Air-FRI: Acceleration of the FRI Protocol on the GPU for zkSNARK Applications

Tanmayi Jandhyala and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario
Canada
{tjandhyala, ggong}@uwaterloo.ca

Abstract. Blockchain holds the potential to support the privacy of its participants by integrating zero-knowledge proofs into its design. zk-SNARK schemes can effectively prove the validity of blockchain transactions to their owners without disclosing user data. However, the efficiency and scalability of the underlying cryptographic protocols that enable these schemes remain a challenge to realize in practice. This paper presents Air-FRI, a novel GPU-enabled software implementation of the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) protocol, which is a core component in post-quantum zkSNARKs that reduces computational complexity in systems with substantial mathematical instances. Existing schemes that implement the FRI protocol entail significant computational times due to large proof sizes. Our optimized solution includes a parallelized computation of Reed-Solomon codewords, the pre-computation of time-intensive finite field operations, non-interactiveness, and the application of a unified Merkle tree commitment to authenticate the entire proof. Together, the implemented optimizations yield a solution that significantly reduces prover and verifier times while minimizing proof size, addressing both scalability and performance challenges in zkSNARK-based algorithms. Performance evaluations conducted by us across two security levels confirm the implementation's high throughput, establishing it as a promising solution for practicable privacy-preservation. Our results show a 93.3% improvement on an average in the speed of the protocol on the GPU as compared to a non-GPU solution for the same parameters, which includes the execution time of all of its sub-phases: the commit phase, the query phase, and the round consistency checks to ensure correctness of the proof. This work establishes a foundation for advancing privacy-preserving post-quantum cryptography, supporting the development of secure, transparent, and decentralized digital infrastructures for the future.

Keywords: FRI · zkSNARKs · GPUs · Software.

1 Introduction

In a digitally-driven society, the balance between data privacy and trust is often compromised. While individuals expect sensitive data, such as medical or financial records, to remain secure and accessible only with explicit consent, these expectations are frequently violated or undermined by institutional misconduct or technological vulnerabilities [8] respectively. Such breaches erode the foundational trust of centralized systems. Blockchain technology [39] offers a decentralized and persistent framework for recording transactions and has been adopted across financial services [43], smart contracts [34], and public services [1]. However, its reliance on distributed public ledgers introduces significant privacy concerns, as all participants must maintain identical copies of data. Addressing these challenges is crucial for organizations to ensure trust while leveraging blockchain's potential [48].

A significant concern within these privacy challenges is the de-anonymization of data recorded on blockchains. Transactions on public ledgers, once verified, are permanently stored in open databases. Past network-based de-anonymization attacks [14,35,15] have linked blockchain transactions to real-world identifiers, such as IP addresses. Additionally, user interactions with the blockchain such as querying transactions, checking account balances, or requesting information about specific blocks can inadvertently reveal their areas of interest. For instance, repeatedly checking the balance of a particular wallet may suggest ownership or a financial association with that wallet [3].

Researchers have developed several cryptographic techniques to address privacy challenges on the blockchain. Among these, Zero-Knowledge Proofs (ZKPs) [27], introduced to prove possession of information without revealing it, play a pivotal role. While initially interactive [4,25], non-interactive proofs were later proposed [17,6], enabling their integration into modern cryptographic systems. Notably, ZCash [32] (based on ZeroCash [10]) was among the first blockchain frameworks to employ ZKPs for user anonymity. Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zkSNARKs) [30,31] extend ZKPs, offering short, efficient proofs that verify knowledge of computations while preserving privacy. Such systems also find utility in zk-oriented digital signature schemes [24].

The challenge is that such systems involve handling extensive computations which must be efficiently verified. To address scalability, Ben-Sasson et al. introduced the ***Fast Reed-Solomon Interactive Oracle Proofs of Proximity*** (FRI) protocol [7], which leverages algebraic structures for verifying mathematical computations, particularly in polynomial commitment schemes [18], where large polynomials are checked for their degree efficiently. FRI employs a transparent setup, meeting the requirements of blockchain infrastructures, and operates as an Interactive Oracle Proof [12] to verify the low-degree property of functions over finite fields with minimal communication complexity and verification costs. The protocol has been adopted in notable works such as Polaris [24], Aurora [11], STARK [9], Fractal [21], Spartan [46], and PREON [20] which builds on Aurora's FRI.

This paper presents a GPU-based implementation of the FRI protocol which employs Reed-Solomon (RS) codewords, as demonstrated in prior works [7,20]. RS codewords encode messages as polynomials over finite fields, evaluated at distinct field points to form codewords. Our implementation, initially programmed in C, incorporates optimizations identified within the protocol and further parallelizes computationally intensive parts on the GPU. Our optimized GPU implementation achieves a remarkable **93%** performance improvement compared to its unoptimized, CPU-only counterpart. Additionally, this work introduces a *specialized* Merkle Tree, designed to accelerate verification while ensuring the protocol's authenticity and integrity.

Our Contributions

The contributions in this paper are **fourfold**, focusing on the optimizing the performance of the FRI protocol in implementation for effective low-degree testing of polynomials for zkSNARK schemes.

1. *Precomputation of inverse elements and implementing repeated hashing of a random element by the prover for every round.*
   The FRI protocol operates through multiple rounds, performing low-degree testing of polynomials represented as Reed-Solomon codewords. At each round, it needs to compute a finite field inverse element. Our first optimization method is to move this computation to a pre-computation done prior to the start of the protocol and the data is stored to be retrieved during the execution of the protocol versus on-the-fly. Secondly, we adopt a method implemented in Factal's [21] FRI and we realize this in our standalone implementation to make the protocol non-interactive by having the prover sample the random element only once before the FRI protocol begins. This also reduces the communication cost, since the prover is given this random element only once at the beginning, facilitating non-interativeness of the protocol in practice.
2. *Implementation and parallelization of a specialized Merkle tree for efficient commitment and verification in the FRI Protocol.*
   We implement an *integrated, verifiable* Merkle tree, designed to efficiently commit all codewords in one go. Specifically, the prover commits to its codewords using a single, specially-constructed Merkle tree, enabling the verifier to verify the authenticity of all committed codewords without requiring the prover to construct an independent Merkle tree for the commitment of each codeword. This implementation not only substantially reduced the proof size but also enhanced the overall performance of the algorithm.
3. *Parallelization of FRI's Commit Phase.*
   As a key contribution, this work presents for the first time to the best of our knowledge, an implementation of the commit phase of the FRI protocol (that follows an additive-FFT for polynomial evaluation) on a GPU, leveraging the parallel computability of all elements in a codeword round. In this approach, each thread is assigned to compute the next codeword element for a specific index during each round, with thread synchronization ensuring that the hierarchical structure of all indices is preserved across all rounds of the protocol.

