南 开 大 学

本 科 生 毕 业 论 文（设 计）

(双学位专用)

中文题目：Plonk 零知识安全协议的研究

外文题目：Research on Plonk Zero-Knowledge Security Protocol

学　　号：1711303

姓　　名：张欣蕊

年　　级：2017 级

专　　业：信息安全

系　　别：信息安全系

学　　院：网络空间安全学院

双修专业：法学

双修院系：法学系

指导教师：苏明

完成日期：2021 年 5 月

关于南开大学本科生毕业论文（设计）的声明

本人郑重声明：所呈交的学位论文，是本人在指导教师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

2021 年 5 月 日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：

2021 年 5 月 日

摘 要

为了探究目前零知识证明的性能最优算法，本文对实现了 zk-SNARK 的 PLONK 协议进行了研究和应用。零知识证明在区块链中的应用十分重要而广泛，如 Zcash 项目利用 zk-SNARK 实现了隐藏交易的发送方、接收方以及交易金额的“隐私计算”；Filecoin 项目实现了利用零知识证明，证明存储提供方真实有效地存储了指定的数据等。本文对理解零知识证明需要的背景知识进行了较为系统的介绍，包括交互与非交互式的零知识证明方式，同态加密、基于椭圆曲线的同态加密、结构化查询字符串 SRS、多项式承诺、Kate 承诺以及双线性映射。同时，本文对 PLONK 协议的实现原理和执行过程进行了研究，并应用 PLONK 协议对一个实际问题进行了具体的电路设计与实现。并且基于此实现进行不同量级电路门数量的扩展，分析了电路门数量对证明过程的运行时间，以及证明等输出文件占用空间大小的影响，并提出了相应的改进思路。

关键词：零知识证明；PLONK；多项式承诺；区块链

Abstract

In order to have an insight on the most optimal construction of zk-SNARK, this paper researches and applies the PLONK algorithm. The application of zero-knowledge proof in the blockchain is very important and extensive. For example, the Zcash project used zk-SNARK to implement the ”privacy computing” ,where the sender, receiver and transaction amounts were hidden. The Filecoin project implemented the zero-knowledge proof to prove that the storage provider had specified data stored truly and effectively. This article provides a systematic introduction to the background knowledge needed to understand zero-knowledge proof, including interactive and non-interactive zero-knowledge proof methods, elliptic curve-based homomorphic encryption, polynomial commitment, bilinear mapping and so on. This paper studies the implementation principle and execution process of the PLONK protocol. This paper also applies the PLONK protocol to a specific circuit design and implementation of a practical problem. Based on this implementation, through expanding the number of circuit gates of different magnitudes, we analyzed the influence of the number of circuit gates on both the running time of the verification process and the size of the verification output files. We also put forward corresponding ideas for performance improvement.

Key Words: Zero-knowledge proof; PLONK; Polynomial Commitment; Blockchain

目 录

第一章　绪论

第一节　研究背景

零知识证明协议可以用于保护用户信息，在过去十年的学术界中依然活跃。与区块链结合时，可用于有效保护用户的相关隐私信息，因此受到更多人的关注。但是零知识证明协议通常由于计算开销大、密文大小膨胀导致空间开销大等原因，不能获得较为理想的性能。新一代的 zk-Snark 零知识证明协议中，Plonk 采用 Rust 开源代码实现，性能较为突出。

因此，研究 PLONK 协议将对区块链及其他情境下的零知识证明过程性能优化提供可能。

第二节　零知识证明研究现状

“零知识证明”与“交互式证明”的概念最初由 Goldwasser，Micali 和 Rackoff[1] 在 1985 年提出。在交互式证明系统中，验证者 V 可以向证明者提出随机问题，V 可以验证超出她计算能力之外的问题，且在交互过程中，P 的私密信息不会被泄露给 V。1991 年，Goldreich 等人[2] 证明了任何一个 NP 问题都有对应的零知识证明算法。2013 年，Parno, Bryan 和 Howell 等人[3] 设计了代表第一代 zk-SNARK 的匹诺曹协议后，更加通用高效的算法不断涌现。以 Groth16 为代表的 zk-SNARKs 技术生成的证明占用空间更小，且验证时间更短，但仍需要提前为每个不同的问题生成新的可信参数，通用性较差。

可更新的通用结构化参考字符串[4]（structured reference string，简称 SRS）的应用为 zk-SNARK 的部署消除了部分障碍。Maller 等人提出的 Sonic[5] 是第一个具有实用性的 zk-SNARK，利用 SRS 为通用算数电路设计了完全简洁的验证。但 Sonic 的证明构建过程仍然开销较大。2019 年由 Gabizon 等人[6] 提出的 PLONK 协议在实现了无需为每个程序单独设计 SRS[7] 的通用 zk-SNARK 的基础上，大大降低了证明方的运行时间。STARK 算法不基于数学难题假设，具有抗量子性，安全性最高，但证明占用的空间也最大。

目前零知识证明已经被广泛应用于区块链中。如 Zerocash[8]（简称 Zcash）应用了基于 zk-SNARKs 的隐私交易[9]，实现了在不透露交易具体信息的基础上，

计算并判断交易是否有效。比特币与以太坊使用了 Schnorr 签名算法。以及包括 Hyperledger Fabric、Quorum 在内的联盟链也都将零知识证明应用于签名算法或隐私层。

第三节　论文的主要工作

本文主要介绍了零知识证明的背景知识，基本概念，以及实现零知识证明的基础理论。主要描述了 zk-SNARK 的最新实现 PLONK 协议的实现原理以及运行步骤。基于 PLONK 的基础 Rust 语言代码，针对某个实际应用场景设计了算数电路并进行了代码实现与证明的运行测试。并且基于此实现进行不同量级电路门数量的扩展，分析了电路门数量对证明过程的运行时间，以及证明等输出文件占用空间大小的影响。

第四节　论文的组织结构

本文共四个章节。第一章介绍了研究背景以及零知识证明的研究现状，概述了本文的主要工作以及文章组织结构。

第二章将零知识证明分为交互式与非交互式两类分别介绍其定义，并介绍了理解 zk-SNARK 需要具备的基本算法和理论。

第三章介绍了 PLONK 协议的实现原理。包括其电路门的结构，电路的组成方式，算数多项式与电路之间的转化关系，以及证明者与验证着各自的执行步骤等。

第四章利用 PLONK 协议对一个实际问题进行了电路的设计与构建。介绍了 PLONK 协议在程序中实现的过程，呈现了证明的构建过程与具体形式。最后对不同加法门数量的电路进行运行测试，观察门的数量与证明生成和验证过程的运行时间是否存在关联。并在此基础之上提出改进思路。

第五章对本文进行了总结与展望。

第二章　零知识证明概述

第一节　零知识证明的定义

零知识证明是一种对于某个关键信息的证明方式。证明者可以向验证者自己知道某个信息，但同时又不必泄露该信息的具体内容。

零知识证明的概念最初由 MIT 的研究人员 Goldwasser, Micali 和 Rackoff 在 1985 年提出[1]。他们认为，当一个证明者 P (Prover) 正在向一个多项式时间的概率算法验证者 V (Verifier) 证明一个命题，如果对于这个验证者在与证明者交互时可以计算的任何内容，她都可以无需通过交互自己计算得到，那么这个证明过程就是零知识的。

用时间机器与三色问题[10] 的例子来说明。假设三色问题的证明者 P 试图用随机展示部分解的方式证明自己知道全部解。该证明者 P 并不掌握该三色问题的解决方案，但其可以通过不断的时间穿越来观看若干种验证过程的答案正确与否，直到找到一个正确的验证答案并穿越回去，用这种作弊的方式完成这次验证。如果一个真正的三色问题验证过程中，验证者 V' 从证明者 P' 那里获取的信息量与时间旅行例子中验证者 V 所能获取的相同，则认为该证明过程是零知识的。

零知识证明需要满足以下性质：

一、完备性（Completeness）：假设证明者 P 和验证者 V 都是诚实的，并且依照证明协议完整地执行了每一步。那么如果陈述 (Statement) 是正确的，P 就可以让 V 确信自己掌握这个知识。

二、可靠性（Soundness）：证明方无法通过作弊的方式使验证者接受错误的陈述。即错误的陈述无法通过验证。

三、零知识性（Zero-Knowledge）：证明结束后，验证方只能得知证明方的陈述是否正确，但无法得知任何其他信息。

对零知识证明的效率主要需要考虑：交互轮数、证明时间、验证时间、证明大小以及是否可以转变成非交互式。对于交互式的零知识证明协议还要考虑交互轮数。

2.1.1 交互式零知识证明

作为零知识证明的最初实现方式，交互式零知识证明需要参与证明的双方执行一系列指定的互动过程。

交互式零知识证明通常要经历几个步骤:

一、承诺（Commit）：假设证明者 P 要证明其知道某个三色问题的解。承诺阶段要求证明者将这个解表示出来，如用卡片上色并摆在桌子上，此后将这个解的每一个节点盖住。此时 P 提供的解不会被泄露，且不可被篡改。

二、挑战（Challenge）：在挑战阶段，验证者 V 可以要求 P 随机打开一条边，即将一条边的两个端点的颜色展示出来。

三、证明（Open）：打开阶段要求 P 展示 V 随机挑选的那条边的两个端点颜色，作为随机挑战的回应。

四、验证（Verify）：如果 V 可以看到 P 展示的两点颜色相同，则认为 P 一定不掌握正确解；如果 P 展示的两点颜色相同，则 P 有可能掌握这个正确解。

仅进行一轮上述验证过程只能确定展示出的边是正确的，但并不能保证 P 知道该三色问题的解。当证明过程所采用的图 $G=(V,E)$ 时，上述证明过程只有 $\frac{1}{E}$ 的概率可以保证 P 知道整张图的正确解。P 仍有 $\frac{E-1}{E}$ 的概率进行了欺骗。

这种原始的交互式零知识证明是一种基于概率的证明方法。为了提高证明结果可靠的概率，需要重复 n 轮以上过程，使 P 用作弊的方式通过证明的概率降低为 $\left(\frac{E-1}{E}\right)^n$。同时，为防止重放攻击，交互式证明过程往往需加入随机挑战。这意味着证明者需要参与每一次的证明过程。

密码学协议中的证明往往基于离散对数等数学难题而非概率，但这种交互式证明方式每次验证均需要证明方参与提供新的证明，依然开销较大，在区块链的应用情境下实用度较低。

2.1.2 非交互式零知识证明

非交互的零知识证明则无需证明者参与每次验证过程。在非交互式零知识证明协议中，P 向 V 提供的证明（Proof）是一个独立的信息。为了避免交互式的随机数挑战过程，非交互式模型的实现要求:

一、P 与 V 都可以访问模拟为随机数生成器的哈希函数。

二、P 与 V 产生的哈希函数输入值相同

Fiat-Shamir 启发式算法可以实现交互式零知识证明到非交互式证明的转化，

将在第二节中介绍。如今广泛采用的零知识证明协议也是基于非交互式的零知识证明实现 zk-SNARKs（Zero-Knowledge Succinct Non-Interactive Argument of Knowledge）。

第二节　零知识证明的基础理论

2.2.1　Schnorr 协议

交互式的 Schnorr 协议[11] 实现了基于离散对数问题的零知识证明。

该协议设置的情境如下。存在一个秩为素数 q 的群 $\mathbb{G}$，P 想要证明她知道群元素 $h = g^x \in \mathbb{G}$ 的离散对数 x。存在关系 $\mathscr{R} = \{(x,h) \in \mathbb{Z}_q \times \mathbb{G} : g^x = h\}$[12]，其中群 $\mathbb{G}$ 与其生成元 g 是 P 与 V 都知道的公共参数。

