

EthereumにおけるVOLE-in-the-Headの検証コスト評価

津田匠貴
Nyx Foundation

December 2, 2025

Abstract

Vector Oblivious Linear Evaluation (VOLE) は、二者間で線形関係を秘匿したまま、ある種の相関を表現する。VOLE-in-the-Head (VOLE-itH) は、従来のゼロ知識証明と比較して線形演算をVOLEプリプロセスに委ねる。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xxx 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・

1 序論

1.1 Ethereumにおけるプライバシーとゼロ知識証明の役割

Ethereumは、スマートコントラクト¹と呼ばれるプログラムを実行できる分散型プラットフォームである。この透明性はシステムの信頼性を担保する一方で、すべての取引記録が公開台帳（ブロックチェーン）により、どのアドレスからどのアドレスへ、いつ、どれだけの資産が移動したかを誰でも追跡可能である。

このプライバシーの欠如という課題に対する強力な解決策として、ゼロ知識証明 (Zero-Knowledge Proof, ZKP) が注目されている。ZKPは、「ある秘密の情報を知っている」ということを証明することによって、ユーザーは自身の取引の詳細を秘匿したまま、その取引が正当であることを検証できる。

Ethereum上でZKPを活用した代表的なアプリケーションに、プライバシーミキサーであるTornado Cashがある。Tornado Cashでは、ユーザーは資産を共通のプールに預け入れ、後で全く新しいアドレスから資産を受け取る。このとき、ユーザーは「自分がある金額をプールに預け入れた正当な預金者である」ことをZKPを用いて証明する。この証明には、預け入れ時のどのアドレスとも結びつく情報が含まれないため、預け入れと引き出しのリンクが断たれる。

このようなアプリケーションを実現するには、Ethereum上でZKPを検証する仕組み、すなわちオンチェーン検証のプロセスは、一般的に以下の手順で行われる。(1) ユーザーが自身のデバイスでZKPを生成する。(2) この証明を、Ethereum上にデプロイされた検証者スマートコントラクトにトランザクションとして送信する。(3) スマートコントラクトは、受け取った証明が有効であるかを検証し、有効であれば後続の処理を行う。しかし、このオンチェーン検証には二つの大きな課題が存在する。

第一の課題は、オンチェーン検証のコストと制約である。スマートコントラクト上での計算コストが高いため、検証アルゴリズムが複雑であったり、証明のデータサイズが大きかったりすると、ガス代は著しく高くなる。また、Ethereumのガス代は変動するため、検証コストも変動する。Ethereum上でZKPを検証するには、事前に検証者スマートコントラクトに検証ロジックをデプロイしておく必要がある。また、Ethereumのガス代は変動するため、検証コストも変動する。Ethereum上でZKPを検証するには、事前に検証者スマートコントラクトに検証ロジックをデプロイしておく必要がある。

第二の課題は、クライアントサイドでの証明生成の負荷である。アプリケーションが広く普及するためには、クライアントサイドでの証明生成の負荷を軽減する必要がある。また、多くの既存のZKP技術（特にzk-SNARKs）は、検証の軽量性と引き換えに、証明生成に高い計算コストがかかる。また、多くの既存のZKP技術（特にzk-SNARKs）は、検証の軽量性と引き換えに、証明生成に高い計算コストがかかる。

したがって、理想的な証明システムは、これら二つの課題を同時に解決する必要がある。すなわち、オフチェーンでの証明生成と、オンチェーンでの検証の両方をサポートするシステムである。また、多くの既存のZKP技術（特にzk-SNARKs）は、検証の軽量性と引き換えに、証明生成に高い計算コストがかかる。

1.2 VOLEベースZKPとVOLE-in-the-Head

VOLE (Vector Oblivious Linear Evaluation) は、セキュア多者計算 (MPC: Secure Multi-Party Computation) の分野、特にYao's Garbled Circuit (GC) と Oblivious Transfer (OT) の研究から派生した暗号学的プリミティブである。これは、二者間で線形関係を秘匿したまま特定の相関を生成することを可能にする。その後、この相関を用いて、従来のSNARKs (Succinct Non-Interactive Argument of Knowledge) で主流であった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した。

その中でもVOLE-in-the-Head (VOLE itH) は、VOLEベースの対話型プロトコルに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検証者側の低い計算コストを実現した。

