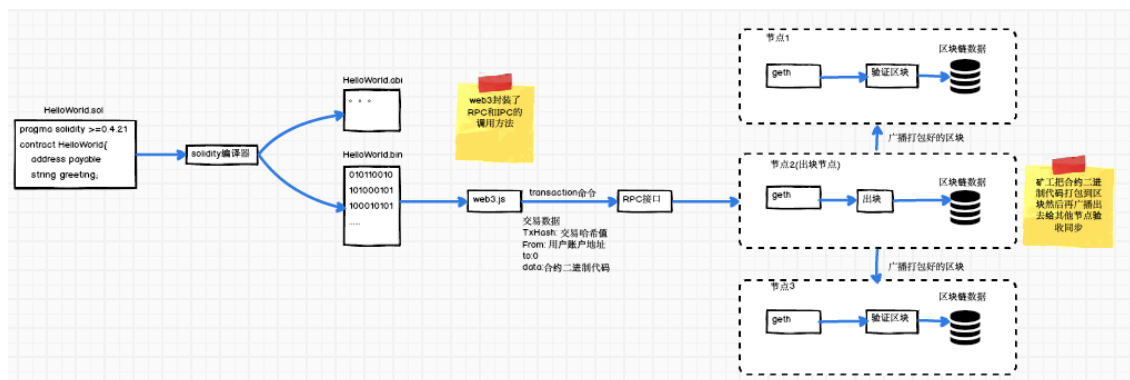


图 2.2 合约部署

最后，当以上两步完成后，我们可以进行合约调用了，合约调用时就可以使用合约中的方法了，如图 2.3 所示：

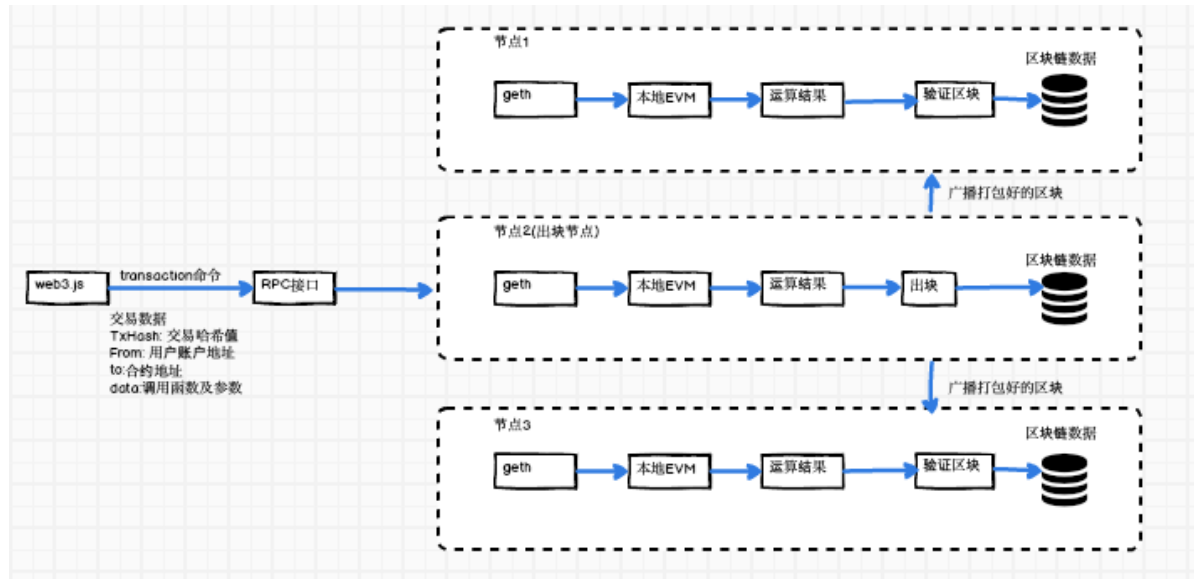


图 2.3 合约调用

2.1.3 去中心化应用（Dapp）

去中心化的区块链应用程序(Dapp)，是指在计算系统上运行的分布式计算机应用程序。DApp 主要通过分布式账本技术(DLT)，也就是以太坊区块链来进行普及的。在以太坊区块链中，DApp 通常被认为是由智能合约而延伸、拓展出来的应用。

根据去中心化应用的一般理论所给出的定义，Dapp 应满足以下标准：

1. 开源：Dapp 通常都是完全开源的，而且需要独立运行，并且没有实体能做到控制它的令牌。Dapp 哪怕在正式发售、运行后，依然可以根据市场的反馈和用户的建议调整其智能合约，但是所有更改必须在用户达成一致意见后才能执行。
2. 为了避免中心点或者中央点因意外发生的故障，Dapp 应用的数据、信息以及操作记录等必须以加密的方式存储在一个公共区块链中。
3. Dapp 应使用加密令牌，这是访问 Dapp 应用所必需的。
4. Dapp 应当根据标准的加密算法生成令牌，以此来作为对应用程序贡献的值节点的证明(例如比特币使用 Proof of Work 算法)。

如表 2.1 所示，Dapp 与传统网络应用开发有以下这些主要区别：

表 2.1 Dapp 与网络应用区别

	传统网络应用	DAPP 应用
前端设计	通常为 HTML+CSS+JavaScript	可使用多种语言、框架编写
后端设计	通常使用 Java、C++ 等高级语言编写	使用 Solidity（主流、面向对象）、 Serpent 等语言编写
数据存储	数据库	区块链
货币	/	可使用已有货币或用户自行定义的 货币

目前，Dapp 的架构主要分为三种类型：轻钱包模式、重钱包模式和兼容模式：

1. 轻钱包模式

轻钱包模式下我们需要有一个开放 Http RPC 协议的节点与钱包通信，这个节点可以是任意链上的节点。轻钱包通常会作为一个浏览器插件存在，插件在运行时会自动注入 Web3 框架，DApp 可以通过 Web3 与区块链节点通信。当 DApp 只是单纯的获取数据时是不需要钱包介入的，但是当 DApp 需要发送交易到链上时需要通过钱包完成对交易签名的过程。

优点：不需要用户同步区块链节点就可以使用

缺点：需要一个公开的节点提供服务，可能会存在安全性问题

2. 重钱包模式

重钱包会自己同步并持有一个区块链节点，提供一个浏览器环境，其他与钱包相似。

优点：自己持有并同步节点，安全性高

缺点：需要持有一个全量的区块链节点

3. 兼容模式

兼容模式可以在轻钱包和重钱包下同时使用，与钱包通信的节点可以选择在钱包外本地持有，也可以自己搭建服务持有并公布节点。

其架构可以总结为图 2.4:

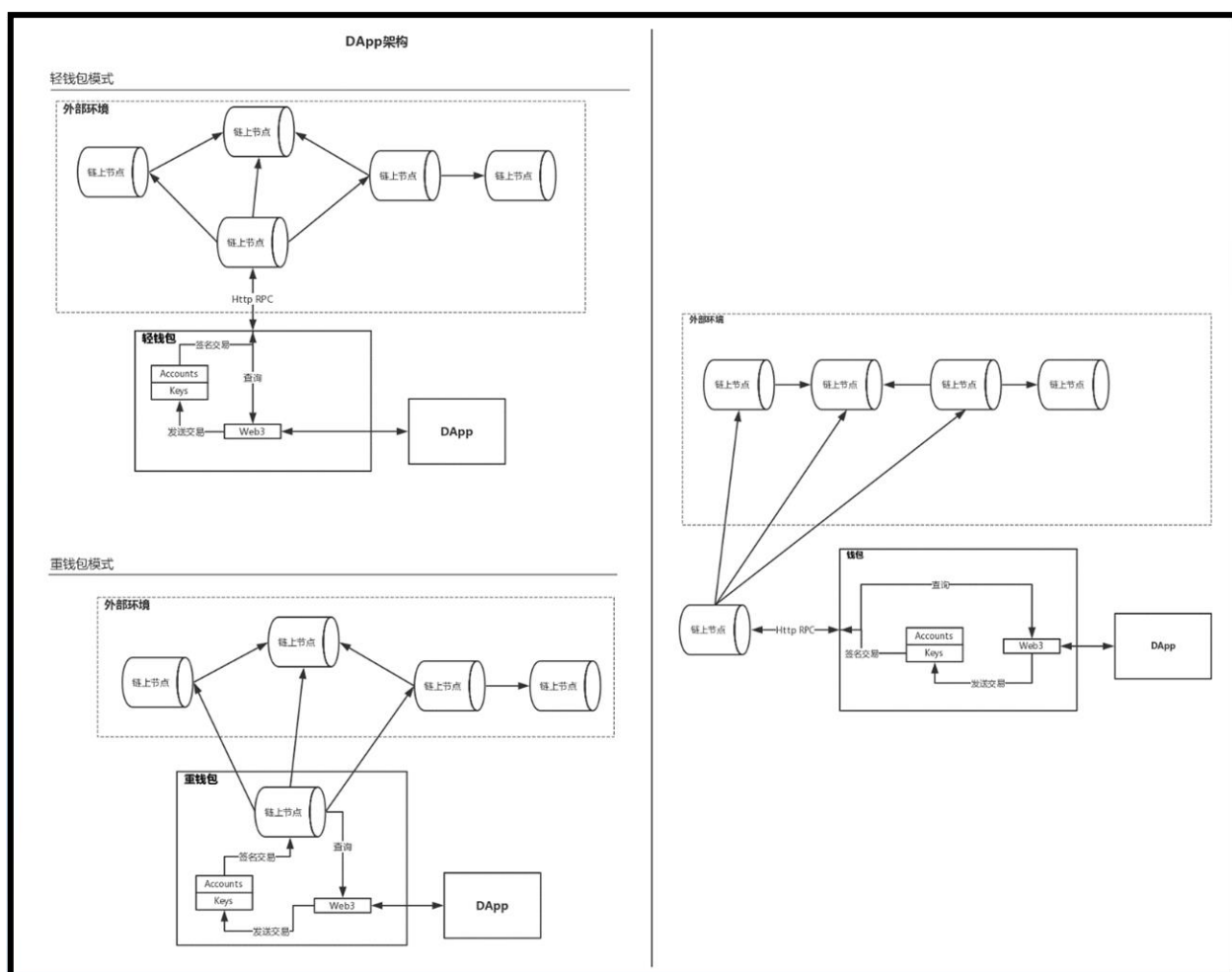


图 2.4 DApp 架构

2.1.4 以太坊介绍

以太坊 (Ethereum)，其标志如图 2.5 所示。自 2015 年推出以来，便获得了广泛关注。以太坊是一个完全开源、去中心化、基于区块链的平台。以太坊拥有自己的加密货币“Ether”。这使得智能合约和分布式应用程序 (DApp)，能够做到成功构造和运行，免除了潜在的许多隐瞒、欺骗或者控制的情况，也排除了引入第三方的干扰。



图 2.5 以太坊

以太坊，不仅是一个平台，也是一种运行在区块链上的图灵完备的语言。以太坊平台是目前市面上最有活力的 DApp 发布平台，帮助许多开发人员构建和发布了 DApp。

在以太坊上运行的 DApp，平台所特有的加密令牌“Ether”运行。早在 2014 年，“Ether”概念刚刚出现时，市场就给予了强烈反响。“Ether”功能、用途十分广泛，是那些希望在以太坊内部开发和运行 DApp 的开发人员所寻求的。目前为止，

“Ether”主要被广泛应用于三个目的：

1. 作为数字货币进行交易，就如同比特币一样
2. 在以太坊内部，用于运行 DApp
3. 用于货币化工作

以太坊设计尊重以下设计原则：

1. 简洁原则

以太坊协议将尽可能简单，即便以某些数据存储和时间上的低效为代价。一个普通的程序员也能够完美地去实现完整的开发说明。这将最终有助于降低任何特殊个人或精英团体可能对协议的影响并且推进以太坊作为对所有人开放的协议的应用前景。添加复杂性的优化将不会被接受，除非它们提供了非常根本性的益处。

2. 通用原则

没有“特性”是以太坊设计哲学中的一个根本性部分。取而代之的是，以太坊提供了一个内部的图灵完备的脚本语言以供用户来构建任何可以精确定义的智能合约或交易类型。想建立一个全规模的守护程序（Daemon）或天网

（Skynet），你可能需要几千个连锁合约并且确定慷慨地喂养它们，一切皆有可可能。

3. 模块化原则

以太坊的不同部分应被设计为尽可能模块化的和可分的。开发过程中，应该能够容易地让在协议某处做一个小改动的同时应用层却可以不加改动地继续正常运行。以太坊开发应该最大程度地做好这些事情以助益于整个加密货币生态系统，而不仅是自身。

4. 无歧视原则

协议不应主动地试图限制或阻碍特定的类目或用法，协议中的所有监管机制都应被设计为直接监管危害，不应试图反对特定的不受欢迎的应用。人们甚至可以在以太坊之上运行一个无限循环脚本，只要他愿意为其支付按计算步骤计算的交易费用。

2.2 密码学算法

2.2.1 哈希算法：SHA-256

哈希算法是一种将输入数据压缩成固定大小的计算函数，计算的结果是称为哈希或哈希值的输出。安全哈希算法（SHA）族指定了一个由六个不同哈希函数组成的族：SHA-0、SHA-1、SHA-224、SHA-256、SHA-384 和 SHA-512。它们获取可变长度的输入消息并将其散列到固定长度的输出。