4. *Implementation and Performance Report*
   As a final contribution, we provide a detailed performance evaluation of our optimized FRI implementation. Our benchmarks measure prover and verifier runtimes across different field sizes and optimization levels, illustrating the tangible performance gains from GPU acceleration and from the integrated Merkle tree. These results demonstrate the practical viability of our approach, offering a pathway for deploying FRI-based zkSNARKs in computation-intensive settings. Table 1 introduces performance comparisons between having no optimizations (CPU-only implementation) versus Air-FRI. The `L3` and `L5` parameter sets defined in Preon [20] correspond to security levels based on the size of the finite field and degree of the tested polynomial. Specifically, `L3` uses a 256-bit field $\mathbb{F}_{2^{256}}$, while `L5` operates over a larger 320-bit field $\mathbb{F}_{2^{320}}$, with correspondingly larger codewords and deeper recursive rounds. These configurations represent realistic zkSNARK security levels, allowing us to evaluate our optimizations under practical, high-assurance cryptographic workloads.

Security Parameters	**CPU Time**	**GPU Time (Air-FRI)**	**Speedup (%)**
`L3`	22.00s	1.49s	93.3%
`L5`	400.00s	13.89s	96.5%

Table 1. Prover time comparison between CPU and GPU implementations of FRI protocol

This work together is termed *Air-FRI*, where 'Air' indicates the fast performance factor of the FRI Protocol, and doubles to mean 'Acceleration'.

Outline. The rest of the paper has the following structure. Section 2 provides background on how the FRI Protocol came to be introduced in zkSNARK schemes. Section 3 mathematically describes the FRI Protocol and its phases. Section 4 discusses our optimizations in depth. Section 5 and Section 6 detail our implementation and its results. While Section 7 presents limitations of our work, Section 8 provides concluding remarks and future work.

2 Background and Related Work

Zero-Knowledge Proofs are a cryptographic tool that allow two parties, a prover P and a verifier V to engage in a 'proof' that attests to the validity of a statement without revealing any information beyond the truth of the statement itself. First introduced [27] in 1985, ZKP systems must satisfy three essential properties, as interpreted from the definitions in the article by [26]: *completeness, soundness,* and *zero-knowledge.*

The evolution of zero-knowledge proofs (ZKPs) introduced non-interactive variants, with Goldwasser and Bellare [6] presenting NIZK, enabling generic theorem validity to be proven without repeated interactions. A NIZK proof system involves a setup phase to establish shared information, followed by a non-interactive phase for proof generation. Building on Blum et al.'s work [17], Rackoff and Simon [44] developed non-interactive proofs of knowledge, securing public-key cryptosystems against chosen ciphertext attacks and demonstrating proofs for **NP** using trapdoor functions. Micali [38] introduced Succinct Non-Interactive Arguments (SNARGs) to enhance efficiency by reducing proof sizes and verification times, which paved way for Bitansky et al. [16] to formalize SNARKs using assumptions like extractable collision-resistant hash functions. zkSNARKs, an advanced class of non-interactive proofs, offer succinct, computationally secure verification and have become pivotal for blockchain privacy by enabling efficient, verifiable transaction ownership without revealing sensitive data. Zero-knowledge digital signatures play a critical role in blockchain by ensuring integrity, authentication, and non-repudiation of transactions. While traditional digital signatures possess zero-knowledge-like properties, non-interactive zero-knowledge proofs offer additional benefits, including *public verifiability*, as highlighted by [6]. This enables proofs to be validated by any party in the setup, whether trusted or transparent.

Practical zkSNARKs rely on Arithmetic Circuits (ACs) and Quadratic Arithmetic Programs (QAPs) [12] to transform computational problems into succinct proofs. QAPs represent circuits as polynomial equations, allowing zkSNARKs to prove correctness without revealing inputs, while maintaining small proof sizes and efficient verification [13]. These characteristics make zkSNARKs highly scalable for blockchain applications. Ben-Sasson [8] introduced zkSTARKs, which ensure transparency and post-quantum resilience without requiring a trusted setup. However, zkSTARKs produce proofs roughly 1000 times larger than zkSNARKs [8,42], and zkSNARKs generally exhibit better performance for smaller proof sizes and verification times. The primary advantage of zkSTARKs lies in their transparent setup and quantum-resistant primitives, such as collision-resistant hash functions.

Efficient zkSNARK systems often rely on the FRI-IOP protocol to handle large polynomials without revealing them, employing Reed-Solomon codes to encode these polynomials into finite field elements [7]. This approach supports transparent polynomial commitment schemes (PCS) for constructing short proofs and ensuring efficient verification with minimal or no interaction. The FRI protocol, utilized in systems like Polaris [24], Aurora [11], and STARK [8], enables zkSNARK scalability by producing logarithmically growing proof sizes while maintaining transparency.

3 The FRI Protocol

In our work, we were able to translate the following information-theoretic math into code that can efficiently run on the GPU. The mathematical workings of the protocol and the notations used in this paper is referenced from the authors of [24] and [5].

3.1 Notations

Let $\mathbb{F} = \mathbb{F}_{2^N}$ be a finite field of characteristic 2, and let $L \subset \mathbb{F}$ be a linear subspace of dimension n, spanned by the basis $\{\beta_0, \beta_1, \cdots, \beta_{n-1}\}$. Each element $\alpha_i \in L$ is formed via binary linear combinations of these basis vectors using coefficients $i_j \in \mathbb{F}_2$, such that $i = i_0 + i_1 2^1 + \cdots + i_{n-1}2^{n-1}$ and $\alpha_i = i_0\beta_0 + i_1\beta_1 + \cdots + i_{n-1}\beta_{n-1}$. The space L thus contains 2^n elements indexed as $L = \{\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}\}$. A Reed-Solomon code of rate $\sigma \in (0,1)$ over L is denoted $\mathsf{RS}[L,\sigma]$ and defined as the set $\mathsf{RS}[L,\sigma] = \{\boldsymbol{f}_L \mid f \in \mathbb{F}[x], \deg(f) < k' = 2^n\sigma\}$, where $\boldsymbol{f}_L = (f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{2^n-1}))$. This code is linear with (the Hamming distance of) at least $2^n - k'$, since any polynomial f of degree less than k' has at most k' roots, and hence at most k' zero entries in the codeword. This follows from the classical Reed-Solomon distance bound [37].

3.2 Phases of the FRI protocol [5,29]

The FRI Protocol is designed to have three functions/phases: (i) the commit phase, (ii) the query phase, (iii) the round-consistency check. We denote this as $FRI(com, query, RC{-}test)$. Our optimizations in the protocol are not presented here yet.

Let $L_0 = L$ and $\mathsf{RS}[L,\sigma]$. Let $f(x) \in \mathbb{F}[x]$ with degree $< \sigma 2^n = 2^k$ ($\sigma = 2^{n-k}$), so $\boldsymbol{f}_L \in RS[L_0,\sigma]$. Given $f(x)$ and $\boldsymbol{f}_L \in RS[L_0,\sigma]$, the protocol intends to prove to a verifier the following NP statement:

$$\deg(f(x)) < 2^k.$$

The process is to iteratively reduce the degree of f *by half* at each step of the proof computation. The protocol begins with the input consisting of a polynomial function $f(x)$, the initial subspace L_0, σ, and the RS codeword $\boldsymbol{f}_L \in RS[L_0,\sigma]$. The output is a binary value $b \in \{0,1\}$, indicating whether the verification passed.

