从模型确权漏洞到分布式隐私计算架构

洪奕迅 3230102930

浙江大学《密码学进阶》课程期末报告

2025 年 12 月 20 日

摘要

本报告深入探讨了大模型确权机制中的安全性漏洞，探索并验证了一种融合分布式零知识证明与隐私计算的整合型架构。本项目的主要贡献在于工程实现与跨协议集成，验证了理论与实践结合的可行性。首先，项目分析了现有基于触发集的统计确权方案存在的“恶意指控”风险，揭示了攻击者如何利用对抗样本迁移性构造虚假证据 [1]。为解决这一信任危机，本项目复现了分布式证明委托机制 Sc-ZK [2] 的核心逻辑，利用多线性扩展将复杂的电路置换转化为本地矩阵运算。此外，针对验证过程中的隐私泄露问题，引入并适配了 PIRANA 协议 [3]，利用恒重码与 SIMD 技术实现高效的批量不经意验证。在此基础上，本项目搭建了一个**原理验证型**仿真系统，复现了“恶意指控”攻击路径，并验证了分布式计算流与隐私查询的性能增益，完成了从理论漏洞分析到系统架构验证的完整闭环。

关键词：模型确权；分布式零知识证明；隐私计算；恶意指控；zk-SNARK

1 引言

随着大语言模型成为关键的数字资产，模型确权的安全性在工业界引发了广泛关注。当前主流的确权方案（如模型水印和指纹）主要依赖于统计学检验。这种机制隐含了一个假设：即用于验证的“触发集”是唯一且不可伪造的。

然而，在本次期末报告的调研中，我关注到刘健老师等人的研究 [1] 揭示了这一假设的缺陷。攻击者可以通过对抗样本的迁移性构造通用的触发集，从而对无辜模型发起“恶意指控”。这一发现表明，仅靠统计学手段无法建立完全可信的确权机制，转向确定性的密码学证明是解决信任危机的关键。

为了探索更安全的确权路径，我研究了将模型推理转化为 zk-SNARK 电路进行证明的方案。针对单机生成大模型证明面临的内存溢出问题，我参考了张秉晟老师最新的分布式证明架构 [2]，该方案利用多线性扩展将复杂的电路计算拆解为本地矩阵运算，为多节点协作生成证明提供了理论支撑。

此外，考虑到验证过程中的隐私保护，即原告不希望泄露触发集，简单的明文验证并不可取。PIRANA 协议 [3] 提出的基于恒重码的隐私检索技术，为实现高效、批量的“不经意验证”提供了思路。整个从漏洞到防范的完整端到端流程可以参照图1。

2 确权漏洞分析

本章深入剖析现有模型确权机制（MOR）的内在缺陷。基于刘健老师等人的研究 [1]，我将指出主流的基于统计检验的确权方案无法抵御“恶意指控”攻击并给出论文中的实证数据。

2.1 统计确权的形式化定义

现有的 MOR 方案主要分为两类：基于参数变动的数字水印和基于行为分析的模型指纹。

以 Adi 等人提出的经典水印方案 [4] 为例，模型所有者通过在训练阶段植入后门，使得模型对特定的触发集 T 输出预设的错误标签。而以 Dataset Inference [7] 为代表的指纹方案，则通过分析模型在特定样本上的预测余量来判定模型是否在私有数据上进行过训练。

尽管实现方式不同，这两类方案的验证逻辑均可

图 1: 端到端系统流程总览。 (1) **恶意指控**：攻击者通过 I-FGSM 算法生成对抗样本；(2) **分布式证明**：利用 Sc-ZK 架构在分布式节点间并行生成确权证明，解决算力瓶颈；(3) **隐私查询**：客户端通过 PIRANA 协议进行不经意验证，防止隐私泄露。

形式化为一个统计检验过程。设原告 $\mathcal{A}$ 提供触发集 $T = \{(\mathrm{x}_i, y_i)\}_{i=1}^{N}$，验证预言机通过计算模型 F 在 T 上的匹配准确率来判定所有权：

$$\mathcal{V}(F, T) = \mathbb{I}(\mathrm{Acc}(F, T) \geq \tau) \tag{1}$$

其中 τ 为判定阈值。该协议的安全性完全建立在一个假设之上：**触发集 T 是不可伪造的**，即只有训练了源模型的人才能构造出高匹配率的样本。

2.2 恶意指控的攻击机制

然而，这一假设忽略了深度学习模型的一个内生特性：**对抗样本的可迁移性**。正如 Goodfellow 等人 [8] 所发现的，针对一个模型生成的对抗扰动，往往能以高概率欺骗结构相似甚至完全不同的其他模型。

攻击者 $\mathcal{A}$ 利用这一特性，可以在没有任何盗版事实发生的情况下，构造一个恶意的触发集 T'，使得一个完全独立训练的良性模型 F_{ind} 也能满足 $\mathcal{V}(F_{\text{ind}}, T') = 1$。这意味着原告可以利用通过迁移性攻击生成的“伪证”，指控任意无辜的第三方模型为盗版。

2.3 恶意触发集的数学构造

为了构造这种通用的恶意触发集，攻击者需要解决一个带约束的优化问题。由于攻击者无法获取嫌疑模型的参数或梯度，通常采用基于查询的替代模型攻击策略。

攻击者本地训练一组参考模型集合 $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$，这些模型与嫌疑模型执行相同的任务。攻击者的目标是寻找一个微小的扰动 δ，叠加在原图 x 上生成 $\hat{\mathrm{x}} = \mathrm{x} + \delta$，使得 $\hat{\mathrm{x}}$ 能够同时激活所有参考模型的特定响应。

定义集成损失函数如下：

$$\mathcal{L}_{total}(\hat{\mathrm{x}}) = \mathcal{L}(F_{\mathcal{A}}(\hat{\mathrm{x}}), y) + \sum_{F \in \mathcal{F}} \beta_F \mathcal{L}(F(\hat{\mathrm{x}}), y) \tag{2}$$

