Dynark: Making Groth16 Dynamic

Tianyu Zhang Yupeng Ouyang Yupeng Zhang
University of Illinois Urbana Champaign
{tianyuz,oyjason,zhangyp}@illinois.edu

Abstract

In recent years, numerous new and more efficient constructions of zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) have been proposed, motivated by their growing practical applications. However, in most schemes, when the witness is changed, the prover has to recompute the proof from scratch even if the new witness is close to the old one. This is inefficient for applications where proofs are generated for dynamically changing witnesses with small changes.

In this paper, we introduce Dynark, a dynamic zkSNARK scheme that can update the proof in sublinear time when the change of the witness is small. Dynark is built on top of the seminal zkSNARK protocol of Groth, 2016. In the semi-dynamic setting where new witnesses deviate from a single original witness, for an R1CS of size n, after a preprocessing of $O(n \log n)$ group operations on the original witness, it only takes $O(d)$ group operations and $O(d \log^2 d)$ field operations to update the proof for a new witness with distance d from the original witness, which is nearly optimal. In the fully-dynamic setting where witnesses are updated sequentially, the update time of Dynark is $O(d\sqrt{n \log n})$ group operations and $O(d \log^2 d)$ field operations, where d is the distance between two consecutive witnesses. Both the proof size and the verifier time are $O(1)$, which are exactly the same as Groth16. Compared to the scheme in a prior work by Wang et al. 2024, we reduce the proof size from $O(\sqrt{n})$ to $O(1)$ without relying on pairing product arguments or another zkSNARK, and the update time and the verifier time of Dynark are faster in practice.

At the core of Dynark is a new algebraic technique that we call *quotient decomposition*. Building on it, our semi-dynamic zkSNARK combines a preprocessing of the original witness with a *sparse zkSNARK*: a Groth16 prover whose cost depends only on the number of non-zero entries in the witness. We then compile it into a fully-dynamic zkSNARK by splitting the preprocessing into small steps that run alongside the updates. Quotient decomposition also gives a preprocessing algorithm that is about $2\times$ faster than that of Feist and Khovratovich, and a Groth16 prover that needs only 4 DFTs instead of 6.

Experimental results show that for $n = 2^{20}$, after a one-time preprocessing of 74.3 seconds, it merely takes 3 milliseconds to update the proof in our semi-dynamic zkSNARK for $d = 1$, and 60 milliseconds to update the proof in our fully-dynamic zkSNARK. These are $1433\times$ and $73\times$ faster than Groth16, respectively. The proof size is 192 bytes and the verifier time is 4.4 milliseconds. The system is fully compatible with any existing deployment of Groth16 without changing the trusted setup, the proof and the verification algorithm.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 3
- 1.2 Technical Overview . . . 4
- 1.3 Related Work . . . 7
- 1.4 Open Problems . . . 8

2 Preliminaries **8**
- 2.1 Polynomials and Lagrange Basis . . . 9
- 2.2 zkSNARK . . . 10

3 Quotient Decomposition **12**

4 Semi-Dynamic zkSNARK **13**
- 4.1 Definition . . . 13
- 4.2 Semi-Dynamic zkSNARK based on Groth16 . . . 15
- 4.3 Cross Term . . . 17
- 4.4 Projection Terms . . . 19
- 4.5 Putting Everything Together . . . 20

5 Fully-Dynamic zkSNARK **22**
- 5.1 Definition . . . 22
- 5.2 Fully-Dynamic zkSNARK based on Groth16 . . . 23

6 Evaluations **27**
- 6.1 Performance of Dynark and Comparison with Gorth16 . . . 27
- 6.2 Comparison with Prior Work . . . 30

A Faster Groth16 Prover **34**

B Sparse Groth16 Prover **35**

1 Introduction

A zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) enables a prover to convince a verifier that an NP statement is true, without revealing any information about the secret witness. zkSNARKs have seen prominent developments in recent years and have found critical applications in many domains. It has been used to improve the privacy and scalability of blockchains and cryptocurrencies [SCG+14, vit]. It has also been applied to prove machine learning computations [ZFZS20, LXZ21, WYX+21, GGJ+23, SLZ24, APKP24], photo authenticity [NT16, DCB25], internet data compliance [GAZ+22] and even the execution of any programs in zero-knowledge virtual machines (zkVMs) [RIS23]. Partly driven by these applications, the efficiency of zkSNARK schemes—especially prover time—has improved by orders of magnitude over the past decade.

However, a major limitation of almost all zkSNARKs is that they are static. To generate a new proof for an updated witness, the prover has to re-execute the proof generation algorithm, and the prover time is the same as generating the proof from scratch, even if only a single element of the witness is changed. This is undesirable in many applications where the witness in a new proof often differs only slightly from the old one. For example, in blockchain applications such as zkBridges [XZC+22] and zkRollups [vit], zkSNARK is used to prove the correctness of off-chain computations and proofs are verified by smart contracts on-chain. Some computations involve the balances of all accounts of a cryptocurrency, and only a small number of accounts changes when a new proof is generated. In machine learning applications such as proving the output of a large language model [SLZ24, QSL+25], the next token is generated by running the model on the prompt together with all previously produced tokens. The computations on the prompt and earlier tokens remain unchanged, and are reused for new tokens and new prompts.

To overcome this limitation, Wang et al. [WPSP25b] introduced the notion of *dynamic zkSNARKs*, where the time to update the proof for a new witness is sublinear in the size of the statement when the distance between the new witness and the old witness is small. They also proposed a new zkSNARK scheme that is dynamic. However, in the basic version of the protocol, the proof size is square-root in the size of the statement. The proof size can be further reduced by composing the zkSNARK with another zkSNARK scheme, but this leads to a high overhead on the concrete efficiency of the scheme. See Section 1.3 for a more detailed explanation.

1.1 Our Contributions

In this paper, we propose an efficient dynamic zkSNARK scheme, Dynark, directly on top of Groth16 [Gro16], one of the most widely used zkSNARKs in practice with extremely small proof size. In our scheme, the proof, the verification algorithm, and the prover key generated by the trusted setup remain exactly the same as Groth16. We view this as an important advantage of our scheme, as it is fully compatible with all existing adoptions of Groth16 in practice. Our main results are summarized below:

- First, we define and build a *semi-dynamic* zkSNARK scheme, where there is one fixed *original witness* and its proof. After a one-time preprocessing of complexity $O(n \log n)$ group operations on the original witness of size n, it only takes $O(d)$ group operations and $O(d \log^2 d)$ field operations to generate a new proof, where d is the distance between the new witness and the original witness. Note that the number of group operations in our scheme is asymptotically optimal. The proof size and the verifier time are both $O(1)$, and are identical to those in Groth16 as mentioned above. Our main tool is *quotient decomposition* (Section 3), a closed form of the quotient of a product of two Lagrange polynomials by the vanishing polynomial. Building on it, our semi-dynamic zkSNARK combines a preprocessing of the original witness with the prover of a *sparse zkSNARK*, run on the difference between the new and the original witness. The sparse zkSNARK is a Groth16 prover whose running time depends only on the

number of non-zero entries in the witness, and may be of independent interest (Appendix B). Quotient decomposition also gives a new algorithm to generate the Groth16 proof with 4 DFTs of size n, while the best known algorithms and implementations have at least 6 DFTs (Appendix A).

We stress that semi-dynamic zkSNARKs are also useful in many applications where the updated witnesses always remain close to the original witness, regardless of the number of updates. For example, when proving computations of different fine-tuned machine learning models from an open-source large language model, the fine-tuned models are always close to the original model.

- Second, we lift our semi-dynamic zkSNARK to a *fully-dynamic* zkSNARK similar to that defined in [WPSP25b]. In this setting, the witness can keep changing and d denotes the distance between two consecutive versions of the witness. We obtain a fully-dynamic zkSNARK scheme with an update time of $O(d\sqrt{n \log n})$ group operations and $O(d \log^2 d)$ field operations, without any amortization. The proof size and the verifier time remain $O(1)$. Our approach provides a compiler to lift any semi-dynamic zkSNARK to a fully-dynamic zkSNARK as long as the semi-dynamic zkSNARK satisfies some additional properties. Our technique can be applied to other works such as the lookup argument in [DGP+24] to make them dynamic in the de-amortized setting.

- Last, we have fully implemented Dynark and benchmarked its performance. For $n = 2^{20}$, with a one-time preprocessing, it merely takes 3 milliseconds to update a proof with $d = 1$ in our semi-dynamic zkSNARK, and 60 milliseconds in our fully-dynamic zkSNARK. Theses are 1433× and 73× faster than recomputing the proof from scratch in Groth16. The update time of the semi-dynamic zkSNARK is faster than Groth16 for distance up to $d = 2^{15}$, while the fully-dynamic zkSNARK achieves faster updates for distance up to $d = 2^8$. The code of our implementation is available at `https://github.com/tnyuzg/dynark`.

1.2 Technical Overview

As observed in [WPSP25b], it is very challenging to build dynamic zkSNARKs from schemes relying on the Fiat-Shamir transformation [FS87]. This is because the change of the witness leads to the change of the transcript that includes the commitment of the witness. Therefore, the randomness derived from the Fiat-Shamir transformation becomes completely different, and the prover has to re-execute the prover algorithm from scratch. This observation rules out many recent zkSNARKs, such as those based on multivariate sumchecks, interactive oracle proofs and error-correcting codes.

In the remaining candidate zkSNARKs based on univariate polynomials, the most challenging step to make dynamic is of the form

$$q(X) = \frac{A(X) \cdot B(X) - C(X)}{v_{\mathbb{H}}(X)},$$

where roughly speaking $A(X)$ is defined by the "left" input of multiplication gates, $B(X)$ is defined by the "right" input, $C(X)$ is defined by the output, $v_{\mathbb{H}}(X)$ is the vanishing polynomial and $q(X)$ is the quotient polynomial of the division (see Section 2.1 for formal definitions). The proof consists of the term $[q(\tau)]_1$, denoting the generator of a group raised to the power of $q(\tau)$, the evaluation of $q(X)$ at a secret point τ. This computation appears in Plonk [GWC19], Marlin [CHM+20], as well as the dynamic zkSNARK in [WPSP25b]. As for the Groth16 [Gro16], the computation is slightly different, as $A(X), B(X), C(X)$ are computed by the R1CS matrices, and the prover key and the proof are different, but the main structure of the computation remains the same. See Section 2.2 for the formal description of Groth16.

As the degree of $C(X)$ is less than $v_{\mathbb{H}}(X)$, this term contributes only to the remainder in the division by $v_{\mathbb{H}}(X)$ and does not affect the quotient polynomial $q(X)$, so we remove it from the equation to simplify the explanation. When the witness changes, the polynomials become $A'(X) = A(X) + \Delta^A(X)$ and $B'(X) = B(X) + \Delta^B(X)$. We express the difference polynomials using the Lagrange polynomials $\ell_i(X)$ as $\Delta^A(X) = \sum_{i \in I_A} \Delta_i^A \ell_i(X)$ and $\Delta^B(X) = \sum_{i \in I_B} \Delta_i^B \ell_i(X)$ where I_A, I_B are indices of non-zero elements in Δ_A, Δ_B respectively. We assume $|I_A| = O(d), |I_B| = O(d)$ for $d << n$ when the update on the witness is small. To generate the new proof, the prover needs to compute $[q'(\tau)]_1$ where $q'(X)$ is the quotient polynomial of the polynomial division[1]:

$$\frac{A'(X) \cdot B'(X)}{v_{\mathbb{H}}(X)} = \frac{(A(X) + \Delta^A(X)) \cdot (B(X) + \Delta^B(X))}{v_{\mathbb{H}}(X)}$$
$$= \frac{A(X) \cdot B(X)}{v_{\mathbb{H}}(X)} + \frac{\Delta^A(X) \cdot B(X)}{v_{\mathbb{H}}(X)} + \frac{A(X) \cdot \Delta^B(X)}{v_{\mathbb{H}}(X)} + \frac{\Delta^A(X) \cdot \Delta^B(X)}{v_{\mathbb{H}}(X)}.$$

The quotient polynomial of the first term is exactly $q(X)$ and $[q(\tau)]_1$ is already computed in the original proof. It remains challenging to compute the quotient polynomials of the last three terms in time proportional only to d.

Projection term. We first show how to efficiently compute the second and third terms, which we name as the projection terms. We observe that $\frac{\Delta^A(X) \cdot B(X)}{v_{\mathbb{H}}(X)}$ is of exactly the same format as the lookup argument proposed in cq [EFG22]. In particular,

$$\frac{\Delta^A(X) \cdot B(X)}{v_{\mathbb{H}}(X)} = \sum\nolimits_{i \in I_A} \Delta_i^A \frac{\ell_i(X) \cdot B(X)}{v_{\mathbb{H}}(X)}$$

and the group element $[\frac{\ell_i(\tau) \cdot B(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ can be precomputed for all $i \in [n]$ in $O(n \log n)$ group operations using the algorithm in [FK23]. With the preprocessing, all the $[\frac{\Delta^A(\tau) \cdot B(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ can be computed by $|I_A| = O(d)$ group operations. The same approach works for $\frac{A(X) \cdot \Delta^B(X)}{v_{\mathbb{H}}(X)}$.

