简洁非交互零知识证明 (zkSNARK) 概述

BerryChen0w0

2024.6.25

目录

1 摘要

本文对简洁非交互零知识证明 (zkSNARK) 的知识体系进行了系统的梳理，从 zkSNARK 的概念解析、实现方法、前沿研究等多方面全面梳理了 zkSNARK 相关知识，构建了基础的 zkSNARK 的知识体系架构。其中，概念辨析方面，从 zkSNARK 名称入手，详细解析了 zkSNARK 的概念、定义和性质。实现方法方面，zkSNARK 主要包括了交互式预言机证明 (IOP) 和函数承诺 (function commitment) 两个主要组成部分，本文分别介绍了这两者，举例说明它们的含义和用法，并整理了现有的主要 IOP 和函数承诺；另外本文介绍了实现非交互和零知识的方法。前沿研究方面，主要介绍了递归 SNARK 和分布式 SNARK 两类提高效率、扩展可处理问题规模的 zkSNARK 设计思路，并对未来发展提出了展望。

2 研究背景和写作目的

当前，零知识证明技术发展迅速，且拥有广泛的应用场景。理论层面上，新的零知识证明协议层出不穷，百家争鸣；应用层面上，零知识证明在区块链交易隐私保护、区块链跨链交易、隐私保护投票、身份认证、隐私计算、联邦学习等众多场景中取得广泛应用，百花齐放。

在理论研究和实践应用中，广受关注的是简洁非交互零知识证明协议 (zero-knowledge Succinct Non-interactive ARgument of Knowledge, zkSNARK)。然而，zkSNARK 协议多种多样，构建它们的基础组件也各不相同。初学者想要入门 zkSNARK 研究领域有较高的门槛，往往遇到概念不清、知识和概念的关系混乱、对知识缺乏整体的把握等困难。完整的 zkSNARK 的知识体系构建和引导，对初学者将起到很大帮助。

目前关于 zkSNARK 协议的综述研究较少，尽管它们都具有很高的质量，但初学者阅读起来往往比较困难。综述论文如 [24, 18]，内容完备，知识体系完整，但因为精炼了 zkSNARK 领域的大量知识，如果没有足够的基础知识储备，则比较难以理解文章精髓。学习读本如 [20] 涵盖了零知识证明的大量知识，娓娓道来循循善诱，但文本长达四五百页，不便于迅速构建起关于 zkSNARK 的知识体系。

因此，本文对简洁非交互零知识证明 (zkSNARK) 的知识体系进行系统地梳理，从 zkSNARK 的概念解析、实现方法、前沿研究等多方面全面梳理了 zkSNARK 相关知识，以期构建基础的 zkSNARK 的知识体系架构，实现以下目标：

1. 提供快速认识 zkSNARK 的基本蓝图，使初学者读完后能够形成对 zkSNARK 体系化的了解。
2. 涵盖 zkSNARK 的概念辨析和理论构建，搭建简单但较完整的 zkSNARK 知识框架。
3. 介绍 zkSNARK 的前沿研究，并提出两个有前景的发展方向。

3 zkSNARK 简介

3.1 零知识证明发展历程

零知识证明 (zero-knowledge proof) 是一个有趣且强大的密码学概念，它允许一方（证明者 (prover)）在不透露关于某陈述 (statement) 本身的任何信息的情况下，说服另一方（验证者 (verifier)）该陈述的真实性。零知识证明自 1985 年提出以来，经历了近四十年的发展，从偏向计算理论的抽象模型，到实

用零知识证明协议的出现，再到如今简洁非交互零知识证明 (zero-knowledge Succinct Non-interactive ARgument of Knowledge, zkSNARK) 大放异彩，在众多领域中取得应用。本节给出零知识证明发展历史的简要介绍。

交互式零知识证明 零知识证明由 Shafi Goldwasser，Silvio Micali 和 Charles Rackoff 在 1985 年首次提出 [10]。他们提出了交互式证明 (Interactive Proof) 的概念，并定义了证明系统 (proof system) 的零知识性。论文中同时给出了二次剩余 (quadratic residue) 和二次非剩余 (quadratic nonresidue) 问题的零知识证明例子。

任意 NP 问题存在零知识证明 1987 年，[9] 证明了所有 NP 问题都有计算零知识的交互式证明协议。文章给出了 3 染色问题的零知识证明协议构造，并证明了协议的完备性。因为 3 染色问题是 NP 完全问题 (NP complete problem)，即任意 NP 问题可以规约到 3 染色问题，从而证明了对任意 NP 问题存在交互式零知识证明。

非交互零知识证明 在 1990 年代，研究人员将这一概念扩展到非交互零知识证明（NIZK）。在非交互证明中，证明者向验证者发送单个消息，使协议在实际场景中更加高效和实用。

简洁非交互零知识证明 2013 年，Pinocchio 协议 [19] 被提出。它是第一个有实用价值的通用零知识证明协议（zkSNARK）。此后，大量 zkSNARK 协议提出，它们在证明大小 (proof size)、证明时间 (prover time)、验证时间 (verifier time)、是否需要可信设置 (trusted setup)、是否抗量子等方面各有优劣。目前较为常用的 zkSNARK 协议有 Bulletproof[]、Groth16[11]、Plonk[8] 等。

3.2 zkSNARK 概念解析

zkSNARK 是 “简洁非交互零知识证明 (zero-knowledge Succinct Non-interactive ARgument of Knowledge)” 的简称，它目前最为通用的一类零知识证明协议。前一节中所述 Pinocchio 协议、Groth16 协议、Plonk 协议等都属于 zkSNARK 协议。注意到 zkSNARK 的全称长而复杂，可以拆分为以下子概念：

1. 知识论证 (argument of knowledge)
2. 非交互 (non-interactive)
3. 简洁 (succinct)
4. 零知识 (zero-knowledge)

本节分别介绍这些子概念的由来和含义，以使读者理解 zkSNARK 的概念。

3.2.1 知识论证 (argument of knowledge)

设我们给定约束关系 R 和一个 x，证明者 (Prover, P) 向验证者 (Verifier, V) 证明存在 w 使得 x 和 w 满足关系 R，即证明存在 w 使 $(x, w) \in R$。

传统证明系统 在传统证明系统中，证明过程是 P 发送 w 给 V，V 验证 $(x,w) \in R$ 是否成立。在这样的证明系统中存在以下两个问题：**证明大小 (proof size) 可能很大；V 的验证过程可能很耗时**。

例如证明两个图同构时，问题可以描述为 $x = (\mathbb{G}_1, \mathbb{G}_2), w = \sigma, R = \{(x,w)|w(x[1]) = x[2]\}$。这里 $\mathbb{G}_1, \mathbb{G}_2$ 是两个图，σ 是两个图的同构映射。此时 P 需要把同构映射 $w = \sigma$ 发送给 V，V 需要验证 $\mathbb{G}_1, \mathbb{G}_2$ 在映射 σ 下是否同构。这里发送的映射 σ 规模（两个图中点到点的映射关系）已经和问题本身的规模（两个图的点和边表示）相当了。更进一步，如果是证明图不同构，则 V 需要验证所有可能的映射下 $\mathbb{G}_1, \mathbb{G}_2$ 都不同构，因为 n 个节点的图可能的映射有 $n!$ 种，此时对 $O(n)$ 的问题，V 所需的验证时间 (verifier time, 后面简记为 v-time) 已经达到 $O(n!)$ 了。