初始状态 P 与 V 掌握的信息分别为

$$P(x, h = g^x),$$

$$V(h = g^x).$$

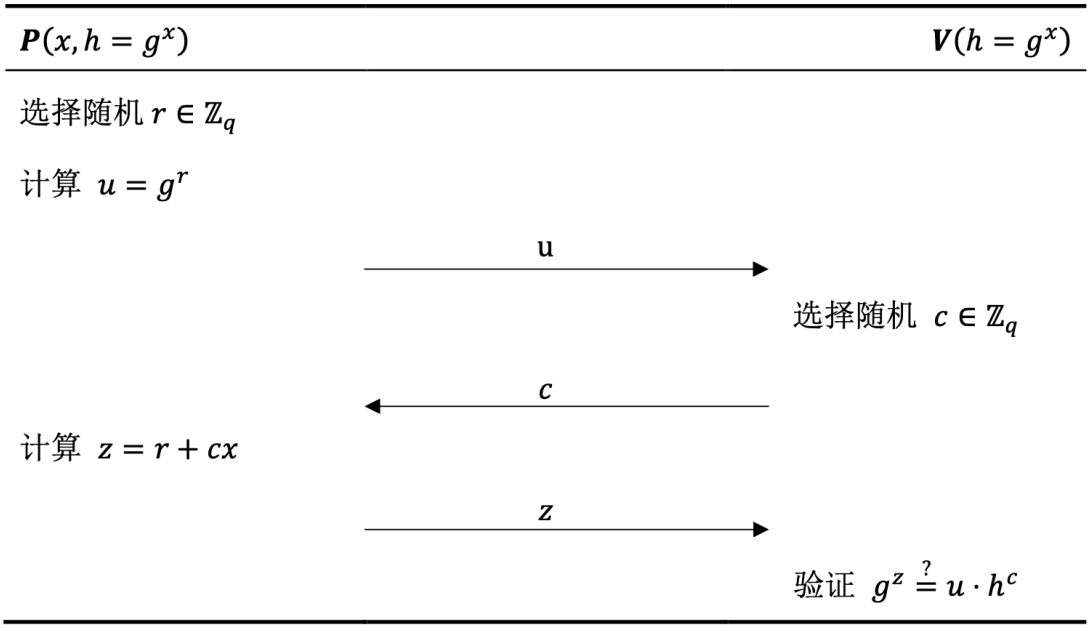

图 2.1: Schnorr 协议交互过程

其完备性表现为，如果 $z = r + cx$, 那么 $g^z = g^{r+cx} = g^r \cdot (g^x)^c = u \cdot h^c$。Schnorr 协议存在被恶意 V 攻击的可能性，导致其失去零知识性[12]，但本文主要介绍零

知识证明协议本身的发展，不对此深入分析。

2.2.2 Fiat-Shamir 启发式算法

Schnorr 协议实现了基于离散对数难题的单次交互式的零知识证明。

Fiat-Shamir 启发式算法[13] 则提出了一种将交互式零知识证明转换为非交互式的方法。主要思想是将 V 提供随机挑战值这一过程，转化为特定的随机数计算方法。保证 P,V 双方在计算时通过特定的哈希算法，以及包括 P 给出的第一条信息在内的输入值，从而获得相同的随机挑战参数。

用 Fiat-Shamir 启发式算法改进后的 Schnorr 协议[14] 不再有交互式的步骤，而是由 P 生成证明 π，发送给 V，再由 V 验证 π 的合法性。

改进后，P 与 V 先后完成的计算如下

表 2.1: P 生成 π 的过程

$P(g,x,h=g^x)$
选择随机 $r \in \mathbb{Z}_q$
计算 $u = g^r$
计算 $c = H(g,h,u)$
计算 $z = r + cx$
生成 $\pi = (u,c,z)$

表 2.2: P 将 π 发送给 V,V 执行

$V(g,h=g^x,\pi=(u,c,z))$
验证 $c \stackrel{?}{=} H(g,h,u)$
验证 $g^z \stackrel{?}{=} u \cdot h^c$

改进后的算法完备性与交互式 Schnorr 协议相同，第一步验证保证了哈希值产生的真实性，第二步是零知识证明的验证过程。

可见 Fiat-Shamir 启发式算法改进后无需 V 提供随机参数。所有进行验证的 V 都可以使用 P 所提供的同一个证明来完成验证。成功实现零知识证明协议的非交互式改进。

2.2.3 基于椭圆曲线的同态加密

一、多项式的表示

对于一个在有限域 $\mathbb{F}_p$ 上的 d 次多项式，其表达式为

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^d, \tag{2.1}$$

其中 $a_0, a_1, \ldots a_d \in \mathbb{F}_p$。

证明关于这个多项式的知识，等价于证明关于其系数 $a_0, a_1, a_2, \ldots, a_d$ 的知识。我们可以通过用 s 取代 x 的方式，计算出多项式 $P(x)$ 在点 $s \in \mathbb{F}_p$ 处的值，即

$$P(s) = a_0 + a_1 \cdot s + a_2 \cdot s^2 + \ldots + a_d \cdot s^d. \tag{2.2}$$

此时 $P(s)$ 可以被看作系数为 $a_0, a_1, a_2, \ldots, a_n$ 的值 $1, s, \ldots, s^d$ 的线性组合。

二、椭圆曲线的同态加密

zk-SNARK 基于椭圆曲线加密算法实现，利用了椭圆曲线自身的加法同态性。为了便于演示同态加密，本章将使用 g^x 代替 $g \cdot x$ 表示椭圆曲线标量乘法。定义 $E(x) = g^x$，g 是椭圆曲线的生成元，该式有如下性质

$$E(ax + by) = g^{ax+by} = g^{ax} \cdot g^{by} = (g^x)^a \cdot (g^y)^b = E(x)^a \cdot E(y)^b. \tag{2.3}$$

可以利用椭圆曲线的该特性进行多项式的加密计算，即

$$g^{1+x+x^2+\ldots+x^d} = g \cdot g^x \cdot g^{x^2} \cdot \ldots \cdot g^{x^d}. \tag{2.4}$$

对于 $1, x, x^2, \ldots, x^d$ 的任意线性组合，即多项式 $P(x)$ ，可以进行如下加密变换

$$g^{P(x)} = g^{a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_d \cdot x^d} = g_0^a \cdot (g^x)^{a_1} \cdot \left(g^{x^2}\right)^{a_2} \cdot \ldots \left(g^{x^d}\right)^{a_d}. \tag{2.5}$$

基于椭圆曲线上的离散对数问题 (ECDLP)，将多项式以椭圆曲线同态加密的方式表示，以实现多项式知识的隐藏。

2.2.4 结构化查询字符串 SRS（Structured Reference String）

SRS 是 P 与 V 双方用于进行证明计算的公共参数，在证明过程是开始前的 Setup 阶段进行设置。假设证明者 P 知道一个多项式 $P(x)$，验证者 V 想得到多项式在 $s \in \mathbb{F}_p$ 处的加密值 $E(P(s))$，有两种方法：

一、P 将多项式 $P(x)$ 发送给 V，由 V 自行计算 $E(P(s))$；

二、V 将 s 发送给 P，P 计算 $E(P(s))$ 后发送给 V。

但我们既不希望 V 知道的多项式的知识，也不希望 P 获得 s 的明文，以保证 P 的证明是基于一个随机盲挑战，防止其在后续证明过程中作弊。

因此，可以进行如下步骤：

一、V 计算出 $E(1), E(s), E\left(s^2\right), \ldots, E\left(s^d\right)$ 发送给 P；

二、P 利用 V 发来的这些值计算出 $E(P(s))$，发送给 V。

但此时，V 并不能保证 P 发来的结果是由她发给 P 的公开参数计算得到的，P 有可能伪造参数进行了计算。而此时 V 手中只有最终的结果 $E(P(s))$，仅凭这一个值无法证明这个结果是由她所提供的机密参数计算得到的。

因此需要引入一个新的参数 α。由 V 计算 $E\left(s^i\right), E\left(\alpha s^i\right)(i \in [0,d])$，然后将 $E\left(s^i\right), E\left(\alpha s^i\right)$ 发给 P。P 用这些值计算得到 $E(P(s))$ 与 $E(\alpha P(s))$，即 $g^{P(s)}$ 与 $g^{\alpha P(s)}$。

此时由 V 验证 $\left(g^{P(s)}\right)^{\alpha} = g^{\alpha P(s)}$，即可证明 P 是用 V 提供的初始参数计算得到的 $E(P(s))$。这个验证过程是交互式的，由 V 提供初始参数以及 α，因此 V 可以直接利用 $g^{P(s)}$ 以及 α 的值计算得到 $\left(g^{P(s)}\right)^{\alpha}$。但在非交互式的证明过程中，需要一个初始化的过程来生成复合计算需要的初始公开参数 $E\left(s^i\right), E\left(\alpha s^i\right)(i \in [0,d])$。

综上，SRS 包含了 P 用于计算和生成加密多项式的一系列参数 $(E(s^i), E(\alpha s^i) i \in [0,d])$ 即 $(g^{s^i}, g^{\alpha s^i} i \in [0,d])$。

2.2.5 多项式承诺

密码学的承诺方案[15] 是一个涉及证明和验证双方的二阶段交互协议。承诺的第一个阶段由承诺的发送方选择一个消息 m，将其加密后，以密文 *commit* 的形式发送给承诺的接收方，加密意味着承诺方无法再更改原始信息 m。第二个阶段由承诺方公开消息 m 与用于加密信息的盲化因子。接收方根据此信息验证其第二阶段收到的 m 与第一阶段承诺的消息是否一致。

多项式的承诺经历承诺（Commit）、随机挑战与生成证明（Challenge & Proof）和验证（Verify）三个过程。

一、承诺

证明方 P 拥有待承诺的多项式 $f(x)$，P 取随机数 r。计算

$$c = f(r), \tag{2.6}$$

公开 (r,c)。

二、随机挑战与生成证明

V 选择随机挑战 z 发送给 P。

P 计算

$$s = f(z), \tag{2.7}$$

$$t(x) = \frac{f(x)-s}{x-z}, \tag{2.8}$$

$$w = t(r). \tag{2.9}$$

P 向 V 返回 $V(s,w)$。

三、验证

验证方 V 验证

$$c - s = w(r-z). \tag{2.10}$$

因为

$$c - s = f(r) - s = t(r)(r-z) = w(r-z). \tag{2.11}$$

此协议是一个较为简单的多项式承诺框架，但在实际使用 P 可以通过构造多项式的方式进行作弊。因此需要对其进行改进，对挑战 z 做盲化以防止 P 通过构造多项式的方式作弊；同时对多项式做有限域的加密映射，以防止 V 对承诺的多项式进行暴力反推，从而保证零知识性。

2.2.6 Kate 承诺

PLONK 协议中应用了多项式承诺作为其零知识证明协议的主协议框架，但其承诺利用 SRS 发生在有限域上。Kate 承诺选择椭圆曲线作为密文空间，利用 SRS 使多项式的输入被隐藏，使 P 难以伪造出满足式（2.10）的多项式。同时设

计了多项式的部分打开，满足了零知识证明的应用场景。

部分打开的 Kate 承诺经历初始化（Setup）、承诺（Commit）、生成证明（Create Witness）和验证（Verify）四个步骤。

一、初始化

选择对称双线性映射的子群, 生成元 $g \in \mathbb{G}$, 配对函数 $e : \mathbb{G} * \mathbb{G} = \mathbb{G}_T$。生成随机秘密 α，生成公开的 SRS 元组 $\{g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^t}\}$。