其中 $F_{\mathcal{A}}$ 为攻击者自己的模型，β_F 为参考模型的权重，y 为攻击者指定的标签。

为了求解该优化问题，刘健老师的工作 [1] 采用了迭代快速梯度符号法（I-FGSM）进行更新。在第 $k+1$ 次迭代中：

$$\hat{\mathrm{x}}_{k+1} = \mathrm{Clip}_{\mathrm{x},\epsilon}\left(\hat{\mathrm{x}}_k - \alpha \cdot \mathrm{sign}(\nabla_{\hat{\mathrm{x}}} \mathcal{L}_{total}(\hat{\mathrm{x}}_k))\right) \tag{3}$$

该公式的物理含义是寻找高维空间中所有同类模型共有的“决策盲区”。通过 Clip 操作将扰动限制在 L_∞ 范数 ϵ 之内，确保生成的触发集不仅相较于原样本差异较小，而且具备极强的跨模型攻击能力。

2.4 漏洞的实证分析

上述优化问题是一个非凸的约束优化，其收敛性在数学上并没有严格的保证，但刘健老师的工作 [1] 对其在 Adi[4]、EWE[5]、Li (b)[9]、DAWN[10]、Lukas[6] 和 DI[7] 等多种主流确权方案进行了实证评估。评估涵盖了 CIFAR-10、ImageNet 和 CelebA 三个数据集。从而我们能够实证地说明这个确权漏洞的存在客观性和严重性。

论文首先确立了不同场景下的判别阈值。如表 1（翻译自论文 [1] **Table 3**）所示，原工作 [1] 根据原告

模型与嫌疑模型的关系，测定了不同确权方案在验证良性模型时的基准表现。显然，“提取”模式下的阈值最高，这对攻击者构成了最严峻的挑战。

然而，表 2（翻译自论文 [1] **Table 4**）的攻击结果表明，原论文构造的恶意触发集具有极强的穿透力。在所有数据集和模型配置下，攻击者的得分不仅超过了较为宽松的“混合阈值”（以粗体显示），在绝大多数情况下甚至击穿了最严格的“提取阈值”（以下划线显示）。

特别值得注意的是表 2 中针对 CelebA 数据集的测试结果。在这一场景下，我们并未训练任何嫌疑模型，而是直接攻击了黑盒的商用 Amazon Rekognition API。尽管我们对该商用模型的内部结构和训练数据一无所知，但仅凭本地模型生成的恶意触发集，就在 DI [7] 方案下实现了 99.9% 的归一化效应值，在 Adi [4] 等其他方案中也均突破了判定阈值。

这一实证分析无可辩驳地证明：基于统计检验的确权机制在面对精心构造的对抗性“伪证”时，几乎没有任何防御能力。只要存在数学上的决策盲区，恶意指控在现实世界中就是可行的。

3 分布式零知识证明架构

为了从根本上解决第二章所述的“恶意指控”问题，必须从概率性的统计检验转向确定性的密码学证明。零知识证明（特别是 zk-SNARK）能够为模型的所有权提供数学上完备的完整性保证。然而，在大模型场景下，生成 zk-SNARK 面临着严峻的计算与内存挑战。

本章基于张秉晟老师的研究 [2]，提出一种可扩展的分布式证明架构，重点阐述如何通过密码学原语的重构来突破“内存墙”与“通信墙”。该架构建立在多线性多项式承诺方案（HyperPlonk [11]）之上，利用 PSS 技术 [12] 重构了底层的密码学原语，其大致的工作框架可以参考图2。

3.1 从单机瓶颈到分布式协作

现代 zk-SNARK 系统（如 Plonk、HyperPlonk）在生成证明时需要保存完整的电路执行轨迹。对于 Transformer 类大模型，其电路规模往往达到数十亿门级别，单台服务器的内存无法容纳如此庞大的数据，导致内存溢出。

现有的协作证明方案如 zkSaaS [13] 虽然尝试引入多方计算，但往往依赖于一个高负载的中心化领导者节点，未能实现完全的负载均衡。

3.2 基于 PSS 的高效并行

为了在分布式环境下实现高效的并行计算，我们采用了 Franklin 和 Yung 提出的打包秘密共享技术 [12]。它是经典的 Shamir 秘密共享方案的推广。

在传统的 SSS 中，一个度数为 d 的多项式仅能隐藏一个秘密值，这导致通信带宽的利用率较低。而在 PSS 中，我们利用拉格朗日插值法，将一个包含 k 个元素的秘密向量 $\mathrm{w} = (w_1, \ldots, w_k) \in \mathbb{F}^k$ 编码进一个多项式 $f(x)$ 中。这里 k 被称为**打包因子**。具体而言，多项式满足：

$$f(-\ell) = w_\ell, \quad \forall \ell \in \{1, \ldots, k\} \tag{4}$$

系统中的 N 个服务器 $S_1, \ldots, S_N$ 分别持有该多项式在不同正整数点上的求值 $y_j = f(j)$ 作为参数 w 的秘密份额 $[w]_i$。

PSS 的核心优势在于其支持 SIMD 操作。利用多项式环的性质，服务器可以在份额上直接进行运算，等效于对原始秘密向量进行运算：

- **线性同态**：本地份额的加法 $[w]_i + [v]_i$ 等价于原始向量的加法 $\mathrm{w} + \mathrm{v}$。

- **乘法同态**：本地份额的乘法 $[w]_i \cdot [v]_i$ 等价于原始向量的点乘 $\mathrm{w} \circ \mathrm{v}$。需要注意的是，乘法操作会将多项式的阶数从 d 翻倍至 $2d$，因此系统需满足 $N > 2d + k - 1$ 以保证可重构性。同时在张秉晟老师的研究 [2] 中证明了 HyperPlonk 的度数通常不超过 4，因此我们可以允许一定程度乘法同态运算造成的度数膨胀而不需要进行频繁的度数规约操作。

在我们的架构中，通过设置 $k = \mathbb{O}(N)$，一次通信和计算可以并行处理 k 个数据项。相较于传统 SSS，PSS 将摊销后的通信开销降低了 k 倍，这是实现大规模电路分布式证明的关键前提。

