班　级　20018031

学　号　20009201080

西安电子科技大学

本科毕业设计论文

题　　目　面向区块链应用的分布式密钥管理机制

学　　院　网络与信息安全学院

专　　业　网络空间安全

学生姓名　谢一飞

导师姓名　马立川

摘 要

随着区块链技术的快速发展，基于区块链与智能合约的分布式应用已经广泛投入使用。然而，传统的非对称式密码技术通常基于某个中心化服务器，这与区块链去中心化的要求背道而驰。本文研究了如何在没有可信第三方参与的情况下，能够在多方参与者写作运行的情况下共同对某一个密钥进行管理。

本文首先介绍了区块链与智能合约的基本概念，以及秘密共享的研究现状。接着，详细阐述了现有的数种秘密共享方案的原理和方法，特别是 Shamir 秘密共享和 Pedersen 可验证秘密共享的协议内容。在此基础上，本文提出了一种面向区块链应用的分布式密钥管理机制，该机制主要包含了三个模块：在分布式群体间共同决定密钥、单一密钥分片的恢复和在不同参与者群体间传递密钥，随后在实现上述三个模块的基础上，结合智能合约，对参与上述三个模块功能的用户进行确认，确保用户间能够达成满足拜占庭容错的共识。此外，本文还探讨了如何使用智能合约来实现密钥管理，包括基于区块链的广播及验证，以及如何在用户间形成拜占庭共识。

最后，本文通过实验验证了所提出方案的有效性。实验结果表明，本文提出的分布式密钥管理机制能够有效地在分布式系统中生成和管理密钥，且各项功能的开销均在可接受的范围内。

关键词：**区块链　　智能合约　　分布式密钥管理　　拜占庭容错　　秘密共享**

ABSTRACT

With the rapid development of blockchain technology, distributed applications based on blockchain and smart contracts have been widely adopted. However, traditional asymmetric cryptography typically relies on a centralized server, which contradicts the decentralized nature of blockchain. This paper explores how to manage a key jointly among multiple participants without the involvement of a trusted third party in a decentralized setting.

Firstly, this paper introduces the basic concepts of blockchain and smart contracts, as well as the current state of research on secret sharing. It then details several existing secret sharing schemes, specifically the principles and methods of Shamir's Secret Sharing and Pedersen's Verifiable Secret Sharing protocols. Building on this foundation, the paper proposes a distributed key management mechanism for blockchain applications. This mechanism primarily includes three modules: jointly determining a key among a distributed group, recovering a single key share, and transferring the key among different participant groups. Subsequently, based on the implementation of these three modules, it combines smart contracts to confirm the users participating in these modules, ensuring that users can reach Byzantine fault-tolerant consensus. Additionally, the paper discusses how to use smart contracts for key management, including blockchain-based broadcasting and verification, and how to form Byzantine consensus among users.

Finally, the paper verifies the effectiveness of the proposed scheme through experiments. The experimental results demonstrate that the proposed distributed key management mechanism can efficiently generate and manage keys in a distributed system, with the costs of various functions remaining within acceptable limits.

Keywords: Blockchain Smart Contract Distributed Key Management Byzantine Fault Tolerance Share Secret

目 录

第一章　绪论

1.1　研究背景

随着区块链技术的发展，大量基于区块链与智能合约的分布式应用已经广泛的被投入使用。然而，传统的非对称式密码技术通常会基于某个中心化的服务器，这与区块链去中心化的要求背道而驰。中心化的处理方式缺乏透明度，随着参与者的增加，中心服务器的性能压力较大，且存在单点故障的风险。如何应对这个问题，成为了区块链技术需要面对的新的挑战。

采用分布式密钥生成（Distributed Key Generation）可以用于解决中心化密钥受到的挑战。分布式密钥生成技术通常基于可验证秘密共享技术[1]进行。它的主要研究内容是如何在不存在可信第三方参与的情况下，由多个参与者共同计算并获得一个密钥，这个密钥无法被任何拥有单一密钥分片的参与者得知，只有收集到足够多的密钥分片，才能够获取生成的密钥。在进行密钥生成时，令多方参与者使用相同的可验证秘密共享技术分享属于自己的秘密分片，并共同决定一个用于生成最终密钥的多项式，从而做到各方参与者公平的共同决定生成的密钥内容。分布式密钥生成中利用了多方计算中密钥共享的概念、秘钥共享将密钥分割成多个部分，每个参与方只持有其中的一部分，需要达到一定阈值才能重构出完整的密钥。而多方计算使得多个参与方能够共同执行计算任务，而不泄露私有输入内容。分布式密钥生成技术在需要多方参与或分布式信任的领域得到应用，如共识协议，分布式存储系统，门限密码学等。其关于密钥的各个过程不需要可信第三方参与，规避了传统密码技术中中心化服务器带来的缺点。

由于分布式系统本身带来的限制，需要引入智能合约技术[2]应对拜占庭将军问题[3]。这是一种自动执行的合约，其合约条款直接写入代码中，当合约触发的条件被满足时，合约将会被自动执行。当前基于区块链的智能合约由于区块链的特性，通常都是可追踪，不可逆且透明的，允许在没有第三方的情况下进行可信任的交易。通过这些前提条件，智能合约可以通过分布式共识算法，在系统中存在恶意节点的情况下满足拜占庭容错（Byzantian Fault Tolerance, 下称 BFT）。一般情况下，可以采用引入冗余信息的方法做做到 BFT，即让多个节点发送相同的信息。

通过对比不同节点发送的信息，可以发现并排除恶意节点发送的错误信息；还可以采用投票机制，让节点对行动计划进行投票，只有达到一定比例的赞成票才能通过行动计划；此外，也可以采用实用拜占庭容错算法 (PBFT)[4]解决拜占庭将军问题

1.2 研究现状

1.2.1 区块链与智能合约

区块链 (Blockchain) 是一种分布式账本，由成为区块的记录组成，这些区块用于记录多台计算机上的交易，通过加密哈希安全的链接在一起。由于分布式存储的特性，任何涉及的区块均无法追溯性的更改。2008 年，Nakamoto Satoshi 提出了第一个区块链以及第一种基于区块链的虚拟货币比特币[5]，随后在 2013 年，根据 Vitalik Buterin 发表的论文[6]，以太坊横空出世。相较于比特币，以太坊允许开发者创建智能合约和去中心化应用，并可以构建投票功能，形成 “去中心化自治组织”。

1996 年，Nick Szabo 使用 “智能合约” 一词指代使用物理资产（如硬件或软件系统）而非法律自动执行的合同。其旨在使合同能够根据约定好的条款自动执行、控制或记录事件操作，目标是降低对于可信第三方的需求、降低仲裁成本和欺诈损失、减少出现的恶意和意外异常。Vitalik Buterin 在以太坊白皮书[6]中提出了一种基于 Solidity 语言的智能合约，相较于 Szabo 提出的较弱版本，其具有图灵完备性。现在大多数加密货币均能够支持在不受信任的各方之间简历更加高级的智能合约。

在基于区块链的智能合约中，参与方认可的合同条款会被转换为可行的计算机程序，条款之间的逻辑联系也被保存在程序中，每个合约语句的执行均会以不可变交易的形式被区块链记录。开发者可以为合约中的每个函数配置访问权限，一旦满足智能合约中的某个条件，触发的语句将会自动执行对应的函数。例如，Alice 和 Bob 约定了一项合同，其中违约金的部分规定当某方违反合同时需要支付 10ETH。当 Bob 触发违约条件时，合约将自动从 Bob 的账户中划转预留的 ETH 到 Alice 的账户中，并将转移记录在案[7]。在以太坊中，编译好的智能合约程序可以被部署到

区块链上，并被以太坊虚拟机执行。

1.2.2 秘密共享

秘密共享（Secret Sharing）是指在群体中分配秘密的方法。1979 年，Adi Shamir 和 George Blakley 独立提出了秘密共享的概念**ShareSecret**[8]。要进行秘密共享，需要使得任意个人均无法掌握关于该秘密的任何可理解信息，但当足够多的参与方将他们的份额重新结合时，该秘密将能够被重建。不安全的秘密共享允许攻击者在每次共享中获取更多信息，而安全的信息共享则是 “all or nothing”，即要么拥有必要数量的共享恢复秘密，要么无法获取任何关于秘密的信息。目前主流的秘密共享方案有：布尔共享、算术共享、Yao 共享和 Shamir 共享[9]。

为了在 BFT-SMR（复制状态机）中保证数据的机密性，存在多种解决方案。直接以加密形式存储数据的问题在于：如果数据在多个用户间共享，则需要密钥分发的服务；当密钥分发由不可信第三方提供时，则需要其它的去中心化 T3P 服务，而这又需要另一个可信第三方的参与。这种情况下，问题会被递归的传递给下一个可信第三方，无法被简单解决。