The full protocol is more complicated and in Section 4.4, we show how to compute the projection terms in Groth16 without changing the prover key generated by the trusted setup. We also propose a better preprocessing algorithm that improves the running time by 2× compared to [FK23].

Cross term. The remaining challenge is the last term named as the cross term:

$$\frac{\Delta^A(X) \cdot \Delta^B(X)}{v_{\mathbb{H}}(X)} = \sum\nolimits_{i \in I_A, j \in I_B} \Delta_i^A \Delta_j^B \frac{\ell_i(X)\ell_j(X)}{v_{\mathbb{H}}(X)}.$$

Here we cannot precompute $[\frac{\ell_i(\tau)\ell_j(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ for all i, j since it would incur an overhead of $O(n^2)$ group operations, which is not practical. Fortunately, we show that $\frac{\ell_i(X)\ell_j(X)}{v_{\mathbb{H}}(X)}$ itself is sparse. That is, when $i \neq j$, the polynomial is of degree $n-2$ and is non-zero only at i and j. As shown in Theorem 3.1, its closed-form is

$$\frac{\ell_i(X)\ell_j(X)}{v_{\mathbb{H}}(X)} = \frac{\omega^{j-i}}{n(1-\omega^{j-i})}\ell_i(X) + \frac{\omega^{i-j}}{n(1-\omega^{i-j})}\ell_j(X).$$

Therefore, with $[\ell_i(\tau)]_1$ in the public parameters for all $i \in [n]$, $[\frac{\ell_i(\tau)\ell_j(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ can be computed in $O(1)$ time. To compute the cross term in the proof, note that the group elements with the same $[\ell_i(\tau)]_1$ can be grouped together, making the complexity $O(d)$ group operations. Computing

[1] Here we abuse the notation to mean the quotient polynomial of polynomial division with a remainder polynomial that is not shown in the equation.

the constants in the equation for all $i \in I_A, j \in I_B$ takes $O(d^2)$ field operations. This can further be reduced to $O(d \log^2 d)$ using an algorithm for sparse convolutions. In Section 4.3, we further show the approach to precompute the terms with $i = j$ in a unified framework with the projection terms above. This technique can also be used to construct a *sparse zkSNARK* on top of Groth16 where the prover time is only $O(m \log^2 m)$ field operations and $O(m)$ group operations for a witness with m non-zero elements. This may be of independent interest, see Appendix B.

Combining the two protocols above, we can update the quotient polynomial in the proof efficiently. All other elements in the proof are linear and can be updated efficiently in $O(d)$ time in the first place. This gives us the semi-dynamic zkSNARK scheme.

From semi-dynamic to fully-dynamic. The reason why the scheme above is not fully dynamic is because of the preprocessing to compute $[\frac{\ell_i(\tau) \cdot B(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ and $[\frac{A(\tau) \cdot \ell_i(\tau)}{v_{\mathbb{H}}(\tau)}]_1$ for $i \in [n]$, which depends on the original witness. The prover cannot rerun the preprocessing for every new witness, which would take $O(n \log n)$ time for every update. The prover could rerun the preprocessing only when the distance between the new witness and the original witness reaches $d = O(\sqrt{n \log n})$, making the update time $O(d\sqrt{n \log n})$ in the amortized setting. Instead of this approach, we propose a new algorithm that de-amortizes the cost and yields a fully-dynamic zkSNARK.

The algorithm is inspired by the dynamic algorithm for convolutions in [RT97]. Let $z = (\mathbb{x}, \mathbb{w})$ be the extended witness of an R1CS. For a sequence of extended witnesses $(z_0, z_1, z_2, z_3 \ldots)$, we divide them into blocks $(B_0, B_1, B_2, \ldots)$ such that the distance between the first witness in a block and the first witness in its previous block is no more than b. We can safely assume that the distance of any single update is no more than $O(\sqrt{n \log n})$, as otherwise it would be more efficient to just rerun the prover algorithm from scratch to generate the proof in our scheme.

In our fully-dynamic construction, we initialize the system by running the semi-dynamic preprocessing once on z_0. For any witness in the first block B_0, we then reuse our semi-dynamic scheme (with preprocessing anchored at z_0) to update the proof, with the update time bounded by $O(b)$ group operations and $O(b \log^2 b)$ field operations. Starting from witnesses in B_1, the prover is going to perform the following two computations:

1. Generate the new proofs still using the preprocessed data of z_0 and our semi-dynamic zkSNARK.

2. Run the preprocessing for the first witness in B_1.

Since the distance within one block is bounded by b, the distance between witnesses in B_1 and z_0 is no more than $2b$, thus the first item to generate the proof can still be computed by $O(b)$ group operations and $O(b \log^2 b)$ field operations. For the second item, we further require that the preprocessing can be computed step-by-step "smoothly". Our preprocessing satisfies this property using the classical butterfly circuit for DFTs and convolutions, see Section 5.2 for the formal protocols. With this property, we evenly divide the computation of the preprocessing into $O(\frac{n \log n}{b})$ per unit distance, and assign $O(\frac{dn \log n}{b})$ of computation to each witness in B_1, where d is used to denote its distance from the previous witness. At the end of B_1, the preprocessing is completed for the first witness in B_1. The preprocessed data of the first witness in B_1 can then be used to generate the proofs for all witnesses in B_2.

The same process is repeated where for any block B_i, the prover has the preprocessed data for the first witness in B_{i-1} ready and generates the proofs using our semi-dynamic zkSNARK. At the same time, the prover runs the preprocessing for the first witness in B_i step by step. By setting $b = O(\sqrt{n \log n})$, we obtain a fully-dynamic zkSNARK scheme with the update time of $O(d\sqrt{n \log n})$ group operations and $O(\sqrt{n \log n} \log^2 n)$ field operations without any amortization. In Section 5.2, we further show how to improve the number of field operations to $O(d \log^2 d)$ utilizing the "locality" property of convolutions. Our approach provides a compiler to lift any

scheme	keygen time	prover time	update time	proof size
[Gro16]	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
[WPSP25b] (Basic)	$O(n)$	$O(n \log n)$	$O(d\sqrt{n} \log n)$	$O(\sqrt{n})$
[WPSP25b] (Dynaverse)	$O(n)$	$O(n \log n)$	$O(d\sqrt{n} \log n)$	$O(\log n)$
[WPSP25b] (Dynalog)	$O(n)$	$O(n \log^2 n)$	$O(d \log^3 n)$	$O(\log n)$
Ours–Semi	$O(n \log n)$	$O(n \log n)$	$O(d)\mathbb{G} + O(d \log^2 d)\mathbb{F}$	$O(1)$
Ours–Fully	$O(n \log n)$	$O(n \log n)$	$O(d\sqrt{n \log n})\mathbb{G} + O(d \log^2 d)\mathbb{F}$	$O(1)$

Table 1: Comparison with Groth16 and [WPSP25b].

semi-dynamic zkSNARK to a fully-dynamic zkSNARK as long as the preprocessing can be computed step by step.

1.3 Related Work

The most relevant prior work is by Wang et al. in [WPSP25b], which initiated the study of dynamic zkSNARKs. They proposed a new permutation argument with a constant proof size that can be updated in time $O(d)$. Unfortunately, in order to handle the multiplication gates, they utilized a bucketing technique that divides the witness into $\sqrt{n}$ buckets of size $\sqrt{n}$, leading to a proof size of $O(\sqrt{n})$. We call this version the basic version and Table 1 shows its comparison with our schemes. The major improvement of our scheme over [WPSP25b] is that the proof size is constant, and is exactly the same as Groth16 with full compatibility. Moreover, in the semi-dynamic setting, our update time is much faster and does not depend on n at all. As we show in Section 6, our prover time, proof size and verifier time are all better than [WPSP25b] in the experiments. The scheme in [WPSP25b] can work with a universal trusted setup, but the running time and the prover key size would become $O(n\sqrt{n})$. In the latest version, the authors proposed an approach to reduce them to $O(n)$, at the cost of relying on the Fiat-Shamir transformation in the random oracle model. The complexity in the table refers to the variant with a per-circuit trusted setup, which is in the same setting as our schemes. In fact, the techniques proposed in our paper can be applied to improve the protocol for checking multiplication gates in [WPSP25b] and reduce its proof size to $O(1)$.

The authors of [WPSP25b] also proposed several variants on top of the basic scheme with better asymptotics. However, all of them utilize GIPA [BMM+21] or another zkSNARK such as Groth16. For example, in one variant named Dynaverse, the proof size and the verifier time are reduced to $O(\log n)$ or $O(1)$ by checking the pairing equations of the basic version using GIPA or Groth16. In another variant named Dynalog, an hierarchical data structure is introduced to reduce the update time to $O(d \log^3 n)$, and the proof size and the verifier time to $O(\log n)$, while increasing the prover time to $O(n \log^2 n)$. Again, it uses Groth16 to prove the correctness of the data structure and the verification of a vector commitment. In these variants, the verification of the inner SNARK involving group operations needs to be implemented in the outer SNARK, incurring a high overhead in the concrete efficiency. They were not implemented in [WPSP25b] and the performance is not clear in practice.

Finally, as a folklore construction mentioned in [WPSP25b], one can build a dynamic zkSNARK using a recursive SNARK over a tree structure (proof-carrying-data). The recursive SNARK of this approach needs to instantiate the random oracle in the SNARK with heuristic security, and the concrete efficiency is not clear in practice either.

In addition to dynamic zkSNARKs, other related works have studied related notions of updatability. The authors of [GGK+25] proposed *split-prover* zkSNARKs, which allow a prover to delegate part of the proof computation to a second prover when the full witness is known later. Our techniques can be used to improve the efficiency of their second prover when the

R1CS is sparse. Updatable lookup arguments were studied in [DGP+24] and updatable KZG vector commitments were studied in [TAB+20, LJGK25]. Notably, [LJGK25] re-casted the preprocessing of [EFG22] as Cauchy-matrix multiplications and obtained the same improvement as our quotient-decomposition approach to speed up batched updates.

Since the early constructions by Kilian [Kil92] and Micali [Mic00], there have been numerous efficient constructions of zkSNARKs in recent years based on various techniques. There are zkSNARKs based on bilinear pairing [PHGR13, Gro16, GWC19], discrete logarithm [BBB+18, BMM+21, BFS20], interactive proofs and sumcheck [WTS+18, XZZ+19, Set20, CBBZ22], interactive oracle proofs [BSCR+19, BSBHR18, ZCF24], error-correcting codes [AHIV17, GLS+23, XZS22, BFK+24] and lattices [BC24]. See [Tha22] for a survey on all recent zkSNARKs. Unfortunately, as explained above, it is very challenging to turn those zkSNARKs with the Fiat-Shamir transformation into dynamic ones. This leaves us with only a few candidates, and Groth16 stands out as one of the most widely used zkSNARKs with extremely small proof size.

Concurrent work. In a new version of [WPSP25c], the authors proposed a new scheme that is different from those in prior versions. The scheme has $O(1)$ proof size without relying on GIPA or Groth16. The construction is essentially a semi-dynamic zkSNARK as defined in our paper, and is made fully-dynamic in the amortized setting. Compared to this protocol, our schemes are more efficient in practice and are fully compatible with Groth16, while the scheme in [WPSP25c] only needs a universal trusted setup.

1.4 Open Problems

Our fully-dynamic zkSNARK has an update time that is square-root in the size of the witness. It remains open to construct a dynamic zkSNARK with a poly-logarithmic update time, without relying on recursive SNARKs. It is particularly challenging to improve our construction, because the preprocessing involves convolutions and there are known lower bounds (square-root in the size of vectors) on dynamic algorithms for convolutions [FHM01]. Dynamic zkSNARKs based on multivariate polynomials could be a potential direction, which is left as interesting future work.

2 Preliminaries

Notations. We use λ to denote the security parameter. $\mathbb{F}$ denotes a finite field of prime order p. PPT stands for probabilistic polynomial time. $\mathsf{negl}(\lambda)$ denotes a negligible function in λ. Let $[n]$ denote the set $\{0, 1, \ldots, n-1\}$ and $[k, n]$ denote the set $\{k, k+1, \ldots, n-1\}$. We use bold letters to denote vectors such as $\mathbf{a}$ or use parentheses notation, such as $(a_i)_{i=0}^{n-1} := (a_0, a_1, \ldots, a_{n-1})$ to explicitly express each element. When $\mathbf{a}$ is sparse, we use $|\mathbf{a}|$ to denote the number of non-zero elements in vector $\mathbf{a}$ and use $I_{\mathbf{a}}$ to denote the set of indices for these non-zero elements. We use lowercase letters to denote elementary polynomials such as $\ell(X)$, uppercase letters to denote synthetic polynomials such as $A(X) = \sum_{i\in[n]} a_i \ell_i(X)$. Let $\circ$ denote the component-wise vector multiplication.

