

EthereumにおけるVOLE-in-the-Headの検証コスト評価

津田匠貴
Nyx Foundation

November 23, 2025

Abstract

Vector Oblivious Linear Evaluation (VOLE) を用いたゼロ知識証明 VOLE-in-the-Head (VOLE-itH) は、線形演算をVOLEプリプロセスに置き換えることで証明生成コストを削減する。一方、Ethereum での公開検証ではオンチェーン検証コストが実用性のボトルネックとなる。本研究では、VOLE-itH のオンチェーン適用に向けた実装ベースの評価として、(1) 証明生成・検証の計算量と証明サイズを SNARK (Groth16) で圧縮した上で Ethereum verifier を実装し、そのガスコストを評価した。
SHA-256/Keccak-F/基本論理回路でベンチマークした結果、VOLE-itH の証明生成は Circom 実装より最大 15.5× 高速だが証明サイズは 6000× 増大した。SNARK で圧縮すると証明サイズを 1,055 バイトに固定でき、オンチェーン検証は 1,055 gas で完了した。これらの結果から、VOLE-itH をブロックチェーン応用に適用する際のコストを評価した。

1 序論

1.1 ゼロ知識証明の進化とオンチェーン検証の課題

ゼロ知識証明は、ある計算が正しく実行されたことを、その計算に関する入力情報を一切明らかにしない性質はプライバシー保護が強く求められる現代のデジタル社会において極めて重要であり、特にスマートコントラクトプラットフォームにおいては、計算の正当性をトランザクションとして示すとともに、ZKPをオンチェーンで検証する際には、証明サイズ、検証計算量、そしてそれに伴うガス代などの課題が存在する。

1.2 VOLEベースZKPとVOLE-in-the-Head

この課題に対し、証明者の計算効率を大幅に向上させる新しいZKPの系統として、VOLE (Vector Oblivious Linear Evaluation) ベースのプロトコルが登場した。これらのプロトコルは、従来のSNARKs (Succinct Non-Interactive Argument of Knowledge) で主流であったR1CS (Rank-1 Constraint System) とは異なるアプローチを取り、特に証明者の計算負荷を軽減することに成功。SNARKsは証明サイズが小さく検証が高速なためオンチェーン検証で広く利用されており、Groth16やFiat-Shamir変換を適用することで、誰でも検証可能な公開証明 (publicly verifiable proof) を生成可能にした画期的な手法である[1]。これにより、証明者側の高い計算効率と

1.3 研究の目的と貢献

VOLEitHは理論的には有望であるものの、その実用性、特にオンチェーン検証における具体的な性質を明確にする。本研究の目的は、VOLEitHの特性を活かした軽量な証明者がEthereum上で検証することが可能である。具体的には、以下の項目を詳細に測定・分析する。

- ・ 証明生成と検証にかかる時間
- ・ 生成される証明のサイズ
- ・ 証明者と検証者の計算負荷 (CPU、メモリ)
- ・ 最終的なオンチェーン検証にかかるガス代

本研究は、VOLEitHのオンチェーン応用における実現可能性と技術的なトレードオフを明らかにする。

2 前提知識と関連研究

本章では、後続のベンチマークと考察を理解するために必要な暗号学的前提を簡潔に整理する。

2.1 Vector Oblivious Linear Evaluation (VOLE)

VOLEは、二者間で線形関係を隠したままMAC相関を生成するプリミティブである。

有限体 F 上で、送信者はベクトル $\mathbf{u}, \mathbf{v} \in F^m$ を、受信者はグローバル鍵 $\boxtimes \in F^m$ を入力とする。プロトコル終了後、受信者は

$$\mathbf{q} = \mathbf{u} \circ \boxtimes - \mathbf{v} \in F^m \quad (1)$$

を満たすペア (\mathbf{u}, \mathbf{q}) を得る (\circ は要素積)。 \mathbf{q} は \mathbf{u} に対する情報理論的MACとなり、 \boxtimes を知らずに \mathbf{q} を固定したまま \mathbf{u} を改ざんすることはできない (binding)。この性質が後述の健全性担保に直接利用される。

一般的に、入力 \mathbf{u} を \mathbb{F}_2 に制限しつつ、 $\mathbf{v}, \mathbf{q}, \boxtimes$ を拡大体 \mathbb{F}_{2^λ} 上に置くサブフィールドV-OLEが用いられる。これはビット演算を保ちつつ、MACは大きな体で保持することで効率と安全