交互式证明系统 (interactive proof system) 在 [10] 中提出的交互式证明系统，试图解决传统证明系统中验证时间 (v-time)。交互式证明系统中，P 和 V 进行多轮交互，在最后一轮结束时，V 判断是否接受 P 的证明（accept 还是 reject）。在交互式证明系统中，P 不需要把 w 发送给 V，而是根据 V 发送来的数据（称为挑战 (challenge)）和 w（称为见证 (witness)）创建响应 (response) 数据（即证明 (proof)）发送给 V。这些响应数据对 V 来说需要是容易验证的。这样，**证明大小 (proof size) 可以小于 w 的规模；验证时间也可以缩短**。不过，这样的效率提高是有代价的——交互式证明系统是概率证明系统，即 P 可能以一定的概率构造出假的证明，骗过 V 的验证。好在实际应用中，可以通过增加交互轮数等方式，把出错的概率降低到很小以至可忽略的水平。

知识证明 (proof of arguement) 前面所说的证明系统中，证明了存在 w 使得 $(x,w) \in R$。但我们更希望 P 能够证明它知道这样的使得 $(x,w) \in R$ 的 w。例如在承诺方案中，Alice 用哈希函数 h 对某个消息 m 做出承诺，承诺值 $y = h(m)$ 公开。之后 Alice 需要向 Bob 打开承诺（一般做法就是 Alice 把 m 发送给 Bob）以证明自己拥有数据 m，而不是仅仅证明 “存在一个 m' 使得 $h(m') = y$” ——毕竟一个哈希值有哈希原像是显然的，“知道某个哈希值的哈希原像” 才是值得证明的事情。

知识证明的具体定义，需要用到提取器 (extractor) 的概念：存在一个提取器 **Ext**，可以从 P 处提取出 w。在4.2节给出详细定义。

3.2.2 非交互 (non-interactive)

前面的证明系统都是交互式的，但交互式证明有一个缺点：一个证明者 P 不能同时向多个验证者 V 进行证明。

如果 P 需要向多个验证者 $V_1, ..., V_m$ 证明，则必须和每一个 V_i 运行一次交互式证明协议。在验证者很多的场景中（例如区块链，每一笔交易需要链上多数节点验证同意），交互式证明的代价难以想象。

为此，我们希望拥有非交互式的证明协议，即证明者 P 生成一个证明 π，验证者 V 验证 π，即可判定是否接受证明。形式化地说，非交互式的知识证明协议 NARK(non-interactive argument of knowledge) 是一个三元组 (S, P, V):

1. $S(C) \to gp$。初始化。这里 C 是电路（即表示一个约束关系）
2. $P(pp, x, w) \to \pi$。P 生成证明 π
3. $V(vp, x, \pi) \to acc/rej$。V 验证证明

3.2.3 简洁 (succinct)

在 NARK 的基础上，我们希望证明 (proof) 的大小尽可能小，验证者的运行时间尽可能短。为此增加“简洁 (succinct)”的要求：

1. $len(\pi) = sublinear(|w|)$
2. $time(V) = O(|x|, sublinear(|C|))$

这里 sublinear 是指小于 $O(x)$ 的级别，例如 $O(x^\alpha), 0 < \alpha < 1$

3.2.4 零知识 (zero-knowledge)

直观的理解“零知识”，就是证明过程中 P 不向 V 透露任何有关 w 的信息，也就是 V 在证明前和证明后能够计算出的东西没有差别。

形式化地定义零知识，需要用到模拟器 (simulator)：定义 V 的视图 (view) 为证明过程中 V 接收到的所有数据，记为 $view_V$。存在模拟器可以模拟出 V 的视图 $view'_V$，若多项式时间的区分器 (distinguasher) 不能区分 $view_V$ 和 $view'_V$（计算不可区分），则 (P,V) 证明协议是零知识的。在4.2节给出详细定义。

4 zkSNARK 定义和性质

在3.2节中，我们已经理解了 zkSNARK 名称中各个概念的含义。本节，我们可以给出 zkSNARK 的形式化定义，并严谨地定义和描述它的各种性质。

4.1 zkSNARK 形式化定义

给定一个约束关系 $\mathcal{R}$,一个 zkSNARK 可以表示为一个多项式时间算法元组 $\Pi = (\textbf{Setup}, \textbf{Prove}, \textbf{Verify})$。其中：

- $\textbf{Setup}(1^\lambda, \mathcal{R}) \to (pp, vp)$。给定安全参数 λ 和关系 $\mathcal{R}$，输出证明者和验证者需要使用的公共参数 pp 和 vp。
- $\textbf{Prove}(pp, x, w) \to \pi$。输入证明参数 pp、公共输入 x、见证 (witness)（即隐私输入）w。若 x 和 w 满足电路 $\mathcal{R}$ 的约束关系，即 $(x, w) \in R$，该”Prove” 算法生成对应的证明 (proof)π。
- $\textbf{Verify}(vp, x, \pi) \to b \in \{0, 1\}$。输入验证参数 vp、公共输入 x、证明 π。该”Verify” 算法用于验证证明 π，输出是一个布尔值 b，若验证通过则 $b = 1$，否则 $b = 0$。

4.2 zkSNARK 的性质

一个 zk-SNARK 体系具有完备性 (completeness)、知识可靠性 (knowledge soundness)、简洁性 (succinctness)、零知识 (zero-knowledge) 性质。[1]

[1]注意到，这里我们不强调“非交互性”，实际上这一点在 zkSNARK 的定义中已经体现了：Prove 和 Verifiy 是独立的两个算法，其运行不需要与另一个算法交互。

- **完备性 (completeness)**。一个诚实的证明者 $\mathcal{P}$，如果它确实知道一个见证 w 满足 $(x,w)\in\mathcal{R}$，那么它应该总能说服验证者 $\mathcal{V}$，即它生成的证明 π 总能被验证者验证通过。形式化地说，对任意 $\lambda\in\mathbb{N}$ 和任意 $(x,w)\in\mathcal{R}$，有：

$$\Pr\left[\mathbf{Verify}(vp,x,\pi)=1 \;\middle|\; \begin{array}{c}(pp,vp)\leftarrow\mathbf{Setup}(1^\lambda,\mathcal{R})\\ \pi\leftarrow\mathbf{Prove}(pp,x,w)\end{array}\right]=1 \tag{1}$$

- **知识可靠性 (knowledge soundness)**。存在一个提取器 (simulator)**Ext**，对任意的证明者 $\mathcal{P}^*$[2]，如果 $\mathcal{P}^*$ 能够生成通过验证的证明 π，则 **Ext** 可以以极大的概率提取出能通过验证的见证 w。即，存在一个多项式时间的提取器 **Ext**，对任意多项式时间的证明者 $\mathcal{P}^*$，有：

$$\Pr\left[(x,w')\notin\mathcal{R} \;\middle|\; \begin{array}{c}(pp,vp)\leftarrow\mathbf{Setup}(1^\lambda,\mathcal{R})\\ \pi\leftarrow\mathbf{Prove}(pp,x,w),\ \mathbf{Verify}(vp,x,\pi)=1\\ w'\leftarrow\mathbf{Ext}^{\mathcal{P}^*}(pp,x)\end{array}\right]=\mathsf{negl}. \tag{2}$$