为了在满足 BFT 的前提下保证数据的机密性，存在多种解决方案如：利用门限加密技术的秘密共享、可验证的秘密共享（VSS）[1]、公开可验证秘密共享（PVSS）[10]、动态主动秘密共享（DPSS）[11]。

可验证的秘密共享（VSS），确保了即使某方参与者是恶意的，也必须存在一个可以明确定义的秘密（遵循协议定义），其它参与者稍后仍然能够恢复秘密值[12]。它添加了一个验证步骤，通过让密钥分发者公布一个与生成秘密部分相绑定的多项式系数的承诺[1]，随后令各方参与者使用他们的部分和公开承诺来验证他们的部分是否正确。类似的方法已经被用于一些面向区块链的工作，例如 KZG-VSSR[13],CALYPSO[14],CHURP[15],FaB-DPSS[16]等。

公开可验证秘密共享（PVSS）方案[10][17]，通过定期更新副本存储份额而不泄露秘密的方式防范自适应的移动敌手。在 PVSS 中，不仅参与者可以验证他们自己的份额，而且任何人都可以验证参与者收到的份额是否正确。这就是说，PVSS 增加了公开验证的功能，使得任何一方（不仅仅是协议的参与者）都可以验证交易

商分配的份额的合法性。

动态主动秘密共享（DPSS），则在主动秘密共享的基础上增加了对份额动态更新的机制。在 DPSS 中，每个参与者的密钥分片可以周期性地动态改变，而统一的原始密钥不受影响。这样每次更换密钥分片后，攻击者在上一个周期内所获得的信息完全失效。如果副本被清理并重新启动[18][19][20]，那么即使敌手收集了足够的份额，由于这些份额会过期，而过期的份额无法与续期的份额组合恢复出秘密。

1.3 本文研究任务与章节安排

本文旨在研究如何高效的实现面向区块链的分布式密钥管理机制。在本文的方案中，由于分布式系统的参与者经常会发生更换，且单一密钥分片如果丢失难以恢复，因此结合现有的 DPSS 方案，形成一套更加安全且适用于区块链应用的分布式密钥管理机制，实现了一套智能合约用于满足区块链应用对于分布式密钥管理的要求, 并针对整体分布式密钥管理机制进行了测试。

全文共分为六个章节, 具体各章节内容简介如下：

第一章：绪论，介绍分布式密钥管理机制的研究背景，并简介区块链和秘密共享的研究现状，引出本文重点在于如何实现一种基于智能合约技术确保拜占庭容错的分布式密钥管理机制，最后介绍了各个章节的内容安排。

第二章：预备知识，介绍 Shamir 秘密共享的原理以及基于其衍生的可验证秘密共享的原理，并介绍了拜占庭容错相关的知识

第三章：面向区块链应用的分布式密钥管理机制，介绍了本文提出的面向区块链应用的分布式密钥管理机制的系统构成和三个主要模块的功能

第四章：方案的实现与评估，对本文提出的密钥管理机制使用代码进行实现，并进行程序上的测试和开销分析。

第五章：总结与展望，对本文的研究工作做概括性总结并对未来相关工作进行展望。

1.4　本章小结

在本章中首先对研究背景进行介绍，并由此引出本文所研究的主题，面向区块链应用的分布式密钥管理机制。在第二部分介绍了国内外对于区块链领域和秘密共享的各种研究方向和研究现状。最后概括了本文的主要研究任务以及本文各个章节的大致介绍。

第二章　预备知识

2.1　秘密共享

2.1.1　Shamir 秘密共享

在 Shamir 秘密共享协议[9]中，各个参与方拥有的秘密分片可以用 $\langle i, s_i \rangle$ 表示，其中 i 是参与者的 id,s_i 是该参与者拥有的秘密分片。秘密分发者需要选取某个秘密多项式

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$$

其中 $f(0) = a_0 = s_0$ 是要分享的原秘密值。其它参与者的秘密分片可以通过计算 $s_i = f(i) = \sum_{k=0}^{n} a_k x^k \bmod p$ 进行计算。当需要恢复秘密时，则需要收集 $t+1 \geq n$ 个用户的秘密分片，对其使用拉格朗日插值法恢复出原多项式 f，并计算 $s_0 = f(0) = a_0$, 得到的结果即为原多项式的值。

2.1.2　拉格朗日插值多项式

拉格朗日插值法是一种通过已知数据点来构造插值多项式的方法，拉格朗日多项式是由拉格朗日插值法决定的多项式，这个多项式可以通过一系列已知的点 (x_i, y_i) 中获取一个多项式 $P(x)$，对这些点有 $P(x_i) = y_i$。

假设存在 $n+1$ 个取值点 $(x_0, y_0), (x_1, y_1) \cdots (x_n, y_n)$，假设任意两个点的 x_j 值均不相同，那么拉格朗日插值法有插值多项式

$$P(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{2-1}$$

其中 $L_i(x)$ 是拉格朗日基函数，有

$$L_i(x) = \prod_{0 \leq i \leq j, i \neq j} \frac{x - x_i}{x_j - x_i} \tag{2-2}$$

由于拉格朗日基函数的特点是 $x = x_i$ 时 $L_i(x) = 1$，$x = x_j$ 且 $j \neq i$ 时，$L_i(x) = 0$，这样 $P(x)$ 在每一个数据点 x_i 处均能够保证有 $P(x_i) = y_i$。

在拉格朗日插值法中，需要 $n+1$ 个点才能够确定一个多项式，且这个多项式是唯一的。而在此前提到的 Shamir 秘密共享算法中，秘密信息隐藏在多项式的常数项中，这样就保证了任何人如果不能收集到足够多的分享，就无法通过插值形式恢复出这个唯一多项式，也就无法找到秘密信息。

2.2 可验证秘密共享

2.2.1 多项式承诺的原理

为了保证共享的秘密不被篡改，Feldman 在 Shamir 秘密共享的基础上添加了对秘密分片的验证机制。验证的原理在于，由于指数运算存在 $g^{a+b}=g^a g^b$，因此用户 P_i 收到的分片 $s_i=f(i)$ 有

$$g^{s_i}=g^{f(i)}=g^{a_0+a_1 i+a_2 i^2+a_3 i^3+\ldots+a_k i^n}=g^{a_0}g^{a_1 i}g^{a_2 i^2}g^{a_3 i^3}\cdots g^{a_k i^n} \bmod p \tag{2-3}$$

而对于多项式的承诺值，有 $A_k=g^{a_k}\rightarrow (A_k)^{i^k}=(g^{a_k})^{i^k}=g^{a_k i^k}$ 因此，将验证承诺的等式2-5右侧展开，有

$$\prod_{k=0}^{n}((A_k)^{i^k})=\prod_{k=0}^{t}((g^{a_k})^{i^k})=g^{a_0}g^{a_1 i}g^{a_2 i^2}g^{a_3 i^3}\cdots g^{a_k i^n} \bmod p \tag{2-4}$$

即当2-3与2-4相等时，说明收到的秘密分片正确。

这种验证机制的核心在于多项式的同构特性：指数的乘积形式与多项式的加法形式同构，以及离散对数问题的困难性。离散对数的困难性保证密钥无法从公开的广播值中反向获取，而指数的乘积形式同构则能够保证分发的密钥分片受到篡改后无法通过公开广播值成功验证。具体来说，公开验证值 $\{A_0,A_1,\cdots,A_{t-1}\}$ 是基于生成元 g 多项式系数 a_i 的指数值。每个参与者可以独立计算并验证 g^{s_i} 是否等于 $\prod k=0^{t1}(A_k)^{i^k} mod p \prod_{k=0}^{t-1}(A_k)^{i^k} \bmod p$。由于指数运算的同态性质，这种验证等价于确认多项式计算的正确性。篡改任何一个分片 s_i 都会破坏这种一致性，因为任何试图伪造验证值的行为都会破坏验证等式的成立性。离散对数问题在大多数有限域上是计算上难解的，这进一步确保了篡改验证值或分片以使验证通过几乎是不可能的，从而保证了分片的完整性和安全性。

2.2.2　Feldman 可验证秘密共享

由于 Shamir 提出的秘密共享方案并没有对秘密进行验证，因此恢复秘密时恶意参与者可以通过修改自身秘密分片使得最终恢复出的秘密与原秘密不同，甚至能够任意的使恢复的秘密进行偏移。为此，Paul Feldman 在 Shamir 秘密共享的基础上，添加了基于离散对数问题的困难性的对原秘密多项式的承诺[1]，保证参与者可以确定他们获得的秘密分片未被篡改，恢复秘密时也可以保证不诚实参与者的秘密分片被剔除，只要保证有 $t+1$ 个诚实参与者，即可保证秘密被正确恢复。