Bilinear pairing. $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h) \leftarrow \mathsf{BilGen}(1^\lambda)$ denotes the generation of a bilinear map where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, and g, h are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable and non-degenerate pairing and $e(g, h)$ is a generator of $\mathbb{G}_T$. Following the notation of [Gro16], we write $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ additively. We write $[a]_1 := a \cdot g$, $[a]_2 := a \cdot h$ and $[a]_T := a \cdot e(g, h)$. With this notation, $g = [1]_1, h = [1]_2$. For all $[a]_1 \in \mathbb{G}_1, [b]_2 \in \mathbb{G}_2$, $[a]_1 \cdot [b]_2 := e([a]_1, [b]_2) = [a \cdot b]_T$. We use $[\mathbf{a}]_1$ to denote the vector of group elements in $\mathbb{G}_1$ by computing $([a_i]_1)_{i=0}^{n-1}$. The same notation applies to $\mathbb{G}_2$ and $\mathbb{G}_T$.

2.1 Polynomials and Lagrange Basis

Assume n is a power of two and $n \mid (p-1)$. Then there exists $\omega \in \mathbb{F}$ such that $\omega^n = 1_{\mathbb{F}}$ and $\omega^j \neq 1_{\mathbb{F}}$ for all $0 < j < n$, which is called a primitive n-th root of unity. Let $\mathbb{H} = \{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$ be the multiplicative subgroup generated by ω.

DFT and iDFT. The discrete Fourier transformation (DFT) of length n is defined as $\widehat{\mathbf{a}} = \mathrm{DFT}(\mathbf{a}) = F \cdot \mathbf{a}$ where F is the Fourier matrix defined as $\left(\omega^{ij}\right)_{(i,j)\in[n]^2} \in \mathbb{F}^{n\times n}$. In this paper, we use two variants of the DFT:

- Field DFT : $\mathbb{F}^n \to \mathbb{F}^n$, where both inputs and outputs are field elements. Computation involves addition and multiplication between $\mathbb{F}$.
- Group DFT : $\mathbb{G}^n \to \mathbb{G}^n$, where both inputs and outputs are group elements. Computation involves group addition between $\mathbb{G}$ and scalar multiplication by elements of $\mathbb{F}$.

For simplicity, we use the term "DFT" to refer to both types when the distinction is clear from context. The inverse discrete Fourier transformation of n is defined as $\mathbf{a} = \mathrm{iDFT}(\widehat{\mathbf{a}}) = F^{-1} \cdot \widehat{\mathbf{a}}$ where matrix F^{-1} is defined as $\left(\frac{1}{n}\omega^{-ij}\right)_{(i,j)\in[n]^2}$. The Fast Fourier Transform (FFT) algorithm can compute both $\mathrm{DFT}(\cdot)$ and $\mathrm{iDFT}(\cdot)$ in $O(n \log n)$ time exploiting the symmetric and recursive structure of F and F^{-1}, whereas a naive matrix-vector multiplication takes $O(n^2)$ time.

Vanishing Polynomial. The vanishing polynomial $v_{\mathbb{H}}(X)$ of $\mathbb{H}$ is defined as $v_{\mathbb{H}}(X) := \prod_{i\in[n]}(X - \omega^i) = X^n - 1$, such that $v_{\mathbb{H}}(h) = 0$ for all $h \in \mathbb{H}$.

Lagrange Basis. The Lagrange basis over $\mathbb{H}$ consists of polynomials $\{\ell_i(X)\}_{i\in[n]}$ of degree $n-1$ where $\ell_i(\omega^j) = 1$ if $i = j$, and $\ell_i(\omega^j) = 0$ otherwise, for all $i, j \in [n]$. The closed form of $\ell_i(X)$ is

$$\ell_i(X) = \prod_{j\in[n]/i}\left(\frac{X-\omega^j}{\omega^i-\omega^j}\right) = \frac{\omega^i}{n} \cdot \prod_{j\in[n]/i}(X-\omega^j) = \frac{\omega^i}{n} \cdot \frac{X^n-1}{X-\omega^i}.$$

Note that $\sum_{i\in[n]} \ell_i(X) = 1$, as both sides have degree less than n and agree on $\mathbb{H}$. The coefficient form of the Lagrange basis is an inverse DFT of the monomials:

Lemma 2.1 (Lagrange Basis in Coefficient Form)**.** *For all $i \in [n]$, $\ell_i(X) = \frac{1}{n}\sum_{j\in[n]} \omega^{-ij} X^j$; that is, $(\ell_i(X))_{i=0}^{n-1} = iDFT\big((X^j)_{j=0}^{n-1}\big)$.*

Proof. Since $(\omega^i)^n = 1$, we have $X^n - 1 = (X - \omega^i)\sum_{j\in[n]} \omega^{ij} X^{n-1-j}$, and therefore

$$\ell_i(X) = \frac{\omega^i}{n} \cdot \frac{X^n-1}{X-\omega^i} = \frac{1}{n}\sum_{j\in[n]} \omega^{i(j+1)} X^{n-1-j} = \frac{1}{n}\sum_{j\in[n]} \omega^{-ij} X^j,$$

where the last step substitutes $j \to n-1-j$. The coefficient $\frac{1}{n}\omega^{-ij}$ of X^j is the (i,j) entry of F^{-1}. □

A degree $n-1$ polynomial $f(X) \in \mathbb{F}^{<n}[X]$ can be expressed as its coefficient form $f(X) = \sum_{i\in[n]} f_i X^i$ or as its evaluation form $f(X) = \sum_{i\in[n]} f(\omega^i)\ell_i(X)$. Since $\left(f(\omega^i)\right)_{i=0}^{n-1} = F \cdot (f_i)_{i=0}^{n-1} = \mathrm{DFT}((f_i)_{i=0}^{n-1})$, it is efficient to convert polynomials between the coefficient form and the evaluation form in $O(n \log n)$ time using $\mathrm{DFT}(\cdot)$ and $\mathrm{iDFT}(\cdot)$.

Circular Convolution. The circular convolution of two vectors $\mathbf{a} * \mathbf{b} = \mathbf{c}$ of size n is defined as $\mathbf{c}[i] = \sum_{j\in[n]} a[j] \cdot b[i-j \bmod n]$ for each $i \in [n]$. This operation corresponds to multiplying two degree $n-1$ polynomials $a(X), b(X)$ with coefficients defined by vectors $\mathbf{a}, \mathbf{b}$ modulo $X^n - 1$, i.e., $c(X) = a(X) \cdot b(X) \mod (X^n - 1)$. The coefficient vector $\mathbf{c}$ of $c(X)$ can be efficiently computed as $\mathbf{c} = \mathrm{iDFT}(\mathrm{DFT}(\mathbf{a}) \circ \mathrm{DFT}(\mathbf{b}))$ in time $O(n \log n)$. Similar to the DFT, we also have two variants of circular convolution:

- Field Convolution : $\mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}^n$, where both the inputs and the output are field elements. Computation involves addition and multiplication between $\mathbb{F}$.
- Group Convolution : $\mathbb{F}^n \times \mathbb{G}^n \to \mathbb{G}^n$, where the first input is field elements, while the second input and the output are group elements. Computation involves group addition between $\mathbb{G}$ and scalar multiplication by elements of $\mathbb{F}$.

2.2 zkSNARK

Indexed Relation. An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance and $\mathtt{w}$ is the witness. The corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ consists of all pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable arithmetic circuits consists of triples where $\mathtt{i}$ is the description of an arithmetic circuit, $\mathtt{x}$ is a partial assignment to its input wires, and $\mathtt{w}$ is an assignment to the remaining input wires that makes the circuit output 1.

Definition 2.2 (zkSNARK)**.** *A zero-knowledge succinct non-interactive argument of knowledge (zkSNARK)* Π *for an indexed relation* $\mathcal{R}$ *is a triple of PPT algorithms* $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$*:*

- $\mathcal{G}(1^\lambda, \mathtt{i}) \to (\mathbf{pk}, \mathbf{vk})$*: On* 1^λ *and index* $\mathtt{i}$*, outputs prover key* $\mathbf{pk}$ *and verifier key* $\mathbf{vk}$*.*
- $\mathcal{P}(\mathbf{pk}, \mathtt{x}, \mathtt{w}) \to \pi$*: On prover key* $\mathbf{pk}$*, instance* $\mathtt{x}$ *and witness* $\mathtt{w}$*, outputs proof* π*.*
- $\mathcal{V}(\mathbf{vk}, \mathtt{x}, \pi) \to 0/1$*: On verifier key* $\mathbf{vk}$*, instance* $\mathtt{x}$ *and proof* π*, outputs accept or reject.*

A zkSNARK satisfies the following properties:

- ***Completeness:*** *A zkSNARK* Π *is perfectly complete if for any* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*:*

$$\Pr\left[\mathcal{P}(\mathbf{vk}, \mathtt{x}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathtt{i}) \\ \pi \leftarrow \mathcal{P}(\mathbf{pk}, \mathtt{x}, \mathtt{w}) \end{array}\right] = 1$$

- ***Knowledge Soundness:*** *A zkSNARK* Π *is knowledge sound if for any PPT adversary* $\mathcal{A}$ *and for any* $\mathtt{i}$ *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *such that the following probability is* $\leq \mathsf{negl}(\lambda)$*:*

$$\Pr\left[\mathcal{V}(\mathbf{vk}, \mathtt{x}, \pi) = 1 \wedge (\mathtt{i}, \mathtt{x}, \mathtt{w}) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathtt{i}) \\ ((\mathtt{x}, \pi); \mathtt{w}) \leftarrow (\mathcal{A} \parallel \mathcal{E}_\mathcal{A})(\mathbf{pk}, \mathbf{vk}) \end{array}\right]$$

- ***Perfect Zero-Knowledge:*** *A zkSNARK* Π *is zero-knowledge if there exists a PPT simulator* $\mathcal{S}$*, such that for any adversarial distinguisher* $\mathcal{D}$*, for any* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*:*

$$\begin{aligned} &\Pr\left[\mathcal{D}(\mathbf{vk}, \mathtt{x}, \tilde{\pi}) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathtt{i}) \\ \tilde{\pi} \leftarrow \mathcal{S}(\mathbf{pk}, \mathbf{vk}, \mathtt{x}) \end{array}\right] \\ =&\Pr\left[\mathcal{D}(\mathbf{vk}, \mathtt{x}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathtt{i}) \\ \pi \leftarrow \mathcal{P}(\mathbf{pk}, \mathtt{x}, \mathtt{w}) \end{array}\right] \end{aligned}$$

- ***Succinctness:*** *The proof size* $|\pi|$ *is sublinear in the witness size* $|\mathtt{w}|$*.*

Algebraic Group Model and q-DLOG Assumption. We analyze the security of our schemes in the *algebraic group model* (AGM) [FKL18]. We use *algebraic adversary* $\mathcal{A}$ to denote a PPT algorithm that given a list of initial group elements $\mathbf{l} \in \mathbb{G}^n$ as input, whenever it outputs a group element $g \in \mathbb{G}$, it also outputs a vector $\mathbf{g} \in \mathbb{F}^n$ such that $g = \sum_{j \in [n]} g_j \cdot l_j$. Finally, as in [Gro16], our security relies on the q-DLOG assumption:

Protocol 1 (Groth16 zkSNARKs)**.**

- *Groth16.*$\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$
  1. *Sample* $\alpha, \beta, \gamma, \delta, \tau \leftarrow \mathbb{Z}_p^*$
  2. *Let* $\mathbf{u} = (u_i(\tau))_{i=0}^{m-1}, \mathbf{v} = (v_i(\tau))_{i=0}^{m-1}, \mathbf{w} = (w_i(\tau))_{i=0}^{m-1}$
  3. *Let* $\boldsymbol{\tau} = \left(\frac{\tau^i(\tau^n-1)}{\delta}\right)_{i=0}^{n-2}$
  4. *Let* $\mathbf{t} = (\mathbf{t}^{\mathsf{v}}, \mathbf{t}^{\mathsf{p}})$
     where $\mathbf{t}^{\mathsf{v}} = \left(\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\gamma}\right)_{i=0}^{k-1}, \mathbf{t}^{\mathsf{p}} = \left(\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\delta}\right)_{i=k}^{m-1}$
  5. *Output* $\mathbf{vk} = ([\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_2, [\mathbf{t}^{\mathsf{v}}]_1)$,
     $\mathbf{pk} = \left(\mathbf{vk}, [\beta]_1, [\delta]_1, \left\{[\tau^i]_2\right\}_{i=0}^{n-1}, [\boldsymbol{\tau}]_1, [\mathbf{t}^{\mathsf{p}}]_1, [\mathbf{u}]_1, [\mathbf{v}]_1, [\mathbf{v}]_2\right)$
- *Groth16.*$\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to \pi$
  1. *Let* $z = (\mathbb{x}, \mathbb{w})$ *and sample* $r, s \leftarrow \mathbb{Z}_p$.
  2. *Compute* $[Q]_1 = \left[\frac{q(\tau)(\tau^n-1)}{\delta}\right]_1$ *where* $q(X)$ *satisfying* $A(X)\cdot B(X) - C(X) = q(X)\cdot v_{\mathbb{H}}(X)$
  3. *Compute* $[A]_1 = [\alpha]_1 + r[\delta]_1 + \sum_{i\in[m]} z_i[\mathbf{u}_i]_1$
  4. *Compute* $[B]_2 = [\beta]_2 + s[\delta]_2 + \sum_{i\in[m]} z_i[\mathbf{v}_i]_2$ *and* $[B]_1$ *similarly.*
  5. *Compute* $[C]_1 = s[A]_1 + r[B]_1 - rs[\delta]_1 + \sum_{i\in[k,m]} z_i[\mathbf{t}^{\mathsf{p}}_{i-k}]_1 + [Q]_1$
  6. *Output* $\pi = ([A]_1, [C]_1, [B]_2)$
- ***Groth16.***$\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$
  1. *Parse* π *as* $([A]_1, [C]_1, [B]_2) \in \mathbb{G}_1^2 \times \mathbb{G}_2$
  2. *Output 1 if* $[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \left(\sum_{i=0}^{k-1} \mathbb{x}_i[\mathbf{t}^{\mathsf{v}}_i]_1\right) \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$*; otherwise output 0.*

Assumption 1 (q-DLOG)**.** *Fix integer* q*. The* q*-DLOG assumption for* $(\mathbb{G}_1, \mathbb{G}_2)$ *states that given*

$$[1]_1, [\tau]_1, \ldots, [\tau^q]_1, \quad [1]_2, [\tau]_2, \ldots, [\tau^q]_2$$

for uniformly chosen $\tau \in \mathbb{F}$*, the probability of an efficient* $\mathcal{A}$ *outputting* τ *is* $\mathsf{negl}(\lambda)$.

R1CS. *Rank-1 Constraint System (R1CS)* is an NP-complete language that generalizes arithmetic circuit satisfiability. The indexed relation $\mathcal{R}_{\text{R1CS}}$ is the set of all triples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((\mathbb{F}, n, m, k, A, B, C), \mathbb{x}, \mathbb{w})$ where $\mathbb{F}$ is a finite field and $A, B, C \in \mathbb{F}^{n\times m}, \mathbb{x} \in \mathbb{F}^k, \mathbb{w} \in \mathbb{F}^{m-k}$. Conventionally, the first element of $\mathbb{x}$ is fixed to 1. The relation holds if and only if $(A\cdot z)\circ(B\cdot z) = C\cdot z$ where $z = (\mathbb{x}, \mathbb{w}) \in \mathbb{F}^m$ is called the extended witness. Sometimes we refer to z as the witness when it is clear from the context.

Groth16. Groth16 [Gro16] is a zkSNARK for R1CS and the scheme is presented in Protocol 1. In Groth16.$\mathcal{G}$, polynomials u, v, w are defined as $u_i(X) = \sum_{j\in[n]} A_{j,i}\ell_j(X), v_i(X) = \sum_{j\in[n]} B_{j,i}\ell_j(X), w_i(X) = \sum_{j\in[n]} C_{j,i}\ell_j(X) \in \mathbb{F}^{<n}[X]$ for all $i \in [m]$. In Groth16.$\mathcal{P}$, the existence of $z = (\mathbb{x}, \mathbb{w}) \in \mathbb{F}^m$ such that $(A \cdot z) \circ (B \cdot z) = C \cdot z$ [2] is equivalent to the existence of a

[2] We slightly abuse the notation by using A, B, C for the R1CS matrices, and $[A]_1, [C]_1 \in \mathbb{G}_1, [B]_2 \in \mathbb{G}_2$ for the proof elements derived from them.

degree $n-2$ quotient polynomial $q(X)$ such that the following equality holds,

$$A(X) \cdot B(X) - C(X) = q(X) \cdot v_{\mathbb{H}}(X) \tag{1}$$

where

$$\begin{aligned} A(X) &:= \sum\nolimits_{i\in[m]} z_i u_i(X) = \sum\nolimits_{i\in[n]} a_i \ell_i(X), \\ B(X) &:= \sum\nolimits_{i\in[m]} z_i v_i(X) = \sum\nolimits_{i\in[n]} b_i \ell_i(X), \\ C(X) &:= \sum\nolimits_{i\in[m]} z_i w_i(X) = \sum\nolimits_{i\in[n]} c_i \ell_i(X) \end{aligned} \tag{2}$$

and for all $i \in [n]$,

$$a_i = (\sum\nolimits_{j\in[m]} A_{i,j} z_j), \quad b_i = (\sum\nolimits_{j\in[m]} B_{i,j} z_j), \quad c_i = (\sum\nolimits_{j\in[m]} C_{i,j} z_j) \tag{3}$$

Note that compared to the original version in [Gro16], $([\mathbf{u}]_1, [\mathbf{v}]_1, [\mathbf{v}]_2)$ are added to **pk** to improve the efficiency of the prover. It does not affect the security of the protocol, as they can be computed from the original **pk** without knowing the secret τ, as suggested in [Gro16].

Lemma 2.3. *Protocol 1 is a zkSNARK by Definition 2.2 under the q-DLOG assumption in the AGM [FKL18].*

3 Quotient Decomposition

Quotient polynomials are central to SNARKs based on univariate polynomials, including Groth16. In this section, we show that the quotients of products of Lagrange polynomials by the vanishing polynomial $v_{\mathbb{H}}(X)$ have a simple closed form, which we call *quotient decomposition*, and on which our update algorithms in Sections 4 and 5 build. Related structure underlies the precomputation of KZG opening proofs [FK23] and of the cached quotients in lookup arguments [EFG22, LJGK25].

Recall from Equation (1) that the quotient polynomial $q(X)$ in Groth16 is determined by the product $A(X)\cdot B(X)$, where $A(X) = \sum_{i\in[n]} a_i\ell_i(X)$ and $B(X) = \sum_{i\in[n]} b_i\ell_i(X)$; the term $C(X)$ has degree less than n and does not affect it. Since $A(X)\cdot B(X) = \sum_{i,j\in[n]} a_i b_j\, \ell_i(X)\ell_j(X)$ and division by $v_{\mathbb{H}}(X)$ is linear, the quotient of $A(X)\cdot B(X)$ is a linear combination of the quotients of the products $\ell_i(X)\ell_j(X)$. Each of these quotients is a polynomial of degree $n-2$, and committing to all n^2 of them would take $O(n^2)$ time and storage.

The key observation is that these quotients nevertheless have a simple structure: for $i \neq j$, the quotient of $\ell_i(X)\ell_j(X)$ is a linear combination of $\ell_i(X)$ and $\ell_j(X)$ alone, with coefficients that depend only on $i-j$. We formalize this observation in the following theorem, using the vector $\boldsymbol{\eta} \in \mathbb{F}^n$ given by $\eta_0 = 0$ and, with indices taken modulo n,

$$\eta_k = \frac{\omega^k}{n(1-\omega^k)} \qquad (k \neq 0). \tag{4}$$

Theorem 3.1 (Quotient Decomposition)**.** *For all $i, j \in [n]$,*

$$\ell_i(X)\,\ell_j(X) = \begin{cases} \phi_{i,j}(X)\, v_{\mathbb{H}}(X), & i \neq j, \\ \phi_{i,i}(X)\, v_{\mathbb{H}}(X) + \ell_i(X), & i = j, \end{cases}$$

where

$$\phi_{i,j}(X) = \eta_{j-i}\,\ell_i(X) + \eta_{i-j}\,\ell_j(X) \qquad (i \neq j), \tag{5}$$

$$\phi_{i,i}(X) = -\sum\nolimits_{k\neq i} \phi_{i,k}(X). \tag{6}$$

Proof. For $i \neq j$, the product $\ell_i(X)\ell_j(X)$ vanishes on $\mathbb{H}$, so it is divisible by $v_{\mathbb{H}}(X) = X^n - 1$. Using $\ell_i(X) = \frac{\omega^i}{n} \prod_{k \in [n]/i}(X - \omega^k)$, its quotient is

$$\phi_{i,j}(X) = \frac{\ell_i(X) \cdot \ell_j(X)}{\prod_{k \in [n]}(X - \omega^k)} = \frac{\omega^{i+j}}{n^2} \prod_{k \in [n]/i,j} (X - \omega^k).$$

This polynomial has degree $n-2$ and vanishes on $\mathbb{H}$ except at ω^i and ω^j, so it equals its Lagrange interpolation $\phi_{i,j}(\omega^i)\,\ell_i(X) + \phi_{i,j}(\omega^j)\,\ell_j(X)$. Since $\prod_{k \in [n]/i}(\omega^i - \omega^k) = \frac{n}{\omega^i}$, we obtain

$$\begin{aligned}
\phi_{i,j}(X) &= \frac{\omega^{i+j}}{n^2} \left(\prod_{k \in [n]/i,j} (\omega^i - \omega^k) \right) \ell_i(X) + \frac{\omega^{i+j}}{n^2} \left(\prod_{k \in [n]/i,j} (\omega^j - \omega^k) \right) \ell_j(X) \\
&= \frac{\omega^{i+j}}{n^2} \frac{1}{\omega^i - \omega^j} \cdot \frac{n}{\omega^i} \cdot \ell_i(X) + \frac{\omega^{i+j}}{n^2} \frac{1}{\omega^j - \omega^i} \cdot \frac{n}{\omega^j} \cdot \ell_j(X) \\
&= \frac{\omega^{j-i}}{n(1 - \omega^{j-i})} \cdot \ell_i(X) + \frac{\omega^{i-j}}{n(1 - \omega^{i-j})} \cdot \ell_j(X) = \eta_{j-i} \cdot \ell_i(X) + \eta_{i-j} \cdot \ell_j(X).
\end{aligned}$$

For $i = j$, the polynomials $\ell_i(X)^2$ and $\ell_i(X)$ agree on $\mathbb{H}$, so $\ell_i(X)^2 - \ell_i(X)$ is divisible by $v_{\mathbb{H}}(X)$; let $\phi_{i,i}(X)$ be the quotient. Summing $\ell_i(X)\ell_j(X)$ over all $j \in [n]$ and using $\sum_{j \in [n]} \ell_j(X) = 1$, we obtain

$$(\sum_{j \in [n]} \phi_{i,j}(X)) \cdot (X^n - 1) + \ell_i(X) = \ell_i(X) \cdot (\sum_{j \in [n]} \ell_j(X)) = \ell_i(X),$$

so $\sum_{j \in [n]} \phi_{i,j}(X) = 0$, that is, $\phi_{i,i}(X) = -\sum_{j \in [n]/i} \phi_{i,j}(X)$. □

Two features of Theorem 3.1 make it useful. First, *sparsity*: for $i \neq j$, the quotient of $\ell_i(X)\ell_j(X)$ involves only $\ell_i(X)$ and $\ell_j(X)$. Second, *shift invariance*: the coefficients depend only on $i-j$, so sums over j are circular convolutions with fixed vectors; for example, $\sum_{j \in [n]} b_j \eta_{j-i} = (\boldsymbol{\eta}' * \mathbf{b})_i$, where $\boldsymbol{\eta}'$, given by $\eta'_k = \eta_{-k}$, is the reversal of $\boldsymbol{\eta}$.

Theorem 3.1 applies to the quotient of any product $A(X) \cdot B(X) = \sum_{i,j \in [n]} a_i b_j\, \ell_i(X)\ell_j(X)$, and how it is best used depends on the sparsity of $\mathbf{a}$ and $\mathbf{b}$:

- *Sparse × dense.* If $\mathbf{a}$ has m non-zero entries and $\mathbf{b}$ is fixed, the quotients of $\ell_i(X) \cdot B(X)$ can be precomputed for all $i \in [n]$ with FFTs, using shift invariance; the quotient of $A(X) \cdot B(X)$ is then a linear combination of m of them. We use this case for the projection terms in Section 4.4.

- *Sparse × sparse.* If $\mathbf{a}$ and $\mathbf{b}$ have m non-zero entries in total, the product involves only $O(m^2)$ pairs (i, j), and by sparsity its quotient is a combination of the $\ell_i(X)$ and $\phi_{i,i}(X)$ at these m indices, with coefficients computable in $O(m^2)$ field operations, or in $O(m \log^2 m)$ with a sparse convolution. We use this case for the cross term in Section 4.3.