二、承诺

待承诺多项式为

$$\varphi(\mathrm{x}) = \sum_{j=0}^{d} a_j x^j (d \leq t). \tag{2.12}$$

计算并输出承诺

$$C = \prod_{j=0}^{d} \left(g^{\alpha^j}\right)^{a_j} = g^{\varphi(\alpha)}. \tag{2.13}$$

三、生成证明

这一步用来生成证明（Witness）。对特定的多项式输入 i（evaluation challenge）计算 $\varphi(i)$（opening evaluation）。令

$$\psi_i(x) = \frac{\varphi(x) - \varphi(i)}{x - i}. \tag{2.14}$$

计算

$$w_i = g^{\psi_i(\alpha)}. \tag{2.15}$$

输出 $Witness = (\varphi(i), w_i)$，即生成的证明。

四、验证

输入 i 处多项式的值 $\varphi(i)$，承诺 C，证明 w_i。

利用配对函数验证

$$e(C, g) = e\left(w_i, \frac{g^{\alpha}}{g^i}\right) * e(g, g)^{\varphi(i)}. \tag{2.16}$$

2.2.7 双线性映射

双线性映射又称配对操作（Pairing）。Kate 承诺的最后一步即通过配对函数做 pairing 验证。

由于具有加法同态的特性，椭圆曲线点之间的线性关系很容易得到验证。例如，假设 G 为群 $\mathbb{G}$ 的生成元，p,q,r 为常数。已知 $P=G*p, Q=G*q, R=G*r$，验证 $5*P+7*Q=11*R$ 等价于验证 $5*p+7*q=11*r$。[16] 但椭圆曲线上的点只支持标量乘法，无法通过这种方式验证形如 $p*q$ 的二次关系。

双线性映射使得椭圆曲线点之间这种二次约束关系可以得到验证。双线性映射关系 e 满足 $e:\mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_T$。其中 $\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T$ 是三个群，$\mathbb{G}_1,\mathbb{G}_2$ 是椭圆曲线上的两个乘法子群，而 $\mathbb{G}_T$ 是有限域中的一个乘法子群。双线性的含义是指，对于 $g\in\mathbb{G}_1, h\in\mathbb{G}_2,\quad a,b\in\mathbb{Z}$，有 $e\left(g^a,h^b\right)=e(g,h)^{ab}$。

在 PLONK 的实现中，预先选择了大小为 r 的群 $\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t$，选择可计算的高效非简并 pairing $e:\mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_t$。$g_1,g_2$ 预先选择的生成元，满足 $e(g_1,g_2)=g_t$。

第三章 PLONK 协议介绍

第一节 PLONK 电路的构建

上一章讨论了如何对已知的多项式构造零知识证明。要利用 zk-SNARK 构造实际问题的零知识证明，则需要先将待证明的问题转化为多项式。用编程语言表示的实际问题在应用 zk-SNARK 之前需要进行处理，将复杂的逻辑转化为一系列代数运算式。在 Zcash 实现的 zk-SNARK[17] 中，这一过程表现为程序 → 算数电路 → R1CS[18][19] 约束 → QAP[20]（Quadratic Arithmetic Program）多项式 → 应用 zk-SNARK。同样地，应用 PLONK 前，需要先将实际问题转化为算数运算，再将算术运算用电路门的形式进行约束，随后将电路门的约束转化为多项式形式的约束，最后对得到的多项式进行零知识的证明。因而以上过程将一个有指定输出的电路约束系统，转化为了一个有特定零根的多项式，并对其进行零知识证明。

电路构建过程属于 PLONK 证明协议的准备阶段，这一构建工作可以不由 P 或 V 来参与完成，只需要明确 P 的证明需求。根据不同的证明需求，搭建不同的数字电路。电路的构建是证明者 P 输入并构建证明的前提。

第二节 PLONK 的门约束

一系列的算术运算可以较为容易地转化为门电路。在算数式转化为离散的门后，还需要将这些门之间的联系建立为新的约束关系。因此，PLONK 将门约束分为两部分内容：单独的算术门约束和复制约束。

算数门约束要求电路中有 n 个算数门，每个门的标号 $i \in [1,n], x \in \mathbb{Z}$。实现将所有线路门用一个通用表达式表示

$$(\mathbf{q_L})_i \cdot \mathbf{a}_i + (\mathbf{q_R})_i \cdot \mathbf{b}_i + (\mathbf{q_O})_i \cdot \mathbf{c}_i + (\mathbf{q_M})_i \cdot (\mathbf{a}_i \mathbf{b}_i) + (\mathbf{q_C})_i = 0. \tag{3.1}$$

$\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}$ 是用于选择电路门类型的系数（selector）。

3.2.1 算术门

对于一个加法运算 $a+b=c$，可以用一个加法门表示（如图3.1）。其中 g_i 表

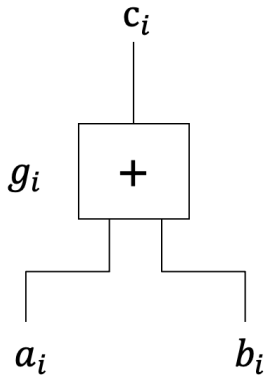

图 3.1: 独立的加法门

示这是电路中的第 i 个门，将 a,b,c 记作 $\mathbf{a}_i,\mathbf{b}_i,\mathbf{c}_i$，分别表示第 i 个门的左输入、右输入和输出三根导线，其中 $\mathbf{a}_i,\mathbf{b}_i,\mathbf{c}_i$ 分别为三根导线上绑定的值。

为导线加上 selector 选择参数，将三个导线分别表示为 $\mathbf{q_L}_i\mathbf{a}_i,\mathbf{q_R}_i\mathbf{b}_i$ 与 $\mathbf{q}_{O_i}\mathbf{c}_i(\mathbf{q_L}_i,\mathbf{q_R}_i,\mathbf{q_O}_i\in\{0,1,-1\})$。从而得到适用于电路的更加通用化的表示形式（如图3.2）：

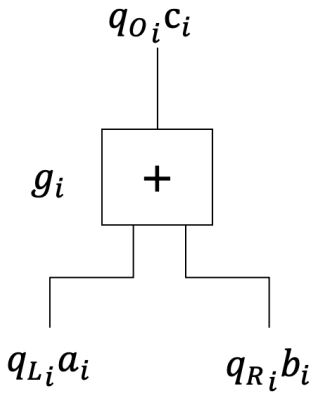

图 3.2: 一个电路中的通用加法门

此时的电路门可以表示为

$$(\mathbf{q_L})_i\mathbf{a}_i\cdot(\mathbf{q_R})_i\mathbf{b}_i=(\mathbf{q}_O)_i\mathbf{c}_i, \tag{3.2}$$

即

$$(\mathbf{q_L})_i\mathbf{a}_i\cdot(\mathbf{q_R})_i\mathbf{b}_i-(\mathbf{q}_O)_i\mathbf{c}_i=0. \tag{3.3}$$

此时 $(\mathbf{q_L})_i = 1, (\mathbf{q_R})_i = 1, (\mathbf{q_O})_i = -1$。由于门的类型仅与 selector 参数有关，因此对于任意的加法门，selector 的值均表示为

$$(\mathbf{q_L})_i = 1, (\mathbf{q_R})_i = 1, (\mathbf{q_O})_i = -1, (\mathbf{q_M})_i = 0, (\mathbf{q_C})_i = 0. \tag{3.4}$$

同样地，对于一个乘法门（如图3.3），其算数表示形式为

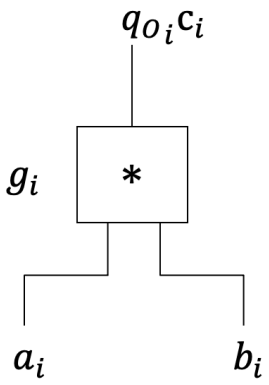

图 3.3: 一个电路中的通用乘法门

$$(\mathbf{q_M})_i \cdot (\mathbf{a}_i\mathbf{b}_i) - (\mathbf{q_C})_i = 0. \tag{3.5}$$

对于任意乘法门，selector 的值均表示为

$$(\mathbf{q_L})_i = 0, (\mathbf{q_R})_i = 0, (\mathbf{q_O})_i = -1, (\mathbf{q_M})_i = 1, (\mathbf{q_C})_i = 0. \tag{3.6}$$

3.2.2 公共输入

公共输入（Public Input，表示为 PI）是指 P 与 V 双方的共有知识。证明生成与证明验证的两个步骤中分别由 P 与 V 各自输入。同时 PI 也是程序用户 V 做验证时唯一的输入。

对于一些涉及公共输入的算数门，其输出不再表示为 c_i，而是单独存入 PI 结构。

$$\mathrm{PI}(x) := \sum_{i\in[\ell]} -\mathrm{x}_i \cdot L_i(x), \tag{3.7}$$

其中 $L_i(x)$ 为拉格朗日基。包含 PI 的加法门为

$$(\mathbf{q_L})_i\mathbf{a}_i \cdot (\mathbf{q_R})_i\mathbf{b}_i - \mathrm{PI}(i) = 0. \tag{3.8}$$

乘法门为

$$(\mathbf{q_M})_i\mathbf{a}_i\mathbf{b}_i - \mathrm{PI}(i) = 0. \tag{3.9}$$

公共参数 $x_1,\ldots,x_\ell \in \mathrm{PI}$ 的赋值方式为，对于 $j \in \ell, i \in [1,n]$

$$(\mathbf{q_L})_i = 1, (\mathbf{q_R})_i = (\mathbf{q_O})_i = (\mathbf{q_M})_i = 0, (\mathbf{q_C})_i = -x_j. \tag{3.10}$$

3.2.3 算数门到多项式的转换

为了将电路中门的约束条件转化为具有特定零根的多项式，约定为电路中的 n 个门顺序标号。$i(i \in n)$ 表示第 i 号门，$\mathbf{a}_i$ 表示第 i 号门的左输入，$\mathbf{b}_i, \mathbf{c}_i$ 分别表示 i 号门的右输入与输出。由此，多项式具有零根 i，等价于多项式满足 i 号门的约束条件。使得多项式满足所有电路门的约束条件，即使多项式拥有零根 $\{1,2,...,n\}$，等价于多项式可以整除

$$\mathrm{Z} = (x-1)(x-2)\ldots(x-n). \tag{3.11}$$

如果一个电路约束系统包含 n 个门，即满足 n 个等式

$$\begin{aligned}
&(\mathbf{q_L})_1 \cdot \mathbf{a}_1 + (\mathbf{q_R})_1 \cdot \mathbf{b}_1 + (\mathbf{q_O})_1 \cdot \mathbf{c}_1 + (\mathbf{q_M})_1 \cdot (\mathbf{a}_1\mathbf{b}_1) + (\mathbf{q_C})_1 = 0 \\
&(\mathbf{q_L})_2 \cdot \mathbf{a}_2 + (\mathbf{q_R})_2 \cdot \mathbf{b}_2 + (\mathbf{q_O})_2 \cdot \mathbf{c}_2 + (\mathbf{q_M})_2 \cdot (\mathbf{a}_2\mathbf{b}_2) + (\mathbf{q_C})_2 = 0 \\
&\ldots \\
&(\mathbf{q_L})_n \cdot \mathbf{a}_n + (\mathbf{q_R})_n \cdot \mathbf{b}_n + (\mathbf{q_O})_n \cdot \mathbf{c}_n + (\mathbf{q_M})_n \cdot (\mathbf{a}_n\mathbf{b}_n) + (\mathbf{q_C})_n = 0.
\end{aligned} \tag{3.12}$$