2.2 VOLE-based ZK

2.3 VOLE-in-the-Head

VOLEitHは、MPC-in-the-Headの開示構造にQuickSilver型のVOLE検査を組み込み、指定検証者VOLEitHは、MPCitHの「開示しない1パーティ」を、VOLEのグローバル鍵 \boxtimes に対応付けることで、指定検証者型のVOLE系プロトコルを公開検証可能にする。鍵となるのが、長さ倍増PRGから構成するGGM treeに基づくAll-but-One Vector Commitmentである。

- ・高さ $h = \log N$ のGGM木の根シードから N 個の葉シード sd_i を生成し、各葉にVOLE用の乱数 r_i を割り当てる。
- ・開示しないインデックス j^* を除き、 P は各葉への経路で必要な兄弟ノードのシードを開示す $N - 1$ 個の r_i を再現させられる (通信量は $O(\log N)$)。
- ・開示されない r_{j^*} が \boxtimes に対応し、 V は公開された r_i の線形結合から $\mathbf{q} = \mathbf{u} \circ \boxtimes - \mathbf{v}$ を再構成する。

この構成により、OTを用いずに非対話でVOLE相関を生成し、MACのbinding性を保ったままFiat-Shamir変換を適用できる。

2.4 SNARKとオンチェーン検証の概観

Groth16を例に、オンチェーン検証は(i) 証明と公開入力をcalldataで受領、(ii)回路固有の検証鍵をコントラクトに格納、(iii) 公開入力とICベクトルのMSMでvk_xを計算、(iv)ペアリングprecompileで等式を判定、の4段階からなる。ガスは主にペアリング (1回あたり~34gas、合計15~20万gas規模)と公開入力数に比例するMSMで支配され、calldata課金は16gas/非ゼロバイトで証明サイズ1055バイトなら約1.7万gasとなる。コードサイズは24KB上限の

2.5 設計指針

- ・証明サイズの固定性: Groth16の証明サイズは一定（本稿では1055バイト）で、回路規模が固定される。
- ・公開入力の削減: MSMコストは公開入力数に線形。ハッシュ圧縮等で項目数を抑える。
- ・検証鍵の配置: コントラクトサイズ上限を考慮し、検証鍵を分離・共有する設計をとる。
- ・フィールド整合性: 証明生成・検証ともBN254を用いる。別曲線を選ぶ場合は専用プリコンパイルを用意する。

3 ベンチマーク設計

本章では、VOLEitHとSNARKを組み合わせたアーキテクチャの性能を定量的に評価するために設計したベンチマークを紹介する。

3.1 測定項目

本研究では、証明システムの性能と実用性を多角的に評価するため、以下のメトリクスを測定対象とする。

メトリクス	説明
証明生成時間	証明者が、ある計算に対する証明を生成するために要する時間
証明検証時間	検証者が、与えられた証明の正当性を検証するために要する時間
証明サイズ	生成された証明データの大きさ
通信オーバーヘッド	非対話型証明において、証明者が検証者に送信する必要がある総データ量
計算負荷	証明生成および検証プロセス中に消費されるCPU使用率および最大メモリ使用量
SNARK制約数	VOLEitHの証明をSNARKに変換する際に生成されるR1CSの制約数。
オンチェーン検証ガス代	生成されたSNARK証明をEthereumのスマートコントラクトで検証する際に消費されるガス代

Table 1: ベンチマーク測定項目一覧

3.2 評価環境

すべてのベンチマークは、以下の統一された環境で実施した。

- ・ハードウェア:
 - CPU: Apple M1
 - メモリ: 16GB
- ・ソフトウェア:
 - 言語: Rust
 - ベンチマークツール: `cargo bench`
 - VOLEitH実装: `soft_spoken` ライブライブ
 - スマートコントラクト開発・テスト: Foundryフレームワーク
 - Solidityバージョン: 0.8.20

3.3 評価対象回路

本ベンチマークでは、プロトコルの基本的な性能と、より実践的な応用における性能の両方を評価

- SHA256回路:
 - 内容: SHA-256。これらは暗号技術で広く利用される標準的なハッシュ関数であり、複数の実装が存在する。
 - 形式: これらの回路は、Bristol Fashion形式で記述されたものを、本研究で利用するVC。
 - 目的: VOLEitHプロトコル単体の性能と、既存のZKP実装（Circom）との比較評価に用いられる。
- E2E評価用基本回路:
 - 内容: 100ゲートおよび1000ゲートのADD（加算）回路とAND（乗算）回路。
 - 目的: エンドツーエンド（E2E）の性能評価、特に回路の規模（ゲート数）と種類（加算、乗算）。