在 Solidity 编写的智能合约中，目前主要支持的哈希算法是 SHA-256 算法。该算法在 Solidity 官方文档中的介绍如图 2.6：



图 2.6 SHA-256 介绍

目前普遍认为，SHA-256 是最安全的哈希函数。此函数表示由给定输入数据产生的可能组合或值。SHA-256 表示为拥有 256 位固定长度的安全散列数值，并且 SHA-256 拥有“抗碰撞”（Collision Resistance）的特性，因此两个不同的用户生成相同哈希值的可能性是极小的。满足 SHA-256 的难度意味着这个散列非常安全，用户妄图寻找原始输入数据几乎是不可能的。

SHA256 具体算法描述如下：

1. 补位

初始信息必须进行补位，以使其长度在对 512 取模以后的余数是 448。也就是说，补位后的消息长度 $Q2 = 448$ 。即使长度已经满足对 512 取模后余数是 448，补位也必须要进行。补位的规则是：先补一个 1，然后再补 0，直到长度满足对 512 取模后余数是 448。总而言之，补位是至少补一位，最多补 512 位。以信息“abc”为例显示补位的过程。

原始信息：01100001 01100010 01100011

补位第一步：0110000101100010 01100011 1

首先补一个“1”

补位第二步：0110000101100010 01100011 10...0

然后补 423 个“0”

我们可以把最后补位完成后的数据用 16 进制写成下面的样子

61626380 0000000000000000 00000000

00000000 0000000000000000 00000000

00000000 0000000000000000 00000000

00000000 00000000

现在，数据的长度是 448 了，我们可以进行下一步操作

2. 补长度

补长度意思是将原始数据的长度补到已经进行了补位操作的消息后面。通常用一个 64 位的数据来表示原始消息的长度。如果消息长度不大于 2^{64} ，那么第一个字就是 0。在进行了补长度的操作以后，整个消息就变成下面这样了（16 进制格式）。

61626380 0000000000000000 00000000
00000000 0000000000000000 00000000
00000000 0000000000000000 00000000
00000000 0000000000000000 00000018

3. 使用的常量

在 SHA256 算法中，需要用到 64 个常量，这些常量是对自然数中前 64 个质数的立方根的小数部分取前 32bit 而来。这 64 个常量如下：

428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2

4. 需要使用的函数

$CH(x, y, z) = (x \text{ AND } y) \text{ XOR } ((\text{NOT } x) \text{ AND } z)$

$MAJ(x, y, z) = (x \text{ AND } y) \text{ XOR } (x \text{ AND } z) \text{ XOR } (y \text{ AND } z)$

$BSIG0(x) = \text{ROTR}^{25}(x) \text{ XOR } \text{ROTR}^{13}(x) \text{ XOR } \text{ROTR}^{22}(x)$

$BSIG1(x) = \text{ROTR}^{6}(x) \text{ XOR } \text{ROTR}^{11}(x) \text{ XOR } \text{ROTR}^{25}(x)$

$$\text{SSIG0}(x) = \text{ROTR}^7(x) \text{ XOR } \text{ROTR}^{18}(x) \text{ XOR } \text{SHR}^3(x)$$

$$\text{SSIG1}(x) = \text{ROTR}^{17}(x) \text{ XOR } \text{ROTR}^{19}(x) \text{ XOR } \text{SHR}^{10}(x)$$

其中 x 、 y 、 z 皆为 32bit 的字。

$\text{ROTR}^2(x)$ 是对 x 进行循环右移 2 位。

5. 计算信息摘要

基本思想：首先将消息分成 N 个 512bit 的数据块，hash 初值 $H(0)$ 经过第一个数据块加密得到 $H(1)$ ， $H(1)$ 经过第二个数据块加密得到 $H(2)$ ……依次处理，最后得到 $H(N)$ ，然后将 $H(N)$ 的 8 个 32bit 依次连接行成 256bit 消息摘要。具体步骤如下：

1) hash 初值 $H(0)$

SHA256 算法中用到的 hash 初值 $H(0)$ 如下

$$H(0)_0 = 6a09e767$$

$$H(0)_1 = bb66ae85$$

$$H(0)_2 = 3c6ef372$$

$$H(0)_3 = a45ff53a$$

$$H(0)_4 = 510e257f$$

$$H(0)_5 = 9b05868c$$

$$H(0)_6 = 1f38d9ab$$

$$H(0)_7 = 5be0cd19$$

2) 计算过程中会用到的三种中间值

64 个 32bit 字的消息调度标记为 w_0 、 w_1 、…、 w_{63} ，8 个 32bit 字的工作变量标记为 a 、 b 、 c 、 d 、 e 、 f 、 g ，包括 8 个 32bit 字的 hash 值标记为 $H(i)_0$ 、…、 $H(i)_7$ 。

3) 工作流程

把初始消息分为 N 个 512bit 的消息块。每个消息块分成 16 个 32bit 的字，依次标记为 $M(i)0$ 、 $M(i)1$ 、 $M(i)2$ 、 \dots 、 $M(i)15$ 然后对这 N 个消息块依次进行处理。处理流程图如图 2.7 所示。

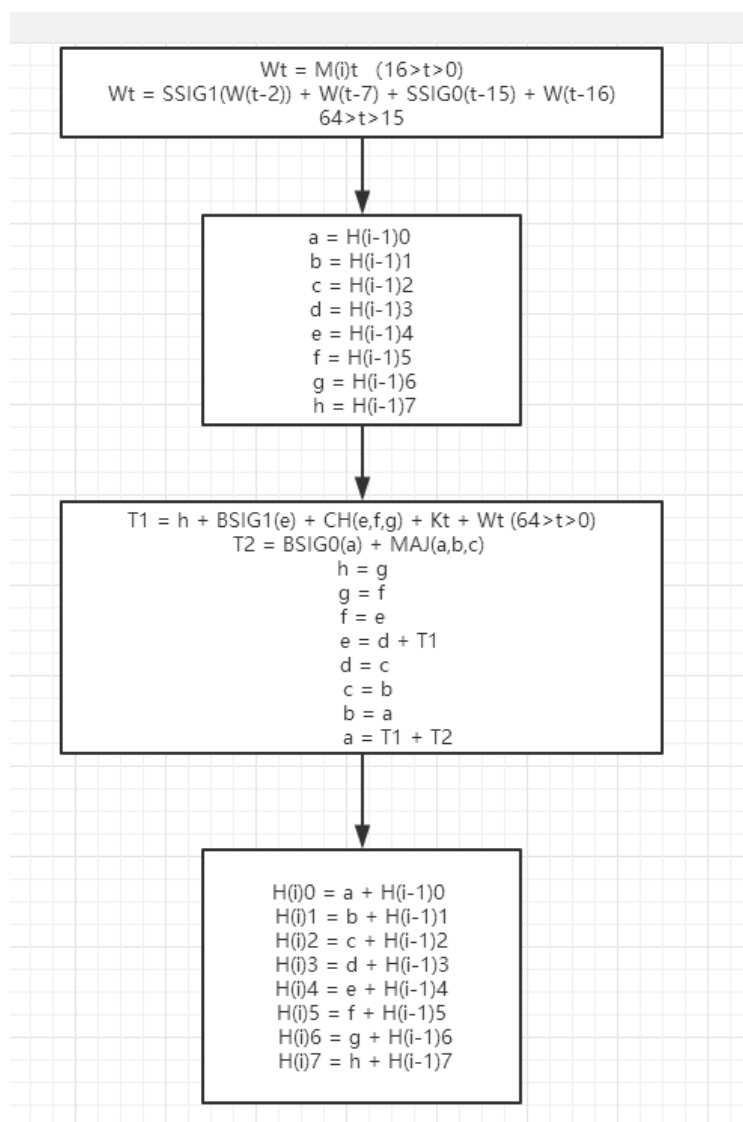


图 2.7 消息块处理流程图

对 N 个消息块依次进行以上四步操作后，会得到 $H(N)0$ 、 $H(N)1$ 、 $H(N)2$ 、 \dots 、 $H(N)7$ 共计 7 个消息块，把 7 个消息块串联起来即可得到最后的 256bit 消息摘要。即加密后的 hash 值。

2.2.2 哈尔（Harn）算法

在整数中，离散对数（Discrete logarithm）是一种基于同余运算和原根的一种对数运算。公钥密码学中很多算法的基础，就是基于寻找离散对数的解。下面举简单的例子说明离散对数的基础算法，见表 2.2:

表 2.2 离散对数算法讲解

步骤	用户	行为
1	Alice、Bob	约定公共的 $q=2739 \cdot (7149-1)/6+1$ 和 $g=7$
2	Alice	选随机数 m ，计算 $7m \pmod q$ ，将其发送给 Bob
	Bob	将收到: $7m=1274021 \cdots 588076766033781$
3	Bob	选随机数 n ，计算 $7n \pmod q$ ，将其发送给 Alice
	Alice	将收到: $7n=1801622 \cdots 531244026803049$

此时 Alice 和 Bob 都能计算出密钥 $7mn \pmod q$ ，但别人难以算出，因为二者自选了 m 和 n 。

而哈尔（Harn）算法正是基于如上的离散对数思想，实现了可广播的多方签名机制。该算法可以帮助完成：系统初始化，单用户签名的产生，单用户签名的验证，多重签名的产生，多重签名的验证等多方内容。

Harn 算法的基础就在于求解离散对数 $y = g^x \pmod p$ 这一数学难题。此前大多数基于离散对数的数字签名方案都需要进行模逆运算，因此计算速度非常慢。而 Harn 提出的这一离散对数算法，签名者不需要计算逆运算，验证者只需要计算两个模指数，计算量大大减少，因此得以在认证方案、密钥交互以及聚合签名领域被广泛使用。

2.2.3 双线性对

设 G_1 是加法循环群， P 为生成元； G_2 是乘法循环群，且它们都是 q 阶，将映射 $e: G_1 \times G_2 \rightarrow G_2$ 称为双线性对，若其满足以下三条性质：

1. 双线性：对于任意的 $P, Q \in G_1$ ， $a, b \in \mathbb{Z}_q^*$ ，均 $e(aP, bQ) = e(P, Q)^{ab}$ 有成立；
2. 非退化性：存在 $P, Q \in G_1$ ，使得 $e(P, Q) \neq 1$ ；
3. 可计算性：对于所有的 $P, Q \in G_1$ ，均存在一个有效的多项式时间算法来计 $e(P, Q)$ 。

如果 $G_1 = G_2$ 则称上述双线性配对是对称的，否则是非对称的。

另外，上述的双线性配对是素数阶的，还存在一种合数阶的双线性配对，最早也是由 Boneh 等人引入密码学领域的。合数阶双线性配对利用子群的正交性可以在实现更加复杂的功能前提下完成安全性证明。

2.3 平台介绍

2.3.1 Eclipse 平台

Eclipse 是一个 IDE 工具，可以帮助我们开发软件。根据定义，集成开发环境 (IDE) 是一种软件应用程序，为计算机程序员提供全面的软件开发工具。还可以在文本编辑器中编写代码，并从命令行进行编译和执行；但是与文本编辑器相比，Eclipse 提供了许多其他有用的特性，使软件的开发更容易、更快。通常情况下，IDE 由调试器、源码编辑器、自动化工具等等工具组成。大多数现代 IDE（如 Eclipse）也提供了智能代码完成特性。

Eclipse 包含基本工作区（workspace）和可扩展的插件系统，用于定制开发环境。因此，除了 IDE 之外，Eclipse 还是一个框架，可以通过使用插件对其进行扩展，以获得更多的特性和功能。我们大多数人使用 Eclipse 只是为了创建或编辑 Java 代码，但是 Eclipse 具有比创建或编辑代码更多的特性，例如，调试、代码存储库集成等。Eclipse 还提供了许多快捷方式来加快软件开发。

2.3.2 Remix IDE 平台

Remix IDE 是一个关于 Solidity 的 IDE，用来编写，编译和调试 Solidity 的代码。它是一个功能强大的、开源的 IDE 工具，它可以帮助用户，直接依靠普通的浏览器编写可靠的智能合约。Remix IDE 是用 JavaScript 编写的，支持在浏览器和本地使用。此外，Remix 还支持测试、调试和部署智能契约等等。

目前有关 Remix IDE 的项目及其所有功能都可以在 [Remix.Ethereum.org](https://remix.ethereum.org) 上找到，并富有官方文档，是一个快捷、方便的开源平台。

2.3.3 Code Runner 平台

CodeRunner IDE 是一个付费的开发工具，它可以运行各类的程序代码。它主要用于非深入的编程练习，或者涉及编程语言较多的计算机课设或软件项目等。目前 CodeRunner IDE 可以编译多于 20 种编程语言的代码，包括但不限于：C, C++, Java, Perl, Ruby, JavaScript, Go, PHP, Python 等等，并且可以基本实现即刻运行。CodeRunner IDE 还包括模糊搜索、选项卡选择占位符和文档片段等多种高级功能，可以帮助编程人员快速入手、学习一种语言并便捷的编译运行。

2.4 框架与工具介绍

2.4.1 Truffle 框架

Truffle 是一个目前世界领先的开发框架以及测试框架。Truffle 还是以太坊的一个高效的资源管理框架。Truffle 使基于以太坊的 Dapp 开发更加便捷、高效，目前 Truffle 有以下这些主要功能：