- **简洁性 (succinctness)**。如3.2.3节所述，简洁性要求证明 π 的长度是亚线性，验证时间至多是线性。

- **零知识 (zero-knowledge)**。如3.2.4节所述，零知识性是指证明过程中，除了“陈述的正确性”以外，并不泄露任何更多的信息。具体来说，存在一个期望多项式时间的模拟器 **Sim**，使得对任意多项式时间的证明者 $\mathcal{V}^*$[3]，下列两个分布是计算不可区分的：

$$\begin{aligned}D_0&=\{\pi_0\leftarrow\mathbf{Prove}(pp,x,w)|(pp,vp)\leftarrow\mathbf{Setup}(1^\lambda,\mathcal{R})\}\\ D_1&=\{\pi_1\leftarrow\mathbf{Sim}^{\mathcal{V}^*}(pp,x,w)|(pp,vp)\leftarrow\mathbf{Setup}(1^\lambda,\mathcal{R})\}\end{aligned} \tag{3}$$

5 符号约定和预备知识

5.1 符号约定

后文中，我们用 $\mathbb{F}_p$ 表示素数 p 阶有限域，$\mathbb{F}_p^{\le d}$ 表示有限域 $\mathbb{F}_p$ 上的次数 $\le d$ 的多项式的全体，即 $\mathbb{F}_p^{\le d}=\{\sum_{i=0}^{d}a_iX^i,a_i\in\mathbb{F}_p\}$。

我们用大写 X 表示变量，小写 x 表示变量的某个取值。例如，$f(X)$ 表示多项式，$f(x)$ 表示多项式 $f(X)$ 在 $X=x$ 点的取值。

记号 $a\leftarrow A$ 表示 a 是由 A 计算得到的结果，$r\xleftarrow{\$}S$ 表示 r 是以均匀分布随机从 S 中选择的一个元素。

描述 zkSNARK 协议（包括 IOP 协议、多项式承诺协议等）时，假设协议中规定证明者 $\mathcal{P}$ 应该发送的值是 $h(r)$，我们用 $\widehat{h(r)}$ 表示协议运行时 $\mathcal{P}$ 实际发送的声称是 $h(r)$ 的值。这是为了描述的方便，避免混淆。

5.2 Schwartz-Zippel 引理

Schwartz-Zippel 引理是交互式（概率）证明中非常重要的基础。本节给出这个引理的描述和简单理解。

[2]这里证明者是任意的，即可能是不诚实的证明者，所谓“不诚实”就是说它可能不按照协议规定的 **Prove** 算法来进行证明（生成 π）

[3]证明者也可以是不诚实的，即它不按照 Verify 的正确操作的做。例如他选择随机数不按照均匀分布来选，想以此套取 P 中关于 w 的信息

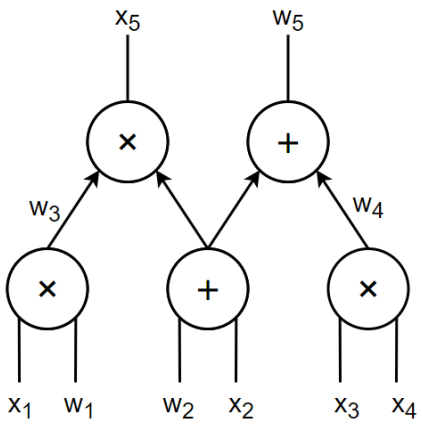

图 1: 算术电路

Schwartz-Zippel 引理 给定非零多项式 $f \in \mathbb{F}_p^{\leq d}[X]$，则对 $r \xleftarrow{\$} \mathbb{F}_p$，有

$$Pr[f(r) = 0] \leq \frac{d}{p}$$

在实际应用中，一般 $p \approx 2^{128}$ 或 2^{256}，而 $d \leq 2^{40}$，则 $\frac{d}{p}$ 非常小，是可忽略的（negl.）。

于是，若 $r \xleftarrow{\$} \mathbb{F}_p, f(r) = 0$，则 f 以极大的概率是零多项式。在协议中我们就可以相信 $f(X)$ 是零多项式。

在6节中将看到，zkSNARK 的计算模型是算术电路，进而将算术电路上的约束关系转化为若干多项式恒等式，则验证 $(x, w) \stackrel{?}{\in} \mathcal{R}$ 就转化为验证多项式恒等式是否成立，也就是验证某多项式是否为零多项式（$f(X) = g(X)$ 就是 $h(X) := f(X) - g(X) = 0$）。chwartz-Zippel 引理的作用是极大地减少验证时间：从验证 $\forall x \in \mathbb{F}_p, h(x) = 0$，变为验证 $r \xleftarrow{\$} X, h(r) = 0$。

6 zkSNARK 的计算模型——算术电路

在零知识证明中，计算模型一般是算术电路 (arithmetic circuit)，也就是把问题抽象成一个电路表示。电路，是由运算门和电线组成的。在算术电路中，只有加法门和乘法门两种运算门。图1是一个算术电路的例子。

把问题转化为电路后，还需要把电路中表示的各种约束条件，转化为数学化的表示，以便应用 zkSNARK 对其进行证明。常用的约束表示有 R1CS[3] 和 QAP[19]。这两种约束表示是等价的，且存在简单的转化。本文仅以图1为例，介绍电路转化为 QAP 表示的方法。

图1中的全部约束可以写为：

$$x_1 \cdot w_1 = w_3$$
$$x_3 \cdot x_4 = w_4$$
$$w_3 \cdot (w_2 + x_2) = x_5$$
$$(w_4 + (w_2 + x_2)) \cdot 1 = w_5$$

其中 x_i 是公共输入，w_i 是见证 (witness)，1 记作 v_{one} 是一个始终赋值为常数 1 的变量。则电路中的

	$l_0(X)$	$l_1(X)$	$l_2(X)$	$l_3(X)$	$l_4(X)$	$l_5(X)$	$l_6(X)$	$l_7(X)$	$l_8(X)$	$l_9(X)$	$l_{10}(X)$
$X=r_1$	0	1	0	0	0	0	0	0	0	0	0
$X=r_2$	0	0	0	1	0	0	0	0	0	0	0
$X=r_3$	0	0	0	0	0	0	0	0	1	0	0
$X=r_4$	0	0	1	0	0	0	0	1	0	1	0

图 2: $l_i(X)$ 取值表

	$r_0(X)$	$r_1(X)$	$r_2(X)$	$r_3(X)$	$r_4(X)$	$r_5(X)$	$r_6(X)$	$r_7(X)$	$r_8(X)$	$r_9(X)$	$r_{10}(X)$
$X=r_1$	0	0	0	0	0	0	1	0	0	0	0
$X=r_2$	0	0	0	0	1	0	0	0	0	0	0
$X=r_3$	0	0	1	0	0	0	0	1	0	0	0
$X=r_4$	1	0	0	0	0	0	0	0	0	0	0

	$o_0(X)$	$o_1(X)$	$o_2(X)$	$o_3(X)$	$o_4(X)$	$o_5(X)$	$o_6(X)$	$o_7(X)$	$o_8(X)$	$o_9(X)$	$o_{10}(X)$
$X=r_1$	0	0	0	0	0	0	0	0	1	0	0
$X=r_2$	0	0	0	0	0	0	0	0	0	1	0
$X=r_3$	0	0	0	0	0	1	0	0	0	0	0
$X=r_4$	0	0	0	0	0	0	0	0	0	0	1

图 3: $r_i(X), o_i(X)$ 取值表

全部变量可以写成向量:

$$\mathbf{z} = (v_{one}, \mathbf{x}, \mathbf{w}) = (1, x_1, x_2, x_3, x_4, x_5, w_1, w_2, w_3, w_4, w_5) =: (z_0, z_1, ..., z_{10})$$