4 結果と分析 (Results and Analysis)

本章では、設計したベンチマークに基づき、VOLEitHの性能を多角的に評価する。まず4.1節でVCを構成し、4.2節で、VOLEitHの証明をSNARKで圧縮しオンチェーン検証するまでのエンドツーエンドプロセスを示す。

4.1 VOLEitH単体性能評価

VOLEitHプロトコル自体の性能を評価するため、標準的な暗号学的ハッシュ関数であるSHA-256とKeccak-Fの回路を用いてベンチマークを実施した。これらの回路はBristol Fashion形式で記述されたものを本研究用に変換したものである。

表1に、Apple M1（メモリ16GB）環境で測定した両回路のベンチマーク結果を示す。

Table 2: VOLEitH単体性能ベンチマーク (SHA-256 vs Keccak-F)

Metric	sha256	keccak_f
Proof Generation Time	95 ms	143 ms
Proof Verification Time	51 ms	74 ms
Proof Size	4,927,342 B (~4.9 MB)	8,416,569 B (~8.4 MB)
Communication Overhead	4,927,407 B (~4.9 MB)	8,416,634 B (~8.4 MB)
Prover Computation Load	0.02% CPU, 118.23 MB	0.04% CPU, 154.14 MB
Verifier Computation Load	0.04% CPU, 138.89 MB	0.04% CPU, 158.1 MB

表2から、回路の複雑性が性能に直接的な影響を与えることがわかる。

Keccak-FはSHA-256よりも複雑な回路構造を持つため、証明生成時間、検証時間、そして証明サイズがSHA-256を上回るコストが必要となった。特筆すべきは証明サイズであり、SHA-256で約4.9MB、Keccak-Fでは約8.4MBにも達する。この巨大なデータサイズは、そのままでは実用的でない。

次に、VOLEitHの性能特性をより明確にするため、既存のZKP実装であるCircom ([5]より) とSHA-256実装との比較を行う。表2に両者の性能比較を示す。

Table 3: SHA-256実装の性能比較 (VOLEitH vs Circom)

実装	証明生成時間	証明サイズ
VOLEitH (本研究)	95 ms	~4.9 MB
Circom (先行研究)	~1,473 ms	~821 Bytes

表3は、VOLEitHの基本的なトレードオフを明確に示している。証明生成時間において、VOLEitHはCircomよりも速い。

一方で、証明サイズに目を向けると、VOLEitHの証明は約4.9MBであるのに対し、Circom (Groth 2016) の結果によると約821Bytesである。

この結果から、VOLEitHはクライアントデバイスのような計算資源が限られた環境での高速な証明が可能である。ただし、この「証明は高速だが、証明自体が巨大」という課題が、本研究でSNARKによる証明圧縮アプローチによって解決される。

4.2 エンドツーエンド (E2E) 性能評価

前節でVOLEitH単体では証明サイズが大きすぎるという課題が明らかになったため、本節ではVOLEitHのE2Eベンチマークを実施する。ベンチマークは、100ゲートおよび1000ゲートのADD回路とAND回路を用いて実施した。

まず、VOLEitHフェーズの性能を表3に示す。

Table 4: E2Eベンチマーク - VOLEフェーズの性能

Metric	100 add	100 and	1000 add	1000 and
Proof Gen. Time	279.012 μ s	476.5 μ s	790.062 μ s	1.649 ms
Proof Ver. Time	68.75 μ s	274.566 μ s	585.6 μ s	1.082 ms
Proof Size	21,361 B	42,491 B	21,319 B	233,175 B
Comm. Overhead	21,426 B	42,556 B	21,384 B	233,240 B

表4から、VOLEフェーズにおいては、回路のゲート数が増加するにつれて、証明生成時間、検証時間ともに増加する。特に、ANDゲート回路はADDゲート回路と比較して、同程度のゲート数であっても証明生成時間が約2倍かかる。