表 1: 不同确权方案 [4, 5, 9, 10, 6, 7] 下的决策阈值（DI 列显示归一化效应大小百分比，其余为 MOR 准确率）

数据集	阈值类型	Adi	EWE	Li (b)	DAWN	Lukas	DI
CIFAR-10	独立	10.0	1.8	23.0	1.0	28.0	90.0
	混合	29.0	32.9	61.5	38.5	57.5	81.4
	提取	48.0	64.0	100.0	76.0	87.0	72.8
ImageNet	独立	15.0	12.0	30.0	3.0	14.0	76.5
	混合	23.5	37.5	65.0	42.5	30.0	69.6
	提取	32.0	63.0	100.0	82.0	46.0	62.6
CelebA	独立	25.7	3.7	55.0	7.0	21.0	20.0
	混合	42.4	2.9	55.5	26.0	28.5	14.1
	提取	59.0	2.0	56.0	45.0	36.0	8.2

3.3 基于矩阵乘法的置换重构

在分布式 zk-SNARK 中，利用 PSS[12] 的同态性质我们可以解决算术运算，但仍需处理电路中复杂的连线关系导致的需要跨节点通信的置换操作。传统的 Plonk 协议依赖置换检查来约束电路连接，这在分布式环境下会导致不同服务器间产生海量的数据 Shuffle，造成严重的网络拥塞。

为了解决这一问题，Sc-ZK [2] 提出了一种创新的代数转化思路。对于任意电路连线 $\sigma : i \to j$，我们定义一个公开的置换矩阵 $M_\sigma \in \{0,1\}^{N\times N}$，其中 $(M_\sigma)_{j,i} = 1 \iff j = \sigma(i)$。此时，对向量 U 的置换操作等价于线性变换：

$$\mathrm{V} = \mathrm{Shuffle}(\mathrm{U}) \iff \mathrm{V} = M_\sigma \cdot \mathrm{U} \tag{5}$$

在分布式环境下，由于 U 被切片为 $\{[\mathrm{U}]_k\}_{k=1}^K$ 分布在不同节点，利用线性同态性，置换操作可以转化为各节点的本地矩阵运算：

$$\mathrm{V} = \sum_{k=1}^{K} (M_\sigma \cdot [\mathrm{U}]_k) \tag{6}$$

这一变换将全网 Shuffle 转化为本地计算，消除了大规模数据传输的需求。

3.4 基于 MLE 的隐式矩阵构建

虽然转化为矩阵乘法避免了通信，但显式存储巨大的置换矩阵 M_σ（维度达 $2^N \times 2^N$）本身又构成了新的内存压力。为了解决这一问题，我们利用了多线性扩展技术。

根据 Thaler 等人的经典求和校验理论 [14]，任何函数都可以被视为其定义域超立方体上的多线性多项式。利用拉格朗日插值，矩阵乘法 $\mathrm{V} = M \cdot \mathrm{U}$ 在 MLE 域上可以表示为：

$$\tilde{V}(r) = \sum_{y\in\{0,1\}^\ell} \tilde{M}(r,y) \cdot \tilde{U}(y) \tag{7}$$

核心突破在于：我们无需显式存储矩阵 $\tilde{M}$。利用置换 σ 的稀疏性（每行仅有一列为 1），$\tilde{M}$ 具有解析解：

$$\tilde{M}(r,y) = \tilde{eq}(r,\sigma(y)) \tag{8}$$

其中 $\tilde{eq}$ 是选择器多项式，$\sigma(y)$ 代表 $M_{\sigma(y),y}$ 处为 1，即预期的置换操作。

这使得服务器可以在 $O(1)$ 的内存复杂度下，动态计算矩阵元素的值，从而彻底打破了存储大模型电路连接关系的内存墙限制。

表 2: 恶意指控对各确权方案的攻击效果（**粗体**表示高于混合阈值，下划线 表示高于“提取”阈值）

数据集	模型配置 (F_S vs. $F_{\mathcal{A}}$)	Adi	EWE	Li (b)	DAWN	Lukas	DI
CIFAR-10	不同结构 & 不同数据	**94.3**	**69.3**	**94.3**	**69.3**	**94.3**	**100**
	相同结构 & 不同数据	**98.0**	**100.0**	**98.0**	**100.0**	**98.0**	**99.1**
	不同结构 & 相同数据	**99.0**	**78.3**	**99.0**	**78.3**	**99.0**	**98.6**
ImageNet	不同结构 & 不同数据	**72.6**	**87.6**	**72.6**	**87.6**	**72.6**	**100**
	相同结构 & 不同数据	**93.7**	**97.0**	**93.7**	**97.0**	**93.7**	**100**
	不同结构 & 相同数据	**84.6**	**89.0**	**84.6**	**89.0**	**84.6**	**100**
CelebA	不同结构 & 不同数据 (Amazon Rekognition API)	**68.4**	**68.0**	**68.4**	**68.0**	**68.4**	**99.9**

3.5 分层分布式求和校验

为了验证全电路约束（即证明 $\sum P(x) = 0$），我们需要在分布式环境下执行 Sumcheck 协议。早期的分布式 Sumcheck 方案（如 zkBridge [15]）要求数据在节点间连续存储，这与我们的 PSS 切片存储方式不兼容。

本项目采用了张秉晟老师等人 [2] 提出的**分层并行策略**。我们将计算树切分为“本地层”与“协作层”：

- **底层（前 $\ell - k$ 轮）**：利用 MLE 的张量积结构，各服务器在本地独立计算子树，此阶段利用数据在存储中的局部性在节点本地完成。利用 MLE 的代数性质，Sumcheck 协议的每一轮折叠本质上是一个线性组合操作。根据原论文 [2] 第 4.1 节的推导，对于第 i 轮折叠（其中 $i \le \ell - k$），服务器 S_j 持有的更新后的份额 $[x_{i+1}]$ 可以完全由上一轮的本地份额计算得出：

$$[\boldsymbol{x}_{i+1}] = (1 - r_i) \cdot [\boldsymbol{x}_i]_{\text{left}} + r_i \cdot [\boldsymbol{x}_i]_{\text{right}} \quad (9)$$

  其中 r_i 是验证者公开广播的随机挑战数。由于采用了 PSS 对数据进行分片，在前 $\ell - k$ 轮中，折叠所需的“左半部分”数据 $[\boldsymbol{x}_i]_{\text{left}}$ 和“右半部分”数据 $[\boldsymbol{x}_i]_{\text{right}}$ 均位于同一台服务器的本地内存中。

