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
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
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
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
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
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
これにより、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

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
この性質を利用することで、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
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**Tornad**
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
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
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

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
しかし、このオンチェーン検証には二つの大きな技術的・経済的課題が存在する。

第一の課題は、**オンチェーン検証のコストと制約**である。 Ethereum仮想マシン（EVM）上で
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
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
これらの制約から、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）**、
この文脈でGroth16のようなzk-SNARKが広く採用されるのは、その証明サイズが回路の規模に関
Ethereumでは、この検証を効率化するために、BN254曲線におけるペアリング演算を高速に実行

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
しかし、多くの既存のZKP技術、特にペアリングなどの公開鍵暗号に基づくzk-
SNARKsは、検証の軽量性と引き換えに、証明生成に膨大な計算リソースと時間を要求する。
複雑なアプリケーションの証明を生成しようとすると、数分から数十分の時間がかかったり、モバ
この課題に対し、VOLE-in-the-Head（VOLEitH）のような対称鍵暗号に基づく証明システムは、

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
本研究は、このトレードオフ関係にある要求を、VOLEitHとzk-SNARKを組み合わせることでどの

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
これにより、従来のSNARKs（Succinct Non-Interactive Argument of Knowledge）で主流であっ
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

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 ver-
ifiable 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]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
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
　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

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)

Vector Oblivious Linear Evaluation（VOLE）は、Oblivious Transfer
(OT) の算術形式と見なされる暗号学的プリミティブであり、証明者と検証者の間で、秘密裡に線形
　kをセキュリティパラメータ、$\mathbb{F}_{2^k}$を要素数2^kの有限体とする。プロトコル完了後、証明者は秘密
$\mathsf{u} \in \mathbb{F}_2^\ell$ とランダムなVOLEタグ $\mathsf{v} \in \mathbb{F}_{2^k}^\ell$ を知り、検証者はグローバルキー
$\Delta \in \mathbb{F}_{2^k}$ とVOLEキー $\mathsf{q} \in \mathbb{F}_{2^k}^\ell$ を知る。これらの値は、以下のVOLE相関式を満たす。

$$q_i = u_i \cdot \Delta + v_i \quad (i = 0, \ldots, \ell - 1) \tag{1}$$

この相関は、証明者が持つランダムビット u_i に対する**線形準同型コミットメントスキーム**として機

秘匿性 (Hiding) 検証者は、q_iとΔ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

拘束性 (Binding) 証明者は、検証者の持つ秘密Δを知らない限り、一度コミットしたu_iを不正な値

VOLEの最も強力な特性は、この線形準同型性にある。同じグローバルキーΔを持つ複数のVOLE相
(u_i, v_i, q_i) と公開定数 $c, c_1, \ldots, c_n \in \mathbb{F}_{2^k}$ が与えられたとき、証明者と検証者は新しい値
$u' = c + \sum_{i=1}^{n} c_i u_i$ に対するVOLE相関 (u', v', q') を、追加の通信なしにローカルで計算できる。

証明者の計算 証明者は新しいVOLEタグv'を以下のように計算する。

$$v' = \sum_{i=1}^{n} c_i v_i \tag{2}$$

検証者の計算 検証者は新しいVOLEキーq'を以下のように計算する。

$$q' = c \cdot \Delta + \sum_{i=1}^{n} c_i q_i \tag{3}$$

VOLEの持つこの強力な線形準同型性は、算術回路の検証コストに直接的な影響を与える。
VOLE相関 ($q_i = u_i \cdot \Delta + v_i$) は、その構造自体が線形な関係式である。そのため、加算やスカラー乗
一方で、乗算のような**非線形演算**は、この線形構造を破壊する。2つのVOLE相関を乗算しても

2.2 VOLE-based ZKと乗算検定

VOLEの持つ線形性（加法準同型性）を利用することで、効率的なゼロ知識証明プロトコル（VOL
based ZK）を構築できる。このパラダイムは、QuickSilver[9]などのプロトコルで採用されている
乗算ゲート、例えば $w_\gamma = w_\alpha \cdot w_\beta$ の検証は、特別な**乗算検定プロトコル**を必要とする。このプロ

証明者の役割 証明者は、乗算ゲートに関わる自身の秘密値（$u_\alpha, v_\alpha, u_\beta, v_\beta$など）に基づいて、中間

検証者の役割 検証者は、自身が持つVOLEキー（$q_\alpha, q_\beta, q_\gamma$）から特定の値$b$を計算する。その後、

$$b \stackrel{?}{=} a_0 + a_1 \cdot \Delta \tag{4}$$

このチェックが成功すれば、証明者が正直に計算を行ったことが高い確率で保証される。も

しかし、この仕組みは、検証者がグローバルキーΔの値を秘密に保つことを前提としている。
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

前節で述べた指定検証者問題を解決し、公開検証可能性を実現するのがVOLE-in-the-Head (VOLEitH) と呼ばれるコンパイラである。これはFAEST [?]やzkPassなどのシステムで採用されており、本研究で利用するSchmivitzライブラリ[12]もこのパ[?] のような効率的なVOLE相関生成プロトコルを、MPC-in-the-Head [?]の手法と組み合わせることで、非対話かつ公開検証可能な証明を構築する。