これは、soft_spokenの実装において、ANDゲートのような乗算処理がADDゲートのような加算処理よりも時間がかかるためである。

次に、SNARKフェーズの性能を表4に示す。このフェーズでは、VOLEitHの証明をSNARK (Groth 2016) によって圧縮される。

Table 5: E2Eベンチマーク - SNARKフェーズの性能

Metric	100 add	100 and	1000 add	1000 and
Proof Gen. Time	272 ms	1,794 ms	324 ms	8,003 ms
Constraints	86,080	3,471,680	86,080	33,942,080
Proof Size	1,055 B	1,055 B	1,055 B	1,055 B
Gas Cost	208,967	208,967	208,967	208,967

表5から、SNARKフェーズではVOLEフェーズとは異なる特性が明らかになる。SNARK証明の生成時間は、回路のゲート数や種類に関わらず約2msである。また、ONCHAIN検証のガス代も208,967 gasで一定であり、これはSNARKの検証が固定コストであることを示す。

これにより、前節で課題となっていたVOLEitHの巨大な証明サイズが大幅に圧縮され、ONCHAIN検証のガス代が約2msに圧縮される。

一方で、SNARK証明の生成時間と制約数には、回路の複雑性が大きく影響している。特に、ANDゲート回路では、制約数が33,942,080に達し、証明生成に8,003 msかかる。

ms（約8秒）を要している。

この関係性をより視覚的に示すため、図1にSNARKの制約数と証明生成時間の関係を示す。

Figure 1: SNARKの制約数と証明生成時間の関係

図1は、SNARKの証明生成時間が、回路の制約数、特に乗算ゲートに起因する制約数の増加によるものである。これは、SNARKの証明生成における主要な計算ボトルネックが、回路の複雑性、特に乗算の多さによるものである。

主な観測事項 本章で得られたE2E測定結果から、以下の特徴が明らかになった。

- ANDゲートはADDゲートよりも大幅に制約数と証明時間を増加させ、VOLEフェーズでも証明時間に影響する。
- ADDのみの回路では制約数がほぼ一定であるのに対し、ANDゲート数に比例してSNARK制約数が増加する。
- SNARKフェーズの証明生成時間が、VOLEフェーズの生成・検証時間を大きく上回り、全体の証明時間の大部分を占める。
- SNARK証明サイズおよびオンチェーン検証ガスは1,055バイトと約209k gasで一定であり、回路規模に依存しない。
- 総証明時間はSNARKフェーズの制約增加に強く影響されるため、複雑な回路ではクライアント側の計算資源を消費する。

4.3 SNARK統合に関する洞察

VOLEitHの証明をGroth16で包むと、証明サイズと検証コストは一定になる一方で、R1CS制約数は回路の構造によって変化する。ここでは制約数の内訳と、制約爆発の要因および緩和策を整理する。

4.3.1 制約数の分解

n を拡張witnessの長さ（秘密入力数と乗算ゲート数の合計）とすると、全体の制約数は

$$16,640 \times n + 2,113,664$$

と表せる。線形項に寄与するガジェットは表6の通りであり、`compute_validation_aggregate`が支障となる。

Table 6: 線形に増加するガジェットの制約数

ガジェット	制約数
<code>compute_d_delta</code>	$128n$
<code>compute_masked_witness</code>	$256n$
<code>compute_validation_aggregate</code>	$16,512n$
合計	$\approx 16,640n$

また、回路サイズに依存しない定数項も無視できない（表7）。乗算ゲートが増えると線形項が増加する。

Table 7: 定数項として加算されるガジェット

ガジェット	制約数
<i>combine</i>	~ 2,097,152
<i>compute_actual_validation</i>	~ 16,384
最終整合性チェック	~ 128
合計	~ 2,113,664

4.3.2 Field Mappingがもたらす制約爆発

SchmivitzにおけるVOLEitHは、 \mathbb{F}_2 、 \mathbb{F}_{2^8} 、 $\mathbb{F}_{2^{64}}$ 、 $\mathbb{F}_{2^{128}}$ といった2進拡大体上で計算を行う。一方で、Groth16のR1CSはBN254の素数体上で定義されるため、各ビット列をBoolean変数列に実装では以下のように、証明内の各値を逐一Boolean配列に射影している。

```
pub fn build_circuit(
    cs: ConstraintSystemRef<Bn254Fr>,
    proof: Proof<InsecureVole>,
) -> VoleVerificationBoolean {
    let witness_commitment_booleans: Vec<Vec<Boolean<Bn254Fr>>> = proof
        .witness_commitment
        .iter()
        .map(|value| f64b_to_boolean_array(cs.clone(), value).unwrap())
        .collect();

    let witness_challenges_booleans: Vec<Vec<Boolean<Bn254Fr>>> = proof
        .witness_challenges
        .iter()
        .map(|value| f128b_to_boolean_array(cs.clone(), value).unwrap())
        .collect();
    // ...
}
```