为了保证秘密分片不被篡改，恢复的多项式不发生变化，秘密分发者需要选取大素数 p、q，$p=2q+1$ 以及 p 的一个生成元 g，并在 Z_q 的生成群中选择系数用于生成秘密多项式。对秘密多项式的每一个系数 a_k 计算 $A_k = g_k^a \bmod p, k = 0 \ldots n$ 作为对多项式的承诺，并将其广播到所有参与者处。这样，秘密分片分发后，各方参与者就可以通过计算

$$g^{s_i} = \prod_{k=0}^{t} (A_k)^{i^k} \bmod p \tag{2-5}$$

是否成立来在不暴露原秘密多项式的前提下判断秘密分片是否被篡改。由于广播的承诺值是直接计算多项式系数和大素数生成元 g 的指数值，这导致承诺本身会泄露一部分原多项式的信息，只是由于离散对数问题的困难性，即使是承诺的创建者也无法更改已经承诺的秘密，恶意参与者也很难从公布的承诺中直接恢复出秘密的内容。

2.2.3　Pedersen 可验证秘密共享

对于秘密共享来说，Feldman 的可验证秘密共享已经可以做到秘密难以泄露且不被篡改，足够安全。但对于分布式密钥生成来说，仅仅使用 Feldman 的可验证秘密共享并不足以保证最终生成密钥的安全性，恶意参与者可以通过某些方式获得最终密钥的信息，具体方法会在本文 3.1.1 节提及。因此我们需要一种能够保证秘密信息完全不泄露的可验证秘密共享协议。

为了保证共享的秘密一定无法暴露，Torben Pedersen 提出了一种可验证秘密共享方案[21]。Pedersen 提出的可验证秘密共享在 Feldman 方案的基础上添加了另

一个辅助多项式

$$f'(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$$

这个多项式系数的选取方式与 Feldman 的可验证秘密共享相同。在分发秘密时，秘密分发者会向其他参与者分别发布 $\langle i, s_i, s'_i \rangle$，其中 $s_i = f(i), s'_i = f'(i)$。在承诺时，需要选择 p 的另一个生成元 h 用于计算对 $f'(x)$ 的承诺 $C_k = g^{a_k} h^{b_k} mod p, k = 0 \ldots n$，并广播给所有参与者。各方参与者在获取秘密分片或需要恢复原秘密时就可以通过计算

$$g^{s_i} h^{s'_i} = \prod_{k=0}^{t} (C_k)^{i^k} \bmod p \tag{2-6}$$

是否成立来判断秘密分片的正确性。与 Feldman VSS 先比，Pedersen VSS 通过引入另一个多项式的方式，保证了承诺能够完美的将原多项式的信息隐藏——g 和 h 的离散对数关系是未知的，承诺中 h 的指数部分事实上是一系列随机数，对秘密多项式没有实际意义，能够在不暴露任何信息的条件下保证秘密分片可以正常被确认。

2.3 拜占庭容错

2.3.1 拜占庭将军问题

拜占庭将军问题[3]最早于 1982 年由 Leslie Lamport 等人提出。问题的基本框架是：一组拜占庭将军分别各率领一支军队共同围困一座城市，各支军队分别可以做出进攻或撤退的决定；然而，由于部分将军选择进攻部分选择撤退的情形会造成严重后果，因此将军们需要以投票的方式决定是否进攻或撤退。由于将军所处城市方向不同，它们只能依靠信使联系。在投票过程中每位将军都将自己投票给进攻还是撤退的信息通过信使分别通知其他所有将军，这样一来每位将军根据自己的投票和其他所有将军送来的信息就可以知道共同的投票结果而决定行动策略。然而，由于将军中可能存在叛徒，信使也有可能被截杀。在这种情况下，如果那些未出现问题的将军仍能通过投票决定战略，便称达到了拜占庭容错。

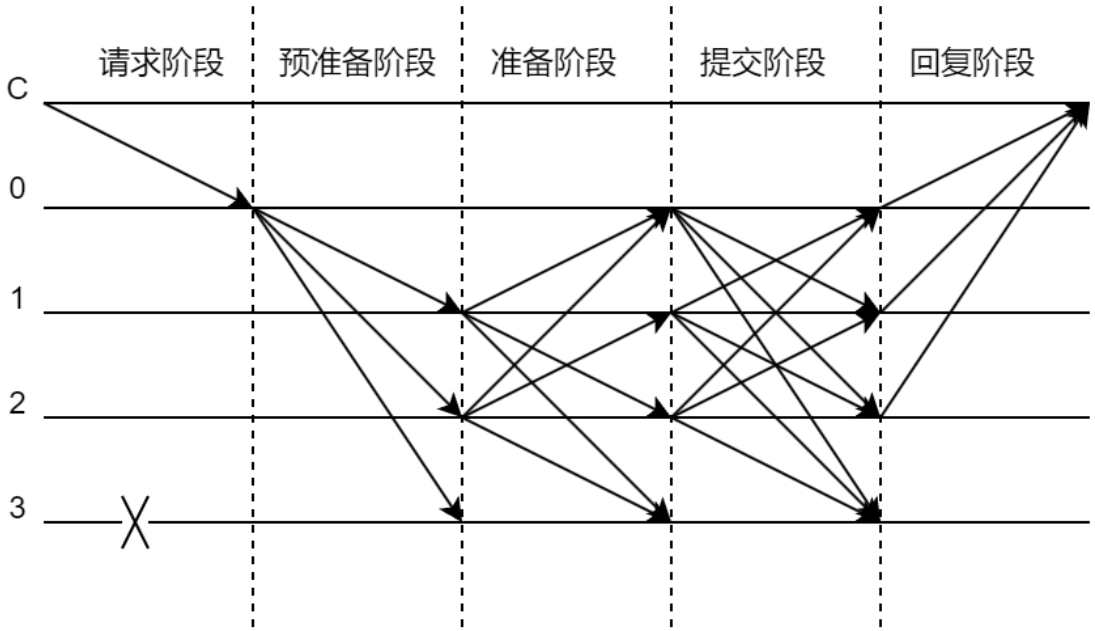

图 2.1　PBFT 算法运行过程

2.3.2　实用拜占庭容错

为了保证能够实现拜占庭容错，实用拜占庭容错算法（Practical byzantine fault tolerance，PBFT)[4]给出了一种解决方案。在 PBFT 算法中，每一方参与者都保存了服务状态的副本，也实现了客户端所有合法的请求的操作。其在分布式系统不出现问题的情况下，最多能够容忍约 $\frac{1}{3}$ 的节点发生故障。即当总共存在 n 方参与者时，整个系统最多能够容忍 $\frac{n-1}{3}$ 个恶意或故障的用户，而不影响分布式共识的正确达成。

在 PBFT 中，存在主节点 (primary) 和副本 (replica) 两种角色，任何一方参与者均可以成为主节点且两者之间可以相互转换。主节点负责协调并管理整个共识的过程，而副本节点则负责执行客户端的请求并维护系统状态的副本。整个协议过程基本上可以分为五个阶段：

1. **请求阶段**：客户端向主节点发送请求。
2. **预准备阶段**：主节点将请求广播给所有副本节点。
3. **准备阶段**：副本节点接收到预准备信息后，将准备信息广播给所有其他节点。
4. **提交阶段**：副本节点接收到足够多的准备信息后，将提交信息广播给所有其

他节点。

5. **回复阶段**：副本节点接收到足够多的提交信息后，执行请求并向客户端发送回复。

其中 2.3. 和 4. 三个部分是 PBFT 协议的三阶段共识流程。

当然，PBFT 算法同样存在一定程度的缺陷，例如如果参与者数量过多，会导致共识效率的下降；且无法防御女巫攻击，需要审核加入节点避免恶意用户用多个账户参与共识。

2.4 本章小结

在本章中首先解释了秘密共享的理念和 Shamir 秘密共享的原理以及拉格朗日插值法的运算过程。随后，在 Shamir 秘密共享的基础上介绍了如何对多项式进行承诺和验证，保证各方参与者收到的秘密分片不受到篡改，并回顾了数种可验证分布式秘密共享协议，重点介绍了可验证秘密共享是如何对用于分发秘密的多项式进行承诺和验证的原理。重点说明了能够满足在不泄露任何关于秘密信息的前提下，做到安全的可验证秘密共享的 Pedersen 的可验证秘密共享，这种协议可以用于分布式密钥生成协议。最后介绍了拜占庭问题，以及满足拜占庭容错的实用拜占庭容错算法。为引出下一章如何采用可验证秘密共享方案和拜占庭共识形成一套面向区块链应用的分布式密钥管理机制协议栈做铺垫。