- **顶层（后 k 轮）**：类似经典的 Sumcheck 协议，仅在最后阶段聚合极少量的数据。

这种分层设计使得绝大部分计算任务都在本地完成。根据原论文的**基准测试数据**（参考 [2] 中 Table 3，本文转录于表 3），在电路规模为 2^{21} 且服务器数量 $N = 128$ 的设置下，zkSaaS [13] 的中心节点面临 90 GB 的通信负担，而采用本架构可将单节点通信量降低至 43 MB。这表明，该架构在广域网环境下具有显著的带宽优势。

表 3: 与 zkSaaS [13] 的性能对比（实验条件：$|C| = 2^{21}, N = 128$）。其中“加速”与“节省”均以单机 Prover 为基准。

方案	角色	局域网加速	广域网加速	空间节省	通信开销
本文方案	S_i (普通节点)	28.0×	25.5×	11.8×	**43 MB**
zkSaaS [13]	S_0 (领导者)	0.9×	0.1×	0.86×	90 GB
	S_i (普通节点)	0.9×	0.1×	16.8×	718 MB

4 基于常重码的高效隐私查询

在分布式 zk-SNARK 解决了确权过程的“完整性”与“可扩展性”后，我们必须面对另一个关键挑战：**验证过程中的隐私泄露**。本章基于刘健老师等人的研究 PIRANA [3]，提出一种基于常重码的高效隐私检索方案，以实现对嫌疑模型的不经意验证。

4.1 验证环节的隐私风险

传统的模型确权协议要求原告 $\mathcal{A}$ 将触发集 T 明文发送给持有嫌疑模型的被告 $\mathcal{S}$。这种交互方式存在严重的安全隐患：

- **探针失活**：一旦恶意的被告 $\mathcal{S}$ 获知了具体的触发

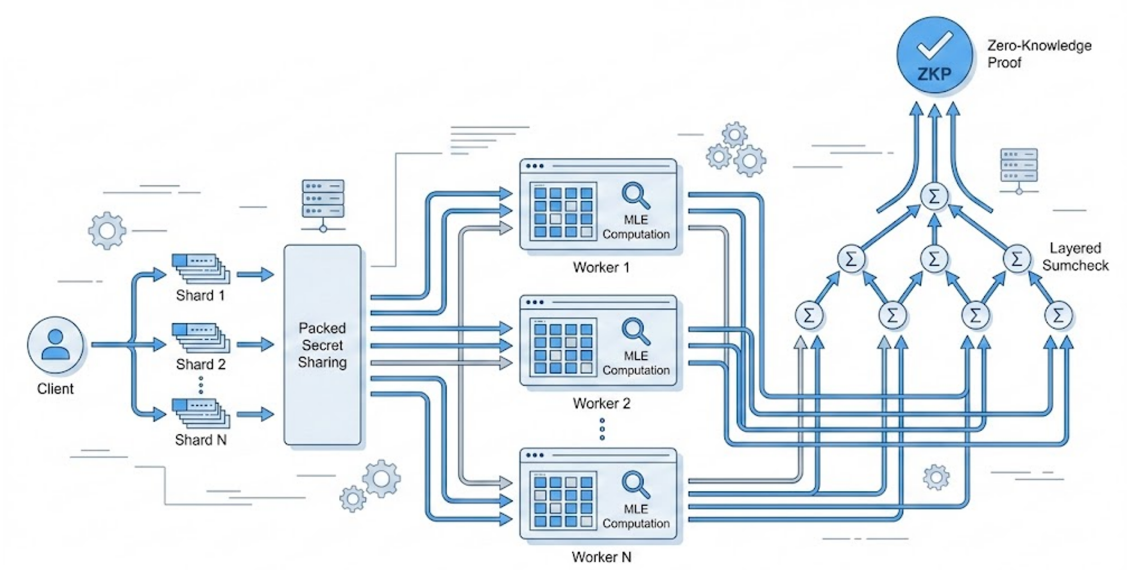

图 2: Sc-ZK 分布式隐私计算架构示意图。左侧展示了客户端通过 PSS 将数据分片；中间展示了各 Worker 节点并行进行本地 MLE 运算；右侧展示了通过分层求和校验聚合生成最终的零知识证明。该架构有效解决了单机内存瓶颈。

集样本，他可以通过针对性的微调或过滤机制，让模型在这些特定样本上输出正确结果，从而“抹除”水印特征，导致验证失效。

- **指纹窃取**：触发集本身往往包含了原告模型的核心知识产权（如特有的对抗样本分布）。明文传输会导致原告的高价值资产泄露。

因此，理想的确权验证应当是不经意的：原告能够查询模型在触发集上的输出，但被告无法得知原告具体查询了哪些样本。这本质上是一个私有信息检索问题，并且我们理想中希望得到的不仅只是一个交集检索问题，而是想要得到数据标签，这在技术上对应了 LPSI。

4.2 PIRANA：恒重码与 SIMD 的同态结合

现有的单服务器 PIR 协议（如 SealPIR [16]）通常将数据库索引编码为独热码。对于大小为 n 的数据库，查询向量稀疏且长度为 n，导致巨大的通信与计算开销。

PIRANA [3] 的核心创新在于引入了常重码。我们将每个索引 $i \in [n]$ 映射为一个码字 $\mathrm{c} \in CW(m,k)$，其中 m 为码长，k 为汉明重量（即码字中 1 的个数）。根据组合数学性质，码长 $m \approx \sqrt[k]{k! \cdot n}$，远小于 n。

在 FHE 密文域下，判断查询码字 x 与数据库索引码字 y 是否相等的算子可以定义为：

$$f(\mathrm{x},\mathrm{y}) = \prod_{j:\mathrm{y}[j]=1} \mathrm{x}[j] \tag{10}$$

仅当查询 x 在所有 k 个对应位置均为 1 时，乘积才为 1，否则为 0。

现有的单服务器 PIR 协议（如 CwPIR [17]）虽然引入了恒重码来降低查询开销，但其核心算子仍然是基于标量的：对于大小为 n 的数据库，服务器需要对每一个条目执行同态乘法，导致计算复杂度高达 $\mathbb{O}(n)$。