- *Dense × dense.* Summing over all pairs gives a closed form of the Groth16 proof quotient in terms of two circular convolutions, which take 4 DFTs in total (Appendix A).

4 Semi-Dynamic zkSNARK

4.1 Definition

Definition 4.1 (Semi-dynamic zkSNARKs)**.** *A semi-dynamic zero-knowledge succinct non-interactive argument of knowledge (semi-dynamic zkSNARK) for an indexed relation $\mathcal{R}$ consists of the following PPT algorithms $\Psi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{U}, \mathcal{V})$:*

- $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$*: On* 1^λ *and index* $\mathbb{i}$*, outputs a public key* $\mathbf{pk}$ *and a verifier key* $\mathbf{vk}$*.*
- $\mathcal{K}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to D$*: On public key* $\mathbf{pk}$*, instance* $\mathbb{x}$ *and witness* $\mathbb{w}$*, outputs preprocessed data* D*.*
- $\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{rand})$*: On public key* $\mathbf{pk}$*, valid extended witness* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_\mathbb{i}$*, outputs the proof* π *and the randomness* rand*.*
- $\mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \to \pi'$*: On public key* $\mathbf{pk}$*, valid extended witnesses* $(\mathbb{x}, \mathbb{w}), (\mathbb{x}', \mathbb{w}') \in \mathcal{R}_\mathbb{i}$*, proof* π *and randomness* rand *outputted by* $\mathcal{P}$*, preprocessed data* D *outputted by* $\mathcal{K}$*, outputs updated proof* π'*.*
- $\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$*: On verifier key* $\mathbf{vk}$*, instance* $\mathbb{x}$ *and proof* π *outputed by* $\mathcal{P}$ *or* $\mathcal{U}$*, outputs accept or reject.*

A semi-dynamic zkSNARK Ψ *satisfies the following properties:*

- ***Completeness:*** *It is perfectly complete if for any* $(\mathbb{i}, \mathbb{x}, \mathbb{w}), (\mathbb{i}, \mathbb{x}', \mathbb{w}') \in \mathcal{R}$*:*

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) = 1 \wedge & (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \mathcal{V}(\mathbf{vk}, \mathbb{x}', \pi') = 1 & D \leftarrow \mathcal{K}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \\ & (\pi, \mathsf{rand}) \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \\ & \pi' \leftarrow \mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \end{array}\right] = 1$$

- ***Knowledge Soundness:*** *It is knowledge sound if for any PPT adversary* $\mathcal{A}$ *and for any* $\mathbb{i}$ *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *such that the following probability is* $\leq \mathsf{negl}(\lambda)$

$$\Pr\left[\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ ((\mathbb{x}, \pi); \mathbb{w}) \leftarrow (\mathcal{A} \parallel \mathcal{E}_\mathcal{A})(\mathbf{pk}, \mathbf{vk}) \end{array}\right]$$

- ***Perfect Zero-Knowledge:*** *It is zero-knowledge if there exists PPT simulators* $\mathcal{S}$ *such that for any adversarial distinguisher* $\mathcal{D}$*, for any* $(\mathbb{i}, \mathbb{x}, \mathbb{w}), (\mathbb{i}, \mathbb{x}', \mathbb{w}') \in \mathcal{R}$ *:*

$$\begin{aligned} &\Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}, \tilde{\pi}) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathbb{i}) \\ \tilde{\pi} \leftarrow \mathcal{S}(\mathbf{pk}, \mathbf{vk}, \mathbb{x}) \end{array}\right] \\ =&\Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \end{array}\right] \end{aligned}$$

and

$$\begin{aligned} &\Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}, \pi') = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \pi' \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}', \mathbb{w}') \end{array}\right] \\ =\Pr&\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}, \pi') = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ (\pi, \mathsf{rand}) \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \\ D \leftarrow \mathcal{K}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \\ \pi' \leftarrow \mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \end{array}\right] \end{aligned}$$

- ***Succinctness:*** *The proof size* $|\pi|$ *is sublinear in the witness size* $|\mathbb{w}|$*.*
- ***Updatability:*** *Let* $|\Delta_\mathbb{x}|, |\Delta_\mathbb{w}|$ *denote the Hamming distances between* $\mathbb{x}, \mathbb{x}'$ *and* $\mathbb{w}, \mathbb{w}'$*. If* $|\Delta_\mathbb{x}| + |\Delta_\mathbb{w}| = o(|\mathbb{x}| + |\mathbb{w}|)$*, the running time of update* $\mathcal{U}$ *should be sublinear in the running time of the prover* $\mathcal{P}$*.*

Definition 4.1 above is adapted from the definition of dynamic zkSNARKs in [WPSP25b]. We combine both the prover key and the updater key into the public key $\mathbf{pk}$ as they are both public and are almost the same. In our definition, we have a preprocessing algorithm $\mathcal{K}$ that takes the original extended witness $(\mathbb{x}, \mathbb{w})$ as input and computes the preprocessed data D. New proofs

are always computed using this preprocessed data of the original witness in the semi-dynamic setting. Note that we ask the prover to output the randomness rand used to generate the proof. This is necessary to re-randomize the proof in sublinear time in our update algorithm. It is not sent to the verifier, and can be omitted if zero-knowledge is not needed. The update algorithm $\mathcal{U}$ takes both the original witness and the new witness as the input, and we assume that they are valid witnesses in the indexed relation. In practice, the new witness can often be computed in sublinear time by traversing the related gates of the updated input in a circuit. Moreover, the size of $\mathbf{uk}, D$ and the witnesses are linear, but the running time of $\mathcal{U}$ can be sublinear with a linear size storage. This is the same model as the sublinear prover time of lookup arguments such as [EFG22], and the logarithmic prover time of the classical Merkle tree [Mer87]. The notion of knowledge soundness extends naturally to the updater, where the adversary is allowed to invoke the update algorithm. Finally, our definition of zero-knowledge requires that the proof generated by $\mathcal{U}$ is indistinguishable from the proof generated by $\mathcal{P}$, which is stronger than the definition in [WPSP25b].

4.2 Semi-Dynamic zkSNARK based on Groth16

In subsequent sections, we present our construction of semi-dynamic zkSNARK on top of Groth16.

Distance between witnesses. By the definition of R1CS, the difference between the original witness $z = (\mathbb{x}, \mathbb{w})$ and a new witness $z' = (\mathbb{x}', \mathbb{w}')$ is defined as $\Delta^z = (z'_i - z_i)_{i\in[m]}$. Δ^z can be represented by its dense form $(i, \Delta^z_i)_{i\in I_z}$, where I_z is the set of indices for which $\Delta^z_i \neq 0$. The Hamming distance between z and z' is denoted as $|\Delta_z| = |I_z| = d$.

In our protocol, we also need the differences

$$\Delta^A = A \cdot (z' - z),\ \Delta^B = B \cdot (z' - z),\ \Delta^C = C \cdot (z' - z),$$

with their corresponding non-zero indices as I_A, I_B, I_C. Their size may not be sublinear even if $|\Delta_z|$ is. This could happen when the i-th column of A, B or C is dense for some $i \in I_z$. In practice, when the computation is modeled as a circuit, this scenario corresponds to the case where one wire connects to many gates in the circuit. For the simplicity of stating the asymptotic complexity, we assume that $|\Delta_A|, |\Delta_B|, |\Delta_C| = O(d)$. However, our protocols still work without this assumption, and in Table 2 of Section 6, we provide the precise calculation of the number of group operations in our schemes. The same assumption is implicit in [WPSP25b]. In fact, their scheme uses the Plonkish arithmetization, which has more redundancy than R1CS and a change in one wire value may lead to more changes in the extended witness.

Updating the proof. After updating the extended witness, our goal is to update the Groth16 proof $\pi = ([A]_1, [B]_2, [C]_1)$ to $\pi' = ([A']_1, [B']_2, [C']_1)$ for z'. Recall that $[A]_1$ and $[B]_2$ in Step 3 and Step 4 of Protocol 1 are computed as:

$$[A]_1 = [\alpha]_1 + r[\delta]_1 + \sum\nolimits_{i\in[m]} z_i[\mathbf{u}_i]_1, \quad [B]_2 = [\beta]_2 + s[\delta]_2 + \sum\nolimits_{i\in[m]} z_i[\mathbf{v}_i]_2,$$

which are linear combinations of the public key $\mathbf{pk}$ with the extended witness and the randomness. They can be easily updated with the new witness. To achieve zero-knowledge, the randomness must be refreshed, thus the new proof becomes:

$$[A']_1 = [\alpha]_1 + r'[\delta]_1 + \sum\nolimits_{i\in[m]} z'_i[\mathbf{u}_i]_1, \quad [B']_2 = [\beta]_2 + s'[\delta]_2 + \sum\nolimits_{i\in[m]} z'_i[\mathbf{v}_i]_2$$

which can be computed as:

$$[A']_1 = [A]_1 + (r' - r)[\delta]_1 + \sum\nolimits_{i\in I_z} \Delta^z_i[\mathbf{u}_i]_1,\ [B']_2 = [B]_2 + (s' - s)[\delta]_2 + \sum\nolimits_{i\in I_z} \Delta^z_i[\mathbf{v}_i]_2$$

The update consists of $O(|\Delta_z|) = O(d)$ group operations.

Similarly, recall $[C]_1$ in Step 5 of Protocol 1 is computed as:

$$[C]_1 = s[A]_1 + r[B]_1 - rs[\delta]_1 + \sum\nolimits_{i\in[k,m]} z_i[\mathbf{t}^{\mathsf{p}}_{i-k}]_1 + [Q]_1$$

where $[Q]_1 = \left[\frac{q(\tau)(\tau^n-1)}{\delta}\right]_1$. Consequently, the updated proof $[C']_1$ becomes:

$$[C']_1 = s'[A']_1 + r'[B']_1 - r's'[\delta]_1 + \sum\nolimits_{i\in[k,m]} z'_i[\mathbf{t}^{\mathsf{p}}_{i-k}]_1 + [Q']_1$$

and the difference from $[C]_1$ is given by $[\Delta^C_{\text{lin}}]_1 + ([Q']_1 - [Q]_1)$ where

$$[\Delta^C_{\text{lin}}]_1 = s'[A']_1 - s[A]_1 + r'[B']_1 - r[B]_1 - (r's' - rs)[\delta]_1 + \sum\nolimits_{i\in[k,m]\cap I_z} \Delta^z_i[\mathbf{t}^{\mathsf{p}}_{i-k}]_1$$

All terms within $[\Delta^C_{\text{lin}}]_1$ can be computed efficiently. The term $s'[A']_1$ can be directly computed using $[A']_1$ from previous steps. Using the same approach to compute and update the term $[B]_2$, the prover also computes and updates $[B']_1$ which is used as a component in the update of $[C]_1$. The complexity of computing all these linear terms is $O(d)$ group operations. Note that this is the step where the updater $\mathcal{U}$ does need to know both the old randomness r, s and the new randomness r', s' in the protocol. That is why $\mathcal{P}$ needs to pass the randomness to $\mathcal{U}$ in our definition.

Quotient term. The main challenge to update the proof lies in updating the quotient term:

$$[Q']_1 - [Q]_1 = \left[\frac{q'(\tau)(\tau^n-1)}{\delta}\right]_1 - \left[\frac{q(\tau)(\tau^n-1)}{\delta}\right]_1 = \left[\frac{(q'(\tau)-q(\tau))(\tau^n-1)}{\delta}\right]_1.$$

Recall that $q(X)$ is defined as:

$$A(X)\cdot B(X) - C(X) = q(X)\cdot v_{\mathbb{H}}(X)$$

where the definition of $A(X)$, $B(X)$ and $C(X)$ can be found in Equation (2) and Equation (3). To use the Lagrange basis, we define the update polynomials as:

$$\begin{aligned}
\Delta^A(X) &= A'(X) - A(X) = \sum\nolimits_{i\in I_A} \Delta^A_i \ell_i(X)\\
\Delta^B(X) &= B'(X) - B(X) = \sum\nolimits_{i\in I_B} \Delta^B_i \ell_i(X)\\
\Delta^C(X) &= C'(X) - C(X) = \sum\nolimits_{i\in I_C} \Delta^C_i \ell_i(X)
\end{aligned}$$

where $\Delta^A = A\cdot\Delta^z, \Delta^B = B\cdot\Delta^z, \Delta^C = C\cdot\Delta^z \in \mathbb{F}^n$ for all $i \in [n]$. Therefore, when updating the quotient polynomial,

$$\begin{aligned}
&(A'(x)\cdot B'(X) - C'(X)) - (A(x)\cdot B(X) - C(X))\\
&= (A(X) + \Delta^A(X))\cdot(B(X) + \Delta^B(X)) - A(X)\cdot B(X) - \Delta^C(X)\\
&= \underbrace{\Delta^A(X)\cdot B(X) + \Delta^B(X)\cdot A(X)}_{\text{projection terms}} + \underbrace{\Delta^A(X)\cdot\Delta^B(X)}_{\text{cross term}} - \Delta^C(X)
\end{aligned}$$

Because the degree of $\Delta^C(X)$ is $n-1$, it does not affect the quotient polynomial and its role is making the remainder polynomial zero (i.e., the witness remains valid after the update). However, the first three terms are all nonlinear in the witness, making them difficult to compute efficiently.