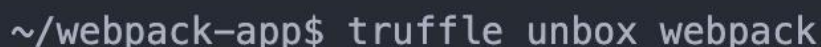
1. 可以完成智能合约的编译与部署
2. Truffle 提供链接处理以及二进制文件管理。
3. Truffle 可以提供合约测试
4. Truffle 部署的 Dapp 是可以扩展得
5. Truffle 环境管理处理良好，基于公链或私链的 Dapp 都可以在其上开发
6. Truffle 提供标准的 npm 包管理
7. Truffle 支持命令行直接运行，十分便利
8. Truffle 帮助实现图形化运行
9. Truffle 支持外部脚本的执行。

2.4.2 Webpack 框架

Webpack 框架是一个 JavaScript 和模块的捆绑器。使用者可以通过该框架将许多模块打包到几个捆绑的脚本资源中。代码分割允许按需加载应用程序的各个部分，通过“加载器”实现；模块可以是 CommonJs, AMD, ES6, CSS, 图像, JSON, Coffeescript, LESS...等等，当然还可以加入你的自定义模块。

Webpack 框架目前是以太坊官方推荐的 Dapp 开发框架之一，适用连接前端与智能合约，方便实现包含合约、迁移、测试、用户界面等多重功能的 Dapp 应用。

对于初学者来说，可以简单的利用图 2.8 所示的终端命令下载 Webpack 的默认模版，再根据自己实际的 Dapp 应用改写。



```
~/webpack-app$ truffle unbox webpack
```

图 2.8 解压 Webpack 模板语句

2.4.3 Ganache

Ganache，它的前身是 Testrpc，是一个虚拟的区块链，它设置了 10 个默认的以太坊地址，包括私钥和所有，并预先加载了 100 个模拟“Ether”。Ganache 本身没有“挖掘”——相反，它会立即确认任何即将到来的交易。这种特性使得迭代开发成为可能——开发者可以为代码编写单元测试，这些代码在这个模拟的区块链上执行，部署智能契约，测试，调用函数，然后将其全部销毁以进行进一步的模拟或新测试，将所有地址返回到初始状态：含有 100 个“Ether”。

Ganache 包含图形化以及命令行两种版本，通常用于开发者搭建私人区块链，模拟 Dapp 在区块链上的运行。

2.4.4 Java-Swing 框架

Java-Swing 框架是 Java 的基础框架之一，属于基础类(JFC)的一部分。该框架主要用于创建 Java 应用程序的图形化界面，完全由 Java 编写，可以提供一个基于窗口的应用界面。Java-Swing 框架构建在抽象窗口工具包的 API 之上，是一款轻量级组件。Java-Swing 包中提供了很多便捷易学的 Java Swing API 的类，如 JButton、JTextField、JRadioButton、JCheckbox、JTextArea、JColorChooser、JMenu 等等。

虽然目前有诸如 Spring、Struts 等更加先进的框架，但轻量级、易编写的 Swing 框架依旧是图形界面设计中值得考虑的选项。

2.4.5 JPBC 库

随着 Java 语言的流行，基于 Java 封装 JPBC 库越来越受到学者们的青睐。JPBC 库封装了一系列重要的密码学以及数学算法，提供了：

1. 一个基于对的密码库(PBC)的端口，由 Ben Lynn 开发的，用来直接在 Java 中执行基于对的密码系统的数学操作。
2. 一种包装器，它允许将成对计算委托给 PBC 库以提高性能。
3. 多线性映射的实现基于 Coron, Lepoint 和 Tibouchi 在整数上的实际多线性映射。该实现支持多线程，并使用内存映射文件保存在主内存需求中。

JPBC 库中包含的密码学接口如下图（图 2.9）：

Package it.unisa.dia.gas.jpbc	
Interface Summary	
Interface	Description
Element	Elements of groups, rings and fields are accessible using the Element interface.
ElementPow	Common interface for the exponentiation.
ElementPowPreProcessing	If it knows in advance that a particular value will be raised several times then time can be saved in the long run by using preprocessing.
Field<E extends Element>	Represents an algebraic structure.
FieldOver<F extends Field, E extends Element>	This interface represents an algebraic structure defined over another.
Pairing	This interface gives access to the pairing functions.
PairingParameters	Represents the set of parameters describing a pairing.
PairingParametersGenerator<P extends PairingParameters>	This interface lets the user to generate all the necessary parameters to initialize a pairing.
PairingPreProcessing	If it is known in advance that a particular element will be paired several times then time can be saved in the long run by using preprocessing.
Point<E extends Element>	This interface represents an element with two coordinates.
Polynomial<E extends Element>	This element represents a polynomial through its coefficients.
PreProcessing	Common interface for all pre-processing interfaces.
Vector<E extends Element>	This element represents a vector through its coordinates.

图 2.9 JPBC 库的密码学接口

2.5 本章小结

本章节概括性的讲述了支持本设计的基础理论，同时介绍了完成该设计所需的重要工具与框架等。虽然看起来复杂、冗长，但 Dapp 的设计本身就是一项集交互、多语言编程以及专业理论知识的综合设计。只有理解并尝试了基本工具的使用，才能妥善的开发 Dapp，并在 Dapp 领域结合专业知识，做出创新。

3. 需求分析及总体设计

本设计立足于合同签署问题，实现了一个基于聚合签名的合同签署系统。在这个系统中有多个角色，这些角色主要分为三类：签署者（ U_i ）、聚合者（ U_c ）以及验证者（ U_v ），他们在自己的节点中，完成自己的任务。同时，这个系统本身的搭建也较为复杂，涉及的模块包含前端、后端、一个以太坊私有链，以及链接它们的交互模块。对于系统整体的需求表现在各角色都能顺利完成自己的任务，对于流程的总体设计体现在帮助前者，并且做到安全、高效、完美运行。

目前粗略的把合同签署需求分为两类，一种是关于大宗交易或商业合同等场景的必须公开身份、必须可追溯责任的签署系统（公平性签署系统）；另一种是可以不公开身份、对于敏感合同签署方可选择保持匿名的签署系统（隐私性签署系统）。关于这两类需求，本设计也提出了不同的算法，保证需求的满足。

综上，本章节将具体分析各方面的需求，并体现基于需求设计系统的全流程思路。

3.1 角色需求分析

3.1.1 签署者（ U_i ）需求分析

首先来分析多方合同签署系统中数量最多的一个角色：签署者 U_i （ $i=3, 4, \dots$ ）。签署者的需求体现在以下几个方面：

1. 签署者需要完成“签名/签署”功能，这也是系统最基本的功能之一。当签署者收到合同信息后，可以选择是否对其签名。

对于该需求，认为需求的功能包含：

- 1) 对于基于公平性签署系统，应在前端设置按钮工具，方便交互。使用户可以连接到区块链上并调用合约。
- 2) 对于隐私性签署系统，应隐藏签署方个人的地址、签名等内容，仅设置一个让签署方完成签名的页面。

2. 签署者应公开部分信息，以方便溯源（仅对于公平性签署系统）。

对于该需求，认为需求的功能包含：

应当公示的信息可以成功显示在前端页面上。

3. 签署者要把签名信息发送给聚合者。

对于该需求，认为需求的功能包含：

后台预留接口，使签署者数据能够成功发送给聚合者。

综上，整理为如下需求表 3.1:

表 3.1 签署者需求表

系统	签名/签署需求	公开信息需求	信息发送需求
公平性签署系统	√	√	√
隐私性签署系统	√	X	√

3.1.2 聚合者(U_c)需求分析

聚合者 U_c 在系统中主要承担三个功能:

1. 需求者需要先检验是否每个签署方都完成签名，并且这个签名是正确的、合法的（仅对于公平性签署系统）。

对于该需求，认为需求的功能包含：

- 1) 聚合者可以调用智能合约里的算法，判断签署方是否正确、合法签名。
- 2) 聚合者若发现签署方签名不符合要求，应提示，可以预留前端提示框。

2. 需求者需要根据算法，完成多方签名聚合

对于该需求，认为需求的功能包含：

聚合者可以调用智能合约里的算法，正确生成聚合签名

3. 需求者需要把已聚合好的签名发送给验证者

对于该需求，认为需求的功能包含：

后台预留接口，使签署者数据能够成功发送给聚合者。

综上，整理为如下需求表 3.2:

表 3.2 聚合者需求表

系统	单用户签名认证	签名聚合	信息发送需求
公平性签署系统	√	√	√
隐私性签署系统	X	√	√

3.1.3 验证者(U_v)需求分析

对于验证者，唯一的需求就是验证聚合签名是否正确、合法。

对于该需求，认为需求的功能包含：

验证者可以调用智能合约里的算法，验证聚合签名并返回结果。

3.2 模块需求分析

3.2.1 前端模块需求分析

根据上文所提到的，不同的签署过程可能有不同的签署需求。因此前端设计也包含了对这两种不同需求的考虑。

针对公平性签署系统，由于提出的要求是“整个签名过程及签名结果都记录在以太坊区块链中”，因此我选择了非常方便快捷的 HTML 语言作为前端编程语言，搭配 CSS 框架的排版，该页面可以非常高效、快速的与智能合约交互。

针对隐私性签署系统，由于使用了盲化算法，需要 JPBC 密码库（这在 Solidity 中非常难以实现，Solidity 语言原生支持的密码算法非常少），因此我选择了直接使用 Java 语言，配合 Swing 框架，制作了客户端。该客户端满足前文提到的要求，保护了签署方的隐私信息，不展示用户的地址、合同信息以及签名信息等。

3.2.2 后端模块（智能合约）需求分析

对于后端模块，在本设计中指智能合约。智能合约一旦部署并在区块链上运行，此次就不可更改；如果用户与区块链上的智能合约交互，则被智能合约的条款约束，要遵守智能合约中规定的内容。根据这一特点，本设计中的后端应利用自身不可篡改、有约束力等特性，回应用户的交互需求、正确接受传入的参数，并向前端返回合法、正确的返回值。对于本设计来说，后端的最核心需求就是智能合约要确保正确地生成聚合签名，并将其记录在区块链上。

3.2.3 交互模块需求分析

本设计中的交互，指的是用户与智能合约间的交互，这也是 Dapp 中前端与后端的交互。交互模块的主要需求包含：

1. 为前端的 HTML 界面提供脚本，以便于签署方基于逻辑的操作。
2. 帮助系统部署 Webpack 框架，以便于与后端智能合约交互（主要指调用、传参等功能）。
3. 模拟基于 JavaScript 的钱包的功能，为签署方生成随机公私钥和地址。

3.3 非功能性需求分析

1. 界面友好

Dapp 系统界面友好、方便进行智能合约签署流程，可以让用户进行直观、简洁的操作。

2. 可扩展性

本文使用 Ganache 搭建区块链网络，Solidity 来撰写智能合约，使用 JavaScript 和 HTML 构建前端界面，因此无论是区块链底层网络，还是智能合约的各种方法亦或是前端显示界面都可以考虑进一步扩展。

3. 数据可靠性

产生的全过程数据最终会存储在底层区块链中，数据无法篡改且永久储存。根据区块链的 POW 共识算法，篡改的数据会由于单个节点的算力不足而变为支链，并随着主链的逐渐延长被舍弃。因此本系统实现了数据的可靠性。

4. 交互需求

针对不同类型的交互，采取不同的方式和方法，运用不同的交互手段和交互管理规则。

3.4.1 功能流程

1. 生成 16 进制, 32 字节的 256bit 随机数。
2. 根据随机数计算公钥、私钥、地址。
3. 选择模式: 公平性签署系统或隐私性签署系统。
4. 进行单人/单方签名。
5. 发送签名信息给聚合者 U_c 。

签署方 (U_i) 流程, 如下图 3.1 所示:

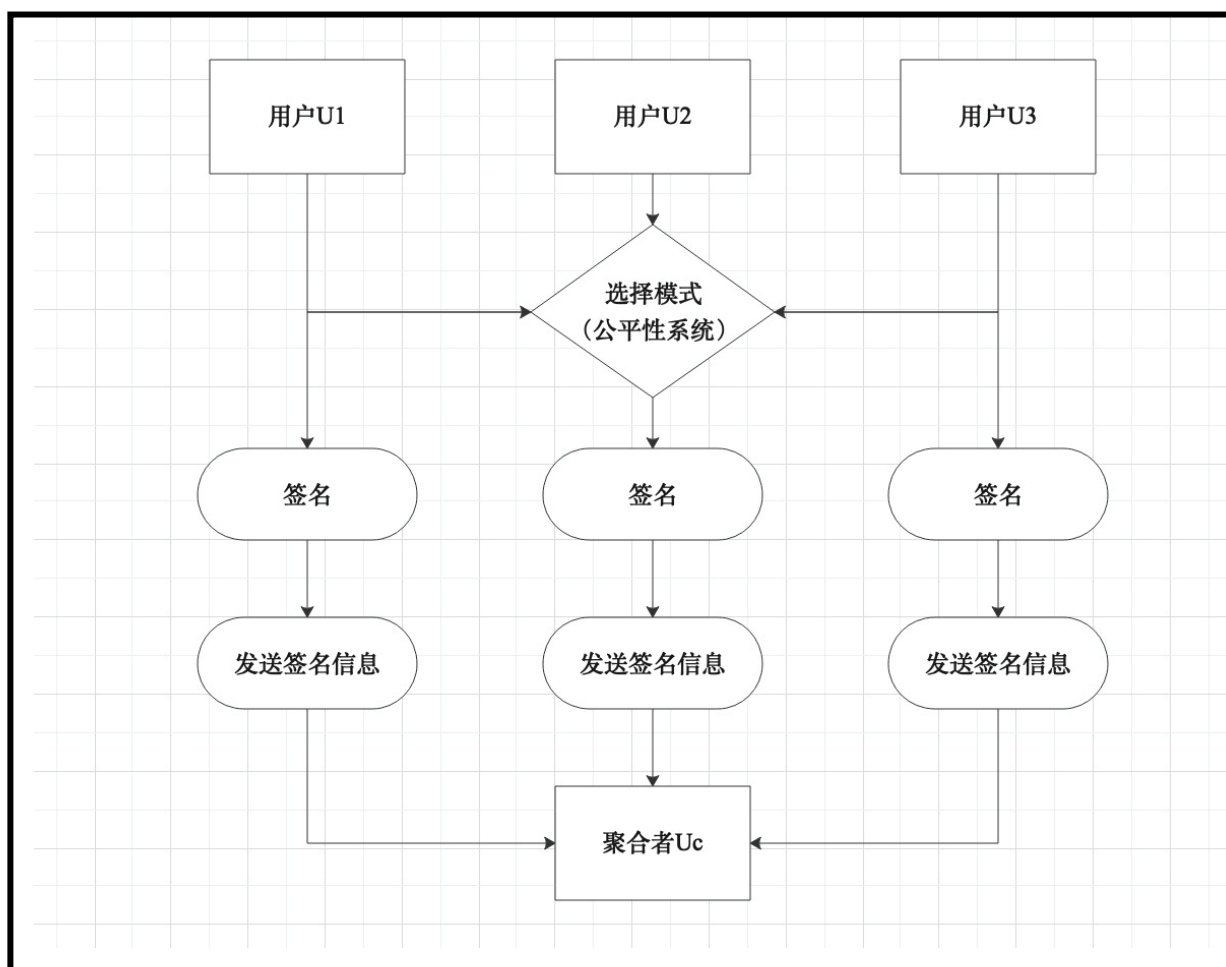


图 3.1 签署方流程

聚合者所经历的流程如下：

1. 收取签署方发送过来的签名。
2. 验证签名是否正确。
3. 若正确，进行聚合签名；若不正确，进行报错提示。
4. 将聚合签名发送给验证者 U_v 。

聚合者（ U_c ）流程，如图 3.2 所示：

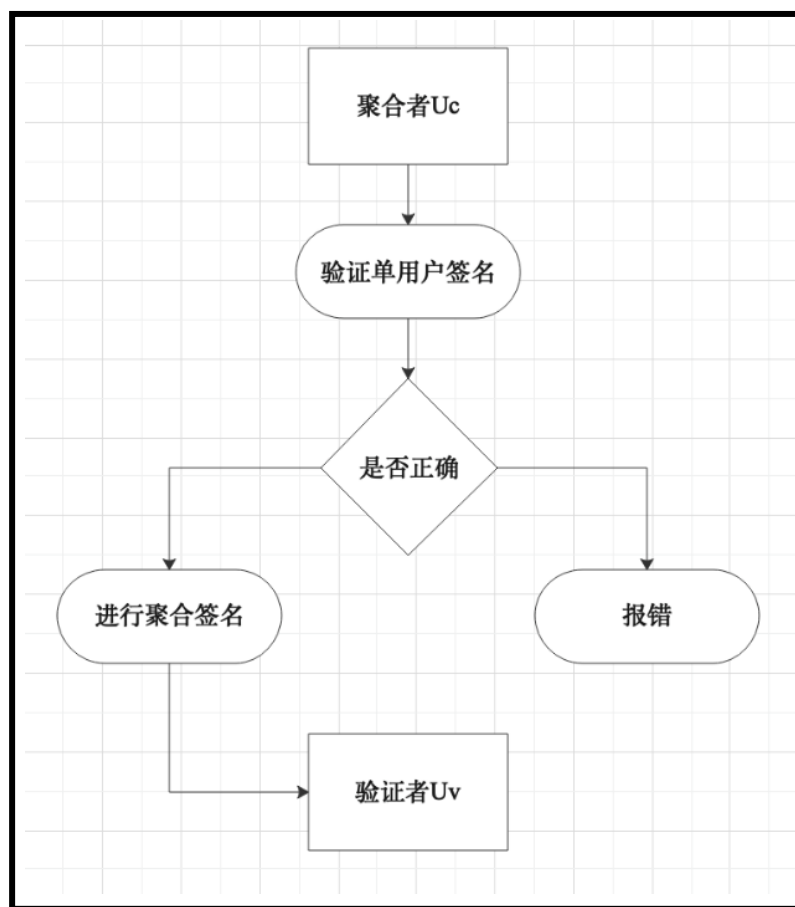


图 3.2 聚合者流程

验证者所经历的流程如下：

1. 收取聚合者 U_c 发送的聚合签名。
2. 验证聚合签名，若正确，则显示；若不正确，则报错。

验证者（ U_v ）流程，如图 3.3 所示：

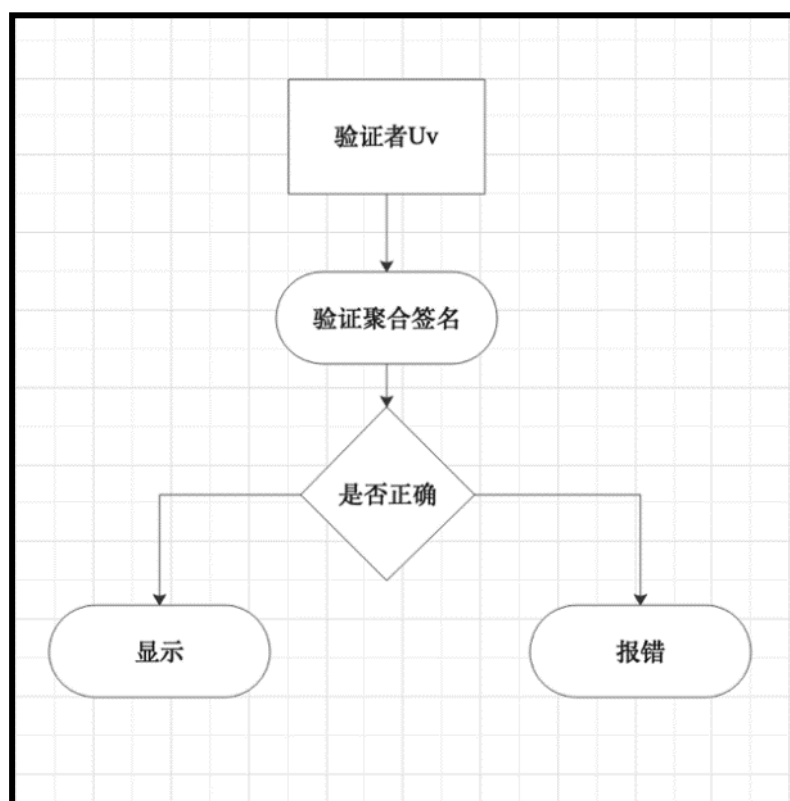


图 3.3 验证者流程

3.4.2 架构流程

在整体架构上，我选择了以太坊官方推荐的 `webpack3.js` 来构建 Dapp：因为对于普通用户来说，我们不能指其像开发者一样，通过命令行来与合约交互。此时 `webpack3.js` 提供了一个桥梁，方便用户与智能合约友好的交流，并保证全过程的可视化。

`webpack3.js` 是以太坊官方所支持、推广的 API。`Web3.js` 可以帮助智能合约的开发者们使用 HTTP 或者 IPC 与本地、远程的以太坊节点交互。`Web3.js` 包括了几个库，诸如“`shh`”以及“`bzz`”库，它们主要都是用来控制 `whisper`、`swarm` 协议的，在本设计中并没有涉及，就不赘述了。本设计中主要应用的两个库是：

1. eth 库

我们需要利用 eth 库来进行智能合约与区块链的交互。在本设计中，我们使用

```
1. var Web3 = require('web3');
```

等语句，先加载系统所需的类库，才能在后续实现不同模块的交互。

2. utils 库

utils 库包含了许多方便 Dapp 开发的工具，以便开发者更好的实现 Dapp 功能。本设计中，我们使用类似

```
2. web3.utils.randomHex(size)
```

的语句（上面的语句可以帮助生成指定位数 16 进制随机数），方便我们进行 Dapp 开发。

Web3 与 Geth 通信使用的是 Json-PRC，这是一种轻量级的 RPC 协议，整个通信的模型可以抽象为下图 3.4：

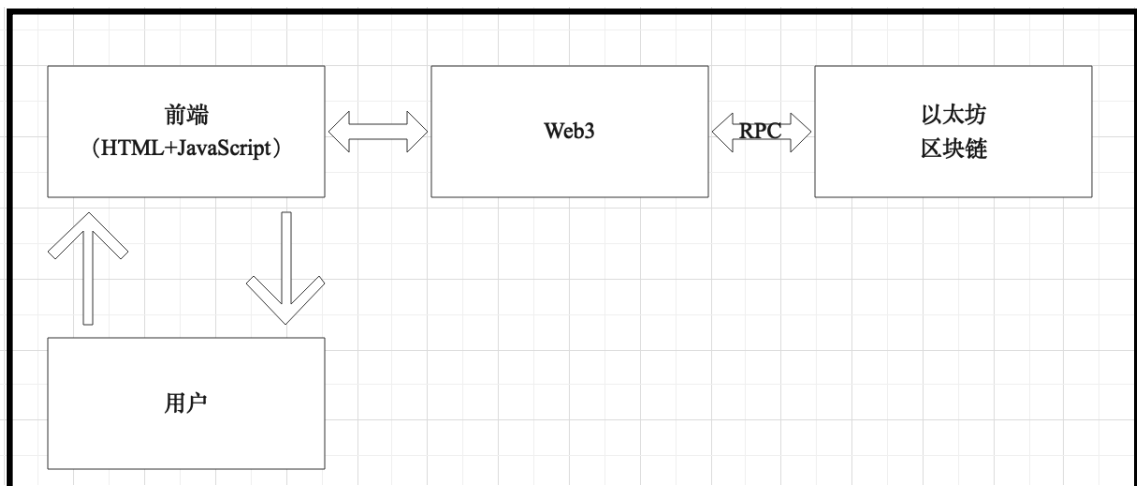


图 3.4 架构流程

3.5 本章小结

在本章节中，我们大致确定了系统的需求以及想要实现该需求的所需的接口、函数等。同时分别从前端模块，后端模块以及交互模块三方面进行了功能性需求分析，以此基础，便可逐步按照需求开始实现毕业设计中的一个功能，设计并实现一个基于聚合签名的合同签署系统，为后续的详细设计讲解与功能测试打下基础。

这一章是在上一章的基础上，从需求入手一步步提炼整体设计结构，把整体设计分为三部分，第一部分是与用户交互的前端页面，第二部分是实现用户与智能合约图形化交互的 web3.js 脚本，第三部分则是记录在区块链上的智能合约。通过对每个模块的解释，逐步实现需求中提到的功能，为下一章中整个系统的实现做铺垫。

4. 详细设计及实现效果

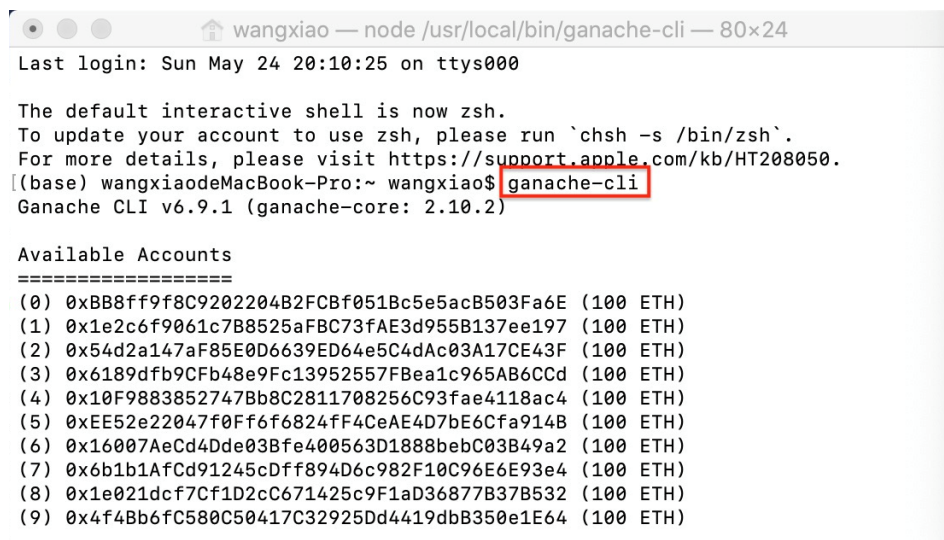
4.1 环境准备

4.1.1 以太坊私链搭建

本设计基于 Ganache-cli 工具搭建私链，以便运行 Dapp。Ganache 的安装及使用代码如下：

```
3. $ sudo npm install -g ganache-cli
4. $ ganache-cli
```