第三章　面向区块链应用的分布式密钥管理

3.1　系统模型

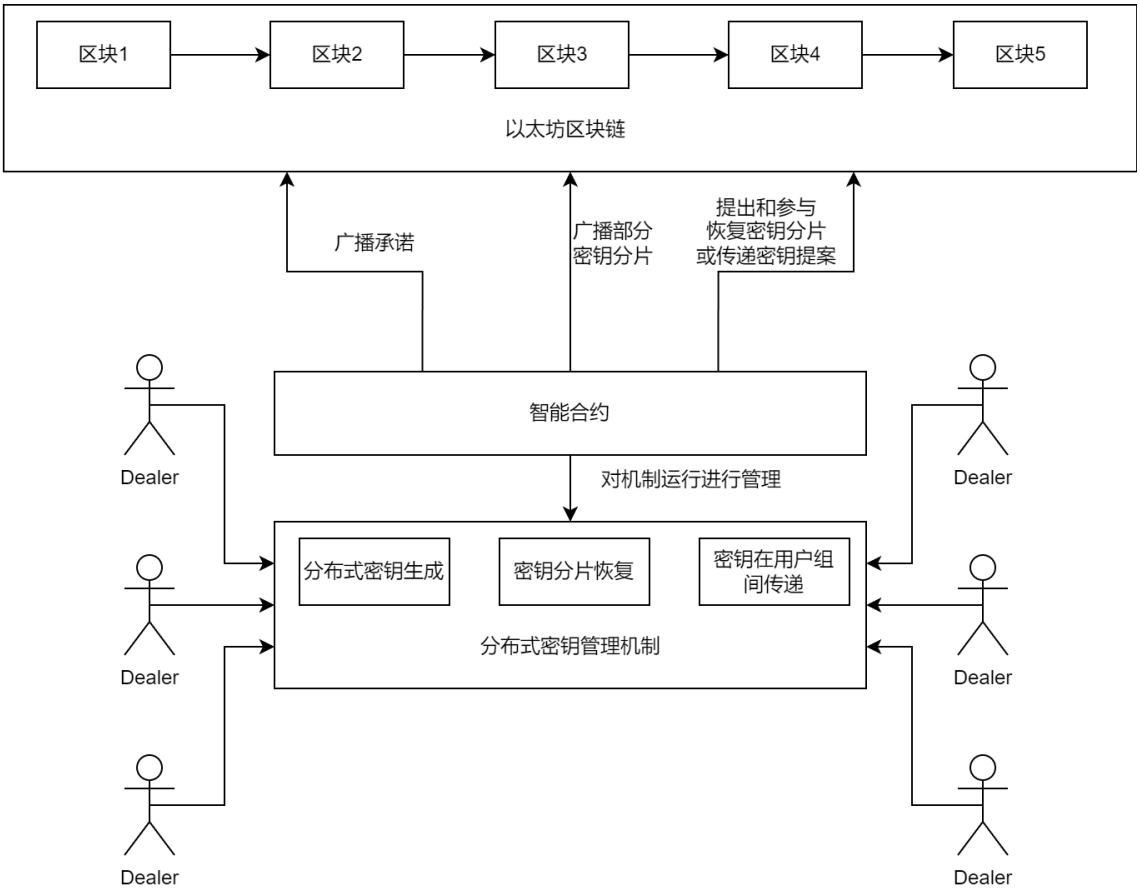

图 3.1　分布式密钥管理机制示意图

在本文中提出的分布式密钥管理机制中，整体上存在三大功能。如图3.1所示，分布式密钥管理机制存在三个主要功能：分布式密钥生成、对单一密钥分片的恢复和密钥在用户组之间传递的功能。各方用户可以参与共同生成密钥，并对密钥分片进行管理。为了在用户组间达成执行恢复密钥或传递密钥的共识，本文中提出的方案中，参与者可以使用智能合约来管理协议中各个功能的执行，将生成密钥时需要广播到其他用户的承诺信息和提出恢复密钥分片或传递密钥的提案使用智能合约提交到区块链上。由于区块链的分布式特性和不可抵赖性，参与者将能够安全的进行整个密钥管理的过程。

敌手模型：考虑存在一种概率多项式自适应敌手，它可以控制网路，并随时

腐化当前视图任意数量的客户端和部分副本。被腐化的客户端和副本可以被操纵任意偏离协议，即容易出现拜占庭错误（Byzantine failures）。这种参与者被称为腐化进程或错误进程。没有故障的被称为城市的进程。对于当前视图 V，敌手最多能够同时控制 $V.t = \left\lfloor \frac{V.n-1}{3c} \right\rfloor$ 个副本。可以认为副本在每个视图后都会清楚状态，则对手可以腐化任意离开系统的副本。对于被破坏的副本，敌手可以知道其存储的私有状态。

3.2 分布式密钥生成

为了实现面向区块链应用的分布式密钥管理，需要采用一种能够由所有诚实参与者共同决定密钥，且能够保证密钥生成过程中做到不泄露和密钥有关的信息，生成的密钥保证均匀分布。Pedersen 曾经提出过一种基于 Feldman 的可验证秘密共享的密钥生成算法[22]可供用于分布式密钥生成。但是根据论文[23]中的内容，攻击者可以在控制了多个用户的情况下，通过投诉取消自身控制的某个用户资格的方式，影响最终密钥值的分布。这样，该算法在敌手的攻击下，无法保证生成密钥的均匀分布，使得协议的保密性和正确性存在问题。

3.2.1 分布式密钥生成算法

在本节将定义本文中采用的分布式密钥生成算法。在此前提到的论文[23]中，Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk 和 Tal Rabin 提出了一种采用 Pedersen 的可验证秘密共享和 Feldman 的可验证秘密共享的分布式密钥生成协议。在协议的过程中，各方参与者首先通过 Pedersen 的可验证秘密共享协议提交一个随机多项式，常数项是他们为共同决定秘密 x 贡献的随机值 z_i，攻击者在这种情况下无法通过诚实方的提交改变 z_i 的值。随后，各方参与者运行 Feldman 的可验证秘密共享和此前提交的随机多项式 $f_i(z)$ 公开验证值 y_i，并由其它参与方验证是否正确，若某一方参与者验证不通过，诚实的参与者可以恢复出其正确的多项式 f_i。这样在诚实各方的共同参与下，协议可以安全的计算出公钥 $y = g^x \bmod p$。这样可以保证获得一个均匀分布的密钥 x 且攻击者无法从 y 本身带有的信息之外获取任何和 x 有关的信息。

算法 3.1 分布式密钥生成

生成密钥 x

1. 各方参与者 P_i 通过 Pedersen 的可验证秘密共享分享一个随机值 z_i:
   (a) P_i 在循环群 Z_q 中选取两个 t 度 $(t \leq \frac{n}{2})$ 的随机多项式 $f_i(z), f'_i(z)$

   $$f_i(z) = a_{i0}+a_{i1}x+a_{i2}x^2+\cdots+a_{it}x^t, f'_i(z) = b_{i0}+a'_{i1}x+b_{i2}x^2+\cdots+b_{it}x^t$$

   令要分享的值 $z_i = f_i(0) = a_{i0}$。并计算要分享的分片 $s_{ij} = f_i(j), s'_{ij} = f'_i(j)(j = 0\cdots n)$ 并发送给另一个参与者 P_j 同时 P_i 广播对这两个多项式的承诺 $C_{ik} = g^{a_{ik}}h^{b_{ik}} \bmod p(k = 0\cdots t)$。
   (b) P_j 收到共享后验证其是否符合

   $$g^{s_{ij}}h^{s'_{ij}} = \prod_{k=0}^{t}(C_{ik})^{j^k} \bmod p \tag{3-1}$$

   若结果不成立，则 P_j 广播对 P_i 的投诉。
   (c) P_i 若收到 P_j 的投诉，则公布 P_j 的共享 s_{ij} 和 s'_{ij}
   (d) 各方参与者若满足如下条件，则被标记为失格参与者，不参与最终的密钥运算：当 P_i 收到超过 t 次投诉 (即其多项式不可能被正确恢复), 或其公布的分享计算无法满足等式3-1
2. 诚实各方参与者建立有效参与者集合 QUAL(诚实各方参与者建立集合的最终结果一致)
3. 最终密钥 x 无法被单一参与者计算得到，但其值有 $x = \sum_{i\in QUAL} z_i \bmod p$。各方参与者将自己的份额设置为 (i, s_i, s'_i) 其中 $s_i = \sum_{j\in QUAL} s_{ji} \bmod p$，$s'_i = \sum_{j\in QUAL} s'_{ji} \bmod p$。