等式中的各项均为一个变量。

对于一类变量，如 $(\mathbf{q_L})_i$，将其表示为二维坐标点 $(i, (\mathbf{q_L})_i)$ 的形式。其中横坐标表示门的标号 i，纵坐标表示 i 号门左输入的 selector 值 $(\mathbf{q_L})_i)_i$。电路构建完成后，可以得到坐标点

$$(0, (\mathbf{q_L})_0),\ (1, (\mathbf{q_L})_1), \ldots,\ (n, (\mathbf{q_L})_n). \tag{3.13}$$

根据以上横纵坐标的对应关系，利用拉格朗日插值的方式可以确定多项式

$$\mathbf{q_L}(x) = (\mathbf{q_L})_0 l_0(x) + (\mathbf{q_L})_1 l_1(x) + \cdots + (\mathbf{q_L})_n l_n(x). \tag{3.14}$$

其中 $l_0(x), l_1(x), \ldots, l_n(x)$ 被称为拉格朗日基，其计算方式为

$$\begin{aligned} l_j(x) :&= \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i} \\ &= \left(\frac{x-x_0}{x_j-x_0}\right)\left(\frac{x-x_1}{x_j-x_1}\right)\cdots\left(\frac{x-x_{j-1}}{x_j-x_{j-1}}\right)\left(\frac{x-x_{j+1}}{x_j-x_{j+1}}\right)\cdots\left(\frac{x-x_n}{x_j-x_n}\right). \end{aligned} \tag{3.15}$$

由此可以计算出满足所有电路门的 $\mathbf{q_L}(x)$ 多项式。其余所有变量均按照同样的方式进行点值到多项式的转换。得到

$$\mathrm{P}(x) = \mathbf{q_L}(x)\mathbf{a}(x) + \mathbf{q_R}(x)\mathbf{b}(x) + \mathbf{q_O}(x)\mathbf{c}(x) + \mathbf{q_M}(x)\mathbf{a}(x)\mathbf{b}(x) + \mathbf{q_C}(x) = 0. \tag{3.16}$$

第三节　复制约束

多项式（3.16）实现了对电路中各个门约束的包含和压缩，但尚未将这些离散的门连接起来组成一个整体的电路。复制约束（Copy constrain）通过置换（permutation）的方式使电路中属于同一根导线，但涉及两个或多个门的变量连接起来。

由于线与线之间的联系不局限于单独某一类输入，如 i 号门的左输入与 j 号门的右输入相等，即 $\mathbf{a}(i) = \mathbf{b}(j)$，因此要将左输入 $\mathbf{a}(x)$，右输入 $\mathbf{b}(x)$ 与输出 $\mathbf{c}(x)$ 三个多项式的自变量索引加以区分，整合到一起。定义变量 $u_i, s_i (i \in [1, N], N = 3n)$，其中 $\{u_1, \ldots, u_n\}$ 表示左输入线值，$\{u_{n+1}, \ldots, u_{2n}\}$ 表示右输入线值，$\{u_{2n+1}, \ldots, u_{3n}\}$ 表示输出线值。$\{s_1, \ldots, s_n\}$ 表示左输入线的标号，$\{s_{n+1}, \ldots, s_{2n}\}$ 表示右输入线的标号，$\{s_{2n+1}, \ldots, s_{3n}\}$ 表示输出线的标号。对于门 i，其左输入线的标号为 s_i，右输入为 s_{i+n}，输出为 s_{i+2n}。

定义置换

$$\sigma(s_1, s_2, \ldots, s_N) = ({S_1}^{\sigma}, \ldots, {S_N}^{\sigma}). \tag{3.17}$$

直觉上，可以认为集合 $\mathrm{U} = \{u_1, \ldots, u_N\}$ 中包含若干个块（block），每个块中所有元素均相等，但这些元素在集合 U 中可以是分散排列的。置换 σ 将线值相等

的导线标号进行挪一位的置换，而导线的值集合顺序不变。因而定义

$$\mathrm{T}(i)=\frac{(u_i+\beta s_i+\gamma)}{(u_i+\beta s_i^{\sigma}+\gamma)}=1, \tag{3.18}$$

$$\mathrm{Z}(i)=\prod_{1<j<i}^{N}\frac{(u_i+\beta s_i+\gamma)}{(u_i+\beta s_i^{\sigma}+\gamma)}=1. \tag{3.19}$$

对于 $x\in[1,N],\mathrm{Z}(x)$ 满足

$$\begin{cases}\mathrm{Z}(x+1)=\mathrm{Z}(x)\mathrm{T}(x)\\ L_1(x)(\mathrm{Z}(x)-1)=0.\end{cases} \tag{3.20}$$

为了实现复制约束的首尾相接特性，将门的标号映射到有限域 $\mathbb{F}_p$ 上的循环子群 $H=\{\omega,\omega^2,...,\omega^n\}$。于是

$$\begin{aligned}&\mathrm{Z}(\omega x)=\mathrm{Z}(x)\mathrm{T}(x)\\ &\mathrm{Z}(\omega x)-\mathrm{Z}(x)\mathrm{T}(x)=0.\end{aligned} \tag{3.21}$$

同时，多项式（3.11）变为

$$\mathrm{Z}_H(x)=(x-\omega)(x-\omega^2)\ldots(x-\omega^n)=x^n-1. \tag{3.22}$$

当 $x\in\mathrm{H},\exists t_g(x)$ 使得 $\mathrm{P}(x)=t_g(x)\mathrm{Z}_{\mathrm{H}}(x)$，同时 $\exists t_c(x)$，满足 $\mathrm{Z}(\omega x)-\mathrm{Z}(x)\mathrm{T}(x)=t_c(x)\mathrm{Z}_{\mathrm{H}}(x)$。

第四节　PLONK 协议执行步骤

上述过程将最终的证明简化为了对门约束与复制约束两类多项式的证明。即 $\exists\mathbf{a}(x),\mathbf{b}(x),\mathbf{c}(x),\mathrm{Z}(x),t_g(x),t_c(x)$6 个只有 P 知道的秘密多项式，使得

$$\begin{cases}\mathbf{q_L}(x)\mathbf{a}(x)+\mathbf{q_R}(x)\mathbf{b}(x)+\mathbf{q_O}(x)\mathbf{c}(x)+\mathbf{q_M}(x)\mathbf{a}(x)\mathbf{b}(x)+\mathbf{q_C}(x)=t_g(x)\mathrm{Z}_{\mathrm{H}}(x)\\ \mathrm{Z}(\omega x)-\mathrm{Z}(x)\mathrm{T}(x)=t_c(x)\mathrm{Z}_{\mathrm{H}}(x).\end{cases} \tag{3.23}$$

3.4.1 预处理

定义 $[x]_1 := x \cdot g$，g 是群 $\mathbb{G}_1$ 的生成元。首先进行 Setup，生成 SRS$(x \cdot [1]_1, \ldots, x^n \cdot [1]_1)$，同时也是 CommitKey。再对电路进行预处理，得到 selector 与 σ 的点值坐标后，由拉格朗日插值法计算出 selector 多项式：

$$\begin{aligned}
\mathrm{q_M}(X) &= \textstyle\sum_{i=1}^{n} q_{Mi}\, \mathrm{L}_i(X) \\
\mathrm{q_L}(X) &= \textstyle\sum_{i=1}^{n} q_{Li}\, \mathrm{L}_i(X) \\
\mathrm{q_R}(X) &= \textstyle\sum_{i=1}^{n} q_{Ri}\, \mathrm{L}_i(X) \\
\mathrm{q_O}(X) &= \textstyle\sum_{i=1}^{n} q_{Oi}\, \mathrm{L}_i(X) \\
\mathrm{q_C}(X) &= \textstyle\sum_{i=1}^{n} q_{Ci}\, \mathrm{L}_i(X) \\
\mathrm{S}_{\sigma 1}(X) &= \textstyle\sum_{i=1}^{n} \sigma(i) \mathrm{L}_i(X) \\
\mathrm{S}_{\sigma 2}(X) &= \textstyle\sum_{i=1}^{n} \sigma(n+i) \mathrm{L}_i(X) \\
\mathrm{S}_{\sigma 3}(X) &= \textstyle\sum_{i=1}^{n} \sigma(2n+i) \mathrm{L}_i(X).
\end{aligned} \tag{3.24}$$

3.4.2 证明者 P 的算法

第一阶段：

选择随机盲因子 $(b_1, \ldots, b_9) \in \mathbb{F}_p$，计算输入的线值多项式

$$\begin{aligned}
\mathrm{a}(X) &= (b_1 X + b_2)\, \mathrm{Z_H}(X) + \textstyle\sum_{i=1}^{n} w_i\, \mathrm{L}_i(X) \\
\mathrm{b}(X) &= (b_3 X + b_4)\, \mathrm{Z_H}(X) + \textstyle\sum_{i=1}^{n} w_{n+i}\, \mathrm{L}_i(X) \\
\mathrm{c}(X) &= (b_5 X + b_6)\, \mathrm{Z_H}(X) + \textstyle\sum_{i=1}^{n} w_{2n+i}\, \mathrm{L}_i(X).
\end{aligned} \tag{3.25}$$

对多项式计算承诺 $[a]_1 := [\mathrm{a}(x)]_1, [b]_1 := [\mathrm{b}(x)]_1, [c]_1 := [\mathrm{c}(x)]_1$，并输出 $([a]_1, [b]_1, [c]_1)$。

第二阶段：

利用 Fiat-Shamir 启发式算法，计算随机挑战 $(\beta, \gamma) \in \mathbb{F}_p$

$$\beta = \mathscr{R}(\text{ transcript },0), \gamma = \mathscr{R}(\text{ transcript },1). \tag{3.26}$$

其中 $\mathscr{R}$ 为特定的哈希函数，transcript 包含目前为止的输出参数字节。随后计算

置换多项式

$$\begin{aligned}&\mathrm{z}(X)=\left(b_7X^2+b_8X+b_9\right)\mathrm{Z_H}(X)\\&+\mathrm{L}_1(X)+\sum_{i=1}^{n-1}\left(\mathrm{L}_{i+1}(X)\prod_{j=1}^{i}\frac{\left(w_j+\beta\omega^{j-1}+\gamma\right)\left(w_{n+j}+\beta k_1\omega^{j-1}+\gamma\right)\left(w_{2n+j}+\beta k_2\omega^{j-1}+\gamma\right)}{\left(w_j+\sigma(j)\beta+\gamma\right)\left(w_{n+j}+\sigma(n+j)\beta+\gamma\right)\left(w_{2n+j}+\sigma(2n+j)\beta+\gamma\right)}\right).\end{aligned}\tag{3.27}$$