1. 利用 npm 工具安装 Ganache 命令行（CLI）版本
2. 输入此命令后，若前一步的安装成功，则证明私有链成功运行，命令行中会出现 Ganache 提供的测试账户等信息，如图 4.1 所示（部分信息）：



```
wangxiao — node /usr/local/bin/ganache-cli — 80x24
Last login: Sun May 24 20:10:25 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) wangxiaodeMacBook-Pro:~ wangxiao$ ganache-cli
Ganache CLI v6.9.1 (ganache-core: 2.10.2)

Available Accounts
=====
(0) 0xBB8ff9f8C9202204B2FCBf051Bc5e5acB503Fa6E (100 ETH)
(1) 0x1e2c6f9061c7B8525aFBC73fAE3d955B137ee197 (100 ETH)
(2) 0x54d2a147aF85E0D6639ED64e5C4dAc03A17CE43F (100 ETH)
(3) 0x6189dfb9CFb48e9Fc13952557FBea1c965AB6CCd (100 ETH)
(4) 0x10F9883852747Bb8C2811708256C93fae4118ac4 (100 ETH)
(5) 0xEE52e22047f0Ff6f6824fF4CeAE4D7b6Cfa914B (100 ETH)
(6) 0x16007AeCd4Dde03Bfe400563D1888bebC03B49a2 (100 ETH)
(7) 0x6b1b1AfCd91245cDf894D6c982F10C96E6E93e4 (100 ETH)
(8) 0x1e021dcf7Cf1D2cC671425c9F1aD36877B37B532 (100 ETH)
(9) 0x4f4Bb6fC580C50417C32925Dd4419dbB350e1E64 (100 ETH)
```

图 4.1 提供的测试账户信息

4.1.2 智能合约部署

前文已经提到过，本设计的实现主要依托于以太坊的 truffle 框架。在个人私有链保持成功运行的情况下，使用 truffle 框架的命令行语句操作，即可实现智能合约的部署。

```
1. $ sudo npm install -g truffle //安装 truffle
2. $ mkdir AggregateSignature //建立工作目录
3. $ cd AggregateSignature
4. $ truffle init //初始化
5. $ cd contracts
6. $ truffle compile //智能合约编译
7. $ truffle migrate //智能合约迁移
```

8. \$ truffle deployed //智能合约部署

1. 安装 truffle（仅第一次）
2. 建立工作目录
3. truffle 框架初始化，生成一些必备的配置文件（在不需要回归初始配置的情况下，对于一工作目录仅第一次使用）
4. 对智能合约（.sol）文件的编译
5. 存入迁移部署脚本
6. 合约部署（可同时部署多个），效果如图 4.2、4.3 所示：

```
app — node • npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash TERM...
Last login: Sun May 24 20:11:22 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) wangxiaodeMacBook-Pro:~ wangxiao$ cd AggregateSignature/
(base) wangxiaodeMacBook-Pro:AggregateSignature wangxiao$ truffle deployed

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name:      'development'
> Network id:        1590322287119
> Block gas limit:   0x6691b7
```

图 4.2 部署截图 1

```
app — node • npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash TERM...
2_AggregateSignature_migration.js
=====

Deploying 'AggregateSignature'
-----
> transaction hash:    0x094977d35bbb468807b13ace1dd49d3881c9f5ded462b685d4a2
2a419e6b8c6d
> Blocks: 0           Seconds: 0
> contract address:    0x29D418b2a20F95BC6d28FDC5f7514084EBD5709B
> block number:        3
> block timestamp:     1590322300
> account:             0xBB8ff9f8C9202204B2FCBf051Bc5e5acB503Fa6E
> balance:             99.97931064
> gas used:            827952
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.01655904 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:          0.01655904 ETH
```

图 4.3 部署截图 2

4.1.3 交互连接配置

1. Truffle 配置: truffle-config.js

Truffle 相关的配置文件，必须配置在工作空间的根目录下，否则会产生一些难以预料的问题。用 JavaScript 脚本编写 Truffle 的配置文件，其它参数可以依照默认值，我们主要关注的是我们声明的 `development` 节点，这个节点在 127.0.0.1 监听，并且行业内默认使用 8545 端口。

```
1. development: {
2.   host: "127.0.0.1",      // Localhost (default: none)
3.   port: 8545,             // Standard Ethereum port (default: none)
4.   network_id: "*",       // Any network (default: none)
5. },
```

2. App 配置:

1) package.json

`package.json` 文件包含包名称、版本号等众多内容，其中值得关注的是“`dependencies`”：依赖包列表。要完成这个配置，需要开发者预先下载适合的 web3 版本。

```
1. {
2.   "name": "app",
3.   "version": "1.0.0",
4.   "description": "",
5.   "private": true,
6.   "scripts": {
7.     "build": "webpack",
8.     "dev": "webpack-dev-server"
9.   },
10.  "devDependencies": {
11.    "copy-webpack-plugin": "^5.0.5",
12.    "webpack": "^4.41.2",
13.    "webpack-cli": "^3.3.10",
14.    "webpack-dev-server": "^3.9.0"
15.  },
16.  "dependencies": {
17.    "web3": "^1.2.4"
18.  }
19. }
```

2) webpack.config.js

在 webpack 的配置文件中，我们设定的模式是“`development`”开发模式，入口文件是与前端 HTML 文件处于同一目录的 `index.js` 文件。该文件负责几乎所有交互内容，是前端与智能合约交互的连接桥梁。

```
1. const path = require("path");
2. const CopyWebpackPlugin = require("copy-webpack-plugin");
3.
4. module.exports = {
5.   mode: 'development',
6.   entry: "./src/index.js",
7.   output: {
8.     filename: "index.js",
9.     path: path.resolve(__dirname, "dist"),
10.  },
11.  plugins: [
```

```
12.     new CopyWebpackPlugin([ { from: "./src/index.html", to: "index.html" } ]),
13.   ],
14.   devServer: { contentBase: path.join(__dirname, "dist"), compress: true },
15. });
```

3) Migrations 脚本:

以下两个由 JavaScript 的脚本，都是部署脚本，比如 initial_migration.js 就是用来部署 Migrations.sol 合约的，其它同理。

Initial_migration.js

```
1. const Migrations = artifacts.require("Migrations");
2.
3. module.exports = function(deployer) {
4.   deployer.deploy(Migrations);
5. };
```

AggregateSignature_migration.js

```
1. const AggregateSignature = artifacts.require("AggregateSignature");
2.
3. module.exports = function(deployer) {
4.   deployer.deploy(AggregateSignature);
5.
6. };
```

4.1.4 Eclipse 环境搭建

关于 Eclipse 的具体安装过程可参照其官方文档，只要安装时选择“Eclipse IDE for Java Developers”并执行默认参数安装即可（如图 4.4）。具体过程在官方文档中都详细可见。关于本设计的 Eclipse 环境配置关键点在于 JPBC 库的导入。



图 4.4 Eclipse 安装

关于 JPBC 库的导入，首先，应遵循官网的安装指导，前往专用连接下载。当 JPBC 相关的 jar 包下载到本地后，我们在已建好的工程下新建一个文件夹“lib”，位置放置于工作空间的根目录下，用它来存放 jar 文件，如图 4.5 所示。

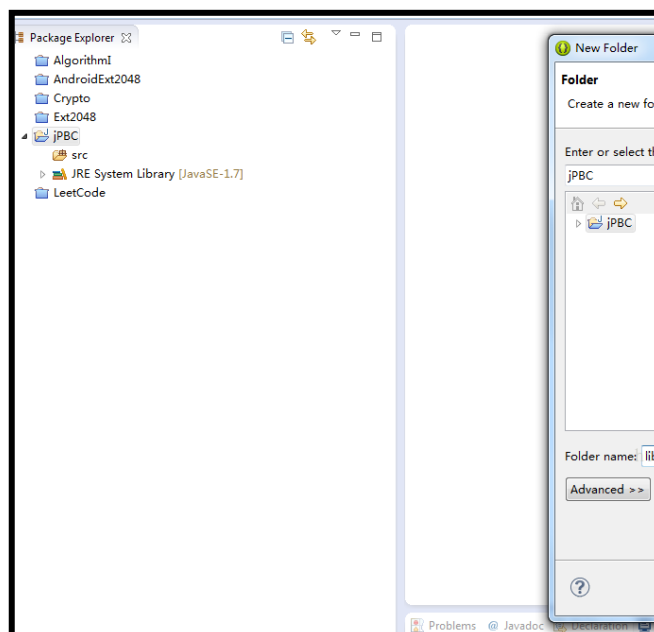


图 4.5 新建文件夹

接下来，我们点击 add jars，将 JPBC 里所有 jar 文件引入工程中，如图 4.6 所示：

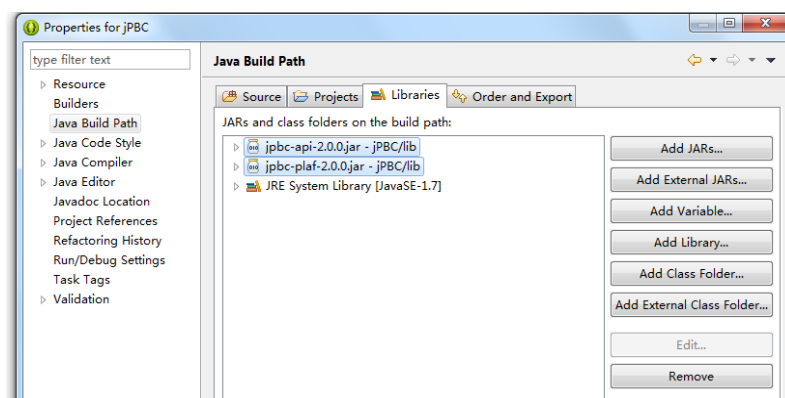


图 4.6 jar 文件引入

最后，我们尝试在 Java 文件的头部导入一些包进行试验：

```
1. import it.unisa.dia.gas.jpbc.Element;  
2. import it.unisa.dia.gas.jpbc.Pairing;  
3. import it.unisa.dia.gas.jpbc.PairingParameters;  
4. import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;  
5. import it.unisa.dia.gas.plaf.jpbc.pairing.a.TypeACurveGenerator;
```

若这些包在导入 Eclipse IDE 后并未报错，则证明 JPBC 库导入成功。

4.2 前端设计与实现

4.2.1 样式与排版

本设计的前端主要利用 HTML+CSS 编程制作。通过 HTML 语言编程，设计出前端的样式，再利用 CSS 工具对前端进行修饰。

首先，先利用 HTML 的布局工具画出整体的外观，包含颜色、组件等，再对他们进行排版，比如本设计的“三分布局”代码如下，实现效果如图 4.7。

```
1. <style>
2.     #header {
3.         background-color:black;
4.         color:white;
5.         text-align:center;
6.         padding:5px;
7.     }
8.     #nav {
9.         line-height:30px;
10.        background-color:#eeeeee;
11.        height:110px;
12.        width:100px;
13.        float:left;
14.        padding:5px;
15.    }
16.    #section1 {
17.        width:1050px;
18.        float:left;
19.        padding:10px;
20.    }
21.
22. }
23. </style>
24.
```

基于聚合签名的区块链合同签署系统

用户1 (U1)

进行签名

?

用户2 (U2)

进行签名

?

用户3 (U3)

进行签名

?

地址为:

(m,ri):

是否签名

图 4.7 布局实现效果

4.2.2交互实现

本设计中的用户-页面-智能合约交互问题，主要由 JavaScript 编写脚本解决。下文将以生成聚合签名这一过程为例，简述交互的实现方式。

首先，我们在界面上“画”出一个生成聚合签名的按钮，代码如下：

```
1. <button onclick="App.MutiSignatureGenerate()">生成聚合签名</button>
```

效果如图 4.8 所示：



图 4.8 聚合签名按钮

在这一句 HTML 语言中，<button>表示了该标签的定义是按钮，如果执行就会生成一个按钮工具。后面的 onclick="App.MutiSignatureGenerate()"表明了此按钮将与 JavaScript 脚本中的 MutiSignatureGenerate()模块交互。

```
2. MutiSignatureGenerate: async function() { //生成聚合签名
3.     const MutiSign=document.getElementById("MutiSign");
4.     var value=await this.meta.methods.Uv_MutiSignGenerate(s1,s2,s3).call
5.     ();
6.     MutiSign.innerHTML = value;
7. }
```

MutiSignatureGenerate()模块就是本设计中生成聚合签名的模块，上面的代码就是为方便讲解而简化的模块代码。为了将生成结果显示在前端页面中，我们现在前端页面预留了 id 为“MutiSign”的标签。接下来，我们通过脚本的“.call()”语句，调用智能合约里的 Uv_MutiSignGenerate()方法，并将最终结果赋值给预设的标签，实现前端界面与智能合约通过 JavaScript 脚本交互，效果如图 4.9 所示：

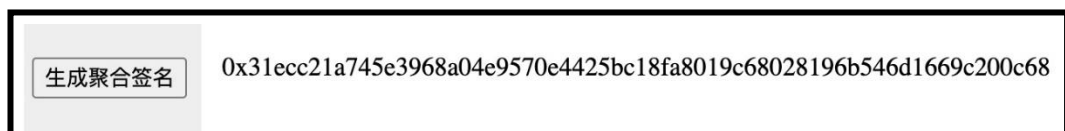


图 4.9 聚合签名展示

通过分析上述例子的流程，我们可以发现，本设计的交互实现机制的流程可以概括为：签署方通过自己的操作与界面交互，由于已经预置了 JavaScript 脚本，因此每当用户有操作时，脚本都会给予响应来做出预期的行为。这些预期的行为可以是来自脚本自身的（比如更改页面的某些内容），也可以是进行下一步深入交互的（比如调取智能合约中的一些方法、函数）。之后再通过脚本，传回前端页面，实现全系统的交互流畅。