PIRANA [3] 通过以下两个核心步骤突破了这一瓶颈：

4.2.1 数据库的矩阵化重排

利用全同态加密案（如 BFV）支持的 SIMD 特性，PIRANA 不再将数据库视为线性的元素列表，而是将其重排为一个矩阵。设 FHE 方案的多项式槽位数为 N（典型值为 4096 或 8192）。我们将包含 n 个元素的数据库重塑为一个 $N \times t$ 的矩阵 M，其中 $t = \lceil n/N \rceil$。对于任意索引 $i \in [n]$，其在矩阵中的位置由行下标 $r = i \pmod N$ 和列下标 $c = \lfloor i/N \rfloor$ 唯一确定。

4.2.2 SIMD 并行计算原理

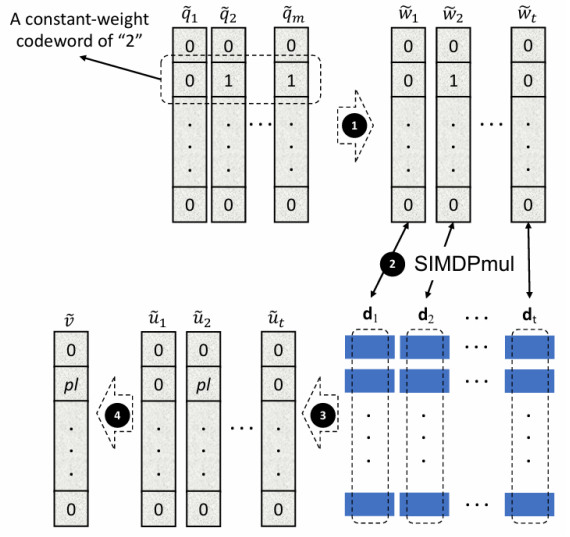

图 3: PIRANA 单次查询工作流示意图（引用自 [3] **Figure 1**）。图中展示了如何将列索引 c 编码为恒重码，并利用 SIMD 乘法并行处理数据库的多个列块 $(d_1, \ldots, d_t)$，从而将 $O(n)$ 的标量乘法降低为 $O(n/N)$ 的密文乘法。

在查询阶段，客户端不再生成针对 n 个元素的查询向量，而是生成针对 t 个列的查询码字。

- **查询压缩**：客户端将目标列索引 c 编码为恒重码 $\mathrm{q} \in CW(m,k)$，并将其加密为 m 个 SIMD 密文。每个密文的第 r 个槽位存储了码字的对应位，而其他槽位则被置零或填充其他查询。
- **列级批处理**：服务器执行同态乘法时，不再是“密文 × 标量”，而是“密文 × 密文（代表一整列数据）”。由于 SIMD 特性，一次密文乘法实际上并行完成了矩阵中一整列 N 个元素的匹配检查。

4.2.3 计算复杂度

通过这种设计，同态相等性测试算子 $f(\mathrm{x},\mathrm{y}) = \prod_{j:\mathrm{y}[j]=1} \mathrm{x}[j]$ 的执行次数发生了质的变化：

$$\mathrm{Mul}_{\mathrm{CwPIR}} \approx (k-1)\cdot n \quad \xrightarrow{\text{SIMD}} \quad \mathrm{Mul}_{\mathrm{PIRANA}} \approx (k-1)\cdot\lceil \frac{n}{N} \rceil \tag{11}$$

理论上，这将计算开销降低了 N 倍。当 $N = 8192$ 时，意味着近 4 个数量级的理论加速。

4.2.4 加速效果实证

为了验证这一理论优势，我们引用原论文的微基准测试数据（翻译自原论文 [3]**Table 3**）。如表 4 所示，在数据库规模 $n = 2^{16}$ 的设置下，PIRANA 的单次查询相比原始的 CwPIR 协议实现了 188.6 倍的端到端加速，且选择向量生成阶段的耗时几乎可以忽略不计。

4.3 基于布谷鸟哈希的批量查询调度

模型确权通常需要验证数百个触发样本，这意味着我们需要多查询支持。如果简单重复运行单次 PIR，效率将无法接受。

原文 [3] 采用了基于 3-way 布谷鸟哈希的批处理策略 [18]。

1. **映射阶段**：将数据库中的 n 个模型指纹映射到 B 个桶中，每个指纹有 3 个候选桶位置。
2. **无冲突调度**：对于原告的查询集 Q（设 $|Q| = L$），我们在二分图上寻找匹配，确保每个桶最多处理 1 个查询请求。

该机制使得系统吞吐量与 $\min(B, \text{SIMD\_Slots})$ 成正比，实现了通信开销不再随查询数量 L 线性增长，达成了“一次通信，批量取回”的效果。

4.4 向 LPSI 的扩展：OPRF 掩码

为了进一步防止被告 $\mathcal{S}$ 的模型输出被暴力遍历，我们将协议扩展为标签隐私集合求交。原文 [3] 借鉴了 Chen 等人 [19] 的思路，引入不经意伪随机函数（OPRF）：

1. **OPRF 盲化**：双方基于私钥 k 计算伪随机标签 $K_i = F_k(x_i)$。
2. **异或掩码**：服务器存储加密后的模型输出 $\text{Enc\_Label} = \text{Label} \oplus K_i$。
3. **PIR 检索**：原告通过 PIRANA 检索密文，并在本地解开掩码。

然而，现有的 SOTA LPSI 方案（如 Cong 等人 [20]）在 Setup 阶段存在严重的性能瓶颈。这是因为

表 4: PIRANA 与 CwPIR 的微基准测试对比。参数设置：$k = 2, N = 2^{13}$，Payload=20KB。

元素数量 n / 数据库大小 (MB)		2^8 / 5.2	2^9 / 10	2^{10} / 21	2^{11} / 42	2^{12} / 84	2^{13} / 170	2^{14} / 340	2^{15} / 670	2^{16} / 1300
CwPIR [17]	选择向量生成 (s)	3.9	7.8	15.5	31.0	61.7	123.1	246.2	492.7	983.3
	内积计算 (s)	0.2	0.4	0.8	1.6	3.3	6.5	13.1	26.2	52.3
	服务端总耗时 (s)	4.1	8.2	16.3	32.6	65.0	129.7	259.4	518.9	1035.6
PIRANA (单次查询)	选择向量生成 (s)	0.001	0.001	0.001	0.001	0.001	0.001	0.027	0.05	0.1
	内积计算 (s)	0.22	0.24	0.28	0.36	0.52	0.86	1.57	2.86	5.39
	服务端总耗时 (s)	**0.22**	**0.24**	**0.28**	**0.36**	**0.52**	**0.86**	**1.6**	**2.9**	**5.49**
加速比		18.6×	34.2×	58.2×	90.6×	125×	151×	162.1×	178.9×	**188.6**×