Commit Phase The commit phase of the FRI protocol involves both the prover and verifier interacting to commit to codewords derived from a polynomial over several rounds. The prover starts with the input $f^{(0)} : L_0 \to \mathbb{F}$, which represents an initial Reed-Solomon (RS) codeword corresponding to the polynomial $f_0(X)$ with a rate ρ. The commitment process is described. For each round $k = 0, \ldots, \mathsf{r}-1$, the following steps occur:

1. Prover Commitment:
   The prover recursively defines codewords $\boldsymbol{f}^{(k)} : L_k \to \mathbb{F}$ for each round.
   It then computes a Merkle tree commitment for each codeword and sends the Merkle root to the verifier.
2. Verifier Challenge:
   The verifier sends a uniformly random element $\alpha^{(k)} \in \mathbb{F}$ to the prover.

3. Codeword Definition:
   The prover constructs the next codeword $\boldsymbol{f}^{(k+1)}$ with domain L_{k+1} such that, for every i where $0 \leq i < |L_{k+1}|$, the following holds:

$$q_k(L_k[2i]) = q_k(L_k[2i+1]) = L_{k+1}[i].$$

   Each element of the new codeword $\boldsymbol{f}^{(k+1)}$ is derived from the previous codeword:

$$f_{k+1}(L_{k+1}[i]) = \frac{f_k(L_k[2i]) - f_k(L_k[2i+1])}{L_k[2i] - L_k[2i+1]}(\alpha^{(k)} - L_k[2i]) + f_k(L_k[2i]). \quad (1)$$

   The term $L_k[2i] - L_k[2i+1]$ is $\beta_0^{(k)}$ [29,5,28].
   At the final round $k = \mathsf{r}$, the prover constructs the codeword $\boldsymbol{f}^{(\mathsf{r})} : L_{\mathsf{r}} \to \mathbb{F}$. This last codeword is directly sent to the verifier as the final commitment.

Query Phase In the query phase, the verifier extracts the Merkle roots, the challenges $\alpha^{(0)}, \alpha^{(1)}, \ldots, \alpha^{(\mathsf{r}-1)}$, and the final codeword $\boldsymbol{f}^{(\mathsf{r})}$. The verifier also gains oracle access to all intermediate codewords $\boldsymbol{f}^{(0)}, \boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(\mathsf{r}-1)}$. In symbol, this is represented as $\boldsymbol{f}^{(k)} = (\boldsymbol{f}_{k,0}, \boldsymbol{f}_{k,1}, \cdots, \boldsymbol{f}_{k,|L|_k-1})$.

First, the verifier performs *Interpolant Verification* by computing the interpolant polynomial $f_{\mathsf{r}}(X)$ using points from $\boldsymbol{f}^{(\mathsf{r})}$. If the degree of $f_{\mathsf{r}}(X)$ exceeds $\rho \cdot |L_{\mathsf{r}}| - 1$, the verifier **rejects** the proof. Second, the *Round Consistency Check* ensures consistency between consecutive codewords through ℓ iterations. In each iteration, a random index $s^{(0)} = i$ is selected, where $0 \leq i < |L_0|$, and for every $0 \leq k < \mathsf{r} - 1$, the index is updated as $s^{(k+1)} = \lfloor s^{(k)}/2 \rfloor$. If $s^{(\mathsf{r})}$ is sampled more than once, the verifier resamples $s^{(0)}$. Lastly, during the *Verification of Codeword Values*, the verifier queries the values $f_{k+1}(L_{k+1}[s^{(k+1)}])$, $f_k(x_1^{(k)})$, and $f_k(x_2^{(k)})$, where $x_1^{(k)} = L_k[2s^{(k+1)}]$ and $x_2^{(k)} = L_k[2s^{(k+1)} + 1]$. The correctness of these queried points is ensured by verifying their Merkle paths.

Round Consistency Check The verifier checks the following equation for every $k \in \{0, 1, \ldots, \mathsf{r} - 1\}$:

$$f_{k+1}(L_{k+1}[s^{(k+1)}]) = \frac{f_k(x_1^{(k)}) - f_k(x_2^{(k)})}{x_1^{(k)} - x_2^{(k)}}(\alpha^{(k)} - x_1^{(k)}) + f_k(x_1^{(k)}).$$

If any of these checks fail, the verifier rejects the proof.
Acceptance: If all checks pass, the verifier accepts the proof, confirming that the original polynomial $f_0(X)$ has a degree no more than $\rho \cdot |L_0| - 1$.

4 Air-FRI

As zkSNARKs progress to support more intricate circuits and transparent setups, achieving computational and memory challenges encountered during their deployment for large-scale cryptographic applications remains a challenge. In traditional zkSNARK implementations, the prover faces substantial overhead when managing large arithmetic circuits or evaluating polynomials, especially when encoding computations into Rank-1 Constraint Satisfiable (R1CS) systems. This bottleneck becomes even more pronounced in protocols like Aurora [11] and Fractal [21], where polynomial commitments using FRI demand significant memory management and intensive field operations [45] [36].

A key challenge addressed in this work is the prover's time complexity. Existing FRI-based zkSNARK systems, while providing post-quantum security and transparent setups, often present a trade-off between proof size and prover efficiency. For instance, Aurora achieves proof sizes in the range of 100-150 KB with logarithmic verification times but incurs increased prover workload scaling with $O(n \log n)$ for large circuits. Similarly, Fractal reduces verification time but imposes considerable computational demands on the prover due to multi-round polynomial evaluations. These limitations underscore the need for novel optimizations to reduce the prover's computational burden while preserving the succinctness and transparency of FRI-based proofs.

This paper introduces optimizations aimed at parallelizing polynomial computations and integrating Merkle tree commitments into GPU-accelerated workflows. By utilizing CUDA-enabled GPU kernels to offload polynomial evaluations and streamline Merkle commitments, our proposed solution reduces memory overhead and prover time without compromising on the transparency and security guarantees inherent to FRI-based systems. These enhancements are expected to accelerate proof generation and improve the scalability of zkSNARK applications, particularly in blockchain environments, where throughput and latency are critical for adoption.

The optimizations proposed in this paper are categorized into four main contributions, as detailed below.

4.1 Precomputation of Inverse Elements and Repeated Hashing of a Random Element

The **line computation** in the FRI protocol plays a critical role in recursively transforming polynomial evaluations across rounds to ensure proximity to low-degree polynomials. This transformation is mathematically expressed by Equation (1), where the verifier provides a uniformly random challenge $\alpha^{(k)}$, and the transformation depends on the denominator term

$$\beta_0^{(k)} = L_k[2i] - L_k[2i+1].$$

Computing the inverse of $\beta_0^{(k)}$ in each round introduces considerable overhead due to the cost of division operations in finite fields $\mathbb{F}_p$. To mitigate this bottleneck, we precompute these inverses before the recursion begins. The stored values are then retrieved during protocol execution, replacing expensive on-the-fly inverse computations with efficient multiplications, which are more favorable in field arithmetic [11,21]. This optimization substantially reduces the prover's computational load, especially for large circuits with many recursive steps.