この変換により、もともと単一の体要素で表現できた計算が数百ビットのAND/XORに展開さ

4.3.3 ANDゲートとwitness_challenge

特にANDゲートを検証する際には、*witness_challenge*と*masked_witness*の全ビットについてANDゲートを計算する。実装の核心は以下の通りであり、128ビット平方の積をBooleanレベルで計算するため、ANDゲートを複数回呼び出す。

```
for (i, challenge_bit) in challenge.iter().enumerate() {
    if i >= 128 { break; }
    for (j, masked_bit) in masked_witness.iter().enumerate() {
        if j >= 128 || i + j >= 128 { continue; }
        let and_result = Boolean::and(challenge_bit, masked_bit)?;
        product[i + j] = Boolean::xor(&product[i + j], &and_result)?;
```

```

    }
}

```

ADDゲートでは`witness_challenge`が不要なため制約数は一定だが、ANDゲートが増えるほど

4.4 技術的ボトルネックと解決策

上記の分析から、Field MappingとGGM木再構成が制約爆発の主要因であることが分かる。本節では、これらを緩和するための具体的な研究方向を整理する。

4.4.1 Field Mapping最適化とLookup Table

Mystique[6]は、機械学習向けに \mathbb{F}_2 と \mathbb{F}_p のデータ変換を効率化するVOLEベースZKであり、Lookup Table (LUT) を導入することでさらに高速化できることが最新研究[7]で示されている。

表8に示す通り、LUTを用いた場合には実行時間が61–130倍短縮し、通信量も最大2.9倍削減でき、VOLEitHのField MappingにMystique型LUTを適用できれば、SNARKフェーズの制約数削減に直

Table 8: MystiqueとLUT拡張の性能比較

関数	プロトコル	実行時間 (s)	通信量 (MB)
指數関数	Mystique with LUT	8.696	99.020
	Mystique	1130.020	291.435
除算	Mystique with LUT	9.837	110.684
	Mystique	617.690	160.428
逆平方根	Mystique with LUT	11.836	147.903
	Mystique	824.639	212.211

4.4.2 GGM木最適化とFolding

Schmivitzでは、VOLEitH検証で最もコストの高いGGM木再構成を簡略化しているが、SNARKでは著者らはGGM木を効率化する手法[8]に加え、FAESTを改良したFAESTERを提案しており、署名本研究で検討したFoldingスキームとこれらの最適化を組み合わせれば、将来的にGGM木再構成部

Table 9: FAESTとFAESTERの比較（セキュリティ128ビット）

スキーム	バージョン	署名サイズ (B)	署名時間 (ms)	検証時間 (ms)
FAEST	Slow	50,063	4.3813	4.1023
	Fast	63,363	0.4043	0.3953
FAESTER	Slow	45,943	3.2823	4.4673
	Fast	60,523	0.4333	0.6103

4.5 総合考察とトレードオフ分析

これまでの分析結果を統合し、VOLEitHとSNARKを組み合わせたアーキテクチャ全体の有効性と、本研究で採用したアーキテクチャは、図2に示すように、証明者側（Prover）で2段階のプロセ

Figure 2: VOLEitH + SNARKによる証明圧縮プロセスの概念図

図2が示す通り、本アーキテクチャは、VOLEitHが生成する巨大な証明（数MBオーダー）を、このアプローチにより、以下の2つの大きな利点を両立することが可能となる。

1. 高速な証明者計算: VOLEitHは、Circomのような従来のR1CSベースのシステムと比較して、これにより、計算能力が限られるクライアントデバイス（例: Webブラウザ、スマートフォン）でも実行可能となる。
2. 低コストなオンチェーン検証: SNARK化された証明は、サイズが小さく、検証コストが回路計算時間に比例するため、高速かつ効率的な検証が可能となる。