提取公钥 $y = g^x \bmod p$

1. 参与者 $i \in QUAL$ 通过 Feldman 的可验证秘密共享暴露 $y_i = g^{z_i} \bmod p$
   (a) 各方参与者 $P_i \in QUAL$ 广播 Feldman VSS 对多项式 $f_i(z)$ 的承诺 $A_{ik} = g^{a_{ik}} \bmod p(k = 0\cdots t)$

(b) 各个参与者验证其他人广播值是否满足

$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik}^{j^k}) \bmod p \tag{3-2}$$

若 P_i 分享的分片被 P_j 计算满足等式3-1但不满足等式3-2，P_j 通过广播秘密分片 $s_{ji}s'_{ji}$ 的方式对 P_i 发起投诉

(c) 当 P_i 收到至少一次投诉后，其它参与者通过重新收集 P_i 发送的分片的方式，通过拉格朗日插值法恢复 P_i 的秘密多项式 $f_i(z)$，z_i 和 $A_{ik} = g^{a_{ik}} \bmod p(k = 0 \cdots t)$。所有其它 QUAL 中的参与者令 $y_i = A_{i0} = g^{z_i} \bmod p$ 作为 P_i 的 y_i 值

2. 各方参与者通过计算 $y = \prod_{i \in QUAL} y_i \bmod p$ 的方式获取公钥 y 的值。

3.3 对密钥分片的恢复和传递

COBRA DPSS[24]旨在为实用拜占庭容错状态机复制（BFT SMR）提供保密层，目标是即使存在活跃对手参与 SMR 也能够防止信息在复制的过程中发生泄漏。在 COBRA 协议栈中，存在分布式多项式生成, 恢复协议和动态重分享协议三个部分。在本文中，采用其恢复和重分享协议来对参与者的密钥分片进行管理。

首先，各参与者生成一个多项式 $f(x)$ 并计算需要分发的份额，然后将这些份额分发给视图中的所有状态机。接着，收集秘密分片并运行共识算法，选出领导者用验证有效的提案建立集合 S 并提交，参与者在接收到 S 中的所有有效提案后接受该集合。最后，等待共识算法决定集合 S，并在必要时请求未收到的提案内容，最终通过选定的多项式集合求和确定最终的多项式 P。具体内容见协议3.2.

3.3.1 单一密钥分片恢复

在恢复协议中，需要恢复密钥的用户 i 会随机生成一个多项式 $f_r(x)$，其中该用户对应分片的值有 $f_r(i) = 0$。该用户使用这个多项式生成用于盲化份额的分片分发给其它参与者。当其他参与者收到后，会将自身分片与盲化分片相加并返回给用户 P_i。当用户 P_i 收集到足够分片后，即可通过插值恢复出多项式 $r_r(x)$，并计算 $r_r(i)$ 来重新获取自己的秘密分片。

协议 3.2　分布式多项式生成

输入 参与者 $r_i \in V$ 各自分别收到 (V, x) 其中 V 是所有状态机的视图，x 是要编码在多项式中的点。

1. 参与者各自生成一个多项式 $f(x)$，并计算需要分发的份额。
2. 各方参与者将运算的份额分发到 V 中。
3. 收集秘密分片并运行 **共识算法**:
   - **a** 假设 $r_l \in V$ 是共识选定的领导者，r_l 使用 $V.t+1$ 个验证有效的提案，建立集合 S 并提交。
   - **b** 假设参与者 $r_i \in V$ 接收到有效的 S 中的所有提案，则该副本接受提案集合 S。
4. 等待**共识算法**决定 S，参与者 r_i 向其它副本 r_j 请求自身未收到的提案内容。
5. 选定的多项式集合进行求和，确定最终的多项式 P。

首先，发起者向其它参与者发出秘密分片恢复的请求，并和决定一个多项式 $f_r(x)$ 用于盲化份额。随后，发起者会计算对应参与者 P_j 的盲化份额 $f_r(i)$，并分发给对应的参与者。这些参与者会将盲化份额与自身的份额 $g(j)$ 相加，得到盲化份额 $f+g$ 并发送给参与者。当参与者得到足够的份额后，会使用拉格朗日插值法恢复出多项式 $r(x) = f(x)+g(x)$。由于 $f_r(i) = 0$，那么 $r(i) = f(i)+g(i) = 0+g(i) = g(i)$，从而恢复出自身密钥分片的值。具体过程见协议3.3。

3.3.2　密钥在参与者群体间的传递

为了在不泄露密钥分片的前提下，安全的将密钥分片在群体之间进行传递，新旧双方用户分别决定一个盲化多项式 $q_{old}(x)$ 和 $q_{new}(x)$ 有 $q_{old}(0) = q_{new}(0)$，原参与者分别计算 $q_{old}(i)$ 并用于将其与自身份额相加用于盲化份额，得到份额 s_i^b。新的参与者组收到对应原参与者组发送的盲份额 s_i^b 后，计算 $s_i' = s_i^b - q_{new}(i)$ 得到新的份额。新的份额值与原份额不同，但最终恢复出的秘密与原秘密相同。

首先，原参与者群组决定一个盲化多项式 $Q(x)$ 并计算相应的盲化分片 $Q(i)$，并计算盲化份额 $R(i)+Q(i)$，其中 $R(i)$ 是参与者持有的份额。这样，如果收集盲化后的分片并进行插值，我们将能够恢复出一个多项式 $(R+Q)(x)$。随后，原参

协议 3.3 秘密分片恢复

发起者 $r_k \in V$

1. 向其他参与者发起秘密分片恢复请求。
2. 等待其它用户回应。

if 收到盲化份额及承诺 **then**

对收到的份额进行验证。

份额数目足够时重建多项式并提取自身的份额。

else if 收到确认 r_j 份额的请求 **then**

将确认 r_j 请求及请求发起者 r_k 的信息 $\langle r_j, r_l, s\rangle$ 发送给 $r_i \in V\backslash\{r_j r_l\}$，并重新发起恢复请求。

end if

其他参与者 $r_i \in V\backslash\{r_k\}$

1. 收到恢复请求，运行协议3.2，共同决定盲化多项式。
2. 若参与副本拥有一个正在恢复的共享，验证该盲化分片 s_i^r。

if 分片验证通过 **then**

计算盲化后分片 $s_i^b = s_i + s_i^r$ 并发送给恢复发起方。

else if 分片验证不通过 **then**

识别该请求来自哪个参与者 r_j，标记为无效请求。

将对无效请求的确认请求返回给 r_k。

end if

收到确认请求 r_i 验证收到 $\langle r_j, r_l, s\rangle$ 的份额是否正确。

if r_i 份额通过验证 **then**

将 r_j 加入忽略列表。

else if r_i 份额未通过验证 **then**

将 r_i 加入忽略列表。

end if

与者群组会将分片发送给对应的新参与者。在新参与者群体中 (其中可能存在原先的参与者)，同样决定一个多项式 $Q'(x)$，有 $Q(0) = Q'(0)$。当收到分片 $(R+Q)(i)$ 后，新的参与者计算 $(R+Q)(i) - Q'(i)$，这样会得到一个新的密钥分片，通过收

集这个分片可以恢复出一个新的多项式 $R'(x) = (R + Q - Q')$。由于 $Q(0) = Q'(0)$，因此会有 $R(0) = R'(0)$，即密钥的值被安全的传递到了新的参与者群组中。具体过程见协议3.4。

协议 3.4　动态重分享

原参与者群组　各个参与者收到信息 (V_{old}, V_{new}, c)。

1. 调用协议3.2生成转移多项式 Q，并计算盲化用份额 s_i^q。
2. 验证新群组 V_{new} 是否为真。若为真，计算 V_{new} 中用户的盲化份额 $s_i^b = s_i + s_i^q$，并对份额进行承诺。
3. 删除份额 s_i 及承诺。

新参与者群组　各个参与者收到信息 (V_{old}, V_{new}, c)。

1. 调用协议3.2生成转移多项式 Q'，并计算份额 $s_i^{q'}$。
2. 如果验证份额为假，运行协议3.3获取 Q' 的有效份额。
3. 等待接收盲化份额 s_i^b 及其承诺，并验证其是否正确。
4. 从所获得的份额中提取 $s'_i = s_i^b - s_i^{q'}$

3.4　使用智能合约实现密钥管理

在前文所述的离线阶段中，各方参与者之间可以相互传递秘密分片以及参与对密钥分片的恢复及对秘密重分享的过程。而是否要运行这个协议需要各方参与者形成共识，如何满足拜占庭容错，达成拜占庭共识是一个分布式密钥管理机制需要应对的问题。在本文的方案中，为了让各方参与者一致运行某个协议，采用基于以太坊的智能合约来保证能够在用户间形成拜占庭共识。在基于拜占庭共识的区块链网络中，智能合约的运行结果能够收到共识算法的验证，且恶意参与者无法篡改用户上传到区块链上的数据。这样，智能合约可以有效保证各方参与者公布到链上的承诺不被篡改，且当用户达成共识时启动运行前文中提到的协议3.2、协议3.3、协议3.4。