计算其承诺 $[z]_1:=[\mathbf{z}(x)]_1$，输出 $([z]_1)$。

第三阶段：

用同样的方法计算商多项式的随机挑战 $\alpha=\mathscr{R}(\text{ transcript })\in\mathbb{F}_p$。计算商多项式

$$\begin{aligned}&\mathrm{t}(X)=\\&\quad(\mathrm{a}(X)\mathrm{b}(X)\mathrm{q_M}(X)+\mathrm{a}(X)\mathrm{q_L}(X)+\mathrm{b}(X)\mathrm{q_R}(X)+\mathrm{c}(X)\mathrm{q_O}(X)+\mathrm{PI}(X)+\mathrm{q_C}(X))\frac{1}{\mathrm{Z_H}(X)}\\&+((\mathrm{a}(X)+\beta X+\gamma)(\mathrm{b}(X)+\beta k_1X+\gamma)(\mathrm{c}(X)+\beta k_2X+\gamma)\mathbf{z}(X))\frac{\alpha}{\mathrm{Z_H}(X)}\\&-((\mathrm{a}(X)+\beta\mathrm{S}_{\sigma1}(X)+\gamma)(\mathrm{b}(X)+\beta\mathrm{S}_{\sigma2}(X)+\gamma)(\mathrm{c}(X)+\beta\mathrm{S}_{\sigma3}(X)+\gamma)\mathrm{z}(X\omega))\frac{\alpha}{\mathrm{Z_H}(X)}\\&+(\mathrm{z}(X)-1)\mathrm{L}_1(X)\frac{\alpha^2}{\mathrm{Z_H}(X)}.\end{aligned}\tag{3.28}$$

其中整合了门约束与式（3.20）的约束。将 $t(x)$ 分成高位与低位三部分 $t(X)=t_{lo}(X)+X^nt_{mid}(X)+X^{2n}t_{hi}(X)$。承诺并输出 $([t_{lo}]_1,[t_{mid}]_1,[t_{hi}]_1)$。

第四阶段：

计算 opening challenge：$\mathfrak{z}=\mathscr{R}(\text{ transcript })\in\mathbb{F}_p$ 以及 opening evaluations：

$$\begin{gathered}\bar{a}=\mathrm{a}(\mathfrak{z}),\bar{b}=\mathrm{b}(\mathfrak{z}),\bar{c}=\mathrm{c}(\mathfrak{z}),\bar{\mathrm{s}}_{\sigma1}=\mathrm{S}_{\sigma1}(\mathfrak{z}),\bar{\mathrm{s}}_{\sigma2}=\mathrm{S}_{\sigma2}(\mathfrak{z}),\\\bar{t}=\mathrm{t}(\mathfrak{z}),\bar{z}_\omega=\mathrm{z}(\mathfrak{z}\omega).\end{gathered}$$

计算线性关系多项式

$$\begin{aligned}&r(X)=\\&\quad\left(\bar{a}\bar{b}\cdot\mathrm{q}_M(X)+\bar{a}\cdot\mathrm{q}_L(X)+\bar{b}\cdot\mathrm{q_R}(X)+\bar{c}\cdot\mathrm{q_O}(X)+\mathrm{q_C}(X)\right)\\&+\left((\bar{a}+\beta\mathfrak{z}+\gamma)\left(\bar{b}+\beta k_{1\mathfrak{z}}+\gamma\right)(\bar{c}+\beta k_{2\mathfrak{z}}+\gamma)\cdot z(X)\right)\alpha\\&-\left((\bar{a}+\beta\bar{\mathrm{s}}_{\sigma1}+\gamma)\left(\bar{b}+\beta\bar{\mathrm{s}}_{\sigma2}+\gamma\right)\beta\bar{z}_\omega\cdot\mathrm{S}_{\sigma3}(X)\right)\alpha\\&+(\mathrm{z}(X))\mathrm{L}_1(\mathfrak{z})\alpha^2.\end{aligned}\tag{3.29}$$

对其计算 evaluation：$\bar{r}=r(z)$。输出 $\left(\bar{a},\bar{b},\bar{c},\bar{s}_{\sigma1},\bar{s}_{\sigma2},\bar{z}_\omega,\bar{t},\bar{r}\right)$。

第五阶段：

计算 opening challenge：$v = \mathscr{R}(\text{ transcript }) \in \mathbb{F}_p$。计算第一类证明多项式 $W_{\mathfrak{z}}(X)$：

$$W_{\mathfrak{z}}(X) = \frac{1}{X-\mathfrak{z}} \begin{pmatrix} \left(\mathrm{t}_{\mathrm{lo}}(X) + \mathfrak{z}^n \mathrm{t}_{\mathrm{mid}}(X) + \mathfrak{z}^{2n} \mathrm{t}_{\mathrm{hi}}(X) - \bar{t}\right) \\ +v(\mathrm{r}(X) - \bar{r}) \\ +v^2(\mathrm{a}(X) - \bar{a}) \\ +v^3(\mathrm{b}(X) - \bar{b}) \\ +v^4(\mathrm{c}(X) - \bar{c}) \\ +v^5(\mathrm{S}_{\sigma 1}(X) - \bar{\mathrm{s}}_{\sigma 1}) \\ +v^6(\mathrm{S}_{\sigma 2}(X) - \bar{\mathrm{s}}_{\sigma 2}) \end{pmatrix}. \tag{3.30}$$

计算第二类证明多项式 $\mathrm{W}_{\mathfrak{z}\omega}(X)$：

$$\mathrm{W}_{\mathfrak{z}\omega}(X) = \frac{(\mathrm{z}(X) - \bar{z}_\omega)}{X - \mathfrak{z}\omega}. \tag{3.31}$$

承诺并输出 $\left([W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1\right)$。

P 返回证明 π：

$$\pi_{\mathrm{SNARK}} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mid}]_1, [t_{hi}]_1, [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{\mathbf{s}}_{\sigma 1}, \bar{\mathbf{s}}_{\sigma 2}, \bar{r}, \bar{z}_\omega \end{pmatrix}. \tag{3.32}$$

最后计算 multipoint evaluation：$u = \mathscr{R}(\text{ transcript })$。

3.4.3 验证者 V 的算法

V 为预处理后的 selector 多项式与 σ 多项式计算承诺，同时也是 VerifierKey：

$$\begin{aligned} &[q_{\mathrm{M}}]_1 := \mathrm{q}_{\mathrm{M}}(x) \cdot [1]_1, [q_{\mathrm{L}}]_1 := \mathrm{q}_{\mathrm{L}}(x) \cdot [1]_1, [q_{\mathrm{R}}]_1 := \mathrm{q}_{\mathrm{R}}(x) \cdot [1]_1, [q_{\mathrm{O}}]_1 := \mathrm{q}_{\mathrm{O}}(x) \cdot [1]_1, \\ &[s_{\sigma 1}]_1 := \mathrm{S}_{\sigma 1}(x) \cdot [1]_1, [s_{\sigma 2}]_1 := \mathrm{S}_{\sigma 2}(x) \cdot [1]_1, [s_{\sigma 3}]_1 := \mathrm{S}_{\sigma 3}(x) \cdot [1]_1 \\ &x \cdot [1]_2. \end{aligned} \tag{3.33}$$

电路包含 ℓ 个 Public Input:$(w_i)_{i \in [\ell]}$。V 利用掌握的知识 $\mathbf{V}\left((w_i)_{i \in [\ell]}, \pi_{\mathrm{SNARK}}\right)$ 执行：

一、验证

$$\begin{aligned}&([a]_1,[b]_1,[c]_1,[z]_1,[t_{lo}]_1,[t_{mid}]_1,[t_{hi}]_1,[W_z]_1,[W_{z\omega}]_1)\in\mathbb{G}_1\\&\left(\bar{a},\bar{b},\bar{c},\bar{\mathbf{s}}_{\sigma 1},\bar{\mathbf{s}}_{\sigma 2},\bar{r},\bar{z}_{\omega}\right)\in\mathbb{F}_p^7\\&(w_i)_{i\in[\ell]}\in\mathbb{F}_p^{\ell}\end{aligned}\tag{3.34}$$

二、分别计算

$$\begin{cases}\beta,\gamma,\alpha,\mathfrak{z},v,u\in\mathbb{F}_p\\ \mathrm{Z_H}(\mathfrak{z})=\mathfrak{z}^n-1\\ L_1(\mathfrak{z})=\frac{\mathfrak{z}^n-1}{n(\mathfrak{z}-1)}\\ \mathrm{PI}(\mathfrak{z})=\sum_{i\in\ell}w_i\,\mathrm{L}_i(\mathfrak{z}),\end{cases}\tag{3.35}$$

计算商多项式 evaluation

$$\bar{t}=\frac{\bar{r}+\mathrm{PI}(\mathfrak{z})-\left(\left(\bar{a}+\beta\bar{\mathbf{s}}_{\sigma 1}+\gamma\right)\left(\bar{b}+\beta\bar{\mathbf{s}}_{\sigma 2}+\gamma\right)(\bar{c}+\gamma)\bar{z}_{\omega}\right)\alpha-\mathrm{L}_1(\mathfrak{z})\alpha^2}{\mathrm{Z_H}(\mathfrak{z})}.\tag{3.36}$$

计算组合多项式承诺的第一部分 $[D]_1:=v\cdot[r]_1+u\cdot[z]_1$：

$$\begin{aligned}&\bar{a}\bar{b}v\cdot[q_{\mathrm{M}}]_1+\bar{a}v\cdot[q_{\mathrm{L}}]_1+\bar{b}v\cdot[q_{\mathrm{R}}]_1+\bar{c}v\cdot[q_{\mathrm{O}}]_1+v\cdot[q_{\mathrm{C}}]_1\\[D]_1:=&+\left(\left(\bar{a}+\beta\mathfrak{z}+\gamma\right)\left(\bar{b}+\beta k_{1\mathfrak{z}}+\gamma\right)\left(\bar{c}+\beta k_{2\mathfrak{z}}+\gamma\right)\alpha v+\mathrm{L}_1(\mathfrak{z})\alpha^2v+u\right)\cdot[z]_1\\&-\left(\bar{a}+\beta\bar{\mathbf{s}}_{\sigma 1}+\gamma\right)\left(\bar{b}+\beta\bar{\mathbf{s}}_{\sigma 2}+\gamma\right)\alpha v\beta\bar{z}_{\omega}\,[s_{\sigma 3}]_1\,.\end{aligned}\tag{3.37}$$

计算完整的组合多项式承诺 $[F]_1$：

$$\begin{aligned}[F]_1:=\;&[t_{lo}]_1+\mathfrak{z}^n\cdot[t_{mid}]_1+\mathfrak{z}^{2n}\cdot[t_{hi}]_1\\&+[D]_1+v^2\cdot[a]_1+v^3\cdot[b]_1+v^4\cdot[c]_1+v^5\cdot[s_{\sigma 1}]_1+v^6\cdot[s_{\sigma 2}]_1\,.\end{aligned}\tag{3.38}$$

计算组合 evaluation 到椭圆曲线子群的映射 $[E]_1$：

$$[E]_1:=\begin{pmatrix}\bar{t}+v\bar{r}+v^2\bar{a}+v^3\bar{b}+v^4\bar{c}\\+v^5\bar{s}_{\sigma 1}+v^6\bar{s}_{\sigma 2}+u\overline{z_{\omega}}\end{pmatrix}\cdot[1]_1.\tag{3.39}$$

三、组合 pairing 验证

$$e\left([W_{\mathfrak{z}}]_1+u\cdot[W_{\mathfrak{z}\omega}]_1,[x]_2\right)\stackrel{?}{=}e\left(\mathfrak{z}\cdot[W_{\mathfrak{z}}]_1+u\mathfrak{z}\omega\cdot[W_{\mathfrak{z}\omega}]_1+[F]_1-[E]_1,[1]_2\right).\tag{3.40}$$

第四章　PLONK 应用实验

第一节　证明电路设计

4.1.1　场景设置

本文设计的零知识证明场景为:

表 4.1: *P* 与 *V* 各自的知识

证明者 *P* 要证明的知识	知道 1024 个人各自的工资，在不透露具体数值的情况下，证明其和为某个公开的数字 *sum*
验证者 *V* 知道的知识	知道这个公开的和 *sum*