为方便讲述交互流程，我制作了交互的时序图，如下图 4.10 所示：

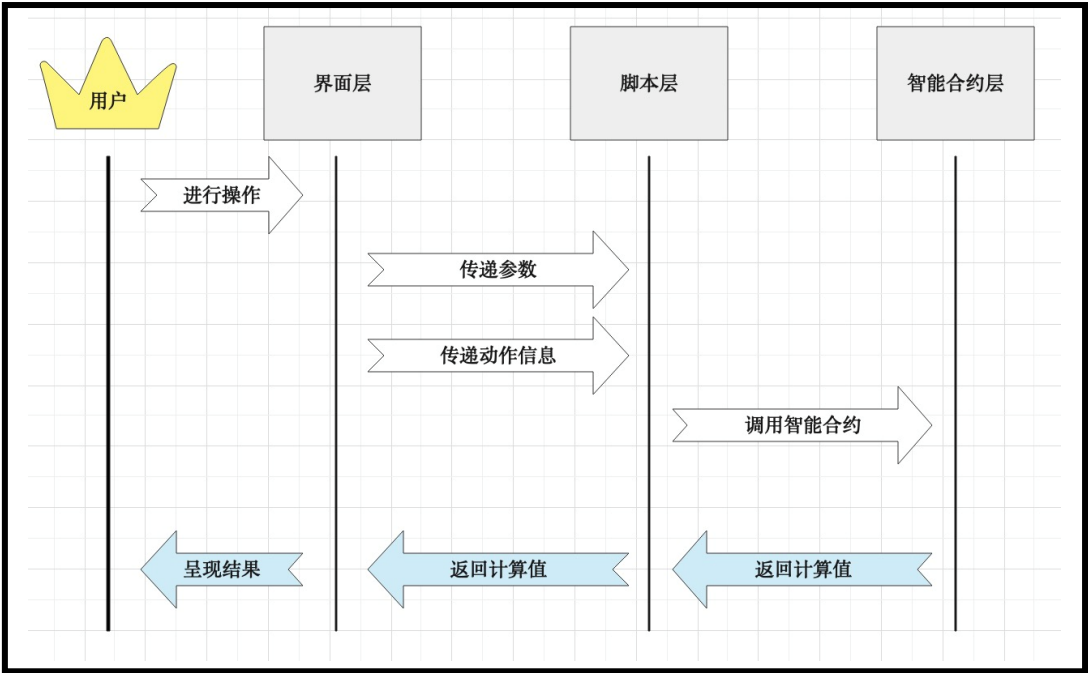


图 4.10 交互时序图

4.2.3 隐私性系统前端的设计与实现

考虑到 Solidity 语言提供的密码库贫乏，隐私性系统主要利用 Java 语言实现。这就涉及到，为了确保用户的交互友好性，我们也需要设计一个前端来实现图形化交互。

在选择绘图框架方面，本设计选择了 Java 的 Swing 框架。其主要设计思路与一般前端基本一致：界面上添加组件，再为组件添加事件。

首先，我们在 main 函数中设计一个基本的“画布”，指定它的长宽、节面的标题等，实现代码如下：

```
1. import javax.swing.JFrame;
2. public class main {
3.     public static void main(String[] args) {
4.         JFrame f=new JFrame("基于聚合签名的区块链合同签署系统");
5.         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6.         f.add(new work());
7.         f.setSize(500, 300);
8.         f.setVisible(true);
9.     }
10. }
```

接下来，我们需要布置界面上的组件，组件都遵循先声明，后添加的原则，即先声明组件的类型、变量名，再设定它们在画布上的位置，最后通过 add（）语句使其加入画布并展示。这一部分的节选代码如下：

```
1. JTextField n1=new JTextField();
2. JLabel l1=new JLabel("输入聚合签名:");
3. JButton check1 = new JButton("验证");
4. JLabel result1=new JLabel("...");
5. n1.setBounds(150, 200, 100, 40);
6. add(n1);
7. setBounds(50, 200, 100, 40);
8. add(l1);
9. check1.setBounds(250, 200, 100, 40);
10. add(check1);
11. add(result1);
```

最后，我们需要添加鼠标事件，使前端与后端进行交互。举例来说，当签署方 U_1 想进行签名时，鼠标点击 U_1 签名按钮，即可调用使 U_1 进行签名操作的相关内容，并保存 U_1 的签名，其节选代码如下：

```
1. public class SListener implements ActionListener {
2.
3.     public void actionPerformed(ActionEvent e) {
4.
5.         if (e.getSource() == b1) {
6.             String s_1=String.valueOf(s1);
7.             try {
8.                 BufferedWriter out = new BufferedWriter(new FileWriter("
s1.txt"));
9.                 out.write(s_1);
10.                out.close();
11.                System.out.println("文件写入成功！");
12.            } catch (IOException k) {
```

```
13.         }  
14.  
15. }  
16. }
```

隐私性系统的前端整体效果如图 4.11 所示：



图 4.11 隐私性系统界面

4.3 后端（智能合约）设计与实现

4.3.1 单用户签名生成

在完成系统参数初始化后，智能合约要提供的第一个功能就是让单一签署方生成对合同的签名。以签署方 U_1 为例，下面的代码展示了签署方 U_1 对合同的签名 s_1 的生成过程。在生成 s_1 的过程中，除了系统初始化状态下，已被各签署方知悉的合同消息 $message$ 、个人公私钥 y_1 、 x_1 等。需要每个用户选取 k_i ， k_i 满足 $k_i \in_R [1, p-1]$ ，计算

$$r_i = g^{k_i} \bmod p \quad (4.1)$$

并将 r_i 发送给系统中的其他签署方 $U_j (j \neq i)$ 。

生成 r_i 的过程用流程图表示为图 4.12：

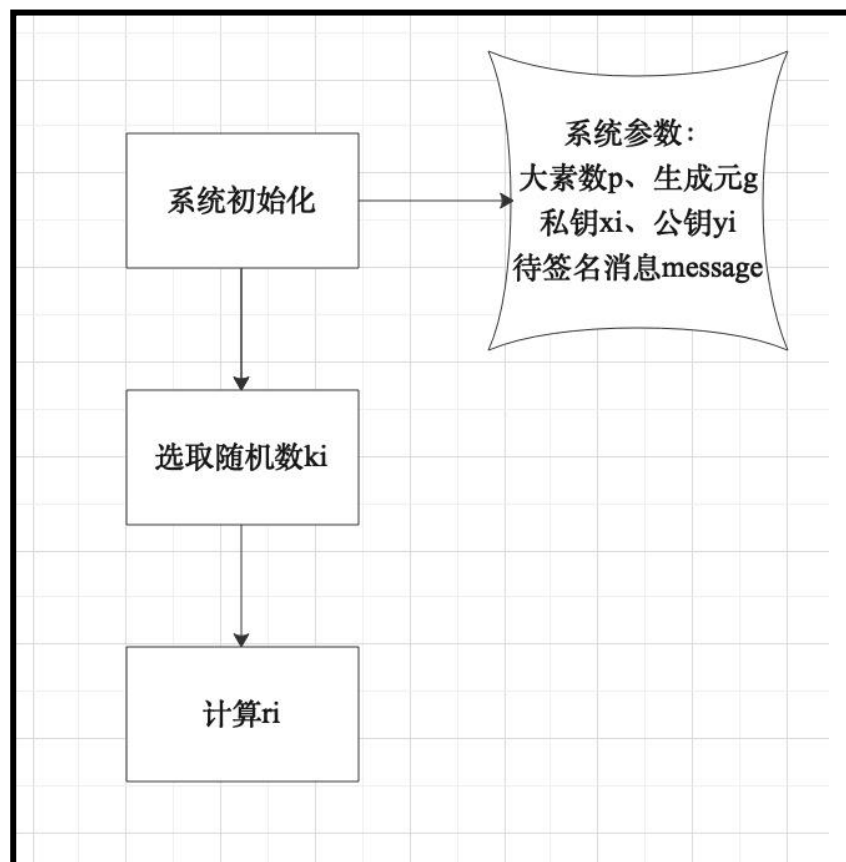


图 4.12 r_i 生成

将此过程转化为代码，可表示为：

```

1. function r1_Generate () public returns(uint){
2.   k1=random();
3.   uint r1=mulmod(g**k1, 1, p);
4.   return r1;
5. }
  
```

当每个用户都收到其它用户的 r_i 后，计算：

$$R = \prod_{i=1}^n r_i^{r_i} \bmod p \quad (4.2)$$

之后再计算

$$s_i = (R + h(m))x_i - r_i k_i \pmod{\varphi(p)} \quad (4.3)$$

将签名信息 $(m, (r_i, s_i))$ 发送给聚合者 U_c 。

上述过程转化为流程图如图 4.13 所示：

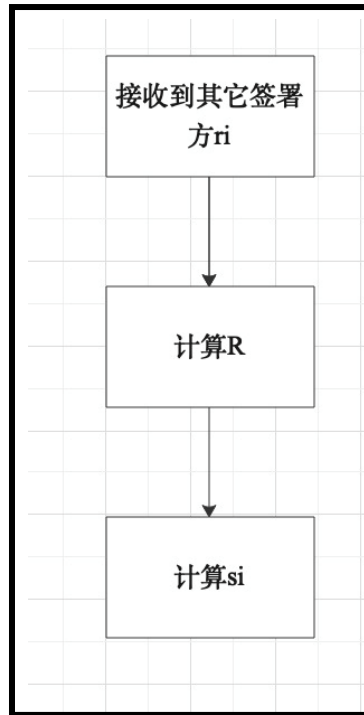


图 4.13 si 生成

再将此过程转化为代码：

```

1. function s1_Generate (uint r_2,uint r_3) public returns(uint){
2.   uint r_1=r1_Generate();
3.   uint r1=r_1;
4.   uint r2=r_2;
5.   uint r3=r_3;
6.   uint R=mulmod(r1**r1, 1, p)*mulmod(r2**r2, 1, p)*mulmod(r3**r3, 1, p);
7.   bytes32 hashmsg=sha256(abi.encodePacked(message)); // the message to be con
   veyed
8.   uint hashnum=uint(hashmsg);
9.   s1=(R+hashnum)*(uint(x1))-mulmod(r1,k1,fai_p);
10.  return s1;
11. }
  
```

当上述步骤完成后，用户界面将提示签名信息发送成功，并公布用户的地址与签名信息，如图 4.14 所示：

用户1 (U1)	
	<input type="button" value="进行签名"/>
	发送完成
地址为:	0x6a2d057c2966e251e132 af3691db153ab65f0d1a
(m , ri):	(BlockChainAggregateSignature , 0x036b6384b5eca791c62761152d0c79 bb0604c104a5fb6f4eb0703f3154bb3d)

图 4.14 用户界面信息

4.3.2 单用户签名验证

在聚合者 U_c 的部分， U_c 的第一个任务就是验证单用户的签名是否正确，如果正确才能进行接下来的提示，否则要给出提示。验证的密码学原理基于离散对数，推导过程如下：

$$si = (R + h(m))xi - r_i k_i \pmod{\varphi(p)} \quad (4.4)$$

$$r_i k_i + si = (R + h(m))xi \pmod{\varphi(p)} \quad (4.5)$$

$$g^{r_i k_i + si} = g^{(R + h(m))xi} \pmod{p} \quad (4.6)$$

$$r_i^{r_i} g^{si} = y_i^{(R + h(m))} \pmod{p} \quad (4.7)$$

由此我们得到了“验证等式”，可以进行验证流程。验证流程图如图 4.15 所示：

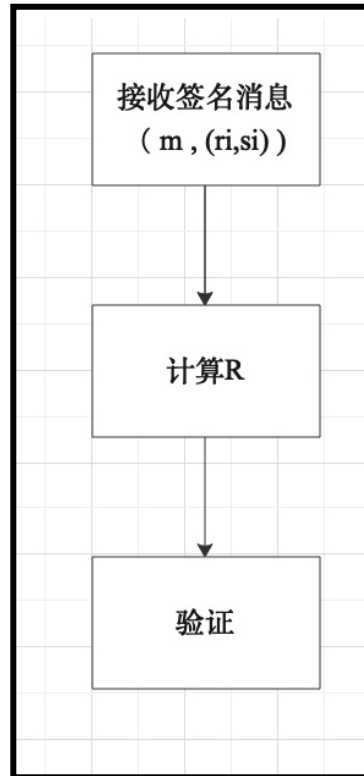


图 4.15 用户验证流程

将流程图转化为代码如下：

```

1. function Uc_SignVerify () public returns(bool,bool,bool){
2.
3.   uint r1=r1_Generate();
4.   uint r2=r2_Generate();
5.   uint r3=r3_Generate();
6.
7.   uint R=mulmod(r1**r1, 1, p)*mulmod(r2**r2, 1, p)*mulmod(r3**r3, 1, p);
8.
9.   bool flag1;
10.  bool flag2;
11.  bool flag3;
12.
13.  bytes32 hashmsg=sha256(abi.encodePacked(message)); // the message to be conveyed
14.  uint hashnum=uint(hashmsg);
15.
16.  if((r1**r1)*(g**s1)==mulmod(y1**(hashnum+R),1,p)) flag1=true;
17.  if((r2**r2)*(g**s2)==mulmod(y2**(hashnum+R),1,p)) flag2=true;
18.  if((r3**r3)*(g**s3)==mulmod(y3**(hashnum+R),1,p)) flag3=true;
19.
20.  return (flag1,flag2,flag3);
21.
22. }
  