¹ 契約の条件確認や履行を自動的に強制するコンピュータプログラム。ブロックチェーン上で実行される。

² Ethereumネットワーク上でトランザクションを実行するために必要な手数料。計算が複雑であるほど高額になる。

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能は不明である。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ことを検証し、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ことを目的とする。

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。
todo: 加法と乗法の計算過程を詳細に説明する。また、加法についてコストがからないことを明記する。

2.1 Vector Oblivious Linear Evaluation (VOLE)

Vector Oblivious Linear Evaluation (VOLE) は、1-out-of-2 Oblivious Transfer (OT) を一般化した暗号学的プリミティブである。OTとは、送信者が持つ2つのメッセージ $\{m_0, m_1\}$ だけを受信者が受け取ることができるプロトコルである。このとき、(1) 送信者は受信者がどちらを選んだか (b の値) を知ることができず、(2) 受信者は選ばなかった方のメッセージを受け取ることができない。このプロトコルは、例えばRSAのような公開鍵暗号に基づいて、以下のように単純な形で構成される。

1. 送信者は、2つのメッセージ m_0, m_1 を持つ。また、それぞれ独立した2組のRSA鍵ペア（公開鍵 N_0, N_1 と秘密鍵 d_0, d_1 ）を持つ。
2. 受信者は、自分の選択ビット $c \in \{0, 1\}$ に基づき、ランダムな共通鍵 k を選び、自分が欲しい方のメッセージ m_c を $k^{e_c} \pmod{N_c}$ で暗号化し、送信者に送る。
3. 送信者は、受信者から受け取った v を、両方の秘密鍵で復号しようと試みる。

- ・ $k_0 = v^{d_0} \pmod{N_0}$
- ・ $k_1 = v^{d_1} \pmod{N_1}$

v は公開鍵 (N_c, e_c) で暗号化されているため、 k_c は正しく共通鍵 k に復号されるが、 k_{1-c} は無関係な値となる。送信者は $m_0 \oplus H(k_0)$ と $E_1 = m_1 \oplus H(k_1)$ を受信者に送る（ H は暗号学的ハッシュ関数）。

4. 受信者は、自身が知っている共通鍵 k を使って、欲しい方のメッセージ $m_c = E_c \oplus H(k)$ を復号する。

この構成において、それぞれのプライバシーは以下のように保証される。

- ・ 受信者のプライバシー: 送信者は、受信者から送られてくる暗号文 v がどちらの公開鍵で暗号
- ・ 送信者のプライバシー: 受信者は、自分が選ばなかった方のメッセージ m_{1-c} に対応する秘密鍵

VOLEは、このOTの概念を一般化し、二者間で線形関係を秘匿したまま、ある種の相関を生成する。証明者（Prover）はベクトル $\mathbf{u} \in \mathbb{F}_2^\ell$ と $\mathbf{v} \in \mathbb{F}_{2^k}^\ell$ を、検証者（Verifier）はグローバルキー $\Delta \in \mathbb{F}_{2^k}$ を入力とする。プロトコル終了後、検証者は $\mathbf{q} \in \mathbb{F}_{2^k}^\ell$ を得て、証明者は (\mathbf{u}, \mathbf{v}) を、検証者は (Δ, \mathbf{q}) を

$$q_i = u_i \cdot \Delta + v_i \quad (i = 0, \dots, \ell - 1) \quad (1)$$

この相関は、 u_i に対する線形準同型コミットメントとして機能し、算術回路の検証に利用できる。

加法準同型性 2つの入力ワイヤ a, b に対するVOLE相関 $(u_a, v_a), (u_b, v_b)$ と (q_a, q_b) が与えられたとき、 $a+b$ に対するコミットメントは、証明者と検証者がそれぞれローカルで値を加算するだけで得られる。 (u_a+u_b, v_a+v_b) を、検証者は $q_c = q_a+q_b$ を計算する。この演算は追加の通信を必要としない。

乗法 一方で、乗算ゲート $c = a \cdot b$ の検証は、非線形な演算であるため、ローカルな計算だけでは実現できない。 $u_a \cdot u_b$ に対応する v_c を生成するために、検証者との間で追加の通信を伴うプロトコルを実行する。

VOLEプロトコルは以下の性質を持つ。

秘匿性 (Hiding) 検証者は q_i から u_i の情報を得られない。これは、証明者が知るランダムな値 v_i が u_i と相関しているためである。

拘束性 (Binding) 証明者は、コミットした u_i とは異なる値 u'_i を開示することができない。これを行うことは、検証者が q_i と u'_i の相関を検証できるためである。

