VOLE-in-the-Headゼロ知識証明のオンチェーン検証における実現可能性と

著者名
所属機関

November 16, 2025

Abstract

VOLE-in-the-Head（VOLEitH）は、線形演算を中心とした構成により証明者計算を大幅に削減し
一方、ブロックチェーン業界では、秘匿送金などでゼロ知識証明技術が採用されており、VOLE
itHのような軽量な証明システムが求められている。 本研究では、こうした理由からVOLEitHゼロ知識
ベンチマークとして、標準的な暗号学的ハッシュ関数（SHA-256, Keccak-
F）および基本的な論理ゲート回路を用いた。 結果として、VOLEitHは既存のゼロ知識証明実装（Ci
さらに、VOLEitHの証明をSNARKで圧縮することにより、証明サイズを1,055バイトに固定し、オン
本稿は、VOLEitHをオンチェーンアプリケーションへ適用する際の技術的なトレードオフを明らかに

1　序論 (Introduction)

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（Zero-Knowledge Proof, ZKP）は、ある計算が正しく実行されたことを、その計算
この「ゼロ知識」という性質は、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

　ブロックチェーン、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
例えば、計算量の多い処理をオフチェーンで実行し、その結果の正当性のみをZKPを用いてオンチ
しかし、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

　TODO: プライバシー保護型ロールアップやオンチェーン監査など、VOLEitHハイブリッドを通
TODO: ガス代とは

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S
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
TODO: SNARKの説明を追加

　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。 これにより、証明者側の高い計算効率とい

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　実現可能性分析と主要な知見

Milestone 1と2では、VOLEitHの証明をオンチェーンで扱うために複数の手法を検討し、最終的に
Wrappingを実装・評価した。本節では、その意思決定過程と主要な洞察を整理する。

2.1　SNARK Wrapping (Groth16)

最も直接的なアプローチは、VOLEitHの検証ロジック全体をGroth16で包む方法である。この手法
gasに固定できた。一方で、制約数は

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

と線形に増加し、特に非線形ゲートを多く含む回路では証明生成時間が急増する。実装は以下のコ

- schmivitz-snark: VOLEitH検証ロジックをarkworksベースのGroth16で証明するためのラッ
- VOLEitH-bench: Groth16圧縮後の証明をEVMで検証し、エンドツーエンド性能を測定する

SHA-256のようにANDゲートが2万個を超える回路では、この制約爆発によりSNARKフェーズの

3　背景 (Background)

本章では、我々の研究の基礎となる暗号学的概念とプロトコルについて解説する。

3.1　ゼロ知識証明の基礎

ゼロ知識証明（ZKP）は、証明者（Prover）が検証者（Verifier）に対し、ある表明が真であるこ

1. 完全性 (Completeness): 証明者の表明が真であるならば、正直な証明者は正直な検証者を必
2. 健全性 (Soundness): 証明者の表明が偽であるならば、不正な証明者が正直な検証者を騙して
3. ゼロ知識性 (Zero-Knowledge): 検証者は、証明の正当性以外には、証明者の持つ秘密情報に

ZKPは、証明者と検証者の間で複数回のやり取りを必要とする「対話型証明システム」と、証明者

3.2　VOLEと関連プロトコル

VOLE（Vector Oblivious Linear Evaluation）は、二者間（SenderとReceiver）のセキュアな計
基本的なVOLEプロトコルでは、Senderが持つアフィン変換 f(x) = ax + b
と、Receiverが持つベクトル u に対し、Receiverが v = au + b を計算する。
この過程で、Senderは u について、Receiverは a, b について何も知ることができない。
　VOLEベースのプロトコルの多くは、LPN（Learning Parity with Noise）仮定の困難性に基づ
LPN仮定とは、ランダムな線形方程式系にノイズが加わったものから、元の線形関係を復元するこ
　VOLEをZKPに応用するために、SPVOLE（Single-Point VOLE）やZP-
VOLE（Zero-Point VOLE）といった派生プロトコルが考案された。 これらは、特定の点でのみ値

3.3　VOLE-in-the-Head (VOLEitH)

VOLE-in-the-Head (VOLEitH) は、VOLEベースのプロトコルをZKPに昇華させた手法である。
その基本的なアイデアは、証明したい算術回路のワイヤ値をProverがコミットし、Verifierがラン
　本研究で利用するsoft_spokenライブラリは、VOLEitHの具体的な実装の一つである。soft_sp
VOLEを巧みに組み合わせることで、効率的な証明生成を実現している。プロトコルの流れは以下

1. **コミットメント**: Proverは、算術回路の各ワイヤの値を表すベクトルにコミットする。
2. **チャレンジ**: Verifierは、ランダムなチャレンジ（乱数）をProverに送信する。
3. **レスポンス**: Proverは、チャレンジに基づき、コミットしたベクトル間の線形関係が成立する
4. **検証**: Verifierは、Proverからのレスポンスと自身が持つ情報を用いて、線形関係が成立する

この対話型のプロトコルを非対話的にするため、Fiat-Shamir変換が用いられる。これは、Verifie
これにより、ProverはVerifierとの対話なしに証明を一方的に生成でき、生成された証明は誰でも