```


当上述步骤完成后，用户界面将提示单用户签名是否通过验证，如图 4.16 所示：



图 4.16 用户签名验证

4.3.3 聚合签名生成

在 U_c 验证了各签署者的签名，并确保其正确后，可以开始生成聚合签名。生成聚合签名的公式是：

$$S = s_1 + s_2 + \dots + s_n \pmod{\varphi(p)} \quad (4.8)$$

将其用代码表示如下，生成聚合签名的效果如图 4.17：

```
1. function Uc_MutiSignGenerate (uint s_1,uint s_2,uint s_3) public view return
   s(uint){
2.   uint S;//MutiSignature
3.   S=mulmod((s_1+s_2+s_3), 1, fai_p);
4.   return S;
5. }
```

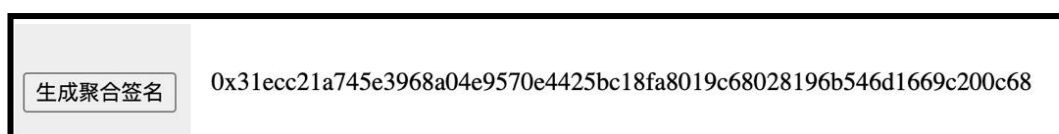


图 4.17 聚合签名生成

生成聚合签名 S 后， U_c 再把聚合签名发送给验证者 U_v 。

4.3.4 聚合签名验证

当聚合者 U_v 收到 U_c 发来的聚合签名后，就可以进行验证。验证的依据基于离散对数问题，简述为

$$\text{Ver}(m, R, S) = \text{True} \quad (4.9)$$

即：

$$Rg^S = \prod_{i=1}^n y_i^{h(m)+R} \quad (4.10)$$

据此，我们就明确了聚合签名验证的流程，如图 4.18 所示：

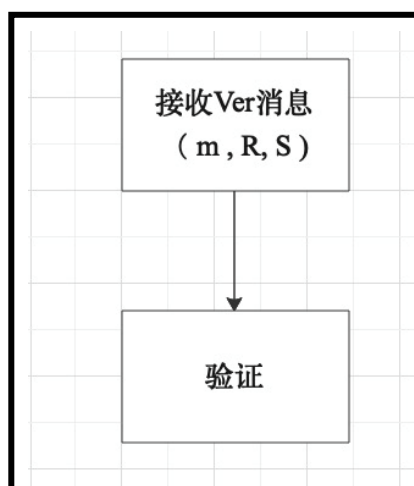


图 4.18 聚合签名验证流程

根据这一流程，本设计编写如下代码：

```

1. function Uv_MutiSignVerify (uint S) public returns(bool){
2.     uint R=mulmod(r1**r1, 1, p)*mulmod(r2**r2, 1, p)*mulmod(r3**r3, 1, p);
3.     bool Mutiflag;
4.     uint a=R*(g**S);
5.     uint b=(y1**(hashnum+R))*(y2**(hashnum+R))*(y3**(hashnum+R));
6.     if(a==b) Mutiflag=true;
7.     else Mutiflag=false;
8.     return Mutiflag;
9. }
    
```

实现的效果如下图 4.19 所示：



图 4.19 聚合签名验证

关于此验证合理性的数学公式推导如下：

$$r_i^{r_i} g^{s_i} = y_i^{(R+h(m))} \pmod{p} \quad (4.11)$$

$$\prod_{i=1}^n r_i^{r_i} g^{s_i} = \prod_{i=1}^n y_i^{(R+h(m))} \pmod{p} \quad (4.12)$$

$$\prod_{i=1}^n r_i^{r_i} \prod_{i=1}^n g^{s_i} = \prod_{i=1}^n y_i^{(R+h(m))} \pmod{p} \quad (4.13)$$

$$R g^{s_1+s_2+\dots+s_n} = \prod_{i=1}^n y_i^{(R+h(m))} \pmod{p} \quad (4.14)$$

4.4 隐私性多方聚合签署系统的设计与实现

经典的聚合签名算法仅考虑了多方合同签署的公平性、聚合性及公开可验证性，对隐私性保护考虑不足，聚合方可验证每个用户的签名，任何用户均对聚合签名进行验证，这样就泄露了单个用户的签名，并且泄露了多方合同签署这一客观事实，难以保证隐私安全。

本文针对隐私安全、公平性、聚合性及公开可验证性等应用需求，提出了一种具有隐私性的多方合同聚合签名方案。首先对经典的聚合签名方案进行了安全性进行分析，基于签署方共享密钥预协商技术，创新性提出并实现了具有隐私保护的多方合同聚合签名方案。

4.4.1 经典聚合签名算法的安全性分析

隐私性签署系统的灵感就是源自于公平性系统的一些缺点与不足。

1. 同态攻击

在公平性系统中，使用了 Harn 算法完成聚合签名。Harn 算法虽然在一般情况下可以做到安全、高效，但实际上难以避免同态攻击：

在公平性系统介绍部分，我们已经提到，Harn 算法需要每个签署方选择一个随机数 k_i ，但当 $k_3 = k_2 + k_1$ 时，由于离散对数 $r = g^k \pmod{p}$ 的性质，就满足了 $r_3 = r_1 r_2$ ，此时攻击者可以连立方程组：

$$s_1 = [x(H(m_1) + r_1) - k_1] \pmod{p-1} \quad (4.15)$$

$$s_2 = [x(H(m_2) + r_2) - k_2] \pmod{p-1} \quad (4.16)$$

$$s_3 = [x(H(m_3) + r_3) - k_3] \pmod{p-1} \quad (4.17)$$

$$k_3 = k_2 + k_1 \quad (4.18)$$

此时有四个方程，四个未知数。这样系统信息就会被破解了。

2. 隐私泄露

在 Harn 聚合签名方案中，由 4.3.2 分析表明，聚合者 U_c 可以得到并且验证每个用户的签名，这样就泄露了用户已经对合同进行签署这一事实；根据聚合签名验证公式（4.10），因为任何用户均知道所有签名方的共享密钥 y_i ，任何用户均可以验证聚合签名的正确性，这样会泄露多方用户合同签署这一客观事实，在有些高隐私性应用中，多方合同签署这一客观事实是不应该被泄露的。

4.4.2 隐私性多方聚合签署系统

本文提出的隐私性多方聚合签署系统如图 4.20 所示：

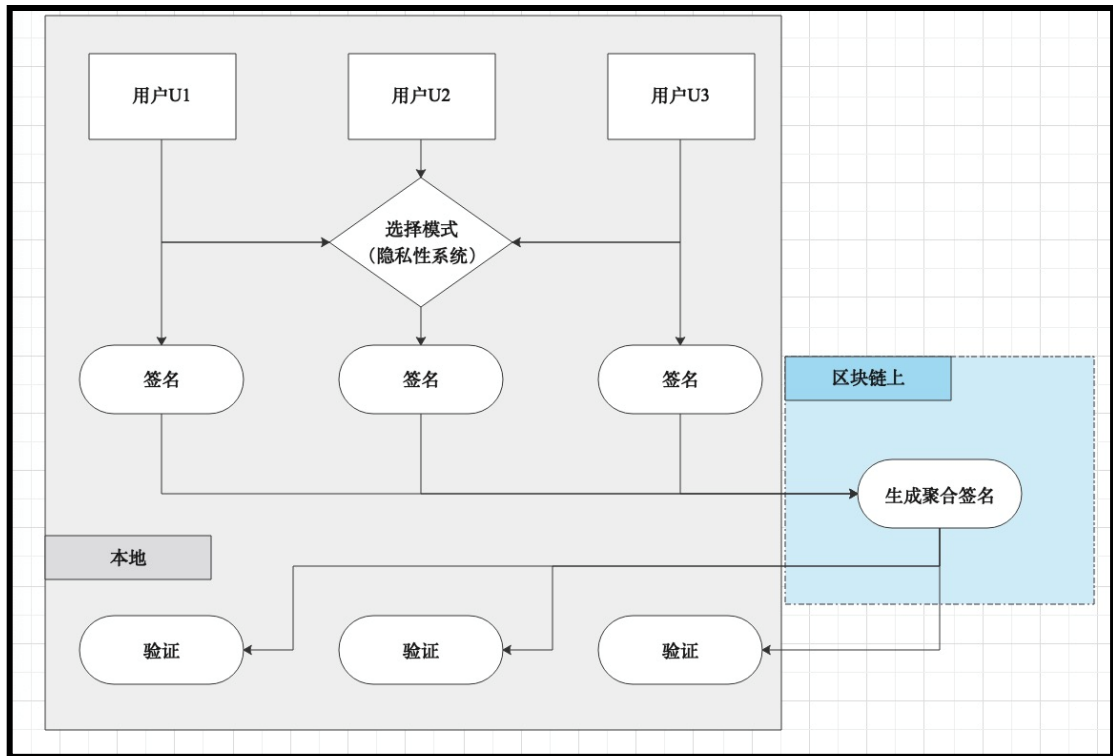


图 4.20 隐私性系统架构图

1. 共享密钥预协商

为了规避上文分析的安全风险，本设计引入了密钥协商过程，希望通过生成一个仅对合同签署方共享的密钥 P_{pub} 来提高系统安全性。这就要求系统，除了签署方自己的私钥 x_i ，需要再选取一个随机数 $x_{k1}, x_{k2}, x_{k3}, \dots$ 然后将此随机数保持私有，通过双线性对的算法与参与的签署方共享，生成密钥，这就免除了公平性系统中提到的聚合者 U_c 以及验证者 U_v 的参与了。

2. 签署方生成签名

此时，签署方生成自己的签名所用的算法是：

$$s_i = (x_i + x_{ki})h(m) \bmod (p-1) \quad (4.19)$$

此签名虽与公平性签名的单用户签名形式类似，但安全性却大大提高：同时加入了用户选取的随机数 x_{ki} 和自己的私钥 x_i ，这样此公式就无法轻易通过方程组连立而破解。并且此签名不支持公开验证，因为 x_{ki} 所对应的 $g^{x_{ki}}$ 是不会公开的。

3. 聚合签名的生成与验证

之后，这一部分生成的聚合签名被拿到区块链上进行生成，生成的结果将会被记录在区块链上，防止日后某方提出异议。聚合签名的验证则可由用户本

地完成，这是因为每个用户都拥有协商好的共享密钥，并且可以得知其它签署方公开的公钥 y_i ，那么用户自己就可以公开验证聚合签名的有效性。少了聚合者 U_c 以及验证者 U_v 的参与，签署系统的隐私性提高了许多，同时基于密钥协商的聚合签名也更加安全，不容易被攻破。

4.4.3 隐私性多方聚合签署系统实现

1. 共享密钥预协商

各签署方首先需要协商一个共享密钥 P_{pub} ，此处我选择了“双线性对”相关算法解决了密钥协商的问题。根据系统初始化时生成的大素数 P ，签署方 U_1 、 U_2 、 U_3 分别选择随机数 $a, b, c \in Z_p^*$ ，执行表 4.1 中描述的步骤：

表 4.1 密钥协商过程

步骤	用户	行为
1	U_1	计算 g^a ，并发送给 U_2 、 U_3
	U_2	计算 g^b ，并发送给 U_1 、 U_3
	U_3	计算 g^c ，并发送给 U_1 、 U_2
2	U_1	受到其它签署者信息后，计算 $K_A = e(g^b, g^c)^a$
	U_2	受到其它签署者信息后，计算 $K_B = e(g^a, g^c)^b$
	U_3	受到其它签署者信息后，计算 $K_C = e(g^a, g^b)^c$

此时根据双线性相关知识，我们可知 $K_A=K_B=K_C$ ，最后协商的公钥就是：

$$P_{pub}=K_A=K_B=K_C= e(g^b, g^c)^a= e(g^a, g^c)^b= e(g^a, g^b)^c \quad (4.15)$$

即：

$$P_{pub}= e(g, g)^{abc} \quad (4.16)$$

利用 JPBC 库封装的一些函数，我们把上述密码学过程转化为代码：

```

1. static int rBits = 160;
2. static int qBits = 512;
3.
4. static TypeACurveGenerator pg = new TypeACurveGenerator(rBits, qBits);
5. static PairingParameters typeAParams = pg.generate();
6. static Pairing pairing = PairingFactory.getPairing(typeAParams);
7.
8. Element a = pairing.getZr().newRandomElement().getImmutable();
9. Element b = pairing.getZr().newRandomElement().getImmutable();
10. Element c = pairing.getZr().newRandomElement().getImmutable();
11. generator = pairing.getG1().newRandomElement().getImmutable();
12.
13. Element ga = generator.powZn(a);
14. Element gb = generator.powZn(b);
15. Element gc = generator.powZn(c);
16.
17. Element ka =pairing.pairing(gb, gc).powZn(a);
18. Element kb =pairing.pairing(ga, gc).powZn(b);
19. Element kc =pairing.pairing(ga, gb).powZn(c);