我们把每个约束条件（也就是每个乘法门）对应到一个随机的域元素上，令第 j 个约束条件对应到 $r_j \xleftarrow{\$} \mathbb{F}$。定义若干个多项式 $l_i(X), r_i(X), o_i(X)$，其中 $l_i(X)$ 对应变量 z_i，令 $l_i(r_j)$ 表示变量 z_i 在第 j 个约束条件（第 j 个乘法门）中是否作为左输入。如果是，则 $l_i(r_j) = 1$；如果不是，$l_i(r_j) = 0$。$r_i(X), o_i(X)$ 同理。

对上面的例子而言,我们就可以得到多项式 $l_0(X), l_1(X), ..., l_{10}(X)$ 如图2。类似地,得到 $r_i(X), o_i(X)$ 如图3所示。

由上面这些 $r_1, ..., r_4$ 点上的取值,我们可以插值(常用 Lagrange 多项式插值法)得到 $l_i(X), r_i(X), o_i(X), i = 0, 1, ..., 10$ 这些多项式的表达式。它们都是 3 次多项式（因为是由 4 个定点插值得到）。现在我们定义多项式 $L(X)$:

$$L(X) = \sum_{i=0}^{10} z_i \cdot l_i(X)$$

类似可定义 $R(X), O(X)$。$L(X), R(X), O(X)$ 称为选择多项式 (selector polynomial)。可以看到 $L(r_j) = \sum z_i \cdot l_i(r_j) = \ j$ 。例如 $L(r_4) = z_2 + z_7 + z_9 = (x_2 + w_2 + w_4)$。则第 j 个约束条件（第 j 个乘法门）对应可以写为多项式关系 $L(r_j)R(r_j) = O(r_j)$。

定义主多项式 (master polynomial) 为

$$p(X) = L(X)R(X) - O(X) = (\sum z_i \cdot l_i(X))(\sum z_i \cdot r_i(X)) - (\sum z_i \cdot o_i(X))$$

则电路的全部约束条件，就转化为 “$p(X)$ 在 $r_j, j = 1, ..., 4$ 上取值都为 0”，即 $\{r_j\}$ 是 $p(X)$ 的零点。这意味着 $(X - r_j)$ 是 $p(X)$ 的因式。定义消亡多项式 (vanishing polynomial)：

$$V(X) = \sum_j (X - r_j)$$

则 $V(X)$ 应当是 $p(X)$ 的因式。即存在多项式 $q(X)$ 使得 $p(X) = V(X)q(X)$。我们称 $q(X) = \dfrac{p(X)}{V(X)}$ 为商多项式 (quotient polynomial)

这样，我们得到了电路约束转化为 QAP 的形式：

$$p(X) = \Big(\sum z_i \cdot l_i(X)\Big) \cdot \Big(\sum z_i \cdot r_i(X)\Big) - \Big(\sum z_i \cdot o_i(X)\Big) = V(X)q(X) \tag{4}$$

7 zkSNARK 协议的构造

zkSNARK 协议拥有较为通用的构造流程，先实现交互式证明，再将其转为非交互式证明，最后实现零知识性质。具体流程如下：

1. 构造交互式预言机证明 (Interactive Oracle Proof)。在交互式预言机证明中，P 可以在预言机模型 (Oracle) 的帮助下向 V 证明某些函数具有某些性质。我们在第8章详细介绍交互式预言机证明。
2. 用函数承诺 (function commitment) 替换 IOP 中 Oracle 的角色，得到交互式的知识论证。函数承诺中，P 可以对某个函数进行承诺，之后 V 可以在某个点上打开承诺，得到函数在这个点的取值。我们在第9章详细介绍函数承诺。
3. 用 Fait-Shamir 变换把交互式证明转化为非交互式证明,得到简洁非交互零知识证明 SNARK(succinct non-interactive argument of knowledge)。
4. 通过添加随机数实现零知识，得到 zkSNARK。我们在第10章详细介绍 Fait-Shamir 变换和引入随机值实现零知识的方法。
5. 把 zkSNARK 应用到算术电路。对算术电路的介绍见第6章。

图4描述了整个构造流程。

8 交互式预言机证明 (Interactive Oracle Proof, IOP)

8.1 IOP 的定义

交互式预言机证明 (IOP) 可以视作交互式证明 (Interactive Proof, IP) 和概率可验证证明 (Probabilistically Checkable Pproofs, PCP) 的结合。我们首先介绍 IP 和 PCP，再引出 IOP 的定义。

8.1.1 交互式证明 (IP)

在交互式证明 (IP)[10] 中，存在证明者 $\mathcal{P}$ 和验证者 $\mathcal{V}$ 两方，$\mathcal{P}$ 向 $\mathcal{V}$ 证明某陈述的正确性。$\mathcal{P}$ 和 $\mathcal{V}$ 之间进行多轮交互，在最后一轮交互结束后，$\mathcal{V}$ 判断是否接受 $\mathcal{P}$ 的证明。在每一轮交互中，验证者 $\mathcal{V}$ 首先发送一个挑战 (challenge)$chall_i$ 给证明者 $\mathcal{P}$，然后 $\mathcal{P}$ 经过计算，产生一个对 $chall_i$ 的应答 (response)$resp_i$ 给 $\mathcal{V}$。如图5展示了交互式证明的一般过程。