它们要求服务器对数据库进行复杂的多项式插值，复杂度高达 $\mathbb{O}(n^2 \log n)$ 计算量巨大。

相比之下，PIRANA 的 Setup 阶段仅需将 Payload 编码为 NTT 格式，计算复杂度极低。如表 5 所示，根据原论文 [3] 的实验数据，在同等硬件条件下，PIRANA 的 Setup 速度相比 Cong 等人的方案提升了 331 倍。

表 5: LPSI Setup 阶段耗时对比（数据源自 [3] 第 6.4 节）。其中 Cong et al. [20] 需要昂贵的多项式插值，而 PIRANA 仅需 NTT 变换。

LPSI 方案	计算资源配置	Setup 耗时
Cong et al. [20]	32 线程	1256.3 s
	单线程	14.4 hours
PIRANA	**单线程**	**157 s**
加速比 (单线程 vs 单线程)		**331**×

这一特性使得 PIRANA 极大地适应了模型参数频繁微调或更新的场景。当模型权重 w 发生变化导致输出改变时，传统方案需要耗费数小时重新构建索引，而 PIRANA 仅需不到 3 分钟即可完成更新，保证了确权系统的实时可用性。

5 仿真系统设计

为了验证上述理论架构的可行性与核心性能指标，本期末展示基于 Python、PyTorch 和 Streamlit 开发了一个可视化的交互式仿真原型系统。该系统集成了攻击复现、分布式计算模拟和隐私查询性能测试三大核心模块，旨在提供从确权漏洞到防御架构的完整演示。

5.1 系统架构总览

仿真系统采用模块化设计，各个功能组件相互独立但逻辑连贯。

- **False Claims**：基于 PyTorch 实现，负责加载深度学习模型，执行 I-FGSM 攻击算法生成恶意触发集，并验证其在独立模型上的迁移性。
- **Scalable ZK**：基于 Python 多线程库 `concurrent.futures` 模拟分布式服务器集群，验证 PSS 分片计算与聚合的负载均衡特性。
- **PIRANA**：模拟 PIRANA 协议的 SIMD 算术逻辑，通过真实执行 CPU 密集型矩阵运算来测量批量查询场景下的性能增益。

5.2 恶意指控攻击复现

该模块旨在复现第二章所述的“恶意指控”攻击。我们在 CIFAR-10 风格的简化场景下，模拟了针对独立训练模型的迁移性攻击。

5.2.1 模型定义与初始化

我们定义了一个具有鲁棒性的卷积神经网络 `RobustCNN` 作为基座模型。为了模拟现实中“不同数据、独立训练”的场景，我们通过对同一组预训

练权重添加不同分布的高斯噪声来生成源模型和嫌疑模型。

Listing 1: 模型变体生成逻辑

```
def create_variant(master_state,
    noise_level=0.05):
    model = RobustCNN()
    new_state = {}
    for k, v in master_state.items():
        # 模拟独立训练带来的参数差异
        noise = torch.randn_like(v) *
            noise_level * v.std()
        new_state[k] = v + noise
    model.load_state_dict(new_state)
    return model
```

5.2.2 集成 I-FGSM 实现

核心攻击逻辑采用了原论文所述的 I-FGSM 算法。注：在我的代码实现中，该函数被命名为 *ensemble_pgd_attack*，这是因为 *I-FGSM* 在数学本质上等价于 L_∞ 范数约束下的 *PGD* 算法。

为了提高对抗样本的迁移性，我们采用了集成攻击策略。攻击者联合一组本地替身模型共同优化损失函数。代码实现如下：在每次迭代中，计算所有替身模型的平均损失，获取梯度符号，并执行步进与裁剪操作。

Listing 2: 基于 I-FGSM 的集成攻击实现

```
def ensemble_pgd_attack(main_model,
    surrogates, image, epsilon, ...):
    # ... (省略初始化代码)
    for _ in range(num_iter):
        total_loss = 0
        # 遍历所有替身模型计算联合损失 (
            Ensemble)
        for net in attack_models:
            output = net(perturbed_image.
                unsqueeze(0))
            loss = F.nll_loss(output,
                target)
            total_loss += loss

        avg_loss = total_loss / len(
            attack_models)
        avg_loss.backward()

        # I-FGSM 核心更新步 (Iterative
            Update & Clip)
        with torch.no_grad():
            # 1. 计算梯度符号 (Sign)
            grad = perturbed_image.grad.
                sign()
            # 2. 沿梯度方向步进 (Step)
            perturbed_image =
                perturbed_image - alpha *
                grad
            # 3. 投影回 epsilon 邻域 (
                Project/Clip)
            eta = torch.clamp(
                perturbed_image - image, -
                epsilon, epsilon)
            perturbed_image = torch.clamp(
                image + eta, 0, 1)
            perturbed_image.requires_grad
                = True
    return perturbed_image
```

实验结果 [图 4] 能看到我们利用集成攻击成功实现了对抗样本的迁移。

图 4: 恶意指控攻击复现界面。左侧为添加扰动的原始图像，右侧为生成的对抗指纹。结果显示被告模型 B 被成功误导（Confidence=0.17）。

5.3 分布式 zk-SNARK 模拟

该模块用于验证第三章提出的 Sc-ZK 分布式架构在计算任务分发上的有效性。

5.3.1 分布式节点仿真

我们使用 `ThreadPoolExecutor` 来模拟 N 个并行工作的服务器节点。每个节点执行 `server_compute_share` 函数，模拟处理分配到的 PSS 份额。为了模拟真实的计算负载，我们根据向量大小插入了 `time.sleep`。

Listing 3: 分布式计算节点模拟

```
class DistributedSumcheck:
    def run_protocol(self):
        # 模拟 PSS 数据分片
        data = np.random.rand(self.
            vector_size).astype(np.float32
            )
        shares = np.array_split(data, self
            .num_servers)

        results = []
        # 使用线程池模拟分布式集群
        with ThreadPoolExecutor(
            max_workers=self.num_servers)
            as executor:
            futures = []
            for i in range(self.
                num_servers):
                futures.append(executor.
                    submit(self.
                    server_compute_share,
                    i, shares[i]))
            # ... (聚合结果)
        return results, true_sum
```