We begin by handling the cross term $\Delta^A(X)\cdot\Delta^B(X)$ in Section 4.3. We then address the two projection terms $\Delta^A(X)\cdot B(X)$ and $\Delta^B(X)\cdot A(X)$ in Section 4.4, and present the full protocol in Section 4.5.

4.3 Cross Term

The cross-term can be expressed as:

$$\begin{aligned}\Delta^A(X)\cdot\Delta^B(X) &= \left(\sum\nolimits_{i\in I_A}\Delta_i^A\ell_i(X)\right)\cdot\left(\sum\nolimits_{i\in I_B}\Delta_i^B\ell_i(X)\right)\\ &= \sum\nolimits_{(i,j)\in(I_A,I_B)}\Delta_i^A\Delta_j^B\ell_i(X)\ell_j(X)\end{aligned}$$

Although the summation only involves $O(d^2)$ term, computing the cross term remains challenging. For each $(i,j)\in(I_A,I_B)$, we cannot efficiently derive $\frac{\ell_i(X)\ell_j(X)}{v_{\mathbb{H}}(X)}$ from $\{X^i\}_{i\in[n]}$. It is possible to precompute all $\{\frac{\ell_i(X)\ell_j(X)}{v_{\mathbb{H}}(X)}\}_{(i,j)\in[n]^2}$ in the preprocessing phase, but it would take $O(n^2)$ time and storage, which is impractical for large n.

Our key insight is that the family of products $\ell_i(X)\ell_j(X)$ for every $(i,j)\in[n]^2$ exhibits a highly structured quotient form. This is exactly the quotient decomposition of Theorem 3.1: for $i\neq j$, the quotient $\phi_{i,j}(X)$ of $\ell_i(X)\ell_j(X)$ is $\eta_{j-i}\,\ell_i(X)+\eta_{i-j}\,\ell_j(X)$, and $\phi_{i,i}(X)=-\sum_{j\neq i}\phi_{i,j}(X)$, where $\boldsymbol{\eta}$ is given in Equation (4) and $\boldsymbol{\eta}'$ is its reversal (Section 3). Hence the quotient of any product of polynomials is a linear combination of $\{\phi_{i,j}(X)\}_{(i,j)\in[n]^2}$, and further of $\{\ell_i(X)\}_{i\in[n]}$ and $\{\phi_{i,i}(X)\}_{i\in[n]}$. Using Theorem 3.1, we can decompose the quotient of the cross term as:

$$\begin{aligned}\Delta^A(X)\cdot\Delta^B(X) &= \sum\nolimits_{(i,j)\in(I_A,I_B)}\Delta_i^A\Delta_j^B\ell_i(X)\ell_j(X)\\ &= \left(\sum\nolimits_{(i,j)\in(I_A,I_B)}\Delta_i^A\Delta_j^B\phi_{i,j}(X)\right)v_{\mathbb{H}}(X)+\sum\nolimits_{i\in I_A\cap I_B}\Delta_i^A\Delta_i^B\ell_i(X)\end{aligned}$$

Therefore, the quotient of the cross term can be expressed as:

$$\begin{aligned}q^{\text{cross}}(X) &= \sum\nolimits_{(i,j)\in(I_A,I_B)}\Delta_i^A\Delta_j^B\phi_{i,j}(X)\\ &= \sum\nolimits_{i\in I}\Delta_i^A\Delta_i^B\phi_{i,i}(X)+\sum\nolimits_{i\in I}\sum\nolimits_{j\in I/i}(\Delta_i^A\Delta_j^B+\Delta_i^B\Delta_j^A)\cdot\eta_{j-i}\cdot\ell_i(X).\end{aligned}\tag{7}$$

Computing this term only takes $O(|I|)$ group operations and $O(|I|^2)$ field operations where $I=I_A\cup I_B$ with preprocessing of $\{\ell_i(X)\}_{i\in[n]}$ and $\{\phi_{i,i}(X)\}_{i\in[n]}$.

Reducing the number of field operations via sparse convolution. We can further reduce the complexity of field operations to $O(|I|\log^2|I|)$ by refining the decomposition to write the quotient as a linear combination of $\ell_i(X)$ for $i\in I$:

$$\begin{aligned}q^{\text{cross}}(X) = &\sum\nolimits_{i\in I}\Delta_i^A\Delta_i^B\phi_{i,i}(X)+\\ &\sum\nolimits_{i\in I}\left(\Delta_i^A\sum\nolimits_{j\in I/i}(\Delta_j^B\cdot\eta'_{i-j})+\Delta_i^B\sum\nolimits_{j\in I/i}(\Delta_j^A\cdot\eta'_{i-j})\right)\cdot\ell_i(X)\end{aligned}\tag{8}$$

In the equation above, the quadratic field operation cost comes from computing $\sum_{j\in I/i}\Delta_j^B\cdot\eta'_{i-j}$ and $\sum_{j\in I/i}\Delta_j^A\cdot\eta'_{i-j}$ for all $i\in I$, which takes $O(|I|^2)$ time. These sums can be interpreted as convolutions of the sparse vectors Δ^A and Δ^B with a special fixed vector $\boldsymbol{\eta}'$ defined in Equation (4), where the input vectors are nonzero only on indices in I and only the output entries corresponding to I are needed. We denote these operations as $\text{SparseConv}_{\boldsymbol{\eta}'}(\mathbf{a},I)$ and $\text{SparseConv}_{\boldsymbol{\eta}'}(\mathbf{b},I)$.

We present an algorithm to compute such sparse convolution in $O(|I|\log^2|I|)$ field operations in Protocol 7 of Appendix B. The algorithm is a special and simplified variant of the algorithms in [GGS87] for the sparse convolution.

Keeping the same prover key as Groth16. The protocol above assumes the existence of $\left[\frac{\ell_i(\tau)(\tau^n-1)}{\delta}\right]_1$ for $i \in [n]$ in $\mathbf{pk}$. However, in the original Protocol 1 of Groth16, the trusted setup outputs $[\boldsymbol{\tau}]_1 = \left(\left[\frac{\tau^i(\tau^n-1)}{\delta}\right]_1\right)_{i=0}^{n-2}$, while the highest degree of $\ell_i(\tau)$ is $n-1$. Therefore, we cannot compute $\left[\frac{\ell_i(\tau)(\tau^n-1)}{\delta}\right]_1$ from $[\boldsymbol{\tau}]_1$ without knowing the secret τ. It is not hard to re-prove the security of Groth16 by providing an extra element $\left[\frac{\tau^{n-1}(\tau^n-1)}{\delta}\right]_1$ in $\mathbf{pk}$.

However, we argue that this is unnecessary, since the degree of the quotient polynomial $q(X)$ remains $n-2$, and the highest degree of $\ell_i(X)$ must cancel out with each other. More specificly, all quotients can be decomposed into a linear combination of $\{\phi_{i,j}(X)\}_{i,j\in[n]}$. For all $i \neq j \in [n]$, the highest degree term of $\phi_{i,j}(X)$ is zero since by Equation (5)

$$\frac{\omega^{j-i}}{n(1-\omega^{j-i})}\frac{\omega^i}{n}X^{n-1} + \frac{\omega^{i-j}}{n(1-\omega^{i-j})}\frac{\omega^j}{n}X^{n-1} = 0$$

Moreover, for every $i \in [n]$, we have that $\phi_{i,i}(X) = -\sum_{j\in[n]/i}\phi_{i,j}(X)$ as shown in Equation (6), so that the highest degree term of $\phi_{i,i}(X)$ is also zero. Therefore, it is enough to only use *truncated lagrange basis* which are defined as:

$$\bar{\ell}_i(X) = \frac{1}{n}\cdot\sum\nolimits_{j\in[n-1]}(\omega^i)^{-j}\cdot X^j$$

Efficient preprocessing. It remains to show how to efficiently compute

$$[\bar{\boldsymbol{\ell}}]_1 := \left(\left[\frac{\bar{\ell}_i(\tau)(\tau^n-1)}{\delta}\right]\right)_{i=0}^{n-1}, \quad [\mathbf{p}]_1 := \left(\left[\frac{\phi_{i,i}(\tau)(\tau^n-1)}{\delta}\right]\right)_{i=0}^{n-1}$$

from

$$[\bar{\boldsymbol{\tau}}]_1 = ([\boldsymbol{\tau}]_1, 0_{\mathbb{G}_1}) \in \mathbb{G}^n, \quad [\boldsymbol{\tau}]_1 = \left(\left[\frac{\tau^i(\tau^n-1)}{\delta}\right]_1\right)_{i=0}^{n-2} \in \mathbf{pk}$$

- Preprocessing Truncated Lagrange Basis. The Lagrange basis can be derived from the monomial basis using the inverse DFT as follows

  $$(\ell_i(X))_{i=0}^{n-1} = \mathrm{iDFT}((X^i)_{i=0}^{n-1})$$

  By linearity of iDFT, we can remove the degree $n-1$ term in monomials to compute the truncated Lagrange basis, and multiply them by the constant $\frac{\tau^n-1}{\delta}$. Therefore, by Lemma 2.1, we can compute $[\bar{\boldsymbol{\ell}}]_1 = \mathrm{iDFT}([\bar{\boldsymbol{\tau}}]_1)$ in $O(n\log n)$ time.

- Preprocessing Diagonals. The naive approach to compute $\phi_{i,i}(X)$ for all $i \in [n]$ would take $O(n^2)$ time by taking an n-size summation for each term. Instead, we exploit their convolution structure. Substituting Equation (5) into Equation (6) and letting $s = \sum_{k\in[n]}\eta_k$,

  $$\phi_{i,i}(X) = -\sum_{j\in[n]/i}\left(\eta_{j-i}\,\ell_i(X) + \eta_{i-j}\,\ell_j(X)\right) = -s\cdot\ell_i(X) - \sum_{j\in[n]/i}\eta_{i-j}\cdot\ell_j(X),$$

  since $j-i$ ranges over all non-zero residues modulo n and $\eta_0 = 0$. The second sum is the i-th entry of $\boldsymbol{\eta} * (\ell_j(X))_{j=0}^{n-1}$, so we can compute $[\mathbf{p}]_1 = -s\cdot[\bar{\boldsymbol{\ell}}]_1 - \boldsymbol{\eta} * [\bar{\boldsymbol{\ell}}]_1$ with one group convolution in $O(n\log n)$ time.

Complexity analysis. The contribution of the cross term to the updated proof $[C']_1$ is given by

$$[\Delta^C_{\mathrm{cross}}]_1 = \sum\nolimits_{i\in I_A\cup I_B}(\Delta^A_i\tilde{\Delta}^B_i + \Delta^B_i\tilde{\Delta}^A_i)[\bar{\boldsymbol{\ell}}_i]_1 + \sum\nolimits_{i\in I_A\cap I_B}\Delta^A_i\Delta^B_i[\mathbf{p}_i]_1$$

where $\tilde{\Delta}^A = \mathrm{SparseConv}_{\boldsymbol{\eta}'}(\Delta^A, I_A\cup I_B)$ and $\tilde{\Delta}^B = \mathrm{SparseConv}_{\boldsymbol{\eta}'}(\Delta^B, I_A\cup I_B)$. The complexity is $O(d)\mathbb{G} + O(d\log^2 d)\mathbb{F}$ with $O(n\log n)\mathbb{G}$ preprocessing time.

4.4 Projection Terms

The projection terms can be written as:

$$\begin{aligned}
&\Delta_A(X) \cdot B(X) + \Delta_B(X) \cdot A(X) \\
&= \Big(\sum\nolimits_{i \in I_A} \Delta_i^A \ell_i(X)\Big) \cdot B(X) + \Big(\sum\nolimits_{i \in I_B} \Delta_i^B \ell_i(X)\Big) \cdot A(X) \\
&= \sum\nolimits_{i \in I_A} \Delta_i^A \cdot (\ell_i(X) \cdot B(X)) + \sum\nolimits_{i \in I_B} \Delta_i^B \cdot (\ell_i(X) \cdot A(X)) \qquad (9)
\end{aligned}$$

The formula is very similar to the sublinear lookup argument in [EFG22]. With a one-time preprocessing on $A(X), B(X)$, the quotient of the projection terms can be computed efficiently. We further optimize this preprocessing by expressing all projection quotients in convolution form. This is summarized in Theorem 4.2, building on Theorem 3.1.