2.2 VOLE-based ZK

todo: VOLE-based ZKの説明を補強する。VOLEの持つ線形性（加法準同型性）を利用することで、ZK-SNARK（VOLE-based ZK）を構築できる。このパラダイムに基づく代表的なプロトコルとしてQuickSilver[9]がある。VOLEベースの証明は、事前にVOLE相関をバッチ生成（pre-processing）しておき、オンラインで検証を行う。

しかし、従来のVOLE-ZKプロトコルは、その健全性（Soundness）を保証するために、検証者が特定の検証者型（Designated Verifier Proof）となり、第三者が検証できる公開検証可能性（Public Verifiability）を持たないという課題があった。

2.3 VOLE-in-the-Head

todo: VOLEitHの説明を補強する。GGM treeについて説明する。VOLE-in-the-Head（VOLEitH）は、上記のような指定検証者型のVOLE-ZKプロトコルを、非対話かつ公開検証可能な形で実現する。VOLEitHは、MPC-in-the-HeadパラダイムとVOLEを組み合わせたもので、証明の健全性を単一の検証者で保証する。

VOLEitHの核となるアイデアは、検証者が秘密情報（ Δ ）を持つ代わりに、証明者自身がコミットメント（ \mathbf{u} ）を生成し、Fiat-Shamir変換[10]によって Δ を導出する点にある。

1. コミットメント: 証明者は、VOLE相関の元となるシード（seed）のベクトルにコミットする。All-but-One ベクトルコミットメントが用いられ、単一のハッシュ値で多数のシードをコミットする。
2. チャレンジ生成: 証明の主要部分が生成された後、Fiat-Shamir変換を用いて、証明全体のトランシュクリプト（ \mathbf{t} ）を生成する。
3. 開示と検証: Δ が定まった後、証明者はコミットメントを開示する。All-but-Oneコミットメントの性質により、 N 個のシードのうち $N-1$ 個を $O(\log N)$ の通信量で開示する。検証者は、開示された情報と自身で計算した Δ を用いてVOLE相関式を検証する。これにより、検証者は \mathbf{q} と \mathbf{u} の相関を検証できる。

2.4 クライアントサイドプルービング

ゼロ知識証明を応用したアプリケーションが広く普及するためには、エンドユーザーが自身のデバイスが特別な高性能ハードウェアを必要とせず、プライバシーを保護しながら自身の正当性(Experience)の鍵となる。

しかし、多くの証明システム、特にペアリングなどの公開鍵暗号に基づくzk-SNARKsは、証明生成に膨大な計算リソース(CPU、メモリ)を要求する傾向がある。複雑なアプリケーションの証明を生成しようとする、数分から数十分の時間がかかったり、モバイルデバイスではこの証明生成の重さが、クライアントサイドでのゼロ知識証明技術の社会実装を妨げる大きな障壁となる。

この課題に対し、VOLE-in-the-Headのような対称鍵暗号に基づく証明システムが解決策として登場した。これらのシステムは、AESなどハードウェアで高速に実行可能な対称鍵プリミティブを計算の基盤とし、この特性により、VOLEitHは計算資源の限られたクライアントデバイス上でも快適なUXを提供できる。

2.5 オンチェーン検証

オンチェーン検証とは、ブロックチェーン上で暗号学的証明の正当性を検証するプロセスを指す。ブロックチェーンは透明性が高い反面、すべてのアクションが公開台帳に記録されるため、ユーザーは個人情報を一切明かすことなく、「ある条件を満たせば、この権利を得る」という権利を行使できる。例えば、ユーザーは個人情報(年齢、国籍、資産状況など)を一切明かすことなく、「ある条件を満たせば、この権利を得る」という権利を行使できる。これにより、プライベートな投票、匿名での資格証明、秘匿性を保った金融取引などが可能となる。

本稿では特に、Ethereum[11]仮想マシン(EVM)上でzk-SNARKの証明を検証するケースを扱う。Ethereumは、スマートコントラクトと呼ばれるプログラムを実行できる分散型プラットフォームである。さらに、ブロックには含められるデータ総量(ブロックガスリミット)に上限があるため、証明自体のサイズが小さく、オンチェーンでの利用には、証明サイズが極めて小さく(簡潔、Succinct)、検証コストが低いことが求められる。