实验结果 [图 5, 图 6] 显示，随着服务器数量 N 的增加，单节点的计算耗时显著下降，且各节点负载均衡，验证了“完全分布式工作负载”的设计目标。

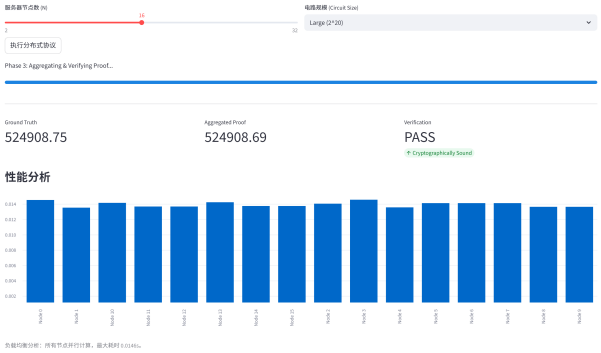

图 5: 分布式计算模拟界面（$N = 16$）。下方柱状图展示了 16 个节点的计算耗时分布，验证了 PSS 方案在负载均衡方面的优异表现。

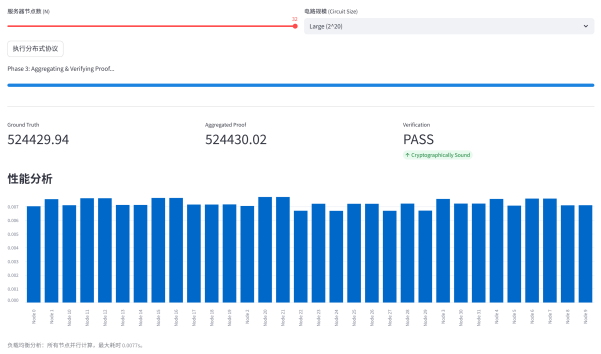

图 6: 分布式计算模拟界面（$N = 32$）。下方柱状图展示了 32 个节点的计算耗时分布，且能看到相同负载节点增加后单节点运行时间明显变短。

5.4 PIRANA 性能基准测试

该模块通过实时运行 CPU 密集型任务，对比了 Naive PIR 与 PIRANA 在批量查询场景下的性能差异。

5.4.1 协议简化与算子模拟

我们基于 Python 实现了 PIRANA 的核心算术逻辑。为了在有限的计算资源下验证 SIMD 的加速比特性，我们采用矩阵乘来模拟同态加密中的密文乘法开销。注：由于未接入真实的 *FHE* 库，本模块侧重于验证恒重码与 *SIMD* 技术带来的**逻辑加速比**，而非绝对的密码学运算耗时。

5.4.2 实时性能测量

为了获得真实的性能数据，我们并未简单返回预设值，而是通过执行矩阵乘法 `np.dot(A, B)` 来模拟同态加密中的密文乘法开销。

在 `measure_execution_time` 函数中，我们分别测量了线性处理和 PIRANA 的耗时。对于 PIRANA，我们模拟了 SIMD 的批处理特性：将 L 个查询打包进 $\lceil L/\text{slots} \rceil$ 个批次中执行。

Listing 4: PIRANA 实时性能基准测试

```
def measure_execution_time(self,
    num_queries_L):
    # 1. 测量 Naive 方法 (线性增长)
    # ... (省略 Naive 测量代码)

    # 2. 测量 PIRANA 方法 (SIMD 摊销)
    start_pirana = time.perf_counter()
    num_batches = math.ceil(num_queries_L
        / self.slots)

    # 模拟 Setup 开销 (如恒重码编码)
    for _ in range(5):
        self._heavy_simulated_op()

    # 真实执行 Batch 处理 (受 SIMD 槽位限
        制)
    for _ in range(num_batches):
        self._heavy_simulated_op()

    t_pirana = time.perf_counter() -
        start_pirana
    return t_naive, t_pirana
```

运行结果 [图 7] 表明，PIRANA 展现出了巨大的性能优势，耗时几乎不随 L 增加而显著上升，体现了 $O(1)$ 的摊销复杂度特性。

图 7: PIRANA 实时性能基准测试。在极端批量场景下（$L = 8192$），PIRANA 的摊销耗时极低且平稳，而 Naive 方法呈线性增长，加速比高达 1367×。

5.5 LPSI 协议功能仿真

除了性能基准测试外，我们还在仿真系统中开发了一个功能演示模块，用于验证基于 PIRANA 的 LPSI 协议的端到端业务逻辑。

5.5.1 业务场景模拟

该模块模拟了一个典型的隐私查询场景：服务器端维护一个包含大量用户的信用评分数据库（Keyword=User_ID, Payload=Score），客户端持有一批待查询的用户 ID 列表。协议的安全目标是：客户端仅能获得交集用户的信用分，且服务器无法获知客户端的具体查询对象。

5.5.2 协议流程仿真

在代码的 `simulate_lpsi` 函数中，我们模拟了“数据生成 → 盲化查询 → 结果匹配”的完整数据流。虽然为了保证 GUI 演示的流畅性，底层的 OPRF 和同态加密运算由逻辑判断代替，但系统界面通过状态流转完整复现了理论章节所述的三个关键阶段：

1. 客户端进行 OPRF 盲化与恒重码编码。
2. 服务器执行批量 PIRANA 检索（SIMD 处理）。
3. 客户端解开掩码并提取交集数据。

代码实现如下，展示了如何通过随机采样模拟交集与差集的处理结果：

Listing 5: LPSI 数据流逻辑模拟

```
def simulate_lpsi(self, server_db_size
    =1000, client_query_size=10):
    # 1. 初始化服务端数据库 (ID -> Score)
    server_data = {f"User_{i}": random.
        randint(300, 850)
                   for i in range(
                       server_db_size)}

    # 2. 生成混合查询 (包含交集与差集)
    existing_keys = random.sample(list(
        server_data.keys()),
        client_query_size // 2)
    non_existing_keys = [f"User_{9000+i}"
        for i in range(...)]
    client_queries = existing_keys +
        non_existing_keys

    # 3. 模拟协议执行结果
    results = []
    for q in client_queries:
```

```
        if q in server_data:
            # 在交集中：OPRF 掩码被正确移
                除，获得 Payload
            status = "Intersection"
            payload = str(server_data[q])
        else:
            # 不在交集中：解密结果为随机值
                或标识符
            status = "Miss"
            payload = "N/A (Masked)"
        results.append({"Query": q, "
            Status": status, "Payload":
            payload})
    return pd.DataFrame(results)
```