Additionally, rather than following an interactive protocol, we implement a non-interactive variant by having the prover sample a random element α_0 once from the extension field at the start of the protocol. This element is reused across all rounds by computing the challenges deterministically through repeated hashing. Specifically, the next round's challenge is computed as

$$\alpha_1 = \text{Hash}(\alpha_0 \parallel \text{root}),$$

and for all subsequent rounds,

$$\alpha_{k+1} = \text{Hash}(\alpha_k), \quad 1 \leq k \leq r,$$

where r denotes the total number of FRI rounds. The verifier reconstructs the challenges using the initial α_0 and the hash function, eliminating the need for back-and-forth communication. This non-interactive design adheres to the **Fiat-Shamir heuristic** [23] and aligns with techniques used in prior work such as Fractal [21] and the framework of Interactive Oracle Proofs [12]. By minimizing interaction, this strategy improves scalability and seamlessly supports GPU-based acceleration for both hashing and proof generation [11,21].

4.2 Integrated FRI Merkle Tree Construction

The primary motivation to the following contribution of our work is to reduce the prover's computational overhead while ensuring the verifier's ability to validate the integrity of the proof through a recursive structure of polynomial evaluations. This construction, as described in the previous section, is based on Reed-Solomon (RS) codewords evaluated over a sequence of domains and committed using a single Merkle tree, ensuring both scalability and security in the zkSNARK setting [11,21].

As our second essential contribution, we construct and employ a verifiable and integrated Merkle tree for simultaneous commitment of all codewords. Specifically, the construction combines codewords and Merkle hash computations into one tree structure. The details are as described below.

RS Tree Structure The recursive nature of the FRI protocol can be represented as a binary tree. The elements of the initial codeword $\boldsymbol{f}_0$ are placed as the leaf nodes at layer 0. Each subsequent layer $k+1$ is constructed from the elements of layer k using the linear combination described earlier in (1). At the final layer r, there are $|L_r|$ nodes. This recursive binary structure is referred to as the RS tree with height r and initial codeword $\boldsymbol{f}_0$. The tree is denoted by $RST(f_0, r)$, where $f_0(x)$ is the original polynomial evaluated over L_0.

Merkle Tree Commitment of RS Codewords To guarantee the integrity of Reed-Solomon (RS) codewords, we construct a Merkle tree that integrates seamlessly with the RS tree structure. This Merkle tree provides an efficient commitment scheme by hashing pairs of codeword elements along with their corresponding RS codeword values at each index at each layer. The construction proceeds as follows.

For layers $0 \le k \le r$, we compute hash values iteratively:

$$\begin{aligned}
h_{1,i} &= h(f_{0,2i}, f_{0,2i+1}), & 0 \le i < |L_1|, \\
h_{2,i} &= h(h_{1,2i} \parallel f_{1,2i}, h_{1,2i+1} \parallel f_{1,2i+1}), & 0 \le i < |L_2|, \\
&\vdots \\
h_{k+1,i} &= h(h_{k,2i} \parallel f_{k,2i}, h_{k,2i+1} \parallel f_{k,2i+1}), & 0 \le i < |L_{k+1}|.
\end{aligned}$$

For subsequent layers, where $r \le k < n$, we compute hashes without appending RS values:

$$h_{k+1,i} = h(h_{k,2i}, h_{k,2i+1}), \quad 0 \le i < 2^{n-k}.$$

The final hash at the top layer provides the Merkle root:

$$h_{n,0} = h(h_{n-1,0}, h_{n-1,1}).$$

This root serves as the commitment for the FRI protocol's Merkle tree, thereby ensuring that all RS codewords are securely and efficiently committed. We use SHA3-256, which outputs 256-bit hashes, and allows an input size of up to 1320 bits, enabling efficient processing of codeword elements: 1024-bit inputs are used for layers $0 < k < r$, while 512-bit inputs are employed for layers beyond r.

Figure 1 shows the structure of our specialized Merkle tree constructed for the purpose of efficiently parallelizing the FRI protocol.

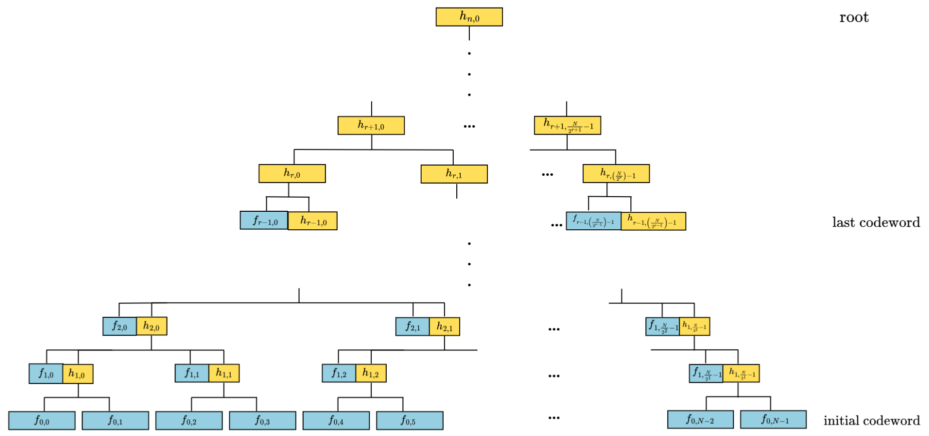


Fig. 1. Integrated FRI Merkle Tree

Time efficiency of this optimization. As compared to that of the CPU implementation (sequentially committing the codeword elements), the number of computations that are reduced is highlighted here. For simplicity, we denote $N = 2^n$, where n is the dimension of the first codeword $\boldsymbol{f}_0$. The number of computations for codewords looks like $\boldsymbol{f}_0 = 2^n, \boldsymbol{f}_1 = 2^{n-1}$, and so on. So, for the entire Integrated FRI Merkle Tree leading up to the root, the total number of sequential computations (including computing all codewords and Merkle hashes) is:

$$2^{n+1} - 1 - n$$

where the final subtraction accounts for structural hash reuse across FRI rounds.

If a separate Merkle tree is constructed for each codeword $\boldsymbol{f}_k$, then each such tree would require:

$$2^{n-k} \text{ codeword elements} + (2^{n-k-1} + \cdots + 1) \text{ hashes} = 2^{n-k+1} - 1 \text{ computations.}$$

Summing this across all n layers, the total number of sequential computations for the Merkle commitment model yields:

$$\sum_{k=0}^{n-1} (2^{n-k+1} - 1) = 2^{n+2} - 1 - n.$$

In contrast, the integrated Merkle tree design requires only $2^{n+1} - 1 - n$ computations, as it avoids redundant hashing at each round by committing to the FRI structure in a unified tree. Thus, the total number of operations saved is:

$$(2^{n+2} - 1 - n) - (2^{n+1} - 1 - n) = 2^{n+1}.$$

This shows that the integrated Merkle tree design improves efficiency by eliminating 2^{n+1} unnecessary operations, making it particularly well-suited for high-performance GPU-based implementations.