この文脈でGroth16が広く採用されるのは、その証明サイズが小さい(回路の規模に関わらず、検証に必要なデータは一定)ため、ブロックチェーンに送信するデータ(calldata)や計算はガス代に直結するため、これらの特性は検証コストを低減させる。

Ethereum上でGroth16の証明を検証するには、Solidityで記述された検証コントラクトが用いられる。この検証ロジックは、Ethereumに予め組み込まれたプリコンパイル済みコントラクトを利用する。具体的には、アドレス'0x08'に配置された'ecPairing'というプリコンパイル済みコントラクトが用いられる。

検証のプロセスは以下の通りである。まず、開発者は証明対象の回路に対応する検証鍵(Verification Key, vk)を含むスマートコントラクト(例: Verifier.sol)をEthereumにデプロイする。

次に、証明を検証したいユーザーは、証明(proof, π)と公開入力(public inputs)をトランザクションとして検証コントラクトに送信する。検証コントラクトは、受け取った証明と公開入力を検証し、その結果に基づき、コントラクトは後続の処理(状態の更新や別のコントラクト呼び出しなど)を実行する。

このようなオンチェーンZKアプリケーション開発において、Circomは算術回路を記述するためのツールであり、これにより、開発者は複雑なペアリング演算のロジックを自ら実装することなく、効率的にオンチェーンアプリケーションを開発できる。

2.6 Groth16

まず、Groth16を理解するために、zk-SNARKs(zero-knowledge Succinct Non-interactive ARGuments of Knowledge)について説明する。zk-SNARKsは、以下の性質を持つ。

- ・ゼロ知識(Zero-Knowledge): 証明が「あるステートメントが真である」という事実以外の情報を漏らすことなく、検証者に対して証明の正当性を示すことができる。
- ・簡潔性(Succinctness): 証明のサイズが非常に小さく、検証時間が短い。これにより、検証コストが低減される。

- ・ 非対話性 (Non-interactive): 事前の一度きりのセットアップの後、証明者と検証者の間でインタラクティブな知識の論拠 (ARgument of Knowledge): 証明者が実際に秘密の入力 (witness) を知っている

Groth16は、このようなzk-SNARKsの中でも特に効率的で広く利用されている方式の一つであり、ブロックチェーンのように検証コストが重視される環境において、特に強力な選択肢となる。

Groth16プロトコルは主に3つのアルゴリズムから構成される：

Setup 回路（通常はR1CS形式）から、証明鍵（Proving Key, pk）と検証鍵（Verification Key, vk）を生成する。このプロセスは回路ごとに一度だけ実行する必要がある、信頼できるSetup）を必要とする。このセットアップで用いられた乱数（"toxic waste"）が漏洩すると、不正な証明が生成可能になるという課題がある。

Prove 証明者は、証明鍵、公開入力（witness）、および秘密入力を用いて、証明（proof, π ）を生成する。

Verify 検証者は、検証鍵、公開入力、および証明 π を受け取り、その正当性を検証する。

Groth16の証明は、特定の楕円曲線（例: BN254）上の3つの群要素（ \mathbb{G}_1 の元2つ、 \mathbb{G}_2 の元1つ）の演算による。検証は、数回のペアリング演算によって行われ、極めて高速に完了する。

この効率性の代償として、回路ごとに異なるTrusted Setupが必要であり、汎用性（universality）が低い。

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について述べる。

3.1 提案アーキテクチャ：VOLEitH証明のSNARKによる圧縮

本研究で評価するアーキテクチャは、VOLE-in-the-Head（VOLEitH）が生成する巨大な証明サイズを圧縮するために、このため、証明者効率に優れたVOLEitHと、証明の簡潔性および検証効率に優れたzk-SNARK（本研究ではGroth16[2]を採用）を組み合わせたハイブリッドアプローチを採用する。このアーキテクチャの究極的な目的は、VOLEitHの利点である「クライアントサイドでの高速な証明生成」を維持しつつ、証明サイズを圧縮することである。証明者が最終的なオンチェーン検証用の証明を生成するまでのプロセスは、以下の4つのステップで構成される。