VOLEitHの効率性の根幹を支えるのが、SoftSpokenOT[?]と呼ばれるプロトコルである。これ
Transfer）から、擬似乱数を用いて極めて多数の安価なOTインスタンスを生成するOT拡張技術の
1個の情報を効率的に転送する」という処理（All-but-One OT）と非常に相性が良い。結果として、

その核心は、検証者が秘密のグローバルキーΔを持つ代わりに、証明者自身がプロトコルの中で
Shamir変換によってチャレンジとしてΔを導出する点にある。このプロセスは、GGMツリー[?]を
but-One Oblivious Transfer (OT) の技術を用いて効率的に実装される。

GGMツリーによるAll-but-Oneコミットメント GGMツリー[?]は、Goldreich、Goldwasser、M
but-Oneコミットメント、すなわち多数の値の中から一つだけを隠して残りをすべて効率的に開示
プロトコルは以下のように進行する。

ステップ1: GGMツリー構築とコミットメント まず、セキュリティパラメータλに対して、入力シ
$G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ を定義し、$G(s) = (s_0, s_1)$ とする。証明者は、ルートシード$s_{root} \in \{0,1\}^\lambda$をランダムに選択する。 深さ$d = \log N$のGGMツリーは、このルートシードから再帰的
$(1 \le j \le d)$のノードは、その親ノードs_{parent}から$G(s_{parent})$によって生成される。具体的には、
$\{0,\dots,N-1\}$のdビット表現を$i_1 i_2 \dots i_d$とすると、リーフ（葉）シードsd_iは、パス$i_1 \dots i_d$を辿
$s_{i_1 \dots i_d}$として計算される。 証明者は、こうして得られたN個のリーフシード$\{sd_i\}_{i=0}^{N-1}$全体に対
$H(sd_0, \dots, sd_{N-1})$を計算し、コミットメントとして公開する。

ステップ2: VOLE値の計算 証明者は、各リーフシードsd_iをさらに別のPRG（G'とする）で展開し
$G'(sd_i)$を得て、自身の秘密VOLE値u, vを以下のように計算する。

$$\mathsf{u} = \sum_{i=0}^{N-1} r_i \tag{5}$$

$$\mathsf{v} = \sum_{i=0}^{N-1} i \cdot r_i \tag{6}$$

ステップ3: チャレンジと開示 (All-but-One OT) 証明の主要部分が構成された後、Fiat-Shamir変換により、証明全体のトランスクリプトからチャレンジ$\Delta \in \{0,\dots,N-1\}$が導出される。このΔが開示を免除されるインデックスとなる。
証明者は、GGMツリーにおいてsd_Δを隠したまま残りのすべてのシードを開示するため、ル
node）のシードを開示情報$pdecom$として証明に含める。Δのビット表現を$\Delta_1 \dots \Delta_d$とすると

$$pdecom = \{s_{\Delta_1 \dots \Delta_{j-1} \bar{\Delta}_j}\}_{j=1}^{d}, \quad \text{ここで } \bar{\Delta}_j = 1 - \Delta_j \tag{7}$$

ステップ4: 検証 検証者は、まず証明者と同じくFiat-Shamir変換でΔを導出する。次に、$pdecom$
Δに対応するリーフシードsd_iを再計算する。この再計算が可能であるのは、iとΔのビット表
$i_j = \bar{\Delta}_j$）から計算を開始できるからである。 検証者は、復元した$N-1$個のリーフシードからh_{com}が正しく再構成できることを確認し、開示の正当性を検証する。
$u_i \Delta + v_i$の検証へと進む。

2.4 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

- **ゼロ知識 (Zero-Knowledge)**: 証明が「あるステートメントが真である」という事実以外のい
- **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
- **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
  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り、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, pi）を生成する。

Verify 検証者は、検証鍵、公開入力、および証明 pi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（ $mathbbG_1$の元2つ、 $mathbbG_2$の元1つ）で構成され、回路の規模に関わらず常に一定のサイズである。 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）

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

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 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

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、VOLE 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

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 この検証ロジックを、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

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 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 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）と

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 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

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 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

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
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
- 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VO
次に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

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
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

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）
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、VOL
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（Grot

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 μs	476.5 μs	790.062 μs	1.649 ms
Proof Ver. Time	68.75 μs	274.566 μs	585.6 μs	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
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、
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
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（G

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
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
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
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、A
例えば、1000 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
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

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
- 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
- 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
- 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

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
compute_d_delta	$128n$
compute_masked_witness	$256n$
compute_validation_aggregate	$16,512n$
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
combine	$\sim 2,097,152$
compute_actual_validation	$\sim 16,384$
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

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AN
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

```
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、Looku
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
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
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

- **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
  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

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
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

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

todo: faest, faester,binius, 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
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
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
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

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
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

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**: Binius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

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.