Verification and Authentication Paths To verify the authenticity of two neighboring points, starting with $s_0 \in L_0$ at layer 0 and extending to $s_k \in L_k$ for $0 < k < r$, the verifier uses authentication paths in the Merkle tree. These paths contain the concatenated hash values of the queried elements and their corresponding RS codewords. Notably, while the leaf layer hashes $\boldsymbol{f}_0$ separately, the intermediate layers $\boldsymbol{f}_k$ (for $1 \leq k < r$) do not require separate hashing, as their values are integrated into the hash tree structure.

The integration of RS codewords with Merkle tree commitments within the FRI protocol ensures both scalability and security. The recursive construction of the RS tree enables efficient polynomial evaluations, while the Merkle tree offers a compact and secure method for committing to the underlying codewords. We believe that these combined structures could be crucial for deploying zkSNARKs in practical applications, such as blockchain networks, where performance and security are of utmost importance.

4.3 Parallel Computation of Codeword Elements on the GPU

zkSNARK systems that have the FRI protocol as one of their core components have large parameter sets to conduct proofs, despite ironically being termed 'succinct'. Depending on the security requirement of the system, arithmetic is performed over finite fields of varying bit-widths. For instance, elements from different domains are represented as 192-bit, 256-bit, or 320-bit field elements, as noted in Preon's documentation [20]. Direct computation of these elements in a sequential fashion is both time- and memory-intensive. To address this, GPU parallelization is employed to distribute the computation of codeword elements across multiple threads, substantially accelerating proof generation.

A novel contribution of this paper is elucidated below. The FRI protocol reduces the codeword size recursively at each round. For a given round k, the next codeword element $f_{k+1}(L_{k+1}[i])$ is computed from two elements of the previous codeword, as described in (1). Each GPU thread computes one element of the next codeword in parallel, avoiding sequential bottlenecks. CUDA kernels assign threads to independently compute each codeword element.

4.4 Implementation and Performance Report

To support our proposed optimizations, we developed a modular and extensible implementation of the FRI protocol tailored for GPU acceleration. The codebase is designed to run both serially on the CPU and in parallel on the GPU, enabling comparative analysis and performance benchmarking. Key computational subroutines including codeword generation, line computations, and Merkle tree hashing are implemented in CUDA-C, while orchestration and memory management are handled in C.

The implementation carefully manages data transfer between host and device memory, pre-allocating buffers for each FRI round to ensure scalability across increasing security parameters. GPU kernels were written with thread-block synchronization to preserve the hierarchical structure of codeword indices across rounds. Where necessary, bit-width precision was customized to handle 256-bit and 320-bit field elements as required by zkSNARK parameter sets.

The modularity of the implementation allows seamless integration into zk-SNARK frameworks such as Polaris [24] or Aurora [11]. In the following section, we outline the software components, toolchains, and dependencies involved in building the system, while Section 6 evaluates the empirical performance under various cryptographic parameter settings.

5 Implementation

The architectural details of the protocol, on a system level, can be broken down into several key modules, each handling different aspects of the protocol.

5.1 Components

Figure 2 shows the overview of the software components that make the implementation. The tiles in green represent code that is borrowed from Preon's [20] implementation. The purple tiles show our code. The blue tile represents open-source library functions.

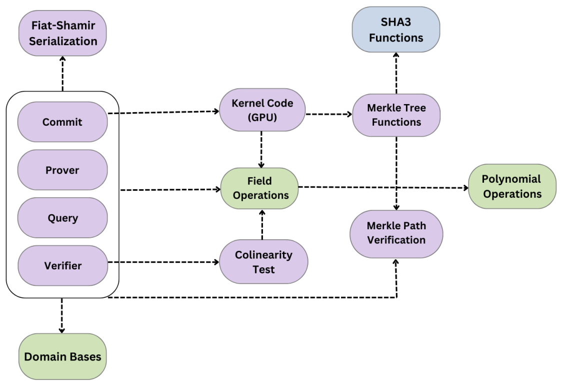


Fig. 2. Software Components of Our Implementation

Implementation Assumptions. The implementation of this FRI protocol is based on several key assumptions that ensure both functionality and performance in cryptographic settings. First and foremost, it is assumed that any entities or participants involved in using this protocol have access to GPU systems capable of executing CUDA-based applications. The use of CUDA-C allows the protocol to take full advantage of modern GPU architectures, enabling the parallel execution of tasks like polynomial evaluation, hashing, and consistency checks in the commit and verify phases. The CUDA toolkit and NVCC compiler are dependencies to compile and execute the code.

We use Keccak [33] (or more specifically, the SHA3-256 variant) to compute cryptographic commitments such as Merkle tree roots and authentication paths. The open-source implementation of Keccack's SHA3-256 is made GPU-compatible for use in our implementation. On the performance side, libcudart facilitates interaction between the code and the GPU, managing tasks such as memory allocation, kernel execution, and synchronization of GPU threads. In particular, it is responsible for handling CUDA-specific operations such as transferring data between the host (CPU) and device (GPU) and launching the parallel kernels that compute the polynomial evaluations and Merkle tree commitments.

Borrowing Preon's [20] Parameters. To make the protocol work, our code uses the FRI Domain parameters, finite field functions, and Polynomial evaluation functions as used in Preon's Optimised Implementation, which is available in the public domain. These functions were made compatible with the GPU nodes using CUDA-related function qualifiers.

5.2 Proof Stream and Verification

The proof stream serves as the main interaction channel between the prover and verifier in the protocol. The prover adds commitments, authentication paths, and related data into the stream, while the verifier retrieves this data during verification. Proper memory management is noted to be essential to ensure that all proof elements remain accessible and valid, particularly when dealing with dynamically generated proof paths. This guarantees that the verifier has all the information needed to validate the proof.

For non-interactiveness of the protocol, a custom serialization method inspired by the Fiat-Shamir heuristic is employed, enabling the prover to sequentially add objects such as codewords, Merkle roots, and sampled values to a global proof stream. The verifier processes these objects in the same order to verify correctness. However, since the serializer used in C does not track data types, the sequence of adding and retrieving objects must be precisely aligned.

5.3 Testing

We highlight and discuss the performance of our implementation. Mainly, the section aims to bring to light the various metrics for our performance testing, how much time the prover and verifier take for their respective execution, and a bit about the memory management that went into the code. We outline the test plan employed for evaluating the performance and correctness of the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) protocol here. The tests are divided into two main categories: unit tests to verify functional correctness and performance tests to assess execution times under various optimization levels. Description of unit tests are assumed to be self-evident and are omitted from this paper.

Performance Test Design The performance tests are divided into two categories: CPU-based tests and GPU-accelerated tests. Each test is run multiple times (up to 100 iterations) to ensure the consistency and reliability of the timing results.

1. **CPU-based tests:** The performance of the C-based FRI protocol is measured for baseline comparison. Initial tests use no optimizations, followed by tests with precomputed inverses, repeated hashes (Fiat-Shamir), and FRI Merkle tree implementation.
2. **GPU-accelerated tests:** Performance is evaluated on the GPU version of the code. The tests include precomputed inverses and parallel line computation, with the final implementation featuring FRI Merkle tree optimizations. These tests assess how GPU parallelism improves the runtime of each phase of the protocol.