Theorem 4.2 (Projection Quotient)**.** *For each $i \in [n]$, the i-th projection quotient $q_i^a(X)$ for the polynomial $A(X) = \sum_{i \in [n]} a_i \ell_i(X)$ is defined by*

$$A(X)\ell_i(X) = q_i^a(X) v_{\mathbb{H}}(X) + a_i \ell_i(X)$$

The vector of all projection quotients $(q_i^a(X))_{i=0}^{n-1}$ can be expressed as

$$(q_i^a(X))_{i=0}^{n-1} = \boldsymbol{\eta} * (\mathbf{a} \circ (\ell_i(X))_{i=0}^{n-1}) + (\boldsymbol{\eta}' * \mathbf{a}) \circ (\ell_i(X))_{i=0}^{n-1} + \mathbf{a} \circ (\phi_{i,i}(X))_{i=0}^{n-1}$$

where $\boldsymbol{\eta}$ and $\boldsymbol{\eta}'$ are defined in Section 3.

Proof. Fix $i \in [n]$. Applying Theorem 3.1 to each product $\ell_j(X)\ell_i(X)$,

$$A(X)\,\ell_i(X) = \sum_{j \in [n]} a_j\,\ell_j(X)\,\ell_i(X) = \Big(\sum_{j \neq i} a_j\,\phi_{j,i}(X) + a_i\,\phi_{i,i}(X)\Big)\,v_{\mathbb{H}}(X) + a_i\,\ell_i(X),$$

so the projection quotient is $q_i^a(X) = \sum_{j \neq i} a_j \phi_{j,i}(X) + a_i \phi_{i,i}(X)$. Substituting $\phi_{j,i}(X) = \eta_{i-j}\,\ell_j(X) + \eta_{j-i}\,\ell_i(X)$ from Equation (5), and using $\eta_0 = \eta_0' = 0$ to extend the sums to all $j \in [n]$,

$$\begin{aligned}
q_i^a(X) &= \sum_{j \in [n]} \eta_{i-j}\,\big(a_j\,\ell_j(X)\big) + \Big(\sum_{j \in [n]} \eta'_{i-j}\,a_j\Big)\,\ell_i(X) + a_i\,\phi_{i,i}(X) \\
&= \big(\boldsymbol{\eta} * (\mathbf{a} \circ \boldsymbol{\ell})\big)_i + (\boldsymbol{\eta}' * \mathbf{a})_i\,\ell_i(X) + a_i\,\phi_{i,i}(X),
\end{aligned}$$

where $\boldsymbol{\ell} = (\ell_j(X))_{j=0}^{n-1}$. This is the i-th entry of the vector identity in Theorem 4.2. □

We can further simplify Equation (9) using $\{q_i^a(X)\}_{i \in [n]}$ defined in Theorem 4.2 into the following

$$\Delta_A(X) \cdot B(X) + \Delta_B(X) \cdot A(X) = \sum\nolimits_{i \in I_A} \Delta_i^A \cdot q_i^B(X) + \sum\nolimits_{i \in I_B} \Delta_i^B \cdot q_i^A(X)$$

With preprocessing of projection quotients, the projection term can be computed in $O(n)$ group operations.

Keeping the same prover key as Groth16. Similar to the cross term, the prover key can be made the same as Groth16 as the monomials with degree $n-1$ in the Lagrange polynomials always cancel out with each other.

Efficient preprocessing. The projection quotients $[\mathbf{q}^{\mathsf{a}}]_1 := \big(\big[\frac{q_i^a(\tau)(\tau^n-1)}{\delta}\big]_1\big)_{i=0}^{n-1}$ and $[\mathbf{q}^{\mathsf{b}}]_1 := \big(\big[\frac{q_i^b(\tau)(\tau^n-1)}{\delta}\big]_1\big)_{i=0}^{n-1}$ can be computed efficiently in $O(n \log n)$ time by applying Theorem 4.2 as follows

$$[\mathbf{q}^{\mathsf{a}}]_1 = \boldsymbol{\eta} * (\mathbf{a} \circ [\bar{\boldsymbol{\ell}}]_1) + (\boldsymbol{\eta}' * \mathbf{a}) \circ [\bar{\boldsymbol{\ell}}]_1 + \mathbf{a} \circ [\mathbf{p}]_1$$

$$[\mathbf{q}^{\mathsf{b}}]_1 = \boldsymbol{\eta} * (\mathbf{b} \circ [\bar{\boldsymbol{\ell}}]_1) + (\boldsymbol{\eta}' * \mathbf{b}) \circ [\bar{\boldsymbol{\ell}}]_1 + \mathbf{b} \circ [\mathbf{p}]_1$$

Our preprocessing of projection quotients is around $2\times$ faster than [FK23] as we use circular convolutions instead of linear convolutions. We highlight that our preprocessing algorithm is not only concretely faster but also conceptually simpler than the one in [FK23], and the core insight to derive this simple preprocessing algorithm is still the perspective of quotient decomposition.

Complexity analysis. The contribution of the projection terms to the updated proof $[C']_1$ is

$$[\Delta^C_{\text{proj}}]_1 = \sum\nolimits_{i \in I_A} \Delta^A_i [\mathbf{q}^{\mathsf{b}}_i]_1 + \sum\nolimits_{i \in I_B} \Delta^B_i [\mathbf{q}^{\mathsf{a}}_i]_1$$

The complexity is $O(d)\mathbb{G}$ with $O(n \log n)\mathbb{G}$ preprocessing time.

4.5 Putting Everything Together

The full protocol of our semi-dynamic zkSNARK is presented in Protocol 2. In $\mathit{SemiDynark}.\mathcal{G}$, we have additional elements $[\bar{\boldsymbol{\ell}}]_1$ and $[\mathbf{p}]_1$ in the public parameters of our protocol over Groth16. All of them can be computed using the public key of Groth16 without knowing the secret τ. Therefore, our protocol can be used on top of Groth16 without rerunning the trusted setup. Alternatively, these elements can be computed more efficiently with τ in a trusted setup, and they do not depend on the R1CS or the witness. We summarize this result in the following theorem.

Theorem 4.3. *Protocol 2 is a semi-dynamic zkSNARK by Definition 4.1 under the q-DLOG assumption in the AGM.*

Proof. A benefit of keeping the same proof, the same prover algorithm and the same verifier algorithm as Groth16 is that the formal security proof is straightforward.

Completeness. The proof computed by $\mathit{SemiDynark}.\mathcal{U}$ is the same as the proof computed by $\mathit{SemiDynark}.\mathcal{P}$ for z' with the same randomness r', s', by the equations in the preceding sections, which is in turn the same as $\mathit{Groth16}.\mathcal{P}$. Therefore, the verifier always outputs 1.

Knowledge soundness. By Lemma 2.3, there exists a PPT extractor $\mathcal{E}_{\mathit{Groth16}}$ for Groth16 in Protocol 1. $\mathcal{E}_{\mathit{SemiDynark}}$ simply runs $\mathcal{E}_{\mathit{Groth16}}$ alongside $\mathcal{A}$ to extract the witness $\mathbb{w}$. Note that all additional elements in $\mathit{SemiDynark}.\mathbf{pk}$ can be computed as linear combinations of elements in $\mathit{Groth16}.\mathbf{pk}$ without knowing the trapdoor τ. Moreover, $\mathit{SemiDynark}.\mathcal{V}$ and $\mathit{SemiDynark}.\pi$ are exactly the same as Groth16. Therefore, by Lemma 2.3, the following probability is negligible:

$$\Pr\left[\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ ((\mathbb{x}, \pi); \mathbb{w}) \leftarrow (\mathcal{A} \parallel \mathcal{E}_{\mathit{SemiDynark}})(\mathbf{pk}, \mathbf{vk}) \end{array}\right].$$

Zero-knowledge. Again notice that the proof generated by $\mathcal{U}$ is exactly the same as the proof generated by $\mathcal{P}$ for the same witness z' and the same randomness (r', s'). In other words, for every (r', s') used by $\mathcal{U}$, there exists a unique randomness used by $\mathcal{P}$ that produces the same proof. Therefore, the proofs are identically distributed, and are thus indistinguishable from each other. In addition, by Lemma 2.3, there exists a simulator $\mathcal{S}_{\mathit{Groth16}}$ that simulates a proof that is indistinguishable from the proof generated by $\mathcal{P}$, without knowing $\mathbb{w}$. Again, although $\mathit{Groth16}.\mathbf{pk}$ is different from $\mathit{SemiDynark}.\mathbf{pk}$, all additional elements can be computed publicly. Combining the two arguments above completes the proof of zero-knowledge in Definition 4.1. Interestingly, our stronger definition of zero-knowledge requiring that the proof generated by $\mathcal{U}$ is indistinguishable from the proof generated by $\mathcal{P}$ actually makes our security proof simpler. □

Complexity. Combining the complexity analyses above, the update time of *SemiDynark* is $O(d)\mathbb{G} + O(d \log^2 d)\mathbb{F}$, after a preprocessing on the original witness of complexity $O(n \log n)\mathbb{G}$. The prover time is $O(n \log n)\mathbb{F} + O(n)\mathbb{G}$, the proof size is $O(1)$ and the verifier time is $O(1)$[3].

[3] Assuming that the size of the public input $|\mathbb{x}|$ is constant.

Protocol 2 (Semi-Dynamic zkSNARK)**.**

- *SemiDynark.*$\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$
  1. $Groth16.\mathbf{pk} \leftarrow Groth16.\mathcal{G}(1^\lambda, \mathbb{i})$.
  2. *Let* $\boldsymbol{\eta}, \boldsymbol{\eta}'$ *be as defined in Section 3.*
  3. *Compute* $[\bar{\boldsymbol{\ell}}]_1 = iDFT([\bar{\boldsymbol{\tau}}]_1)$ *where* $[\bar{\boldsymbol{\tau}}]_1 = ([\boldsymbol{\tau}]_1, 0_{\mathbb{G}_1})$ *and* $[\boldsymbol{\tau}]_1 \in Groth16.\mathbf{pk}$
  4. *Compute* $[\mathbf{p}]_1 = -\left(\left(\sum_{i\in[n]} \eta_i\right) \cdot [\bar{\boldsymbol{\ell}}]_1 + \boldsymbol{\eta} * [\bar{\boldsymbol{\ell}}]_1\right)$
  5. *Output* $\mathbf{pk} = (Groth16.\mathbf{pk}, [\bar{\boldsymbol{\ell}}]_1, [\mathbf{p}]_1), \mathbf{vk} = Groth16.\mathbf{vk}$
- *SemiDynark.*$\mathcal{K}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to D$*:*
  1. *Compute* $\mathbf{a} = A \cdot z, \mathbf{b} = B \cdot z, \mathbf{c} = C \cdot z \in \mathbb{F}^n$ *where* $z = (\mathbb{x}, \mathbb{w})$
  2. *Compute* $[\mathbf{q}^{\mathsf{a}}]_1 = \boldsymbol{\eta} * (\mathbf{a} \circ [\bar{\boldsymbol{\ell}}]_1) + (\boldsymbol{\eta}' * \mathbf{a}) \circ [\bar{\boldsymbol{\ell}}]_1 + \mathbf{a} \circ [\mathbf{p}]_1$
  3. *Compute* $[\mathbf{q}^{\mathsf{b}}]_1 = \boldsymbol{\eta} * (\mathbf{b} \circ [\bar{\boldsymbol{\ell}}]_1) + (\boldsymbol{\eta}' * \mathbf{b}) \circ [\bar{\boldsymbol{\ell}}]_1 + \mathbf{b} \circ [\mathbf{p}]_1$
  4. *Output* $D = ([\mathbf{q}^{\mathsf{a}}]_1, [\mathbf{q}^{\mathsf{b}}]_1)$
- *SemiDynark.*$\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{rand})$*:*
  1. $\pi \leftarrow Groth16.\mathcal{P}(Groth16.\mathbf{pk}, \mathbb{x}, \mathbb{w})$ *where* $\mathsf{rand} = (r, s)$ *used in* $Groth16.\mathcal{P}$.
  2. *Output* (π, rand).
- *SemiDynark.*$\mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \to \pi'$*:*
  1. *Let* $z = (\mathbb{x}, \mathbb{w}), z' = (\mathbb{x}', \mathbb{w}')$ *and* $\Delta^z = z' - z$.
  2. *Compute* $\Delta^A = A \cdot \Delta^z, \Delta^B = B \cdot \Delta^z \in \mathbb{F}^n$. *Let* I_A *and* I_B *be the nonzero indices set.*
  3. *Compute* $\tilde{\Delta}^A = SparseConv_{\boldsymbol{\eta}'}(\Delta^A, I_A \cup I_B), \tilde{\Delta}^B = SparseConv_{\boldsymbol{\eta}'}(\Delta^B, I_A \cup I_B) \in \mathbb{F}^n$.
  4. *Sample* $r', s' \leftarrow \mathbb{F}$.
  5. *Compute* $[A']_1 = [A]_1 + (r' - r)[\delta]_1 + \sum_{i\in I_z} \Delta_i^z [\mathbf{u}_i]_1$.
  6. *Compute* $[B']_2 = [B]_2 + (s' - s)[\delta]_2 + \sum_{i\in I_z} \Delta_i^z [\mathbf{v}_i]_2$ *and* $[B']_1$ *similarly.*
  7. *Compute* $[\Delta^C_{lin}]_1 = s'[A']_1 - s[A]_1 + r'[B']_1 - r[B]_1 - (r's' - rs)[\delta]_1 + \sum_{i\in[k,m]\cap I_z} \Delta_i^z [\mathbf{t}^{\mathsf{p}}_{i-k}]_1$.
  8. *Compute* $[\Delta^C_{cross}]_1 = \sum_{i\in I_A\cup I_B} (\Delta_i^A \tilde{\Delta}_i^B + \Delta_i^B \tilde{\Delta}_i^A)[\bar{\boldsymbol{\ell}}_i]_1 + \sum_{i\in I_A\cap I_B} \Delta_i^A \Delta_i^B [\mathbf{p}_i]_1$.
  9. *Compute* $[\Delta^C_{proj}]_1 = \sum_{i\in I_A} \Delta_i^A [\mathbf{q}^{\mathsf{b}}_i]_1 + \sum_{i\in I_B} \Delta_i^B [\mathbf{q}^{\mathsf{a}}_i]_1$
  10. *Compute* $[C']_1 = [C]_1 + [\Delta^C_{lin}]_1 + [\Delta^C_{cross}]_1 + [\Delta^C_{proj}]_1$
  11. *Output* $\pi' = ([A']_1, [C']_1, [B']_2)$
- *SemiDynark.*$\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$*:*
  1. *Output* $Groth16.\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi)$