证明过程需要：*P* 输入 1024 个人各自的工资数值以及其总和 *sum*，*V* 输入公开的工资和 *sum* 以及 *P* 提供的证明进行验证。

4.1.2　电路设计

实现 1024（2^{10}）个输入的相加，首先需要 512 个加法门对其两两相加，得到 512（2^9）个中间结果，用 olayers[0] 表示。再构建 256（2^8）个加法门，将得到的 512 个中间结果两两相加得到 256 个新的中间结果，存入 olayers[1]。不断累加，得到 olayers[9] 即输出的和 *sum*，同时也是 Public Input。电路总共需要 1023 个加法门。

PLONK 由 Rust 语言实现。构建结构 TestCircuit 用于存储 *P* 的 1024 个输入和:

```
1 pub struct TestCircuit {
2     ilayer:Vec<BlsScalar>, // '用户输入层'
3     olayers:Vec<Vec<BlsScalar>>,  // '中间结果层'
4     e: JubJubScalar,
5     f: JubJubAffine,
6 }
```

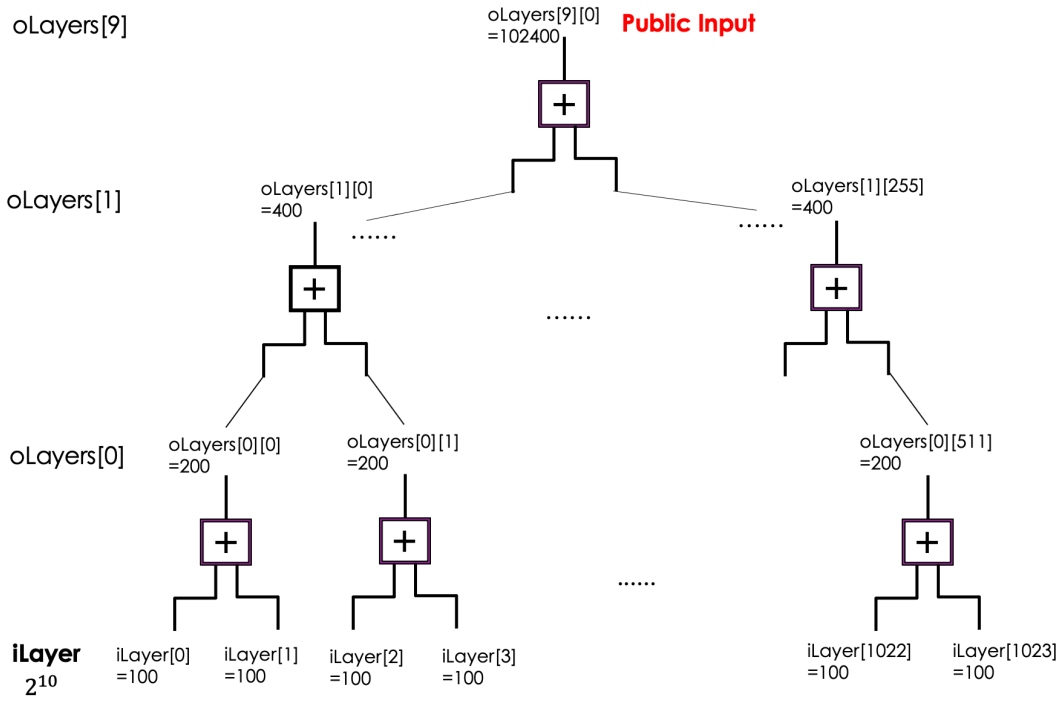

图 4.1: 电路示意图

4.1.3 Setup

首先 P 与 V 双方明确证明需求，根据电路大小，执行 setup 步骤，生成包含 CommitKey 与 OpeningKey 的 SRS。其中 CommitKey 用于对多项式系数做承诺

表 4.2: CommitKey，本电路中 $\beta = 2$

CommitKey
$\{g_1, g_1{}^{\beta}, g_1{}^{\beta^2}, \ldots, g_1{}^{\beta^{degree}}\}([\beta^i]_1, i \in [0, degree])$

OpeningKey 由 Verifier 用于做验证

表 4.3: OpeningKey，本电路中 $\beta = 2$

OpeningKey
$\{g_1, g_2, g_2{}^{\beta}, g_{2_{prepared}}, g_{2_{prepared}}{}^{\beta}\}$,($g_{2_{prepared}}$ 用于做 pairing)

4.1.4 电路构建

电路构建的阶段无需 P,V 双方参与。当明确电路需求后，即可以根据电路图确定各个门的类型，即 selector 值。由程序的公共部分利用 composer 结构为

selector 数组赋值。本电路利用遍历数组的方式，循环生成电路门。此阶段需要重新编写代码初始电路的构建代码，为电路添加门约束并为每根导线赋随机初值。以中间层 olayers 的赋值为例：

Listing 4.1: 为中间层导线赋随机初值

```
let mut olayers_vec = Vec::with_capacity(po);
for i in (0..=(po-1)).rev(){
    let mut layer =
    Vec::with_capacity(base_two.pow(i as u32) as usize);
    for j in 0..(base_two.pow(i as u32)){
        value = rng.gen();
        layer.push(BlsScalar::from(value));
    }
    olayers_vec.push(layer);
}
```

以一个加法门为例，调用函数 *poly_gate* 添加算数门约束：

Listing 4.2: 添加加法门

```
composer.poly_gate(
                ilayer[ii],
                ilayer[ii+1],
                olayers[0][ii/2],
                BlsScalar::zero(),
                BlsScalar::one(),
                BlsScalar::one(),
                -BlsScalar::one(), // q_o = -1
                BlsScalar::zero(),
                None,  // pi
            );
```

函数 *gadget* 用于完成整个电路的门约束生成，其中调用的 *poly_gate* 用于添加单个门，填充公式 (3.12) 中的各个数值。此时由于 P,V 尚未参与电路的输入，因此这一阶段只是建立电路门的约束关系，线值用随机数填充。与此同时将同一根导线上的门输入添加到构建置换 Permutation 结构的 variable_map 中。

Listing 4.3: 添加导线到置换结构中

```
1 pub(crate) struct Permutation {
2     pub(crate) variable_map:
3         HashMap<Variable, Vec<WireData>>,}
```

如导线 **a** 关联了 2 号门的输出、3 号门的右输入和 5 号门的左输入，则它在 variable_map 中的存储形式应为：< **a**, [Right (3), Output (2), Left (5)] >。

随后执行 *preprocess*，利用 ifft 对数组进行拉格朗日插值变换，转化为多项式 (3.16)。生成 ProverKey 与 VerifierKey。ProverKey 用于后续 P 构建证明时的计算，是系数与点值形式的电路多项式（包含 selector 多项式与 σ 多项式的明文）。VerifierKey 用于 P 输出证明 π 后，V 验证时的计算过程，包含电路多项式的承诺 (4.6)。其结构如表所示

表 4.4: ProverKey，包含各种类型约束的子 ProverKey

ProverKey
n （电路门个数）
arithmetic
logic
range
fixed_base
permutation
variable_base
v_h_coset_4n

每个子 key 包含了系数与点值形式的具体多项式。如 arithmetic 包含了 $\mathbf{q_M}, \mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_C}, \mathbf{q_4}, \mathbf{q}_{arith}$7 个多项式的系数形式与点值形式。其中 $\mathbf{q_4}$ 为具体实现代码中的第三个输入门 seletor，$\mathbf{q}_{arith}$ 为在公式 (3.16) 基础上添加的对算数门的选择参数。即

$$(\mathbf{q_L}(x)\mathbf{a}(x) + \mathbf{q_R}(x)\mathbf{b}(x) + \mathbf{q_O}(x)\mathbf{c}(x) + \mathbf{q_M}(x)\mathbf{a}(x)\mathbf{b}(x) + \mathbf{q_C}(x))\mathbf{q}_{arith} = 0. \quad (4.1)$$

VerifierKey 的结构为

表 4.5: VerifierKey，包含各种类型约束的子 VerifierKey

VerifierKey
n （电路门个数）
arithmetic
logic
range
fixed_base
variable_base
v_h_coset_4n

每个子 key 包含了对各个多项式的承诺（Commitment）。如 arithmetic 包含了 7 个 selector 多项式的承诺。对证明过程中生成的 VerifierKey 进行解析并输出，得到文本文件，其中的算数门子 key 为：

表 4.6: 本例中 VerifierKey 生成的 arithmetic 子 key

Selector polynomial	Commitment
$\mathbf{q_M}$	8362848c9a0a622504b468490c6335999a09627c4db2d38e 8e7cc1a4160e5e3214374d9ef22f1bade6b7d9d8021cd786
$\mathbf{q_L}$	9684bcf46facc2cb95f1ac0d1e234344a59d6ed2b2a5826d b5be8de9057db3612a4a93336558aebb0fa0be00adbaaf2a
$\mathbf{q_R}$	b0898cba31c0cc419e36e11d5fb1848b1720c467df9dacc2 5d02b3af4b90d056f699422e66c8ae34cd275161ad89e1d9
$\mathbf{q_O}$	b0898cba31c0cc419e36e11d5fb1848b1720c467df9dacc2 5d02b3af4b90d056f699422e66c8ae34cd275161ad89e1d9
$\mathbf{q_C}$	910b7f0987832de73ecbc0af7497dfafb0af599a971113aa 99ea0195633a111ebb45b50be3bee66a2a93746280a6cfc2
$\mathbf{q_4}$	8c2aede0a6f1cca1e58620f7106860bd9ac3b8a157a19fbf 604d50f09f1c34b357a3d18f85a3eb168071ce59f0a855d7
$\mathbf{q_{ariht}}$	96976b408304058c097b9b162d8e07c3ebc16913e4ef8279 57759d62d85a47fbbc80e19675ceaaf6d4d4da6203df78cb

第二节　证明构建与验证过程

在证明的构建步骤中，为 TestCircuit 实现函数 *gen_proof*，证明者 *P* 向 TestCircuit 对象中输入 ilayer 与 olayers 中的每个数值。重新利用 *gadget* 函数生成包含真实 **a**,**b**,**c** 等线值的证明电路，并将该电路信息加入到对象 prover 中。

prover 调用函数 *prove* 计算证明。*prove* 函数首先调用 *preprocess_prover*，再进入 *preprocess_shared* 将待证明的 selector 等电路公共多项式用 ifft 变为系数形式，

```
1 let q_m_poly =
2 Polynomial::from_coefficients_slice(&domain.ifft(&self.q_m));
```

该函数同时为生 Verifier 生成了 transcript，

```
1 verifier_key.seed_transcript(transcript);
```

并利用电路公共多项式的的系数计算出这些多项式的承诺，生成 VerifierKey。

```
1 let q_l_poly_commit =
2 commit_key.commit(&q_l_poly).unwrap_or_default();
```

随后回到 *preprocess_prover* 将多项式转换回点值形式。再通过 *prove_with_preprocessed*，先将点值形式多项式通过 ifft 转化为系数形式，

```
1 let w_l_poly =
2 Polynomial::from_coefficients_vec(domain.ifft(w_l_scalar));
```

再利用 CommitKey 和包含 transcript 信息的 ProverKey 为得到的多项式计算承诺，写入 *P* 的私有变量 ProverKey。

计算承诺的函数是由 CommitKey 结构实现的 *commit*：

```
1 let w_l_poly_commit = commit_key.commit(&w_l_poly)?;
```