3.4.1 基于区块链的广播及验证

在密钥生成阶段，各方参与者需要将自身对多项式的承诺 C_{ik} 广播到所有参与者中。为了保证参与者能够获取到不被篡改的广播内容，如对多项式的承诺的内容、收到投诉后公布的密钥分片的内容等，在本方案中会将这些广播数据存储到区块链上。由于区块链上的信息具有不可篡改性，且所有参与区块链的用户均可以读取区块链中任意一个区块的信息，因此将这些内容上传到区块链上可以有效保证这些数据的正确性。但是在区块链，尤其是共有区块链上存储大量数据，需要的交易次数相对较多，会产生相对较高的 gas 费用。因此，本方案在广播阶段，各方参与者通过网络直接向其它参与者发送广播内容，仅在区块链上存储广播内容的哈希值，从而降低存储数据带来的消耗。

参与者在生成密钥阶段，会将广播承诺值的哈希值通过事件的方式存储到区块链上，并能够由其它参与者接收。其它参与者若发现哈希值与收到的广播值的哈希值不同，则同样通过合约事件的方式，将需要抱怨参与者的地址以及密钥分片通过合约存储到区块链上。当收到投诉时，参与者同样通过合约事件广播对应密钥分片哈希值的方式应对投诉。具体过程见3.5。

合约 3.5 广播验证

发起者: 发起者地址，广播数据的哈希值

参与者: 确认哈希值是否正确，广播抱怨

发起者通过智能合约事件将数据存储到区块链上

参与者监听到事件被触发

if 参与者发现哈希值与广播值不符 **then**

通过广播分片的方式提出投诉

end if

发起者收到投诉

通过广播密钥分片的形式应对投诉

3.4.2 提案的发起和参与

在进行密钥分片的恢复和动态重分享时，同样需要使用智能合约来进行管理，使得参与者能够达成是否恢复分片或更新委员会参与者的共识。当用户由于某种

原因丢失了他们的密钥分片时，他们可以向智能合约发出请求，请求其他用户协助恢复其密钥。随后，其余用户可以选择是否参与协助恢复该用户的密钥分片，当选择参与的用户数目达到 t 时，该参与者将会能够恢复他遭到破坏的分片。

另外，当需要更换密钥管理委员会的成员时，可以通过在区块链上进行选举新的委员会成员来实现。区块链技术的去中心化特性保证了选举过程的公平性和透明性，从而确保了委员会成员的合法性和信任度。一旦新的委员会成员产生，新旧两批参与者可以通过运行动态重分享协议的方式将密钥安全的进行传递。

发起者会将自身的提案通过网络发送给其它参与者，并将提案的哈希值通过智能合约的方式存储到区块链上。此外，传递到区块链上的还有参与者地址的列表，用于保证合约不会让非参与者投票。随后，各方参与者会对提案进行投票表决。一段时间后，提案的发起者可以查看提案结果，若超过了秘密门限数目的参与者同意提案，那么他们可以共同执行恢复或重分享协议。最后，提案被自动销毁。具体过程见3.6。

合约 3.6　进行提案

发起者: 发起者地址，提案的哈希值

参与者: 进行投票

发起者将提案信息发送到区块链上

参与者监听到提案信息

参与者通过合约对提案进行投票

发起者统计提案结果

if 赞成票 $\geq$ 门限数 **then**

　提案通过，执行协议3.3或协议3.4

else if 赞成票 $\leq$ 门限数 **then**

　提案不通过

end if

投票关闭，合约被销毁

3.5 本章小结

在本章中介绍了分布式密钥管理机制的整体构成。首先介绍了生成密钥采用的分布式密钥生成算法和对参与者密钥分片管理所需要进行的操作，并重点描述了各个协议的运行过程。随后介绍了如何使用智能合约公开且安全对密钥进行承诺和处理对密钥分片的投诉，这样能够保证密钥的管理安全、透明且不可篡改。这些步骤构成了一个完整的分布式密钥管理机制，能够保障保密钥生成和管理的安全性、可靠性和可追溯性。为下一章节对方案进行实施做准备。

第四章　方案的评估与实现

为了对本文中提出的密钥管理机制进行测试，本文针对第三节中提出的分布式密钥生成过程以及对密钥分片的恢复和重分享过程以及各项算法细节，采用 Python 语言进行模拟，并引入了基于 python 的 gmpy2 库对密钥所需要的大数进行高精度的数学运算。此外，关于智能合约的部分，采用在线 IDE 对所写合约进行简单测试。

本章节将介绍方案的实施环境与代码方案，并分析其所需要消耗的时间开销以及合约在区块链上广播数据所需要的消耗。由于条件限制，无法满足实际或采用虚拟机的方式部署一个分布式计算环境，因此分布式密钥管理的主要计算阶段采用代码模拟多参与者在分布式密钥管理方案中的行为实现，以此为依据判断单一用户在参与整体密钥管理活动中所需要的各种开销。

4.1　实验环境与实施方案

4.1.1　实验实施环境

本文的方案在 Windows10 环境下使用 Python3.10[25]语言实现。使用 solidity[26]编写了相应的智能合约，并搭建了一个私有区块链用于部署该合约。本文通过 Python 模拟实现了一种分布式密钥生成的方式由于密钥需要对大数进行高精度运算保证安全性，因此采用了基于 Python 的 gmpy2[27]、numpy[28]库模拟运行实验分布式密钥生成及管理中，本地运算所需要执行的部分。

4.1.2　实验实施方案

在实验代码中存在四个主要的部分: 用于模拟参与者行为的 user 类，用于分布式密钥生成阶段的 “PedersenVSS” 类以及实现了恢复和重分享的 recover 和 reshare 类构成。在 user 类中，用户可以创建一个 “PedersenVSS” 对象，并将随机决定的密钥值 x 通过 Pedersen 的可验证秘密共享的过程将密钥分片分发给其它参与者；可以对接收到的密钥分片通过调用 “PedersenVSS” 对象中函数的方式进行验证和投

诉；通过和其它用户协作的方式提取出公钥 y 的过程；以及对其它用户提出的恢复和重分享过程发起响应，完成密钥分片的安全传递。在 “PedersenVSS” 类中，实现了密钥分片的计算、分发，验证和对密钥进行恢复的过程。除此之外，还存在一个用于运行这些代码的 main.py 文件，用于模拟实验中出现的各个过程。

为了比较不同用户数目参与过程时需要的时间，在 main.py 中还存在一个计时器，用于记录整个流程消耗的时间。由于代码设计上采取一次处理一个用户行为的方式，因此可以认为单一用户消耗的时间开销有

$$t_{user} = \frac{t_{total}}{number\ of\ participants}$$

4.2 智能合约方案

在分布式密钥管理机制中，智能合约被用于确保密钥分片的安全生成和对于恢复密钥分片或在组间传递密钥分片提案的管理。在本节中存在两个智能合约，第一个合约用于在密钥生成阶段将广播值的信息传递给其它用户，从而使其能够确认该合约有效；第二个合约用于用户提出对单一密钥分片的恢复或在用户群间动态重分享密钥分片的提案，当超过阈值（即恢复密钥所需最低人数）同意提案时，即可进行密钥分片的恢复或传递。

如下的合约 DKG 是在密钥生成阶段用于广播相关数据的合约。其中存在三个事件：Commit、ComplaintX、Share。在密钥分发阶段，参与者需要对其它用户广播对多项式的承诺。为了保证广播的值不被篡改，可以通过将其哈希值存储到区块链上的方式，让其它用户能够验证收到的广播值的正确性。此外，采用事件可以让其它用户使用工具监听区块链的变动状况，保证其能够及时的对广播值进行响应，并进行相应的计算。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DKG {
```

```
// 多项式承诺的Hash值
bytes32 public commitHash;

// 事件：多项式承诺
event Commit(address owner, bytes32 cHash);

// 事件：生成密钥x阶段投诉
event ComplaintX(address owner, address complained);

// 事件：应对投诉，广播分片值
event Share(address complainer, address complained, bytes32 sHash);

// 在GenerateX阶段广播投诉
function broadcastComplaintX(address complain) public {
        // 广播投诉事件
        emit ComplaintX(msg.sender, complain);
}

// 在ExtractY阶段广播投诉
function broadcastComplaintY(address complain, bytes32 sHash) public {
        // 广播秘密分片
        emit Share(msg.sender, complain, sHash);
}

// 广播多项式承诺
function broadcastCommit(bytes32) public {
        // 触发承诺事件，将Hash上传到链上
        // 其它参与者监听此事件，等待广播的Hash
        emit Commit(msg.sender, commitHash);
}
```