1. **ステップ1：VOLEitH証明の生成** まず、証明者は与えられた計算（算術回路）に対し、VOLEitH証明 π_{VOLEitH} を生成する。このプロセスは、対称鍵暗号ベースの軽い計算で構成されるため非常に高速だが、生成された証明サイズは非常に大きい。
2. **ステップ2：VOLEitH検証回路のR1CS化** 次に、ステップ1で生成された π_{VOLEitH} の正当性を検証するための検証ロジックを、SNARKが扱うことができる形式であるR1CS（Rank-1 Constraint System）の算術回路として表現する。この「検証回路」は、 π_{VOLEitH} を入力とする。
3. **ステップ3：SNARK証明の生成** 証明者は、ステップ2で構築した「VOLEitH検証回路」に対して、SNARK証明 π_{SNARK} を生成する。この π_{SNARK} は、「ある有効な π_{VOLEitH} を知っており、それに対する検証計算が正しく実行されたことを証明する」ためのものである。Groth16の特性により、 π_{SNARK} は回路の複雑さに関わらず一定の小さなサイズ（約1KB）で表現できる。
4. **ステップ4：オンチェーン検証** 最終的に、コンパクトなSNARK証明 π_{SNARK} と、計算の公開入力を用いて、スマートコントラクトは、事前にデプロイされた検証鍵を用いて π_{SNARK} を検証する。この検証は、非常に高速に行われる。

この一連のプロセスにより、クライアント側では証明生成の大部分を軽量なVOLEitHプロトコル
本研究では、このアーキテクチャをRust言語で実装した。VOLEitHの実装にはschmivitzライブラリ

3.2 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

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
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
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

Table 1: ベンチマーク測定項目一覧

3.3 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
 - VOLEitH実装: schmivitz ライブラリ

3.4 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
 - 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VO
 - 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
- ・ 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

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VOL次

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、VOLEこれは、証明者の計算効率を重視するVOLEベースのプロトコルの特性を強く反映している。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（Gro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

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VO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、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、検証時間が大幅に増加する。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高コストであるためである。

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 (Zk-SNARK) にコンパイルする。

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。最も注目すべきは、最終的なSNARK証明のサイズが、回路のゲート数や種類に関わらず1,055バイトであること、また、オンチェーン検証のガス代も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ためである。これにより、前節で課題となったVOLEitHの巨大な証明サイズが大幅に圧縮され、オンチェーン検証のガス代も大幅に削減される。

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 (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比例して増加することを示している。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、特に乗算の多さであるためである。

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フェーズでも証明生成時間が大幅に増加する。
- ・ 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、全体の証明生成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フェーズの制約増加に強く影響されるため、複雑な回路ではクライアント

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

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

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
<code>combine</code>	$\sim 2,097,152$
<code>compute_actual_validation</code>	$\sim 16,384$
最終整合性チェック	~ 128
合計	$\sim 2,113,664$

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に展開される。

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を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

```

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)
指数関数	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）で証明生成が可能となる。
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

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特に、回路が多く乗算（ANDゲート）を含む場合、SNARKの制約数が急増し、SNARK証明の生成コストが高くなる点である。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、サーバーやブロックチェーンにアップロードする。例えば、プライバシーを保護した上での本人確認（分散型ID）、プライベートな状態を持つ

- ・ 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、トランザクションコストの削減が求められる。

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、「証明者の高速化」と「検証者のコスト削減」の両方を達成する。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慎重な評価が必要である。

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

todo: faest, faester, binus, Universal SNARK について説明する。

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異なり、検証者のコストを削減する。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、回路のサイズを大幅に削減できる。これにより、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、証明の生成と検証の速度である。

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の高速化」と「検証者のコスト削減」の両方を達成する。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評価する。

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 上の演算を高速化する。
 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、検証プロセスを高速化する。
 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SNARKの導入を検討する。
- 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と競合する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、オンチェーンゲーミングなどのユースケースに適用する。

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>.
- [9] Yang, K., Kohl, P., and Song, D. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '21), pp. 233-250, 2021. DOI: 10.1145/3460120.3484550.

- [10] Fiat, A., and Shamir, A. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Advances in Cryptology – CRYPTO ’ 86, pp. 186-194, 1987. DOI: 10.1007/3-540-47721-7_12.
- [11] Wood, G. Ethereum: A Secure Decentralised Generalized Transaction Ledger. Ethereum Project Yellow Paper, 2024. <https://ethereum.github.io/yellowpaper/paper.pdf>.
- [12] adust09. Swanky: A suite of tools for developing zero-knowledge circuits and applications. GitHub Repository. <https://github.com/adust09/swanky>. Accessed: 2025-12-01.