得到各个多项式的承诺以及利用 transcript 生成的挑战值计算 evaluation，得到证明 π（3.32）。

表 4.7: π: Commitments and Opening proofs

线值多项式	Commitment
$[a(x)]_1$	90c09c18dd7de4afce1e24f9d840fca2c4b06f3a48d2c5cf 26ac7b8be002c64477b3327b53ef02767d272efabb9e9c63
$[b(x)]_1$	809e950480589a7128245ee2dd643df147772499d39f7148 0855458928e49743a253abce9054e73a98b7dce34c3dd654
$[c(x)]_1$	b0df2dbf6b6db0240f61c5eb89183dd00b45e699b14fd432 3aa655ba1fafca0d6f67201eb14f97f67e47985d09fca3e3
$[d(x)]_1$	afea663f4e7a0f55cac66c88edaac4bd2d920db1c2bb6cb6 23f328c953ecd95ee62023f4e7eff0f20b5668bc36371693
置换多项式	Commitment
$[z(x)]_1$	ad1f40489540b911a1c41734d742c2ebfeec12603b9f3f9e 7d58b91b398ad9c95dfec84dcc9be5d27bb2247665f0dd9c
商多项式	Commitment
$[t_1(x)]_1$	890bd5d6aa2ec57e374b94a1f481304efd8af7e971c4d514 10221035717b1c106bb6e4ca746c93493207c971170c2cc4
$[t_2(x)]_1$	98584eac0a0cee7fc5d85890412a75f29c9f738f0a463d62 8e937560c965b35a63d1ef4beedb4edb5b93b4f79801fe17
$[t_3(x)]_1$	812bad46209e586308da6714da24500079cede0a40fa91c2 148d7adb09df1b73b1d6eec1f0eaf332e207db58c4602a09
$[t_4(x)]_1$	a317aee9768052e6ef51a22b7cf4c3b4322b27c2da720a6f 72db085d58011b8e1cb584a0635d843049473233f4ebbb5c
Opening proof	Commitment
$[W_{\mathfrak{z}}]$	8724cb5b23001a325c8fa5b2ddc25ebd28babec78bad0a5f e9c09a2bbf521e8344fa7bf1b69b1629e7ea93a224cf911e
$[W_{\mathfrak{z}\omega}]$	81893127a089ac4ba1142efcdf0f789ae7e3aed8563163ff 7c1188590429f662cff1048104ef026d676e150fb2c5ed2e

表 4.8: π: Opening evaluations

Opening Evaluations	Evaluation
$\bar{a}$	b8f585e45b8baa92b513a1369239f20a 9a88c4d34dd274cf1117f5091ca89539
$\bar{b}$	30a93089a73e6142c5df0a4ddd7d80b2 0475bdb734f9c6bf260c8156fb97be50
$\bar{c}$	48bf6e93569cc68338198a8a1431de32 c9f6e595be8fba2a6a5fc7e930271565
$\bar{d}$	3d46d3c5d1dc5500aa39b4a5d57bddae da56022c45cc5d1744650f5926d7f052
$\bar{s}_{\sigma 1}$	5cb7f2da36b0c72446785ce4126a4826 76b33b74c7385f8c7b67d8112774b239
$\bar{s}_{\sigma 2}$	0079d24c7b1f5c3698d40f507b6032fd 989ad8822005abb54c7e58d75b566619
$\bar{r}$	f82bd03ac02febdd4e3639cf344045bc e6507c671d2435486eae23fff2a2e760
$\bar{z}_{\omega}$	38e96d1f7b56f870ca64e2ffa1aa9401 443215fe045ce86d94230aabd3032560

在证明的验证环节，V 输入 Public input，调用函数 *verify_proof* 开始验证。对象 proof 调用函数 *verify* 输入参数 verify_key,opening_key 和 public_input。依次生成 transcript 并按顺序生成挑战值 $\beta, \gamma, \alpha, \mathfrak{z}$。计算 Zero 多项式 $Z_H(\mathfrak{z})$，$\mathfrak{z}$ 处的第一个拉格朗日多项式 $L_1(\mathfrak{z})$，$\mathfrak{z}$ 处的商多项式 $\bar{t}$。随后依次计算 $[D]_1$(公式3.37), $[F]_1$(公式3.38), $[E]_1$(公式3.39)，并利用 opening_key 调用函数 *batch_check* 验证 pairing（公式3.40）。

第三节　运行性能

4.3.1　性能的重要性

对于通过分布式存储与计算实现的区块链技术而言，空间储存性能与计算运行时间性能均会对其实用性产生很大影响。零知识证明作为区块链技术的重

要组成部分之一，对其时间与空间性能的提升是十分值得关注的。

当使用基于区块链技术产生的加密货币，如 Zcash，进行匿名交易时，其单笔交易的成立需要运用零知识证明技术进行隐私保护和合法性认证。为了满足实际使用中对大量交易进行快速确认的需求，加密货币在实用性上需要解决的问题之一是交易确认的速度。目前纽约证券交易所平均每秒处理的交易量大约为 261.12 笔，即 261.12 tps（Transaction per second），VISA 更是达到了 2000tps。这意味着如果要满足同等水平的交易平台应用需求，对单笔交易的认证时间（Proof verification time）应达到 3.8ms 甚至 0.5ms 的级别。运用零知识证明技术对交易进行合法性认证的速度应当向这一标准靠拢，甚至超越。而 Zcash 目前的平均交易吞吐量为 27tps，比特币仅为 7tps[21]，难以满足达到如上要求。

4.3.2 性能测试与分析

本节在 1024 输入加法门电路的基础上进行了进一步设计。改变输入个数与电路门个数，对不同门数量的电路中生成证明（*gen_proof* 函数）与证明验证（*verify_proof* 函数）两部分的运行时间进行测试。同时，将每个电路运行过程中生成的 ProverKey 与 VerifierKey 进行了解析并写入文本文件，比较门的数量与生成的文本文件占用空间大小之间的关系。

本文实验的测试平台为 Macbook pro(15-inch, 2017)，内存 16 GB，处理器为 2.8 GHz 四核 Intel Core i7。

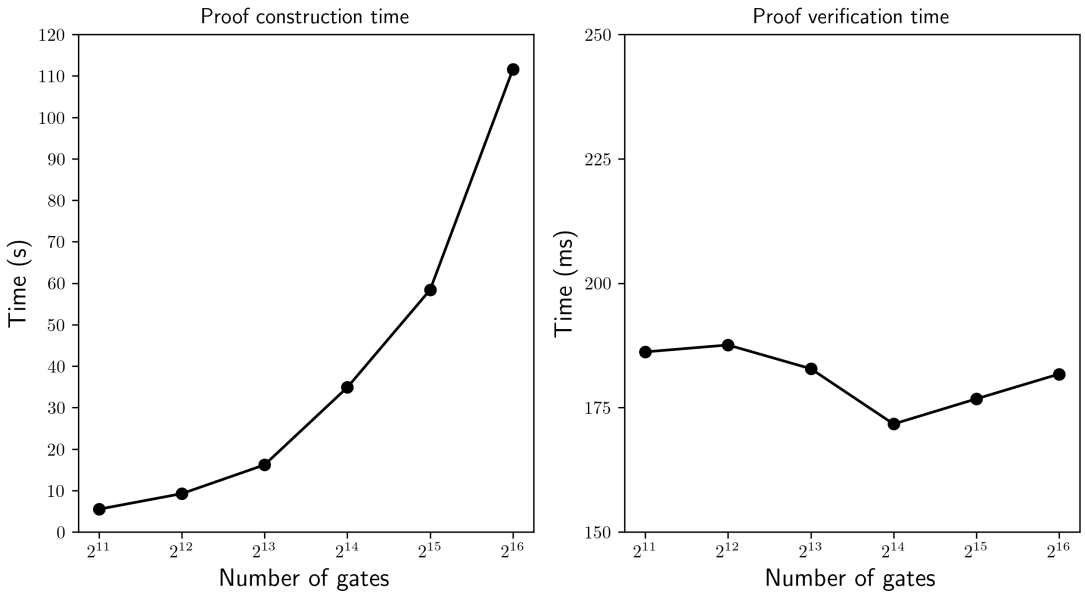

图 4.2: 时间测试

由图（4.2）可见对于不同大小的电路，PLONK 协议的证明生成时间（Proof construction time）会随着电路门个数变多而增加。并且电路门数量的增长速率是以 2 为底的指数式增长，证明的生成时间也呈现了等比例的增长趋势。可以推测证明生成时间与电路门数量 n 成正比，其时间复杂度为 $O(n)$。但验证时间（Proof verification time）与门的数量在当前测试的范围内不具备明显的相关性，验证时间相对稳定。

PLONK 协议在执行验证的步骤中首先经历了对固定大小的承诺的计算，计算步骤固定为挑战值代入多项式计算；随后的双线性对验证过程同样仅涉及对大小固定的承诺的计算，并且 pairing 操作与电路门个数 n 无关。因此验证过程的时间复杂度为 $O(1)$。