```

2. 签署方各自生成签名

接下来，我们利用每个签署方自己的私钥 X_i 和选定的随机数 X_{ki} （ $X_{ki}=a, b, c, \dots$ ），以及合同内容的哈希值，计算各自的签名：

```
1. static String message="BlockchainAggregateSignature";
2. static int msg=message.hashCode();
3.
4. s1=Math.floorMod((x1+xk1)*msg,p-1);
5. s2=Math.floorMod((x2+xk2)*msg,p-1);
6. s3=Math.floorMod((x3+xk3)*msg,p-1);
```

3. 聚合签名的生成与验证

聚合签名的生成过程依旧是在区块链上调用智能合约完成，与上文相同，在此就不再次描述了。在盲化的聚合签名验证方面，主要遵循下面公式进行验证：

$$g^S = \prod_{i=1}^u y_i^{h(m)} \cdot P_{pub}^{h(m)} \quad (4.17)$$

由于盲化作用，验证时无法知道谁签过名了，只能验证聚合签名是否正确，保护了用户的隐私。

4.5 系统测试

4.5.1 公平性系统测试

点击“进行签名”按钮，将由上至下依次显示“发送完成”、地址、消息 m 以及 r_i 的信息，如下图 4.21 所示：

用户1 (U1)	
<input type="button" value="进行签名"/>	
发送完成	
地址为:	0x6a2d057c2966e251e132 af3691db153ab65f0d1a
(m , ri):	(BlockchainAggregateSignature , 0x036b6384b5eca791c62761152d0c79 bb0604c104a5fb6f4eb0703f3154bb3d)

图 4.21 签名测试

点击“检验签名”按钮，则可以显示出每个用户是否都进行了签名。已签名的用户会显示“已签名”；未签名的用户则显示“undefined”，意为系统微定义该用户签名。效果如图 4.22 所示：

	用户1 (U1)	用户2 (U2)	用户3 (U3)
	<div>进行签名</div>	<div>进行签名</div>	<div>进行签名</div>
	发送完成	?	发送完成
地址为:	0x60a066d10b7ce81d7ac2ad9874daac46e2282f6b		0x7f9e9bf9dec12c5cef146f93a5eee56772ee647f
(m , ri):	(BlockChainAggregateSignature , 0x036b6384b5eca791c62761152d0c79bb0604c104a5fb6f4eb0703f3154bb3d)		(BlockChainAggregateSignature , 0xce6d7b5282bd9a3661ae061feed1dbda4e52ab073b1f9285be6e155d9c38d4)
<div>是否签名</div>	已签名	undefined	已签名

图 4.22 检验签名测试

检验用户签名过程同理：点击“用户签名验证”按钮，若该用户通过验证，则显示“通过验证”，否则显示“undefined”，如图 4.23 所示：

	聚合者(Uc):		
<div>用户签名验证</div>	通过验证	undefined	通过验证

图 4.23 用户签名验证测试

若验证全部通过，则点击“生成聚合签名”按钮，可以生成聚合签名，如图 4.24 所示：

	聚合者(Uc):	
<div>用户签名验证</div>	通过验证	通过验证
<div>生成聚合签名</div>	0x31ecc21a745e3968a04e9570e4425bc18fa8019c68028196b546d1669c200c68	

图 4.24 聚合签名生成测试

反之，若之前有用户未签名或者签名验证未通过，则页面会弹框提示“请先完成之前的步骤”，如图 4.25 所示：

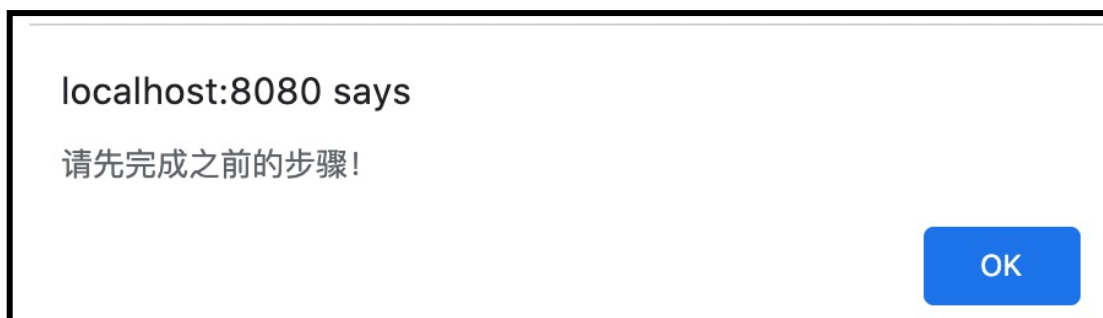


图 4.25 报错弹窗生成测试

最后，点击“聚合签名验证”按钮，即进行聚合签名验证测试，若测试通过，则界面会显示“通过验证”，如图 4.26 所示：

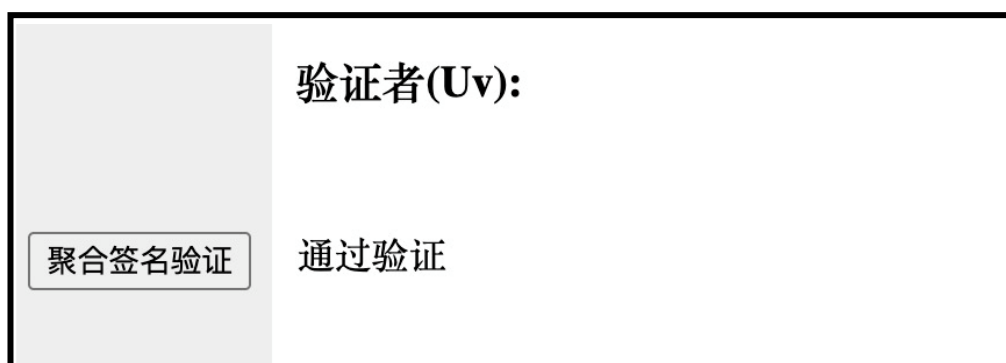


图 4.26 聚合签名验证测试

4.5.2 隐私性系统测试

隐私性系统主要依托于 Java-Swing 框架编写的客户端（如图 4.27 所示）：



图 4.27 隐私性系统客户端测试

在客户端上，我们点击“ U_1 签名”、“ U_2 签名”、“ U_3 签名”按钮，即可实现生成单用户签名。当签名成功写到后台预设的存储地点时，命令行会提示写入成果，如图 4.28 所示：



图 4.28 写入后台测试

之后，单签署方签名需要传到区块链上聚合，这个功能与“公平性系统”原理相同，在此就不赘述了。最后的验证阶段，输入生成的聚合签名，即可验证是否通过验证，如图 4.29 所示：

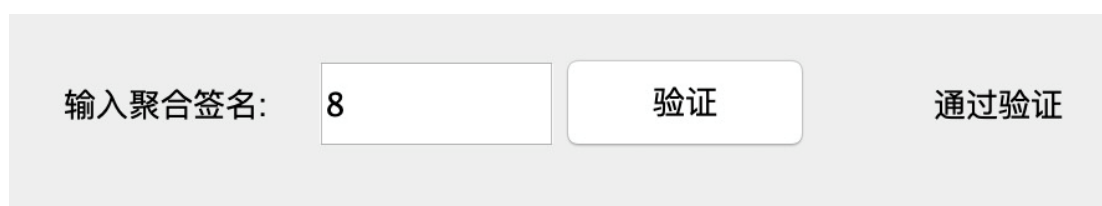


图 4.29 验证测试

4.6 本章小结

本章是整篇论文的最后一个部分，也是最重要的一个部分。本章结合前三章所介绍的知识与各类工具，详细的介绍了如何依据理论知识设计并实现全系统。本章首先仔细分析了第三章中提出的需求，之后根据需求编写代码，考量了前端、后端以及交互部分的不同功能如何实现，最后成功制作了公平性系统和隐私性系统。完成了全系统的基本功能，同时进行测试，保证运行达到设计时的预期。

5. 成果总结与展望

5.1 成果总结

本次研究成果主要包括：

1. 利用短短几周时间学习了区块链相关知识，尤其是以太坊区块链的相关知识性质。
2. 学习了一些此前未接触过的密码学算法，通过查阅资料以及推导公式，将这些密码学算法成果应用到区块链上。
3. 根据教程，成功设计并实现了一个基于私链的 Dapp。编程完成包括前端、后端以及交互模块的所有内容，并且均为原创，工作量达到了毕业生应有的水准。
4. 通过撰写论文，对自己的毕业设计进行了讲述、总结、归纳。

5.2 研究展望

虽然我成功的完成了毕设提出的要求，不过我也发现了本次设计中很多不完善的方面，希望在以后的学术生涯中，利用更多的知识、经验解决这些问题：

1. 本 Dapp 此次仅设计了 Web 客户端，事实上，Dapp 也可以在 IOS 系统或 Android 系统上部署、使用。以后如果掌握了移动端开发的相关技能，希望可以完成移动端不同系统的 Dapp 开发，真正把 Dapp 代入人们的日常生活，使得现实世界中的合同签署系统更加方便
2. 本 Dapp 虽然能满足多方用户的签署，但目前还是只能做到在系统初始化时固定签署方数量，真实情况下，合同签署应该可以自定义签署方数量。希望以后可以学习更高级的 Dapp 开发技能，以便添加更多附加功能。

结论

智能化生活的脚步已经不再是“越来越近”了，我们现在就正在步入智能化时代。云计算、5G、人工智能、区块链等等新兴技术的兴起，正在深刻的改变人们的生活模式和思维模式。本设计此次解决的传统的合同签署问题，是现有生活模式下的一个典型问题，也是传统技术出现一些弊端的一个缩影：技术总是在进步的，正式新技术的不断推进，就技术带来的问题才得以解决，人类的科技、社会才能不断发展。

本论文中提出并设计的区块链多方合同签署系统，就是利用了目前的新兴技术：区块链。通过查阅资料，分析特性，我了解到区块链拥有共识算法机制、难以篡改等等特性，因此想到利用区块链来解决合同签署问题。最终设计出了基于离散对数算法的公平性系统和基于盲化的隐私性系统这两个系统，它们分别解决了合同签署中的公平、溯源、追责问题以及隐私保护、匿名的需求。

此外，本设计的创新之处就在于，我将设计移植到了区块链上，把系统设计成一个 Dapp。如果只是在网站上重复的把算法复刻一遍，这就失去了毕业设计的创新性，完全变成软件课程的大作业了。本设计大胆尝试了目前少有人实践的合同类 Dapp，巧妙利用区块链的特性解决实际问题，是一次在区块链领域应用范畴上的大胆尝试。

参考文献

- [1]林冠宏. 区块链以太坊 DApp 开发实战[M]. 清华大学出版社:北京, 2019:15.
- [2]高莹, 吴进喜. 基于区块链的高效公平多方合同签署协议[J]. 密码学报, 2018, 5(5):556-567.
- [3]Antoine Joux.A One Round Protocol for Tripartite Diffie-Hellman[J].Journal of Cryptology,2004,17:263-276.
- [4]Zuhua Shao.Fair exchange protocol of signatures based on aggregate signatures[J].Computer Communications,2008,31:1961-1969.
- [5]Duc-Phong Le,Guomin Yang,Ali Ghorbani.A New Multisignature Scheme with Public Key Aggregation for Blockchain[C].2019 17th International Conference on Privacy, Security and Trust (PST).Fredericton, NB, Canada:IEEE Xplore,2019.26-28.
- [6]杨青, 张惠玲, 吴成晶. 可证安全的广播多重盲签名方案[J]. 计算机与数字工程, 2017, 333:1357-1359.
- [7]唐春明, 高隆. 区块链系统下的多方密钥协商协议[J]. 信息安全, 2017(12):17-21.
- [8]Angelo De Caro.The Java Pairing Based Cryptography Library (JPBC)[EB/OL].<http://gas.dia.unisa.it/projects/jpbc/>,2013-12-04.

致谢

毕业设计既是大学学术生涯的终点站，又是未来学术道路上的新起点。经历了四年的学习、磨练，毕业设计对于我来说，是一次对于过去学习效果的检验，也是一个展示自己学术水平的舞台。

首先，最应该感谢的就是我的毕业设计指导教师：周艺华老师。早在选题之前，周老师便与我进行了沟通，为我详细的介绍了毕业设计的相关知识和课题难度。在毕业设计正式开始之后，周老师更是全程辅导，为我提供了丰富的研究思路，热情、及时的回答我提出的问题。每周的线上指导，周老师都认真的检查我的进度，并为我提出建设性意见。在疫情这一特殊时期，多亏了周老师认真、负责的教导，我才能按时并保质的完成毕业设计。

其次，我要感谢每一位曾教导过我的老师。毕业设计的完成，离不开过去每门专业课的知识，感谢老师们的教导为我打下良好的计算机基础，让我能够更好、更快的上手毕业设计。

当然，应该被感谢的还有我的各位同学们。在四年的学习生活中，我们一起上课、学习、玩耍、竞赛。值得铭记的不仅有同吃同住，打闹嬉戏的情景；更有共摘桂冠，为校争光的瞬间。这些都将成为我一生难以忘却的美好记忆。

本科生涯即将画上句号，完成毕业设计的过程中，每当遇到难点或难以突破的瓶颈时，我总是会回想起北京工业大学的校训“不息为体，日新为道”，我们即使离开学校，在自己未来的岗位上也要永不放弃、有所追求、积极创新，成为对社会有用的人才。

最后，由衷感谢陪伴我度过本科生涯最后一次展示的答辩组老师们，您们辛苦了！