Test Environment & Test Metrics.
The tests were conducted on a system with the following configuration. The system's CPU was a standard ARM processor with 8 CPU cores and 8GB of unified memory. For GPU acceleration, an NVIDIA A40 GPU was used, equipped with $10,752$ CUDA cores and 48GB of VRAM [40]. The time was measured in seconds, and the system clock was used for recording execution times. The performance evaluation of the FRI protocol focuses on execution times across different implementations. Key metrics include **prover time**, measuring the time taken for proof commitment, and **verifier time**, capturing proof validation tasks. To ensure reliability, the *average time* is calculated over multiple iterations, while the *minimum* and *maximum times* represent the best- and worst-case scenarios. Additionally, the *speedup factor* quantifies performance gains from GPU acceleration compared to CPU implementations. These metrics provide a comprehensive assessment of the protocol's performance.

6 Results

This section presents the results for various parameters and briefly compares them with existing implementations.

6.1 Performance of the Code Run with `L3` Parameters

The code for our implementation is majorly written for Preon's [20] `L3` parameters.
Parameter Mapping. Throughout this section, we instantiate the symbolic parameters used in earlier sections as follows: (1) $N = 2^{17}$ is the length of the initial codeword, corresponding to the size of the evaluation domain $L \subset \mathbb{F}_{2^{256}}$. (2) $n = \log_2 N = 17$ is the height of the Merkle tree used to commit to the initial codeword. (3) $k = 2^{12}$ is the degree bound of the polynomial being tested in the FRI protocol. (4) The field $\mathbb{F} = \mathbb{F}_{2^{256}}$ is used throughout the protocol's `L3` security implementation to represent codeword elements and evaluate polynomial values. (5) The codeword is reduced over $r = 12$ rounds of FRI folding, from size $N = 2^{17}$ down to 2^5, with each round halving the domain size. (6) An expansion factor of 32 is used, implying a rate $\rho = \frac{k}{N} = \frac{2^{12}}{2^{17}} = \frac{1}{32}$.

To verify the correctness of the protocol, 14 co-linearity tests are conducted during the process. The hash function used in the Merkle tree commitment processes inputs of 1024 bits and generates outputs of 256 bits.

Table 2 shows the progressive decrease in Average Time taken to run the protocol for the same parameters with each optimization implementation. Rows 1, 2, 3 show the times taken by the implementation in C, while 4 and 5 clearly show the advantages of our major and novel optimizations using the GPU. For the implementation of the protocol for the low-degree test of 4096, we measure the individual prover and verifier times over 10 iterations to account for additional overhead incurred outside the isolated prover and verifier functions, including memory transfers and proof stream serialization costs, the results of which are

Table 2. Performance Comparison of the FRI protocol on CPU vs GPU for `L3` Parameters

S.No.	Category	Iterations	Average (s)	Max (s)	Min (s)
1	C Code - No Optimizations	100	22.516272	24.487644	21.283454
2	C Code - Precomputed Inverses & Fiat-Shamir	100	17.949764	18.285129	17.674678
3	C Code - Precomputed Inverses, Non-interactiveness, & Verifiable FRI Merkle Tree	100	13.466491	14.387460	12.974454
4	GPU Code - Precomputed Inverses, Non-interactiveness, Parallel Line Computation	100	9.355615	10.225553	9.138932
5	**GPU Code - Parallel Line Computation & Verifiable FRI Merkle Tree**	**100**	**1.490716**	**1.626001**	**1.459887**

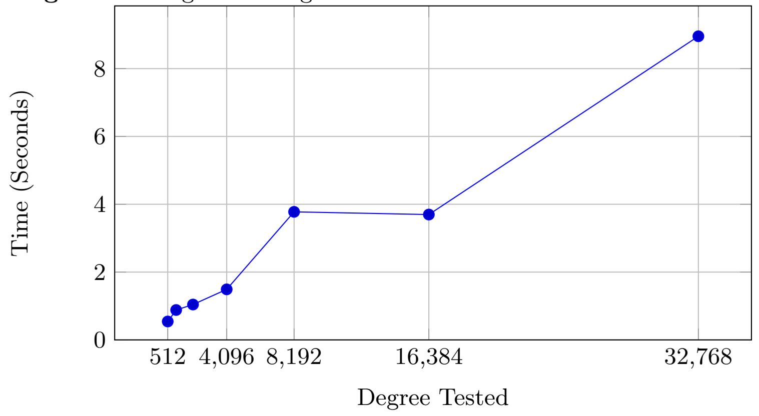
shown in Table 3. As such, the timings in Table 3 represent a lower bound on computational performance, whereas Table 2 captures end-to-end system-level costs.

Metric	Prover Time (s)	Verifier Time (s)
Minimum	0.66909	0.16506
Maximum	0.67360	0.16651
Average	0.67026	0.16586

Table 3. Minimum, Maximum, and Average Times for Prover and Verifier for degree 2^{12} over 10 iterations

The Figure 4 below shows the times for various degrees as tested on the GPU. One of the implications of an uneven curve could be the fact that time efficiency is also dependent on the number of threads and blocks that a GPU kernel is allocated before invocation. This is in-turn dependent on the length of the input. Effective scheduling and memory coalescing techniques can reduce bottlenecks in such GPU workloads by ensuring that memory access is streamlined across multiple threads, thus improving throughput for numerically-driven computations.

Fig. 3. FRI Degree Testing Time on the GPU with `L3` Parameters



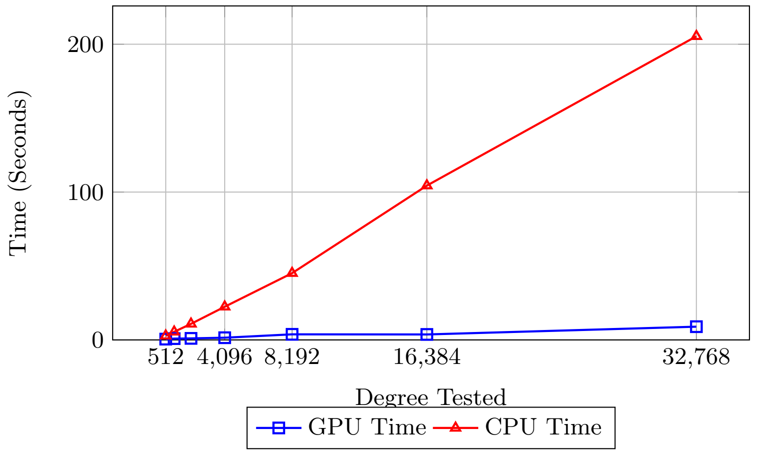


Fig. 4. CPU vs GPU Execution Times for FRI Degree Testing for `L3` Parameters

Based on the CPU and GPU times for the same degrees, we measure the percentage in performance efficiency termed as the 'Speedup Factor' (Table 4). Since the degree test for 2^{12} is the main focus of our implementation, we consider the speedup factor of this degree as the benchmark for our work.