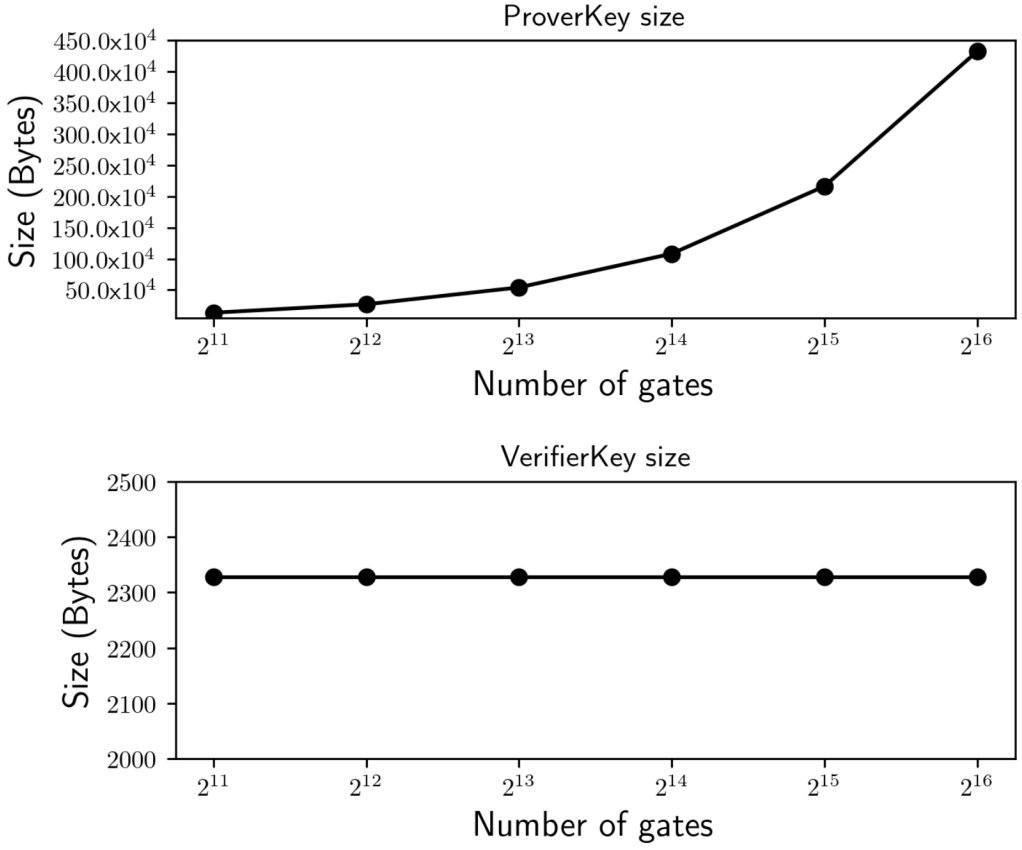

图 4.3: 空间测试（Keys）

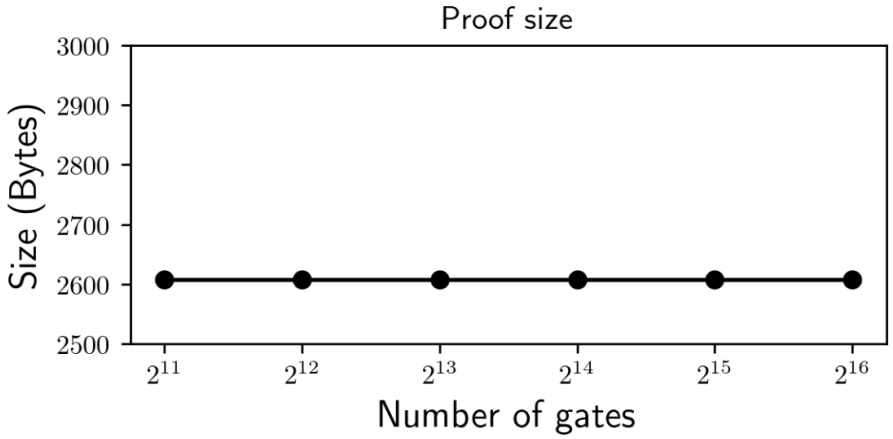

图 4.4: 空间测试（Proof）

由图（4.3）与（4.4）可见，VerifierKey 与 Proof 的大小是一个常量，其空间复杂度为 $O(1)$，占用的空间不会随电路中门数量的变化而改变。这是因为 VerifierKey 中存储了固定个数多项式的承诺，而承诺本身是一个椭圆曲线中的点，其大小是固定的。承诺的结构如下:

```
1     pub(crate) struct Commitment(
2     /// The commitment is a group element.
3     pub(crate) G1Affine,
4 );
```

而 ProverKey 的大小会随着电路门数量的增多而随之增加。ProverKey 中存储的主要内容为各个多项式的系数形式与点值（Evaluation）形式。需要存储的多项式系数个数与多项式的阶数成正比。而多项式的阶数即电路中门的个数。当电路门的数量增加，同时意味着多项式的阶数随之增加，ProverKey 中需要存储的多项式系数数量会同步等比例增长。因此，当电路门个数 n 按照以 2 为底的指数增长时，ProverKey 文件大小会不断等比例以 2 倍的速度增大，其空间复杂度为 $O(n)$。但 ProverKey 的内容属于证明生成时的中间环节，实际应用中不会被存储在区块链上，因此对算法应用的性能影响较小。

4.3.3 性能优化思路

对 *gen_proof* 过程中的各个环节（Round 1-5）的运行时间进行测试， 由图（4.5）可见，*gen_proof* 各个环节中时间开销最大的 Round 3，生成商多项式的部分（公式3.28），以及 Round 1，计算线值多项式并对其进行承诺的步骤（公

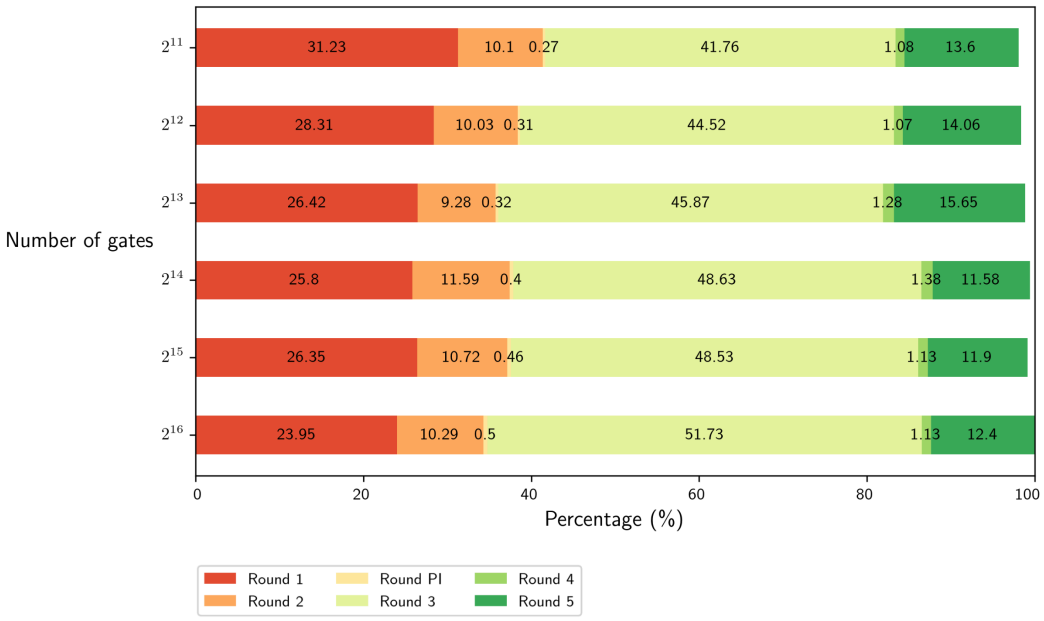

图 4.5: 不同数量门电路生成证明的各阶段时间测试

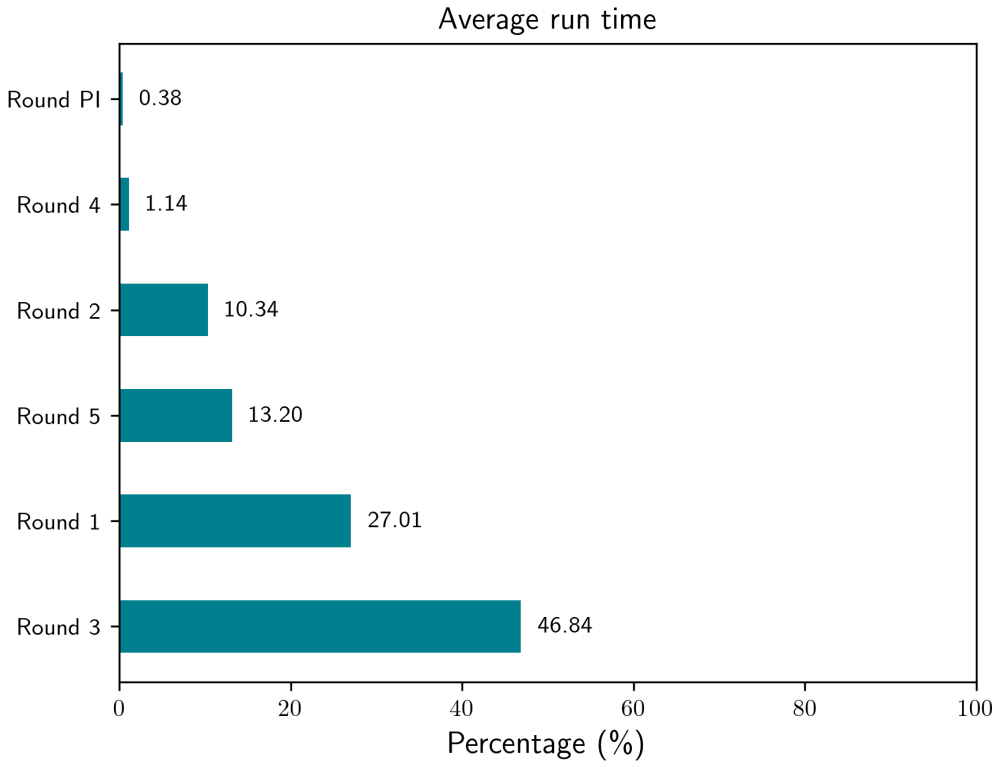

图 4.6: 生成证明的阶段平均时间测试

式3.25)。其中 Round 1 中的操作涉及四步:

1. 将线值变量转换为用于计算的数据结构。

2. 利用 ifft 将点值形式的线值多项式转化为系数形式。

3. 对系数形式的多项式计算承诺。

4. 将承诺加入到 transcript。

对这四个步骤的运行时间进行进一步测试，发现步骤 3，计算多项式承诺步骤的运行时间占 Round 1 总时间的的 97.10%。承诺步骤需要利用 CommitKey，对 4 个多项式进行总共 4n 个椭圆曲线点与大整数的乘法计算。

而在 Round 3 的计算中，也涉及了对 4 个 n 次多项式 t_1,t_2,t_3,t_4，即商多项式的 4 个幂次区间，进行承诺的计算步骤。这部分的开销与 Round 1 的第三步大致相同，但在计算承诺前，Round 3 比 Round 1 额外多了一部分对商多项式的计算开销。由图（4.5）可见随着门数量的增加，Round 1 耗时占比有降低的趋势，而 Round 3 的耗时占比反而有增加趋势。Round 3 中除去承诺之外的商多项式计算步骤可能是导致这一趋势的原因，这意味着商多项式计算的时间复杂度要高于其他的计算步骤，大于 $O(n)$。因此对商多项式的计算进行复杂度的降低将对大型电路的性能提升有明显改善。

第五章　总结与展望

零知识证明的应用前景十分广阔，特别是在区块链与隐私计算等技术的应用情境中。对证明算法性能的提升是使其得到更广泛应用的关键点。

PLONK 现有的代码在对多项式进行点值到系数形式转换的过程使用了快速傅立叶逆变换（ifft），但在电路门较少的情境中单独编写拉格朗日插值函数可能是一个对特殊场景下运行性能具有潜在提升的设计方式。

参考文献

[1] S Goldwasser, S Micali and C Rackoff. The knowledge complexity of interactive proof systems, 17th. In: Annual ACM Symp. on Theory of Computing, 1985.

[2] Oded Goldreich, Silvio Micali and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM (JACM), 1991, 38(3): 690 ∼ 728.

[3] Bryan Parno et al. Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, 2013: 238 ∼ 252.

[4] Jens Groth et al. Updatable and universal common reference strings with applications to zk-SNARKs. In: Annual International Cryptology Conference, 2018: 698 ∼ 728.

[5] Mary Maller et al. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019: 2111 ∼ 2128.

[6] A. Gabizon, Zachary J. Williamson and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. IACR Cryptol. ePrint Arch. 2019, 2019: 953.

[7] Vitalik Buterin. Understanding PLONK.

[8] Eli Ben Sasson et al. Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, 2014: 459 ∼ 474.

[9] 星想法. 零知识证明 - zk-SNARK 应用场景分析.

[10] Matthew Green. Zero Knowledge Proofs: An illustrated primer, 2014. https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/.

[11] Claus-Peter Schnorr. Efficient signature generation by smart cards. Journal of cryptology, 1991, 4(3): 161 ∼ 174.

[12] Dima Kogan. Lecture 5: Proofs of Knowledge, Schnorr's protocol, NIZK.

[13] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques, 1986: 186 ∼ 194.

[14] Heidroon Ong and Claus-Peter Schnorr. Fast signature generation with a Fiat Shamir—like scheme. In: Workshop on the Theory and Application of of Cryptographic Techniques, 1990: 432 ∼ 440.

[15] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference, 1991: 129 ∼ 140.

[16] Vitalik Buterin. Exploring Elliptic Curve Pairings. https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627.

[17] Zcash. What are zk-SNARKs? https://z.cash/technology/zksnarks/.

[18] Jens Groth. On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques, 2016: 305 ∼ 326.

[19] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Annual International Cryptology Conference, 2017: 581 ∼ 612.

[20] Rosario Gennaro et al. Quadratic span programs and succinct NIZKs without PCPs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2013: 626 ∼ 645.

[21] Mateusz Raczyński. What Is The Fastest Blockchain And Why? Analysis of 43 Blockchains.