实验结果 [图 8] 能够看到我们的仿真程序能够很好完成取得预期数据而不泄露原始信息的目标。

图 8: LPSI 协议执行演示。绿色行表示查询 ID 在交集中，客户端成功解密 Payload；红色行表示不在交集中，Payload 仍为掩码状态。

两部分的演示结果直观地证明了 PIRANA 架构的灵活性：它不仅能作为底层的 PIR 原语提供加速，还能通过集成 OPRF 层，平滑扩展为应用层的 LPSI 协议，从而在保护模型参数的同时保护查询者隐私。

6 总结与展望

6.1 工作总结

本期末项目针对大模型确权场景下的信任危机与计算瓶颈，提出了一套端到端的分布式隐私计算架构。

- **确权漏洞**：证实了基于统计检验的传统确权方案存在“恶意指控”漏洞 [1]，证明了引入密码学确定性证明的必要性。
- **架构创新**：基于 Sc-ZK 架构 [2]，利用多线性扩展将电路置换转化为本地矩阵运算，结合分层求和校验协议，成功将单机 OOM 的风险分散至分布式集群，实现了计算与存储的线性扩展。
- **隐私增强**：集成了 PIRANA 协议 [3]，利用恒重码与 SIMD 技术实现了对触发集的高效不经意验证。

6.2 挑战与解决方案

尽管存在逻辑闭环，但在迈向工业级落地的过程中，本架构仍面临两个严峻的实际挑战。

6.2.1 挑战一：从半诚实到恶意安全性

当前的分布式架构基于半诚实假设，即假设服务器会忠实执行协议但试图通过中间数据推导隐私。然而在现实的 Proof-as-a-Service 场景中，恶意的服务器可能会伪造计算结果（例如，故意通过错误的证明来通过验证，或者破坏 PSS 的重构）。

解决方案：引入信息论 MAC 验证。参考 Weng 等人在 **AntMan** [21] 中的工作，我们可以为每一个 PSS 份额附加一个信息论安全的消息验证码。

具体而言，对于秘密值 $[x]$，服务器需额外维护 $[\alpha \cdot x]$（其中 α 是全局密钥）。在每一轮计算结束后，通过验证 $\alpha \cdot \sum x_i \stackrel{?}{=} \sum \gamma_i$ 来保证计算的正确性。这种方法仅增加极少量的通信开销（通常为 2×），即可将安全性提升至恶意模型。

6.2.2 挑战二：客户端的初始化瓶颈

在当前的架构中，客户端需要负责将庞大的模型参数 w 进行 PSS 分片并分发给 N 个服务器。对于一个 70B 参数的大模型，客户端需要上传 $N \times 140$GB 的数据。这对于轻量级设备而言是不可接受的“带宽墙”，甚至抵消了委托计算带来的优势。

解决方案：硬件加速与静默预处理。为了解决这一问题，未来的工作可以探索两个方向：

1. **基于 PCG 的静默预处理**：利用伪随机相关生成器，客户端只需分发短种子，服务器即可在本地扩展出相关的随机掩码，从而将通信复杂度从线性压缩至亚线性。

2. **异构硬件加速**：参考 Zhang 等人的 **PipeZK** [22] 架构，在服务器端部署 FPGA/GPU 集群。利用 FPGA 的流水线特性加速 NTT 和 MSM 运算，从而掩盖广域网下的通信延迟。这将使得系统能够容忍更高的网络延迟，降低对客户端带宽的依赖。

综上所述，从“统计确权”走向“密码学确权”是充满可能的大模型版权保护的未来方向。通过结合分布式计算、零知识证明与隐私检索技术，我们为构建可信、安全、高效的 AI 基础设施提供了有力的理论与实践支撑。

7 产物说明

本期末展示的所有仿真代码已在 Github 开源，在安装必要包依赖后即可利用 `Streamlit` 参考 Listing6 运行并在浏览器实时预览和进行仿真：

注：对于第一部分的攻击仿真，可能会出现除 *0* 外所有可用 ϵ 值都可以造成迁移攻击成功的情况，这是我们训练源模型和嫌疑模型的逻辑决定的（他们是在相同结构上添加了不同的随机噪声），多运行几次就能看到小 ϵ 失败而大 ϵ 成功的期望结果。

Listing 6: 运行命令

```
pip install -r requirements.txt
python -m streamlit run demo.py
```

参考文献

[1] Jian Liu, Rui Zhang, Sebastian Szyller, Kui Ren, and N. Asokan. False Claims against Model Ownership Resolution. In *33rd USENIX Security Symposium (USENIX Security)*, 2024.

[2] Liu, X., Zhou, Z., Wang, Y., Pang, Y., He, J., Zhang, B., et al. Scalable Collaborative zk-SNARK and Its Application to Fully Distributed Proof Delegation. In *34th USENIX Security Symposium (USENIX Security)*, 2025.

[3] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. PIRANA: Faster Multi-query PIR via Constant-weight Codes. In *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[4] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by back-dooring. In *27th USENIX Security Symposium*, 2018.

[5] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium*, 2021.

[6] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2020.

[7] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset inference: Ownership resolution in machine learning. In *International Conference on Learning Representations (ICLR)*, 2021.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[9] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN. In *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[10] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. DAWN: Dynamic adversarial watermarking of neural networks. In *Proceedings of the 29th ACM International Conference on Multimedia*, 2021.

[11] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *Advances in Cryptology –EUROCRYPT 2023*, 2023.

[12] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing (STOC)*, 1992.

[13] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In *32nd USENIX Security Symposium*, 2023.

[14] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology – CRYPTO 2013*, 2013.

[15] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[16] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[17] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In *31st USENIX Security Symposium*, 2022.

[18] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[19] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[20] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. AntMan: Interactive Zero-Knowledge Proofs with Sublinear Communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[22] Ye Zhang, Shuo Chen, and G. Edward Suh. PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture. In *48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.