Degree	**CPU Time (s)**	**GPU Time (s)**	**Speedup Factor (%)**
512	2.798574	0.543291	80.59%
1024	5.476513	0.881827	83.90%
2048	10.856021	1.043323	90.39%
4096	**22.425451**	**1.490716**	**93.35%**
8192	45.155425	3.776132	91.64%
16384	104.365670	3.696402	96.46%
32768	205.366330	8.953130	95.64%

Table 4. Speedup Factor (%) of GPU over CPU for Various Degrees for `L3` Parameters

6.2 Performance on `L5` Parameters

For `L5` security, Air-FRI's results show notably seminal performance despite large codeword lengths to begin the protocol with.

The parameters for the `L5` implementation of the FRI protocol are carefully chosen to ensure robust performance and security. The finite field used is $\mathbb{F}_{2^{320}}$, and the protocol tests polynomials with a degree of 2^{16}. An expansion factor of 32 is applied, with the initial codeword length set to 2^{21} $(2,097,152)$ and the least codeword length reduced to 2^5 (32) after successive rounds. The protocol undergoes 16 rounds of codeword reductions, with 10 co-linearity tests conducted to verify correctness. To secure commitments, a Merkle tree with a height of 21 is constructed, where each node involves a hash computation that takes 1152 bits as input and produces a 256-bit output.

Tested on the metrics mentioned above, Table 5 shows the prover and verifier times measured over 10 iterations on the GPU. As compared to `L3`'s results, it is shown that the proof generation takes only a few seconds more even when the codeword constraints are large. This shows significant promise in realizing our implementation in *post-quantum secure* algorithms which require such large constraints and strong security levels.

Metric	GPU Prov. (s)	GPU Verif. (s)	CPU Prov. (s)	CPU Verif. (s)
Minimum	13.85859	1.12372	~400	~1.9
Maximum	13.88316	1.12699		
Average	13.87129	1.12488		

Table 5. Minimum, Maximum, and Average Times for Prover and Verifier over 10 iterations using `L5` parameters ($\mathbb{F}_{2^{320}}$, degree 2^{16}). The estimated sequential CPU prover time was approximately 400 seconds, which is not practical at all.

6.3 Performance of Current FRI Implementations

To compare our test results with the existing FRI implementations, which are currently compatible to be run only on the CPU, we ran the pre-written tests of the libiop library [45], which Aurora [11], Ligero [2], and Fractal [21] use to run their entire protocol. The tests are generated with the library's build and metrics like the vector size and subgroup size **are much lesser** as compared to those tested for in our implementation. Table 6 shows the times for the three algorithms that use the libiop library for their low-degree test implementation. Further, to compare the entire performance, we also consider the Merkle tree test times for these algorithms.

Table 6. Performance Comparison of Existing FRI Implementations on the CPU

Implementation	Prover Time (s)	Verifier Time (s)	Vector Size	Subgroup Size
Aurora (libiop)	0.077	0.006	2^9	2^{12}
Fractal (libiop)	0.071	0.009	2^9	2^{12}
Ligero (libiop)	0.049	0.022	2^9	2^{12}

The Merkle tree test for libiop implementations recorded an execution time of 1.008 seconds, with verification requiring approximately 1 second per index in a single tree.

Note: Preon's [20] FRI implementation is tested for a large polynomial of degree 2^{12} with a subgroup size of 2^{17}, while the Aurora, Fractal, and Ligero implementations from the libiop library are tested for a smaller polynomial degree of 2^9

within a subgroup of 2^{12}. In scale, when the size of the extension field exponentially increases, the prover time approximately doubles (see 3). This is why, although Preon calls Aurora for its inner working, including for running the FRI Low-Degree Test, the time taken by Preon's code is much greater to account to the exponential growth in codeword sizes and to cater to parameters of larger extension basis. This is shown in Table 7 below.

These timings on the CPU shown in Table 7, when compared with our implementation the GPU, has significant performance improvements through our optimizations. This shows promise that our software-oriented solution for accelerating the FRI protocol can serve algorithmic efficiency when carefully deployed in those zkSNARK schemes which contain the FRI as one of their core components.

Metric	**Prover Time (seconds)**	**Verifier Time (seconds)**
Minimum	126.926	0.803
Maximum	139.683	1.167
Average	129.248	0.875

Table 7. Preon's Prover and Verifier Times for Testing of Degree 2^{12} over 50 Iterations on the CPU

7 Limitations

Our implementation does not account for software-induced vulnerabilities, such as susceptibility to side-channel attacks, which remains a limitation. Further, the proof system dynamically allocates memory in C to manage objects of varying sizes, but there is a risk of memory mismanagement if more memory is allocated than utilized. To enhance memory safety while leveraging GPU performance, Rust and Rustacuda [47] could be explored as alternatives. Additionally, the integrated Merkle tree implementation does not facilitate on-the-fly computation of authentication paths, requiring the prover to either store the Merkle tree data or access all codewords across FRI rounds during the query phase, increasing memory management complexity and security vulnerabilities.

8 Conclusions and Future Work

This paper presents novel and efficient optimizations of the FRI protocol for low-degree polynomial testing, implemented on the GPU. Implementation parameters, particularly the domain elements, are adapted from Preon's values [20]. Key optimizations include the pre-computation of domain base inverses to accelerate codeword generation and the sampling of a random element by the prover, which is repeatedly hashed by the verifier during co-linearity checks, enabling a non-interactive protocol. Further performance gains are achieved by parallelizing codeword computations and constructing a verifiable Merkle tree on GPU kernels using NVIDIA's CUDA-C, significantly reducing proof size and improving scalability. Compared to a CPU-based implementation, our GPU-based approach achieves a 93.3% efficiency gain for codeword computations in $\mathbb{F}_{2^{256}}$, reducing prover time from an average of 22 seconds to 1.49 seconds. For $\mathbb{F}_{2^{320}}$, corresponding to `L5` security parameters, the prover time is 13.89 seconds, only 11 seconds higher than for `L3` parameters, despite larger codewords and more reduction rounds. As expected, verifier times remain significantly lower than prover times, reflecting the protocol's efficiency-focused design. These results underscore the importance of optimization in cryptographic implementations, particularly for scalable systems. While this implementation was developed for integration with Polaris [24], it remains broadly applicable to any system utilizing the FRI protocol as a core component. Future work may explore additional performance refinements and broader deployment contexts.

Further Reduction of the Proof Size. For a finite field byte size of 256, the current implementation shares approximately 9,000 objects with the verifier via the proof stream, primarily for co-linearity tests across all rounds. Reducing this number could involve adjusting localization parameters and decreasing the number of co-linearity tests while maintaining protocol correctness.

Implementing Zero-Knowledge with this FRI Protocol. To integrate this code with any zkSNARK-based system, the zero-knowledge property can be achieved by masking the polynomial whose degree is being tested. At each reduction step of the FRI protocol, an additive masking polynomial can be applied to conceal the original polynomial. This ensures that the verifier learns nothing beyond the degree bound. The masking terms are designed to cancel during verification, maintaining proof integrity while preserving privacy.

Comparison of Energy Consumption between CPU and GPU Implementations of the Protocol. While GPUs improve protocol performance, it is important to assess their energy consumption to understand the ecological trade-offs. Uncoalesced global memory accesses are known to significantly impact energy use in NVIDIA GPUs [19], and nearly half of a 100A GPU's energy is consumed by a static kernel [22]. Future work should aim to quantify the energy differences between CPU and GPU implementations, balancing performance with sustainability.