```
}
```

如下的合约 Voting 是用于对用户群进行提案的合约。合约的拥有者首先需要上传一个所有参与者地址的列表，用于确保仅有拥有密钥分片的用户可以对提案进行投票。当合约被部署到区块链上时，其会发起一个事件，并将提案内容的哈希值存储到区块链上。这样其余参与者在监听到提案出现时，会可以对提案进行投票。此外，合约拥有者可以在收集到足够多的赞成票后决定停止合约的运行，并将合约关闭并销毁。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Voting {
        address public owner;
        address[] public participants;
        //提案的哈希值，防止输入过长字符串无限消耗gas费用
        bytes32 public proposal;
        mapping(address => bool) public voters;
        uint public yesVotes = 0;
        uint private threshold;

        //事件：发起投票
        event Vote(address owner, bytes32 proposal);

        //事件：提案结果
        event Result(address owner, bytes32 proposal, bool result);

        constructor(bytes32 _proposal, uint256 num_p, address[] memory
            _participants) {
                owner = msg.sender;
                proposal = _proposal;
```

```
        emit Vote(msg.sender, proposal);
        threshold = num_p / 2;
        participants = _participants;
}

// 修饰符，结束函数只有发送者可以调用
modifier onlyOwner() {
        require(msg.sender == owner, "Caller is not the owner");
        _;
}

// 修饰符，投票函数仅合格者可以调用
modifier onlyParticipants(){
        bool isParticipant = false;
        for(uint i = 0; i < participants.length; i++){
                if(msg.sender == participants[i]){
                        isParticipant = true;
                        break;
                }
        }
        require(isParticipant, "Only participant can vote.");
        _;
}

// 投票函数
function vote(bool _vote) public {
        require(voters[msg.sender] == false, "You have already voted
            .");
        voters[msg.sender] = true;
```

```
            if (_vote == true) {
                yesVotes++;
            }
        }

        //控制投票停止
        function endVoting() public onlyOwner {
            if (yesVotes > threshold) {
                emit Result(owner, proposal, true);
            } else {
                emit Result(owner, proposal, false);
            }
            selfdestruct(payable(owner));// 结束运行后销毁合约
        }
}
```

4.3 时间开销分析

在本实验中，通过对拥有 3、5、7、9 个用户（门限为 2、3、4、5 个用户）的分布式系统进行测试，测试在不同参与者参与的情况下，本方案中用户在不同阶段的运行效率。由于无法部署一个真实的分布式系统，这里的数据均不考虑网络传输等可能带来的时延。在实验中，将所有阶段分别运行 10 次，并取平均值作为该情况下该阶段的运行耗时。

在密钥生成阶段不同参与者数量下，最终成功生成密钥用时如图4.1所示。随着参与者数目的增多，生成多项式的度数会随之增加，计算对多项式承诺的运算量随之增加，增加速率保持线性增长。此外，对密钥分片验证过程的运算量同样线性增长。因此随着用户数目增加，单一用户运算用时呈线性增长，总耗时呈指数增长。

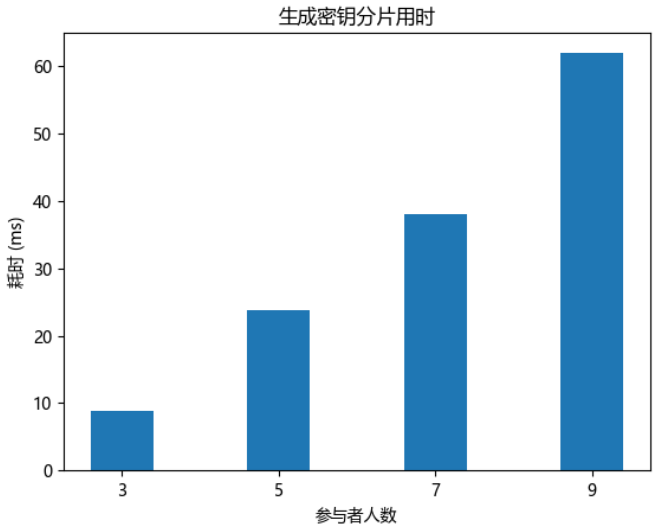

图 4.1　不同用户数量生成密钥 x 用时

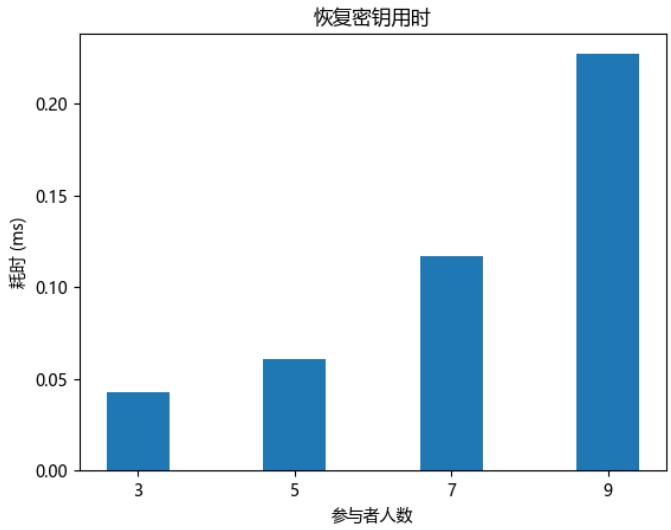

图 4.2　不同用户数量生成密钥 x 用时

在不同用户数目的情况下，通过插值法恢复密钥用时如图4.2所示。由于恢复密钥时采用拉格朗日插值公式计算原多项式，因此单一用户恢复密钥所需要的时间与拉格朗日插值法有关。因为拉格朗日插值法需要计算基函数，因此其计算复杂度为 $O(n^2)$，即其时间消耗随用户数目增加呈指数增长。

在提取公钥 Y 阶段不同参与者数量下，最终提取出 Y 用时如图4.3所示。与生成密钥的情况相同，每增加一名用户，计算 y 值需要多与一项进行相乘，运算耗时呈线性增加。

在运行 Recover 协议时，不同参与者数量下，最终成功恢复密钥分片用时如

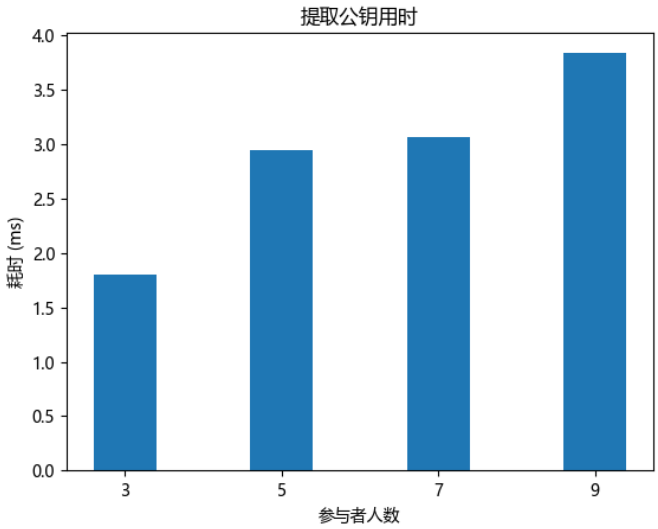

图 4.3 不同用户数量提取公钥 y 用时

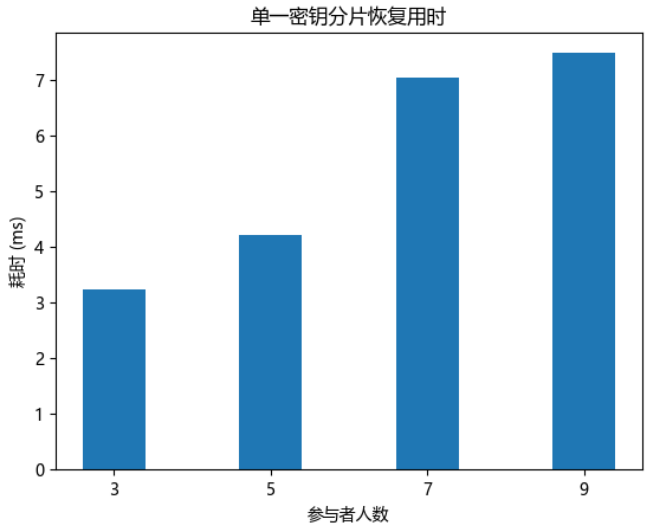

图 4.4 不同用户数量恢复秘密分片用时

图4.4所示。由于提取密钥分片同样需要进行拉格朗日插值，因此其消耗时间同样随用户数目增加呈指数增长。

在进行 Reshare 过程中，不同参与者数量下，最终成功安全传递密钥用时如图4.5所示。它在计算上的开销只有计算盲化多项式的值、计算盲分享和从盲分享中提取新的分片值的过程。这意味着其计算开销随着用户数量的增加呈线性增长。