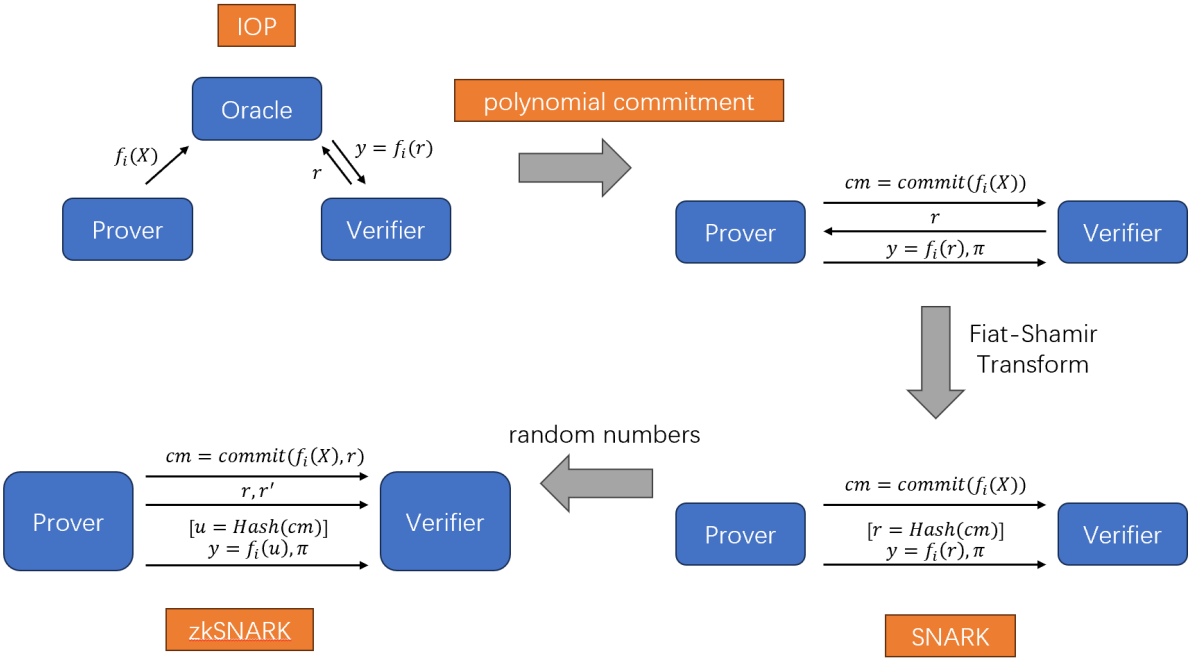

图 4: zkSNARK 构造流程

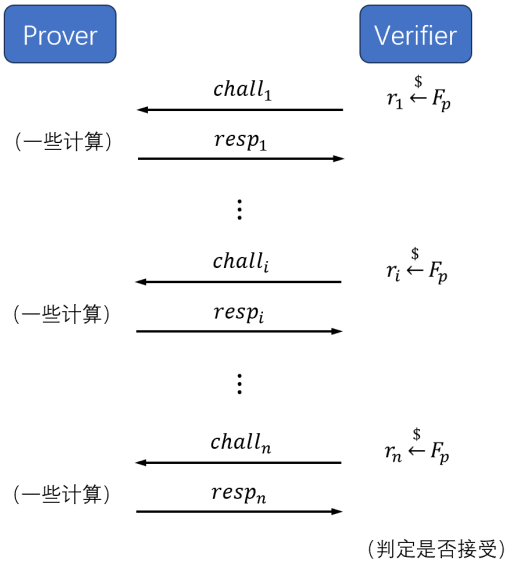

图 5: 交互式证明 (IP)

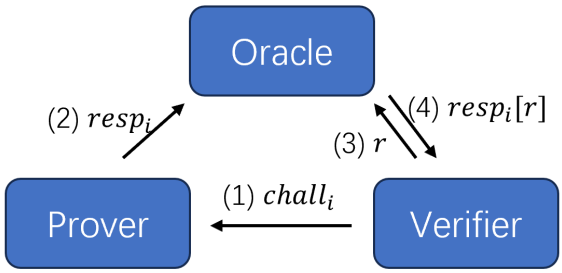

图 6: 交互式预言机证明 (IOP)

8.1.2 概率可验证证明 (PCP)

概率可验证证明 (PCP) 由论文 [7, 1] 等给出。PCP 中，证明者 $\mathcal{P}$ 生成证明 (proof)，概率多项式时间的验证者 $\mathcal{V}$ 可以以预言机形式查询证明的某些位置[4]。$\mathsf{PCP}[r, q]$ 表示这样的语言类：$\mathcal{V}$ 使用至多 r 比特的随机数（即证明长度至多 2^r 个比特），查询证明中的至多 q 个位置。注意，PCP 不是交互式的，而是 $\mathcal{P}$ 直接生成整个证明 (proof) 发送给 $\mathcal{V}$。

8.1.3 交互式预言机证明 (IOP)

交互式预言机证明 (IOP) 有 Eli Ben-Sasson 等人在 2016 年给出形式化定义 [4]。IOP 中证明者 $\mathcal{P}$ 和验证者 $\mathcal{V}$ 进行多轮交互，在每一轮交互中，$\mathcal{V}$ 不需要完整读取 $\mathcal{P}$ 发送的消息，而是可以以预言机形式访问 $\mathcal{P}$ 的消息，即可以访问消息的任意位置，并得到该位置的信息。实际上，$\mathcal{P}$ 并不发送完整的消息给 $\mathcal{V}$，而是发送一个消息的预言机 (Oracle)，$\mathcal{V}$ 可以向预言机查询消息在任意位置的值。

为方便理解 Oracle，可以把它视作一个绝对可靠的第三方。$\mathcal{P}$ 把消息发送给 Oracle，$\mathcal{V}$ 可以向 Oeacle 查询消息的某些位置。图6展示了 IOP 每一轮交互的过程。注意，把 Oracle 视作可靠的第三方是一种便捷的理解方式，实际上密码学中的 Oracle 是一个理论模型，并不存在“第三方”。

8.1.4 多项式 IOP(polynomial IOP, pIOP)

我们在第6章已经看到，zkSNARK 是把问题转化为电路，进而转化为多项式约束，然后对多项式约束做证明。因此，zkSNARK 使用的 IOP 是对多项式做证明的 IOP，即多项式 IOP(polynomial IOP, pIOP)。pIOP 是这样的 IOP：$\mathcal{P}$ 发送的消息都是对多项式的预言机。即，每一轮交互中，$\mathcal{P}$ 发送的是对某多项式 $f_i(X)$ 的预言机（可以理解为图6中的 $resp_i := f_i(X)$），$\mathcal{V}$ 查询 $f_i(X)$ 在任意点 $r \in \mathbb{F}_p$ 上的取值（可以理解为图6中的 $resp_i[r] := f_i(r)$）。

8.2 IOP 协议举例——sumcheck 协议

sum-check 协议 [17] 是一个经典的 IOP 协议。

给定一个 v 个变量的多项式 $g(X) = g(X_1, X_2, ..., X_v) \in \mathbb{F}_p^{\leq d}$，sum-check 协议是证明

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_v \in \{0,1\}} g(b_1, b_2, ..., b_v) \tag{5}$$

[4]所谓“预言机形式”，可以理解为：$\mathcal{V}$ 可以访问证明的任意位置，且一定会得到这个位置的准确信息。但同时，$\mathcal{V}$ 的这种访问不会向 $\mathcal{V}$ 泄露任何关于证明内容的分布的信息

Description of Sum-Check Protocol.

- At the start of the protocol, the prover sends a value C_1 claimed to equal the value H defined in Equation (4.1).
- In the first round, $\mathcal{P}$ sends the univariate polynomial $g_1(X_1)$ claimed to equal

$$\sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1,x_2,\ldots,x_v).$$

  $\mathcal{V}$ checks that

$$C_1 = g_1(0)+g_1(1),$$

  and that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, rejecting if not. Here, $\deg_j(g)$ denotes the degree of $g(X_1,\ldots,X_v)$ in variable X_j.
- $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$, and sends r_1 to $\mathcal{P}$.
- In the jth round, for $1 < j < v$, $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial $g_j(X_j)$ claimed to equal

$$\sum_{(x_{j+1},\ldots,x_v)\in\{0,1\}^{v-j}} g(r_1,\ldots,r_{j-1},X_j,x_{j+1},\ldots,x_v).$$

  $\mathcal{V}$ checks that g_j is a univariate polynomial of degree at most $\deg_j(g)$, and that $g_{j-1}(r_{j-1}) = g_j(0)+g_j(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_j \in \mathbb{F}$, and sends r_j to $\mathcal{P}$.
- In Round v, $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial $g_v(X_v)$ claimed to equal

$$g(r_1,\ldots,r_{v-1},X_v).$$

  $\mathcal{V}$ checks that g_v is a univariate polynomial of degree at most $\deg_v(g)$, rejecting if not, and also checks that $g_{v-1}(r_{v-1}) = g_v(0)+g_v(1)$.
- $\mathcal{V}$ chooses a random element $r_v \in \mathbb{F}$ and evaluates $g(r_1,\ldots,r_v)$ with a single oracle query to g. $\mathcal{V}$ checks that $g_v(r_v) = g(r_1,\ldots,r_v)$, rejecting if not.
- If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ halts and accepts.