一方で、このアーキテクチャには考慮すべきトレードオフも存在する。最大のトレードオフは、証明者は、高速なVOLEitH証明生成に加えて、SNARK証明を生成するための追加の計算コストを負担する。特に、回路が多くの乗算（ANDゲート）を含む場合、SNARKの制約数が急増し、SNARK証明の生成時間が長くなる。したがって、本アーキテクチャは、以下のような特性を持つユースケースにおいて特に有効である。

- ・ クライアントサイドでの証明生成: ユーザー自身のデバイスで証明を生成し、サーバーやブロードキャストする。例えば、プライバシーを保護した上で本人確認（分散型ID）、プライベートな状態を持つデータの検証。
- ・ オンチェーンコストの最小化が重要: ブロックチェーンのスケーラビリティが重視され、トランザクションの大きさを最小化する。

結論として、VOLEitHとSNARKを組み合わせたハイブリッドアプローチは、「証明者の高速性」と「証明圧縮」の両立を実現する。その性能は回路の特性、特に乗算の数に大きく依存するため、アプリケーションを設計する際には注意が必要である。

5 結論と今後の展望 (Conclusion and Future Work)

5.1 結論

本研究では、証明者効率の高いVOLE-in-the-Head (VOLEitH) プロトコルと、証明圧縮に優れたSNARKを組み合わせたハイブリッドアプローチを提案した。ベンチマークを通じて、以下の主要な知見が得られた。

1. VOLEitHの基本的なトレードオフ: VOLEitHは、CircomのようなR1CSベースのシステムと比較して、この巨大な証明は、単体ではオンチェーン検証の大きな障壁となる。
2. SNARK圧縮の有効性: VOLEitHの証明をSNARK (Groth16) で圧縮することにより、回路の構造を理解する必要がなく、VOLEitHのオンチェーン応用の道が拓かれる。
3. アーキテクチャのボトルネック: エンドツーエンドのプロセスにおける主要なボトルネックは、データの入出力と、データの暗号化/復号化である。

結論として、VOLEitHとSNARKを組み合わせたハイブリッドアプローチは、「クライアントサイドでの証明生成」と「オフチェーンの証明圧縮」の両立を実現する。本研究は、その具体的な性能データとトレードオフを明らかにすることで、このアプローチの実用性を示すものである。

5.2 今後の展望

本研究の成果を踏まえ、特に優先度の高い研究課題は以下の3点である。

1. Field Mapping最適化: Mystique型のデータ変換やLookup TableをVOLEitHに取り入れ、 \mathbb{F}_2
2. GGM木再構成の高度化: FAESTERのような最適化とFoldingスキームを組み合わせ、検証フ
3. 代替SNARK/証明システムの検討: BinusやRecursive SNARKなど、二進演算に適した新しい

これらに加えて、以下のエンジニアリング課題にも継続的に取り組む必要がある。

- SNARK証明生成の最適化: VOLEitHからR1CSへの変換フローを高速化し、Plonk/Halo2とい SNARKやSolidity verifierのガス最適化を検討する。
- アプリケーション駆動の評価: 分散型ID、プライベートトランザクション、オンチェーンゲー
- セキュリティ整合性の確保: VOLEitHはLPN仮定に基づくポスト量子耐性を持つ一方で、Gro

参考文献 (References)

References

- [1] Baum, C., et al. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. Cryptology ePrint Archive, Paper 2023/996, 2023. <https://eprint.iacr.org/2023/996>.
- [2] Groth, J. On the Size of Pairing-based Noninteractive Arguments. In EUROCRYPT 2016, pp. 305-326, 2016. DOI: 10.1007/978-3-662-49896-5_11.
- [3] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. IACR Cryptology ePrint Archive, 2019.
- [4] Grassi, L., et al. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In USENIX Security Symposium, 2021. <https://www.usenix.org/conference/usenixsecurity21/presentation/grassi>.
- [5] Iden3. Circom: A Circuit Compiler for Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2023/681, 2023. <https://eprint.iacr.org/2023/681>.
- [6] Haitner, Y., et al. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. Cryptology ePrint Archive, Paper 2021/730, 2021. <https://eprint.iacr.org/2021/730>.
- [7] Fu, H., et al. Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning. Cryptology ePrint Archive, Paper 2025/507, 2025. <https://eprint.iacr.org/2025/507>.
- [8] Beullens, W., et al. One Tree to Rule Them All: Optimizing GGM Trees and OWFs for Post-Quantum Signatures. Cryptology ePrint Archive, Paper 2024/490, 2024. <https://eprint.iacr.org/2024/490>.