根据以上数据我们可以了解到，随着参与用户数量的增加，单一用户计算开销发生了显著的上升，尤其是恢复密钥分片或密钥的过程，其随用户增加时间开销呈指数增长。因此本方案如果在实际场景中应用，可能需要严格控制参与者数

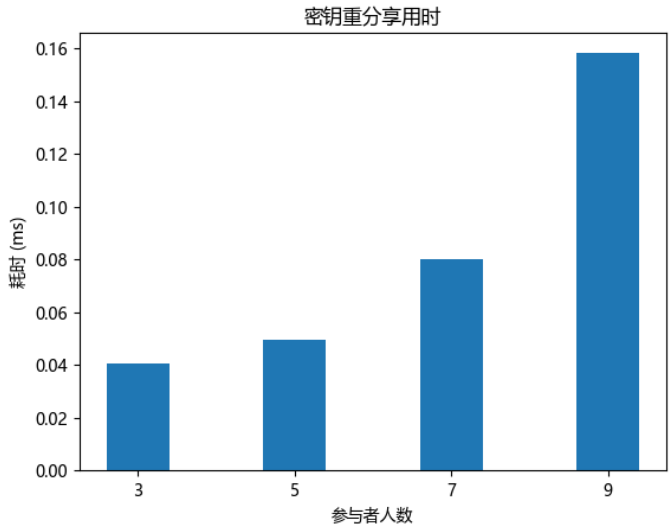

图 4.5　不同用户数量传递密钥用时

量以控制单一用户的计算开销。

4.4　本章小结

在本章中，对本文提出的方案设计了一个实验用于实现并评估方案是否可行。首先本文给出了实验的实施环境，并介绍了实验整体上的实施方案。随后介绍了一种可能的实现前文中提到智能合约的实现，最后对运行方案后，在拥有 3、5、7、9 个参与者的情况时，方案中各个过程的时间开销。

第五章　总结与展望

随着区块链技术的发展，区块链应用已经得到了广泛的应用，伴随着以太坊和智能合约的进步，各种区块链应用如雨后春笋般出现。相对应对于群体的分布式安全管理和身份认证成为了需要被解决的问题如何在去中心化的前提下，对密钥进行安全管理仍然是一个亟待解决的问题。本文提出了一种方案，可以在群体间共同决定如何对密钥进行管理，这种方案有助于区块链应用的可用性和安全性。

通过本文的研究，希望能够提供一种能够用于分布式群体之间的利用区块链技术进行管理的密钥管理的机制，并希望能够帮助区块链应用能够应用于更多场景。此外，也期待对该机制进行更深入的研究和优化，以满足不断变化的安全需求。

致　谢

时光荏苒，岁月如梭，从 2020 到 2024，四年的时光转瞬即逝，我的本科生涯也即将落下帷幕。回首往昔，我在生活中和学习上都受到了许多人带来的帮助。

首先，我要感谢我的父母。没有他们的关心和意见，我可能会遇到更多的挫折，走更多的弯路。他们是我坚实的后盾，任何时候都会给我最大的支持。他们的理解和包容让我在遇到困难时有了坚持下去的勇气。

同时，感谢马立川老师给我的毕业设计和论文写作的指导。马老师的专业知识和严谨的态度令我获益匪浅，他的耐心指导和无私帮助使我的毕业设计能够顺利完成。感谢马老师自始至终一次又一次耐心的指导和帮助。

此外，我在大学期间遇到了一群志同道合的朋友。从带我成长的学长，到共同进步的同学，我们在学校中共同面对生活和学业上的困难，也一起感受过成功的喜悦。他们给我带来了多彩的生活，如果我的大学生涯没有他们的身影，可能我在只能度过一个索然无味的四年。

最后，我要感谢我的电脑。从初中毕业到大学毕业，它和我共同经历了无数个日日夜夜，可以说伴随了我最久的时间。即使它可能有些脱离时代，但仍然能够完美的完成我交给它的一切任务。

回首往昔，我已经经历许多；展望未来，永远相信美好的事情即将发生。

参考文献

[1] FELDMAN P. A practical scheme for non-interactive verifiable secret sharing[C/OL]//28th Annual Symposium on Foundations of Computer Science (sfcs 1987). 1987: 427-438. DOI: 10.1109/SFCS.1987.4.

[2] SZABO N. Smart Contracts: Building Blocks for Digital Markets[C/OL]//. 2018. https://api.semanticscholar.org/CorpusID:198956172.

[3] LAMPORT L, SHOSTAK R, PEASE M. The Byzantine generals problem[G]//Concurrency: the works of leslie lamport. 2019: 203-226.

[4] CASTRO M, LISKOV B. Practical byzantine fault tolerance and proactive recovery[J]. ACM Transactions on Computer Systems (TOCS), 2002, 20(4): 398-461.

[5] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system[J]. Bitcoin.–URL: https://bitcoin.org/bitcoin. pdf, 2008.

[6] BUTERIN V, et al. A next-generation smart contract and decentralized application platform[J]. Ethereum。–URL:https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014, 3(37): 2-1.

[7] ZHENG Z, XIE S, DAI H N, et al. An overview on smart contracts: Challenges, advances and platforms[J]. Future Generation Computer Systems, 2020, 105: 475-491.

[8] BLAKLEY G R. Safeguarding cryptographic keys[C]//Managing requirements knowledge, international workshop on. 1979: 313-313.

[9] SHAMIR A. How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612-613.

[10] STADLER M. Publicly verifiable secret sharing[C]//International Conference on the Theory and Applications of Cryptographic Techniques. 1996: 190-199.

[11] MARAM S K D, ZHANG F, WANG L, et al. CHURP: dynamic-committee proactive secret sharing[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 2369-2386.

[12] BESSANI A N, ALCHIERI E P, CORREIA M, et al. DepSpace: a Byzantine fault-tolerant coordination service[C]//Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008. 2008: 163-176.

[13] BASU S, TOMESCU A, ABRAHAM I, et al. Efficient verifiable secret sharing with share recovery in BFT protocols[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 2387-2402.

[14] KOKORIS KOGIAS E, ALP E C, GASSER L, et al. Calypso: Private data management for decentralized ledgers[J]. Proceedings of the VLDB Endowment, 2021, 14(4): 586-599.

[15] MARAM S K D, ZHANG F, WANG L, et al. CHURP: dynamic-committee proactive secret sharing[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 2369-2386.

[16] GOYAL V, KOTHAPALLI A, MASSEROVA E, et al. Storing and retrieving secrets on a blockchain[C]//IACR International Conference on Public-Key Cryptography. 2022: 252-282.

[17] 刘文杰, 郑玉, 陈遥. 基于 ECC 可公开验证的多方秘密共享方案[J]. 计算机工程与设计, 2008, 29(14): 3.

[18] CASTRO M, LISKOV B. Practical byzantine fault tolerance and proactive recovery[J]. ACM Transactions on Computer Systems (TOCS), 2002, 20(4): 398-461.

[19] SOUSA P, BESSANI A N, CORREIA M, et al. Highly available intrusion-tolerant services with proactive-reactive recovery[J]. IEEE Transactions on Parallel and Distributed Systems, 2009, 21(4): 452-465.

[20] GARCIA M, BESSANI A, NEVES N. Lazarus: Automatic management of diversity in bft systems[C]//Proceedings of the 20th International Middleware Conference. 2019: 241-254.

[21] PEDERSEN T P. A threshold cryptosystem without a trusted party[C]//Advances in Cryptology —EUROCRYPT' 91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10. 1991: 522-526.

[22] PEDERSEN T P. Non-interactive and information-theoretic secure verifiable secret sharing[C]// Annual international cryptology conference. 1991: 129-140.

[23] GENNARO R, JARECKI S, KRAWCZYK H, et al. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems[C]//STERN J. Advances in Cryptology — EUROCRYPT '99. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999: 295-310.

[24] ZHAO J, GONZALEZ A, AMID A, et al. COBRA: A Framework for Evaluating Compositions of Hardware Branch Predictors[C/OL]//2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 2021: 310-320. DOI: 10.1109/ISPASS51385.2021.00053.

[25] Python Language Reference, version 3.10[Z]. http://www.python.org.

[26] solidity[EB/OL]. https://soliditylang.org/.

[27] gmpy2: Fast multiple-precision arithmetic for Python[EB/OL]. https://pypi.python.org/pypi/gmpy2/.

[28] HARRIS C R, MILLMAN K J, van der WALT S J, et al. Array programming with NumPy[J/OL]. Nature, 2020, 585(7825): 357-362. https://doi.org/10.1038/s41586-020-2649-2. DOI: 10.1038/s41586-020-2649-2.