图 7: sum-check 协议

即给定 $g(X)$，$\mathcal{P}$ 证明 H 是由上述求和正确计算得到的结果。

$\mathcal{V}$ 当然可以自己计算这个求和的值 H，验证它和 $\mathcal{P}$ 发送的声称是 H 的值 $\widehat{H}$ 是否相等。但计算这个求和的时间成本是 $O(2^v)$，这对 $\mathcal{V}$ 来说不够高效。交互式的 sum-check 协议通过与 $\mathcal{P}$ 交互，$\mathcal{V}$ 的时间复杂度可以降低到 $O(v)$。

sum-check 协议的详细描述如图7所示 [20]。

在第 j 轮中，$\mathcal{P}$ 发送一个声称是 $g_j(X_j) := \sum_{b_{j+1},b_{j+2},\ldots,b_v} g(r_1,r_2,\ldots,r_{j-1},X_j,b_{j+1},\ldots,b_v)$ 的多项式 $\widehat{g_j(X_j)}$，$\mathcal{V}$ 检查：

- $\mathsf{deg}(\widehat{g_j}) \overset{?}{\leq} \mathsf{deg}(g)$。这是保证 $\widehat{g_j}$ 的次数不超过原始的多项式，从而确保 Schwartz-Zippel 引理可用。
- $\widehat{g_j}(0)+\widehat{g_j}(1) \overset{?}{=} \widehat{g_{j-1}}(r_{j-1})$。这是确保 $\widehat{g_j}$ 是由 $\widehat{g_{j-1}}$ 正确生成的。

之后 $\mathcal{V}$ 选择一个随机数 $r_j \overset{\$}{\leftarrow} \mathbb{F}_p$ 发送给 $\mathcal{P}$，$\mathcal{P}$ 计算 $g_{j+1}(X_{j+1}) := \sum_{b_{j+2},\ldots,b_v} g(r_1,r_2,\ldots,r_j,X_{j+1},b_{j+2},\ldots,b_v)$

直到最后一轮，$\mathcal{V}$ 额外检查 $\widehat{g_v(r_v)} \overset{?}{=} g(r_1,\ldots,r_v)$。若检查通过，则由 Schwartz-Zippel 引理，可以以极大概率认为 $\widehat{g_v(X_v)} = g_v(X_v)$，进而可以以极大概率认为 $\widehat{g_{v-1}(X_{v-1}} = g_{v-1}(X_{v-1})$，进而... 可以以极大概率认为 $\widehat{g_1(X_1)} = g_1(X_1)$。又因为 $\widehat{H} := C_1 = \widehat{g_1}(0)+\widehat{g_1}(1)$，于是可以以极大概率认为 $\widehat{H} = H$。

一般情况下，多项式 g 对每个变量 X_i 的次数 $\mathsf{deg}_i(g)$ 都是较小的常数值。于是，sum-check 协议的 $\mathcal{V}$ 时间复杂度是 $O(v)$，$\mathcal{P}$ 的时间复杂度是 $O(2^v)$。

9 函数承诺 (function commitment)

9.1 函数承诺的定义

函数承诺 (function commitment) 是对函数的承诺方案。具体来说，对一组函数 $\mathcal{F}$，$f \in \mathcal{F}$，函数承诺是一个多项式时间算法元组 $\Pi = (\textbf{Setup}, \textbf{Commit}, \textbf{Eval})$：

- $\textbf{Setup}(1^\lambda) \to gp$。输入安全参数 λ，生成公共参数 gp。
- $\textbf{Commit}(gp, f, r) \to \mathsf{com}_f$。对函数 $f \in \mathcal{F}$ 做承诺，生成承诺值 com_f。这里的 r 是一个随机数，用于实现隐藏性 (hiding)（“隐藏性”定义见本节后文）。
- $\textbf{Eval}(\mathcal{P}, \mathcal{V}, \mathsf{com}_f, x \in X, y \in Y)$：这里 X 是函数的定义域，Y 是值域。**Eval** 算法是证明者 $\mathcal{P}$ 向验证者 $\mathcal{V}$ 证明 $f(x) = y$。

  $\mathcal{P}(gp, f, x, y, r) \to \pi$。证明者生成对 $f(x) = y$ 的证明 π。

  $\mathcal{V}(gp, \mathsf{com}_f, x, y, \pi) \to b \in \{0, 1\}$。验证者验证证明，若验证通过则 $b = 1$，否则 $b = 0$。

函数承诺具有绑定性 (biding)、隐藏性 (hiding) 两条性质。

- 绑定性 (biding)。即一个承诺 com 对应一个函数，难以找到两个函数 f 和 g，使得 $\mathsf{com}_f = \mathsf{com}_g$。形式化地说：

$$\Pr\left[\mathsf{com}_f = \mathsf{com}_g \,\middle|\, \forall f, g \in \mathcal{F}\right] = \mathsf{negl}. \tag{6}$$

- 隐藏性 (hiding)。即承诺 com 并不泄露被承诺的函数 f 的任何信息。形式化地说，对任意 $f, g \in \mathcal{F}$，下列两个分布不可区分：

$$D_f = \{\mathsf{com}_f \leftarrow \textbf{Commit}(gp, f, r_f) | r_f \xleftarrow{\$} \mathbb{F}_p\} D_g = \{\mathsf{com}_g \leftarrow \textbf{Commit}(gp, g, r_g) | r_g \xleftarrow{\$} \mathbb{F}_p\} \tag{7}$$

9.2 函数承诺协议举例——KZG 多项式承诺

KZG 协议 [12] 是一个非常经典且常用的函数承诺协议。它是一个多项式承诺协议，即被承诺的函数是（单变量）多项式。

KZG 协议描述如下：给定素数 p 阶双线性群 $\mathbb{G}, \mathbb{G}_T$，g 是 $\mathbb{G}$ 的生成元，e 是双线性映射 $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$。函数簇（多项式簇）$\mathcal{F} := \mathbb{F}_p^{\leq d}[X]$。对 $f \in \mathcal{F}$ 做 KZG 承诺：

Algorithm 1 $\textbf{Setup}(1^\lambda) \to gp$

1: $\tau \xleftarrow{\$} \mathbb{F}_p$
2: $gp = (g, g^\tau, g^{\tau^2}, ..., g^{\tau^d})$
3: delete τ
4: **return** gp

Algorithm 2 Commit$(gp, f, r) \to \mathsf{com}_f$

$f(X) = f_0 + f_1 X + f_2 X^2 + ... + f_d X^d$

$\mathsf{com}_f = g^{f(\tau)}$

$= g^{f_0 + f_1 X + f_2 X^2 + ... + f_d X^d}$

$= (g)^{f_0} \cdot (g^{\tau})^{f_1} \cdot (g^{\tau^2})^{f_2} \cdot ... \cdot (g^{\tau^d})^{f_d}$

return com_f

Eval 要证明 $f(u) = v$。如果 $f(u) = \hat{v}$，那么 u 是 $f(X) - \hat{v}$ 的零点，即存在 $q(X) \in \mathbb{F}_p^{\leq d}[X]$ 使得 $f(X) - \hat{v} = (X - u)q(X)$。于是：

Algorithm 3 Eval : $\mathcal{P}(gp, f, x, y, r) \to \pi$

compute $q(X) = \frac{f(X) - \hat{v}}{X - u}$

$\pi := \mathsf{com}_q = g^{q(\tau)}$

return π

Algorithm 4 Eval : $\mathcal{V}(gp, \mathsf{com}_f, x, y, \pi) \to b \in \{0, 1\}$

if $e(\mathsf{com}_f / g^{\hat{v}}, g) = e(g^{\tau - u}, \pi)$ **then**

  $b := 1$

else

  $b := 0$

end if

return b

9.3 现有主要函数承诺协议概述

现有函数承诺协议主要可以分为三类：基于双线性对 (bilinear pairing) 的，基于离散对数 (discrete logarithm) 的，基于哈希 (hashing) 的。