3.4　SNARKsとオンチェーン検証

VOLEitHによって生成された証明は、証明者の計算効率は高いものの、証明サイズが非常に大きい
4.1節で詳述）。このままではオンチェーン検証は現実的ではないため、証明をさらに圧縮する技
　ここで登場するのがSNARK（Succinct Non-interactive Argument of
Knowledge）である。SNARK、特に現在主流のGroth16などは、証明サイズを数百バイト程度ま
これは、計算の正当性の問題を、特定の性質を持つ多項式の存在問題に変換し、その多項式に対す
　SNARKの検証は、Ethereum Virtual Machine (EVM) 上でスマートコントラクトとして実装で
これにより、VOLEitHの巨大な証明をSNARKで圧縮し、そのコンパクトなSNARK証明をオンチ

3.5　信頼モデルと安全性仮定

TODO: CRSセットアップ、透明性、LPN系と楕円曲線系の安全性仮定を比較し、VOLEitHとSNA

3.6　オンチェーン検証に必要な運用前提

TODO: コントラクトデプロイ、鍵管理、Blobやカリブレーションデータの配信方法など、実運用

4　ベンチマーク設計 (Benchmark Design)

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

4.1　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

- **証明生成時間 (Proof Generation Time)**: 証明者が、ある計算に対する証明を生成するために要
- **証明検証時間 (Proof Verification Time)**: 検証者が、与えられた証明の正当性を検証するために
- **証明サイズ (Proof Size)**: 生成された証明データの大きさ。単位はバイト（B）。
- **通信オーバーヘッド (Communication Overhead)**: 非対話型証明において、証明者が検証者に
- **計算負荷 (Computation Load)**: 証明生成および検証プロセス中に消費されるCPU使用率（%）
- **SNARK制約数 (Number of constraints)**: VOLEitHの証明をSNARKに変換する際に生成される
- **オンチェーン検証ガス代 (On-Chain Verification Gas Cost)**: 生成されたS-NARK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量。

4.2　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 ライブラリ
  - 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

4.3　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

- **大規模暗号回路**:
  - 内容: SHA-256 および Keccak-F。これらは暗号技術で広く利用される標準的なハッシュ
  - 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

5　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

5.1　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 1: 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1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、そして証明サイズのい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（[1]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 2: 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2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

5.2　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

まず、VOLEitHフェーズの性能を表3に示す。

表3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

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（

表4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 gasで一定であり、これはSNARKの検証が固定コストで行われることを示している。これにより、

Table 3: 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

Table 4: 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

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
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

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

5.3　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
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、

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、
Webブラウザ、スマートフォン）でも、複雑な計算の証明を現実的な時間で生成できる可能

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

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、
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

・ **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

6 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
Mappingを整理する。

6.1 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5の通りであり、compute_validation_aggregateが支

Table 5: 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6）。乗算ゲートが増えると線形項が

Table 6: 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	$\sim 2,113,664$

6.2　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

```
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

6.3　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

```
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

7　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

7.1 Field Mapping最適化とLookup Table

Mystique[2]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[3]で示されている。表7に示す
130倍短縮し、通信量も最大2.9倍削減できる。VOLEitHのField MappingにMys-
tique型LUTを適用できれば、SNARKフェーズの制約数削減に直結すると期待される。

Table 7: 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

7.2 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

Table 8: 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

さらに、Milestone 1で検討したFoldingスキームを適用すれば、GGM木の各レイヤを段階的に

8 関連研究 (Related Work)

本研究は、VOLEベースのゼロ知識証明をオンチェーンで実用化するための性能評価を行ったもの

8.1 他の主要なZKPシステムとの比較

現在、ZKPシステムには多様なアプローチが存在し、それぞれが異なるトレードオフを持つ。

- zk-STARKs: STARKsは、透明性（Trusted Setupが不要）とポスト量子耐性を大きな特徴と
  Setupが必要）と組み合わせている点で、純粋なSTARKsとは異なるアプローチを取ってい
- Plonk/Halo2: これらのシステムは、Groth16のような特定の回路ごとにTrusted
  SetupをSNARKとは異なり、一度のTrusted Setup（Universal
  Trusted Setup）で様々な回路に再利用できるという利点を持つ。これにより開発の柔軟性

・ R1CSベースのSNARKs (例: Circom/Groth16): 本研究でも比較対象とした、現在最も広く使

8.2 VOLEベースZKPに関する先行研究

VOLEベースのZKPは、その証明者効率の高さから近年活発に研究されている分野である。
extttsoft_spokenの元となった論文をはじめ、 extttmozzarella、 extttmac-
n-cheeseといった多くの実装が提案されている。これらの研究の多くは、プロトコルの理論的な改
extttsoft_spokenを実際に用い、SNARKによる圧縮を経てオンチェーンで検証するまでのエンドツ

8.3 オンチェーンZKP検証に関する既存の取り組み

オンチェーンでのZKP検証は、ZKロールアップ（例: StarkNet, zkSync,
Polygon zkEVM）によって大きな成功を収めている。これらのプロジェクトは、大量のトランザク
STARKsやR1CSベースのSNARKsを利用している。
本研究のアプローチは、これらの大規模なロールアップとは異なり、個別のアプリケーションが

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

9.1 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
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
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

9.2 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

・ **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、Gr

参考文献 (References)

References

[1] Iden3. Circom: A Circuit Compiler for Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2023/681, 2023. https://eprint.iacr.org/2023/681.

[2] 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.

[3] 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.

[4] 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.