Memory Profiling of the GPU Code. Since the protocol's performance relies heavily on shared memory between the host (CPU) and device (GPU kernel), a robust implementation should incorporate memory profiling results to make optimization decisions about thread and block usage, ensuring efficient GPU kernel launches across varying codeword lengths and Merkle tree computations. Specifically, a GPU-oriented memory profiling using nvprof or nsys profile [41] tools provided within the cuda-toolkit can be employed.

Integrating Air-FRI with Full-Fledged zkSNARKs and Blockchain Applications. An important direction for future work is the integration of Air-FRI into complete zkSNARK systems, particularly for scalable blockchain applications. This requires extending Air-FRI's GPU-accelerated commitment and verification phases to support the various models in zkSNARKs. Additionally, replacing the current FFT implementation (borrowed from Preon [20]) with a more efficient FFT algorithm such as one using specialized bases and subfield structures [29] could further accelerate polynomial evaluation and improve overall system performance.

Acknowledgement

This work is supported by MITACS-BTQ research grant.

References

1. Akins, B., Ng, J., Verdi, R.S.: The use of blockchain for public services. Journal of Public Economics (2013)
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 acm sigsac conference on computer and communications security. pp. 2087–2104 (2017)
3. Aslam, S., Tošić, A., Mrissa, M.: Secure and privacy-aware blockchain design: Requirements, challenges and solutions. Journal of Cybersecurity and Privacy **1**(1), 164–194 (2021). `https://doi.org/10.3390/jcp1010009`
4. Babai, L.: Trading group theory for randomness. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC). pp. 421–429 (1985)
5. Badakhshan, M., Luo, G., Jandhyala, T., Gong, G.: Ursa minor: The implementation framework for polaris. In: International Workshop on the Arithmetic of Finite Fields (WAIFI). Ottawa, Canada (June 2024), accepted for publication
6. Bellare, M., Goldwasser, S.: New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 194–211. Springer New York (1990)
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018), `https://doi.org/10.4230/LIPIcs.ICALP.2018.14`
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. In: Advances in Cryptology – CRYPTO 2018. pp. 201–234 (2018)
9. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018), `https://eprint.iacr.org/2018/046`
10. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014)
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38. pp. 103–128. Springer (2019)
12. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14. pp. 31–60. Springer (2016)
13. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 781–796. SEC'14, USENIX Association (2014)

14. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin p2p network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 15–29. CCS '14, Association for Computing Machinery (2014). https://doi.org/10.1145/2660267.2660379, https://doi.org/10.1145/2660267.2660379
15. Biryukov, A., Tikhomirov, S.: Deanonymization and linkability of cryptocurrency transactions based on network analysis. In: 2019 IEEE European symposium on security and privacy (EuroS&P). pp. 172–184. IEEE (2019)
16. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 326–349. Association for Computing Machinery (2012). https://doi.org/10.1145/2090236.2090263, https://doi.org/10.1145/2090236.2090263
17. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. p. 103–112. Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/62212.62222, https://doi.org/10.1145/62212.62222
18. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. Cryptology ePrint Archive (2020)
19. Chen, J., Li, B., Zhang, Y., Peng, L., Peir, J.k.: Statistical gpu power analysis using tree-based methods. In: 2011 International Green Computing Conference and Workshops. pp. 1–6 (2011). https://doi.org/10.1109/IGCC.2011.6008582
20. Chen, M.S., Chen, Y.S., Cheng, C.M., Fu, S., Hong, W.C., Hsiang, J.H., Hu, S.T., Kuo, P.C., Lee, W.B., Liu, F.H., et al.: Preon: zk-snark based signature scheme (2023)
21. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 769–793. Springer (2020)
22. Delestrac, P., Miquel, J., Bhattacharjee, D., Moolchandani, D., Catthoor, F., Torres, L., Novo, D.: Analyzing GPU Energy Consumption in Data Movement and Storage . In: 2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP). pp. 143–151. IEEE Computer Society (Jul 2024). https://doi.org/10.1109/ASAP61560.2024.00038, https://doi.ieeecomputersociety.org/10.1109/ASAP61560.2024.00038
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
24. Fu, S., Gong, G.: Polaris: transparent succinct zero-knowledge arguments for r1cs with efficient verifier. Proceedings on Privacy Enhancing Technologies (2022)
25. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity. Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS) pp. 174–187 (1986)
26. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology **7**(1), 1–32 (1994)
27. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012, https://doi.org/10.1137/0218012

28. Gong, G.: Fri. Unpublished note (December 2023), department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada.
29. Gong, G.: Uni/multi variate polynomial embeddings for zksnarks. Cryptography and Communications pp. 1–32 (2024)
30. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 321–340. Springer (2010)
31. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)
32. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N., et al.: Zcash protocol specification. GitHub: San Francisco, CA, USA **4**(220), 32 (2016)
33. Keccak Team: Keccak specifications summary (2024), `https://keccak.team/keccak_specs_summary.html`, accessed: November 11, 2024
34. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 839–858 (2016)
35. Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in bitcoin using p2p network traffic. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security. pp. 469–485. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
36. Levinin, A.: libiop-fp2: A c++ library for iop-based zksnarks over elliptic curves. `https://github.com/levanin/libiop-fp2` (2023), accessed: October 13, 2024
37. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. Elsevier (1977)
38. Micali, S.: Cs proofs (extended abstract). Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS) pp. 436–453 (1994)
39. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin **4**(2), 15 (2008), `https://bitcoin.org/bitcoin.pdf`
40. NVIDIA Corporation: NVIDIA A40 datasheet. `https://images.nvidia.com/content/Solutions/data-center/a40/nvidia-a40-datasheet.pdf` (2020), accessed: October 23, 2024
41. NVIDIA Corporation: Cuda profiler user's guide. `https://docs.nvidia.com/cuda/profiler-users-guide/` (2024), accessed: November 12, 2024
42. Panait, A.E., Olimid, R.F.: On using zk-snarks and zk-starks in blockchain-based identity management. In: Innovative Security Solutions for Information Technology and Communications: 13th International Conference, SecITC 2020, Bucharest, Romania, November 19–20, 2020, Revised Selected Papers 13. pp. 130–145. Springer (2021)
43. Peters, G.W., Panayi, E.: Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. Journal of Banking Regulation **16**(3-4), 217–233 (2015). `https://doi.org/10.1057/jbr.2015.19`
44. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Annual international cryptology conference. pp. 433–444. Springer (1991)
45. SCIPR Lab, contributors: libiop: A c++ library for iop-based zksnarks. `https://github.com/scipr-lab/libiop` (2017–2024), accessed: October 13, 2024

46. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550 (2019), `https://eprint.iacr.org/2019/550`
47. The RustaCUDA Developers: Rustacuda: A safe wrapper for the cuda driver api (2024), `https://docs.rs/rustacuda/latest/rustacuda/`, accessed: November 10, 2024
48. Zhang, R., Xue, R., Liu, L.: Security and privacy on blockchain. ACM Computing Surveys (CSUR) **52**(3), 1–34 (2019)