基于双线性对的函数承诺（多项式承诺） 。上述 KZG 协议就是基于双线性对的函数承诺协议。这类协议的优点是：证明 (proof) 大小是常数级，这是所有函数承诺中最小的。另外，KZG 还有一个优点是它可以批量承诺，即多多个多项式在多个点上的取值，可以整合到一个承诺中 [5]，于是也只需要一个常数大小的 proof 即可。

缺点是：1) 需要可信初始化设置 (trusted setup)，即需要一个可信第三方运行 **Setup** 算法。前面 KZG 协议中我们看到，**Setup** 完成后需要删除 τ。而在缺乏信任的实际应用场景中，往往不存在完美的可信第三方。2) 不抗量子。这是因为双线性对需要使用特殊的密码学群 $\mathbb{G}$ 和 $\mathbb{G}_T$，它们是不抗量子的。

基于离散对数的函数承诺 。Bulletproofs[6] 是这类函数承诺的典型例子。除了提供函数承诺之外，Bulletproofs 还可以提供范围证明，即证明某值在某范围内，这是 Bulletproofs 协议的特别之处。其他基于离散对数的函数承诺还有 Hyrax[21], Dory[15] 等。

	Cryptographic tool	P-time	Proof size	V-time	setup	Post-quantum
KZG	pairing	$O(d)$	$O(1)$	$O(1)$	universal	no
Bulletproofs	Discrete logarithm	$O(d)$	$O(\log d)$	$O(d)$	transparent	no
Hyrax		$O(d)$	$O(\sqrt{d})$	$O(\sqrt{d})$		
Dory		$O(d)$	$O(\log d)$	$O(\log d)$		
FRI	hash	$O(d \log d)$	$O(\log^2 d)$	$O(\log^2 d)$	transparent	yes
Ligero		$O(d)$	$O(\sqrt{d})$	$O(\sqrt{d})$		
Brakedown		$O(d)$	$O(\log^k d)$	$O(\sqrt{d})$		
Orion		$O(d)$	$O(\log^2 d)$	$O(\sqrt{d})$		

图 8: 主要函数承诺协议汇总

这类函数承诺不需要可信初始化设置。我们把不需要可信初始化设置的 SNARK 协议称为“透明”(transparent) 的。但这类协议同样不抗量子，因为离散对数问题不是抗量子的。

基于哈希的函数承诺 。典型的例子是 FRI 协议 [2]，FRI 协议也是基于哈希的函数承诺中证明大小最小的。其他协议后 Ligero, Brakedown, Orion 等。这类协议既是透明的，又是抗量子的，在安全性上表现最好。但它们的证明大小较大，验证时间也较长，因而目前实际应用中采用不多。

图8整理了主要函数承诺的类型、性能和安全性表现。

10 实现非交互和零知识

10.1 实现非交互——Fiat-Shamir 变换

zkSNARK 是非交互式的证明协议，但在构造 zkSNARK 时，往往是先构造出交互式的协议，然后将其转化为非交互式。Fiat-Shamir 变换就是从交互式转为非交互式的便利工具。

对一个有 Prover 和 Verifier 两方的交互式协议，如果 Verifier 向 Prover 发送的所有消息都是均匀分布的随机值，则称这个协议是公币协议 (public coin protocol)。

对公币的交互式协议，用 Fiat-Shamir 变换可以将其变为非交互式协议，即让第 i 轮中 V 发送的随机值由前 i 轮中交互过的所有消息的哈希值来确定：

$$r_i = hash(x_i, i, r_{i-1})$$

10.2 实现零知识

经过前面的步骤，我们已经得到非交互式的简洁的知识证明了 (SNARK)。最后一步是实现零知识性。基本思路是向协议中添加随机值，使得含有“知识”的数据被混淆成一个“随机”值。我们以向 KZG 添加零知识性 [23] 为例说明这一点。

前述 KZG 协议 [12] 不是零知识的，是因为承诺值 $\mathsf{com}_f = g^{f(\tau)}$ 是确定性的，即它和 $f(X)$ 函数一一对应。例如，如果 $\mathsf{com}_f = 0$，则攻击者根据 Schwartz-Zippel 引理，即可推断出 $f(X)$ 大概率是零多项式。这就泄露了关于 $f(X)$ 的知识。

添加零知识的思路是引入随机值。把 com_f 改为 $\mathsf{com}_f = g^{f(\tau)+r\eta}$，其中 $r, \eta \xleftarrow{\$} \mathbb{F}_p$ 是两个随机值，称为随机化子 (randomizer)。这样，com_f 和 $f(X)$ 就没有一一对应关系了，$f(\tau) + r\eta$ 也成为一个 $\mathbb{F}_p$ 上的随机值，于是 com_f 在攻击者看来也成为一个随机值。

引入随机值后，相应地需要修改 **Eval**：验证 $f(X)+r\eta-\hat{v} = (x-u)(q(X)+r'\eta)+\eta(r-r'(x-u))$，证明 (proof)$\pi = g^{q(\tau)+r'\eta}, g^{r-r'(\tau-u)}$

11 zkSNARK 前沿研究和展望

笔者对 zkSNARK 的前沿研究并没有全面的把握，这里只是给出两个笔者感兴趣的方向：递归 SNARK 和分布式 SNARK。这两者都是在提高 SNARK 的效率、扩展它可以处理的问题规模等方面的努力，且都有不错的成果产出。

11.1 递归 SNARK(recursive SNARK)

考虑到一些 SNARK（像 groth16,plonk-KZG），证明大小较小，但证明时间长；另一些（基于 FRI, hash 等）证明时间短，但证明大小较大，递归 SNARK 的思路是把两者结合起来，以获得证明时间较短、证明大小也较小的 SNARK。

思路是，先用一个证明大小大、证明时间短的 snark(P,V) 对问题 C 生成一个 proof，然后再用一恶搞证明大小小、证明时间长的 snark(P',V')，证明这个 proof 能够通过 V 验证，即 P' 是证明 V(proof)=1 的。这样两个 snark 结合在一起，构成一个 recursive snark，它的证明大小就比较小（是 P' 产生的 proof'），P 证明时间也比较短（因为 P 比较快，对问题 C 证明；而 P' 虽然慢，但是是对算法 V 做证明，这个问题规模较小，所以耗时不会太长）。

recursive snark 可以不只两层，使用这种 recursion 的思路，可以套很多层。这可以用来解决增量计算的问题。例如，区块链上的 zkRollup，layer2 向 layer1 提交交易证明时，是把 layer2 上的若干交易打包在一起，生成一个零知识证明 π 来证明 “所有这些交易都是合法的”，把交易和 π 提交到 layer1，之后 layer1 验证 π 通过，即可确认打包的所有交易的合法性。

普通的思路是，打包所有交易后统一生成证明，但这样为所有交易生成证明，问题的规模（电路规模）就比较大，这样证明时间就比较长，证明的大小可能也比较大。如图9。而使用递归 SNARK，就可以为若干个交易先生成证明 π_1，然后再产生若干笔交易后，生成 π_2，如此下去，每一个 π_i 的生成时间和证明大小都比较小。等积累到一定数量的交易后，生成一个 “$\pi_1, \pi_2, ..., \pi_n$ 都是可以通过验证的合法证明 (proof)” 的证明 π，作为所有交易的合法性证明，提交到 layer1。这样，总的证明时间较短，证明大小也比较小。如图10。

Nova[14] 和 SuperNova[13] 是典型的递归 SNARK 协议。

11.2 分布式 SNARK

这种方案侧重于 zkSNARK 协议的工程实现方面，通过分布式地实现 zkSNARK 协议算法（特别是 **Prove** 算法，这是当前 zkSNARK 协议的效率瓶颈），提高协议的效率和能够处理的问题规模。

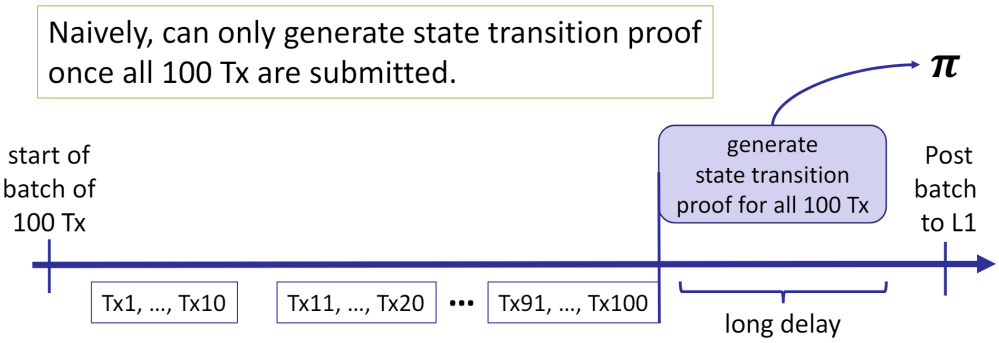

图 9: zkRollup 的简单思路

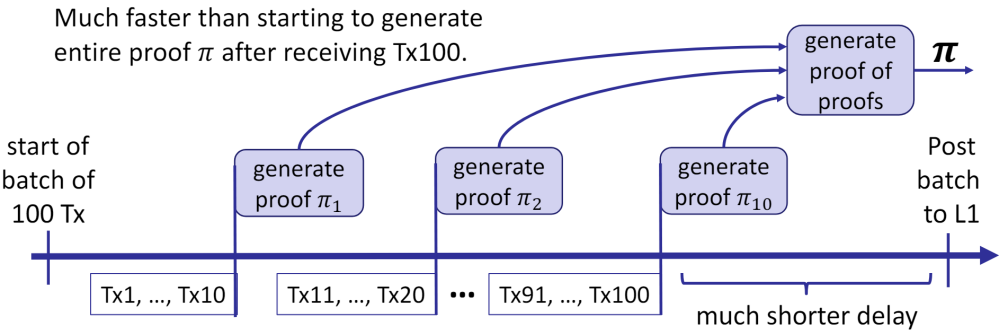

图 10: zkRollup 的递归 SNARK 思路

DIZK[22] 是首个使用分布式实现思路提高 zkSNARK 协议的效率和能够处理的问题规模的。文章使用 Apache Spark 分布式架构对 Groth16 协议 [11] 设计了分布式实现，效果是：1） 能够处理的问题规模（电路规模）比当时最佳协议大两个数量级；2） 平均每个门的处理时间（证明时间）比当前最佳快两个数量级。

Pianist[16] 不仅在工程上分布式实现了 SNARK，而且针对 SNARK 协议进行了改进。文章提出了分布式 pIOP，将其与分布式多项式承诺方案结合。文章基于 Plonk 协议做了分布式转化和改进，并在 zkRollup 场景中进行了测试，实验表明 Pianist 实现了很高的性能优化。

11.3 zkSNARK 研究展望

我认为，zkSNARK 有两个很有前景的研究方向：不需要可信设置的（即透明 (transparent) 的）、抗量子的 zkSNARK，即透明 (transparent) 的 zkSNARK；提高 zkSNARK 的效率和扩展可以处理的问题的规模。

透明的、抗量子的 zkSNARK 。当前高效的 zkSNARK 构造，一般是用双线性对的或基于离散对数的函数承诺。这两类函数承诺构造处的 zkSNARK 都不抗量子，且其中基于双线性对是函数承诺还需依赖可信设置。“可信第三方” 在实际的缺乏信任的使用场景中是不存在的，因此设计透明的 zkSNARK 很有必要。同时，量子计算、量子计算机的发展很快，抗量子的 zkSANRK 研究也很有价值。目前有的透明且抗量子的 zkSNARK 一般基于编码和哈希函数，它们的证明大小和验证时间都不够理想（至少是对数级），需要设计高效的透明抗量子 zkSNARK。

提高 zkSNARK 效率、扩展可处理问题的规模 。当前的 zkSNARK 效率瓶颈主要在证明者 $\mathcal{P}$ 生成证明的步骤，同时计算设备的存储资源有限，这两点限制了 zkSNARK 在大规模的问题（即问题转化为的电路规模很大）上的应用。在实际应用场景中，问题的规模往往是非常大的，现有的 zkSNARK 为他们生成证明可能需要数十分钟甚至数小时，这是不可接受的。为此，需要进一步提高 zkSNARK 的效率，并扩展它们可以处理的问题的规模。

为实现这种提升，现有两种思路，一种是设计递归 SNARK，将大问题拆分为小问题；另一种是采用分布式设计实现 zkSNARK 协议。这两种思路目前都在起步阶段，其设计仍然较为简单（例如 DIZK，仅仅是对 groth16 协议进行了工程上的分布式实现），却已经取得了较好的效果。未来在这两个方向上有很大的发展空间。

参考文献

[1] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 21–32, New York, NY, USA, 1991. Association for Computing Machinery.

[2] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *Electron. Colloquium Comput. Complex.*, 2017.

[3] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge (extended version).

[4] E. Ben-Sasson, A. Chiesa, and N. Spooner. *Interactive Oracle Proofs*, page 31–60. Jan 2016.

[5] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Paper 2020/081, 2020. `https://eprint.iacr.org/2020/081`.

[6] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018.

[7] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[8] A. Gabizon, Z. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive,IACR Cryptology ePrint Archive*, Jan 2019.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. *International Cryptology Conference,International Cryptology Conference*, Jan 1987.

[10] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, page 291–304, Jan 1985.

[11] J. Groth. *On the Size of Pairing-Based Non-interactive Arguments*, page 305–326. Jan 2016.

[12] A. Kate, G. M. Zaverucha, and I. Goldberg. *Constant-Size Commitments to Polynomials and Their Applications*, page 177–194. Jan 2010.

[13] A. Kothapalli and S. Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758, 2022. `https://eprint.iacr.org/2022/1758`.

[14] A. Kothapalli, S. Setty, and I. Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*, page 359–388. Jan 2022.

[15] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments.

[16] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs.

[17] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, Dec 2002.

[18] A. Nitulescu. zk-snarks: A gentle introduction.

[19] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. *IACR Cryptology ePrint Archive,IACR Cryptology ePrint Archive*, Jan 2013.

[20] J. Thaler. Proofs, arguments, and zero-knowledge, 7 2023.

[21] R. Wahby, J. Thaler, M. Walfish, S. Nyu, and N. Georgetown. Doubly-efficient zksnarks without trusted setup.

[22] H. Wu, W. Zheng, A. Chiesa, R. Popa, and I. Stoica. Dizk: A distributed zero knowledge proof system. *IACR Cryptology ePrint Archive,IACR Cryptology ePrint Archive*, Jan 2018.

[23] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vram: Faster verifiable ram with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 908–925, 2018.

[24] 李威翰, 张宗洋, 周子博, and 邓🄵. 简洁非交互零知识证明综述. *密码学报*, 9(03):379–447, 1 月 2022.