Improving the prover time of Groth16. With our better understanding of quotient polynomials, we also propose a better algorithm to generate the Groth16 proof with only 4 DFTs in Appendix A.

5 Fully-Dynamic zkSNARK

Semi-dynamic zkSNARKs allow efficient proof updates from an initial witness $\mathbb{w}$ and its preprocessed data D. Their efficiency, however, relies on the assumption that the updated witness remains close to the initial one. They become inefficient in scenarios where a long sequence of updates makes the witness deviate significantly from the initial witness. To address the problem, in this section, we present the construction of our fully-dynamic zkSNARK.

5.1 Definition

Definition 5.1 (Fully-dynamic zkSNARKs)**.** *A fully-dynamic zero-knowledge succinct non-interactive argument of knowledge (fully-dynamic zkSNARK) for an indexed relation $\mathcal{R}$ consists of the following PPT algorithms $\Psi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{U}, \mathcal{V})$:*

- $\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$*: On 1^λ and index $\mathbb{i}$, outputs a public key* $\mathbf{pk}$ *and a verifier key* $\mathbf{vk}$*.*
- $\mathcal{K}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to D$*: On public key* $\mathbf{pk}$*, instance* $\mathbb{x}$ *and witness* $\mathbb{w}$*, outputs preprocessed data D.*
- $\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to (\pi, \mathsf{rand})$*: On public key* $\mathbf{pk}$*, instance* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_\mathbb{i}$*, outputs the proof π and the randomness* rand*.*
- $\mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \to (\pi', D', \mathsf{rand}')$*: On public key* $\mathbf{pk}$ *and valid extended witnesses* $(\mathbb{x}, \mathbb{w}), (\mathbb{x}', \mathbb{w}') \in \mathcal{R}_\mathbb{i}$*, proof π and preprocessed data D, outputs updated proof π' and updated data D'.*
- $\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$*: On verifier key* $\mathbf{vk}$*, instance* $\mathbb{x}$ *and proof π outputed by $\mathcal{P}$ or $\mathcal{U}$, outputs accept or reject.*

A fully-dynamic zkSNARK Ψ satisfies the following properties:

- ***Completeness:*** *It is perfectly complete if for any $\mathbb{i}$, any sequence $(\mathbb{i}, \mathbb{x}_0, \mathbb{w}_0), \ldots, (\mathbb{i}, \mathbb{x}_k, \mathbb{w}_k) \in \mathcal{R}$:*

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi_0) = 1 \wedge & (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \mathcal{V}(\mathbf{vk}, \mathbb{x}_k, \pi_k) = 1 & D_0 \leftarrow \mathcal{K}(\mathbf{pk}, \mathbb{x}_0, \mathbb{w}_0) \\ & (\pi_0, \mathsf{rand}_0) \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}_0, \mathbb{w}_0) \\ & \text{for } i = 1, 2, \ldots, k, (\pi_i, D_i, \mathsf{rand}_i) \leftarrow \\ & \mathcal{U}(\mathbf{pk}, D_{i-1}, \mathbb{x}_{i-1}, \mathbb{w}_{i-1}, \mathbb{x}_i, \mathbb{w}_i, \pi_{i-1}, \mathsf{rand}_{i-1}) \end{array}\right] = 1$$

- ***Knowledge Soundness:*** *It is knowledge sound if for any PPT adversary $\mathcal{A}$ and for any $\mathbb{i}$ there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that the following probability is $\leq \mathsf{negl}(\lambda)$*

$$\Pr\left[\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) = 1 \wedge (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ ((\mathbb{x}, \pi); \mathbb{w}) \leftarrow (\mathcal{A} \parallel \mathcal{E}_\mathcal{A})(\mathbf{pk}, \mathbf{vk}) \end{array}\right]$$

- ***Perfect Zero-Knowledge:*** *It is zero-knowledge if there exists PPT simulators $\mathcal{S}$ such that for any adversarial distinguisher $\mathcal{D}$, for any $\mathbb{i}$, any sequence $(\mathbb{i}, \mathbb{x}_0, \mathbb{w}_0), \ldots, (\mathbb{i}, \mathbb{x}_k, \mathbb{w}_k) \in \mathcal{R}$:*

$$\Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}_0, \pi_0) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{S}(1^\lambda, \mathbb{i}) \\ \pi_0 \leftarrow \mathcal{S}(\mathbf{pk}, \mathbf{vk}, \mathbb{x}_0) \end{array}\right]$$

$$= \Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}_0, \pi_0) = 1 \;\middle|\; \begin{array}{l} (\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \pi_0 \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}_0, \mathbb{w}_0) \end{array}\right]$$

and

$$\Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}_k, \pi_k) = 1 \;\middle|\; \begin{array}{l}(\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ \pi_k \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}_k, \mathbb{w}_k)\end{array}\right]$$

$$= \Pr\left[\mathcal{D}(\mathbf{vk}, \mathbb{x}_k, \pi_k) = 1 \;\middle|\; \begin{array}{l}(\mathbf{pk}, \mathbf{vk}) \leftarrow \mathcal{G}(1^\lambda, \mathbb{i}) \\ (\pi_0, \mathsf{rand}_0) \leftarrow \mathcal{P}(\mathbf{pk}, \mathbb{x}_0, \mathbb{w}_0) \\ D_0 \leftarrow \mathcal{K}(\mathbf{pk}, \mathbb{x}_0, \mathbb{w}_0) \\ \textit{for } i = 1, 2, \ldots, k, (\pi_i, D_i, \mathsf{rand}_i) \leftarrow \\ \mathcal{U}(\mathbf{pk}, D_{i-1}, \mathbb{x}_{i-1}, \mathbb{w}_{i-1}, \mathbb{x}_i, \mathbb{w}_i, \pi_{i-1}, \mathsf{rand}_{i-1})\end{array}\right]$$

- ***Succinctness:*** *The proof size $|\pi|$ is sublinear in the witness size $|\mathbb{w}|$.*
- ***Updatability:*** *Let $|\Delta_\mathbb{x}|, |\Delta_\mathbb{w}|$ denote the Hamming distances between $\mathbb{x}, \mathbb{x}'$ and $\mathbb{w}, \mathbb{w}'$. If $|\Delta_\mathbb{x}| + |\Delta_\mathbb{w}| = o(|\mathbb{x}| + |\mathbb{w}|)$, the running time of update $\mathcal{U}$ should be sublinear in the running time of the prover $\mathcal{P}$.*

Compared to Definition 4.1 of semi-dynamic zkSNARKs, the only difference in Definition 5.1 is that the update algorithm $\mathcal{U}$ also updates the preprocessed data D to D' for the updated extended witness $z' = (\mathbb{x}', \mathbb{w}')$, so that D' can be utilized for future updates based on z', making the scheme fully dynamic. Our construction also needs the updated randomness rand' to achieve zero-knowledge, and we denote it separately from D' in our definition. The completeness and the zero-knowledge incorporate a sequence of updates, but are essentially similar to Definition 4.1. Our definition is almost the same as the one in [WPSP25b], except that our zero-knowledge requires that the proof generated by $\mathcal{U}$ is indistinguishable from the proof generated by $\mathcal{P}$.

5.2 Fully-Dynamic zkSNARK based on Groth16

Fully-dynamic from semi-dynamic in the amortized setting. A naive approach to build fully-dynamic zkSNARKs from semi-dynamic zkSNARKs is to recompute the preprocessed data D' for the extended witness z' after its distance becomes b-far away from the initial extended witness z. This ensures that the distance of any new witness in subsequent updates is at most $O(b)$ away from the most recent preprocessed witness with the corresponding preprocessed data. Then using our semi-dynamic zkSNARK in the previous section, the update cost is at most $O(b)$ $\mathbb{G}$ and $O(b \log^2 b)$ $\mathbb{F}$. Meanwhile, the preprocessing cost is amortized to $O(\frac{n \log n}{b})$ $\mathbb{G}$ per unit-distance update. By setting $b = \sqrt{n \log n}$, the total amortized update time is $O(\frac{dn \log n}{b} + b)\mathbb{G} + O(b \log^2 b)\mathbb{F} = O(d\sqrt{n \log n})\mathbb{G} + O(\sqrt{n \log n} \log^2 n)\mathbb{F}$.

However, the amortized approach can lead to instability of the update time, which is undesirable in applications that require predictable update costs such as blockchain systems. We show how to construct fully dynamic zkSNARKs in the de-amortized setting below.

De-amortizing the updates. We adopt the ideas in [RT97] for dynamic algorithms to de-amortize the update cost. We divide a sequence of extended witnesses $(z_0, z_1, z_2, z_3, \ldots)$ into blocks $(B_0, B_1, B_2, \ldots)$, such that the distance between the first witness in block B_i, denoted by z_{B_i}, and the first witness in block B_{i+1}, denoted by $z_{B_{i+1}}$, is at most b for all blocks. If the distance of any single update is larger than b, then we simply rerun the preprocessing of our semi-dynamic zkSNARK and this case is consistent with the complexity of our fully-dynamic zkSNARK as we show below. Therefore, for simplicity, we assume the distance of each update is less than b and the sequence can always be divided into the blocks above.

In the naive approach above for the amortized setting, the updater basically computes the full preprocessed data for the first witness z_{B_i} of every block B_i, and uses it to compute the proofs for all witnesses in the block. This computation can be de-amortized using the two algorithms below:

- Stepwise DFT. We first observe that the main computations in the preprocessing—namely the DFTs and iDFTs—are *smooth*, meaning that they can be divided into b smaller consecutive steps. We use the classical butterfly network DFT algorithm, as shown in Protocol 3. It consists of $\log n$ rounds, where in each round the elements are paired by their indices and processed through n butterfly gates, and the butterfly gates within a round are independent of each other. The stepwise DFT step-DFT$_b(t, \mathbf{x})$ executes only $\frac{n \log n}{b}$ of these gates per step, where b is the number of steps to break the DFT into, $\mathbf{x}$ is the input of the DFT, and t is the index of the current step. By running step-DFT$_b(t, \mathbf{x})$ for $t = 0$ to $b-1$, the output is exactly DFT$(\mathbf{x})$. The same holds for step-iDFT.

  With this stepwise DFT algorithm, we present the preprocessing of our semi-dynamic zkSNARK in the stepwise manner in Protocol 3. The main computations are step-DFT and step-iDFT in Steps 3 and 5. However, the preprocessing also includes element-wise products, which can be trivially divided into any number of steps. That is why we need to introduce b_0, b_1, b_2, b_3 such that $b = b_0 + 2b_1 + b_2 + b_3$ to balance the computation of different phases. d is the distance between two consecutive witnesses, and the complexity of the stepwise preprocessing is $O(\frac{dn \log n}{b})\mathbb{G}$. **state** denotes the intermediate results to keep between the steps.

  We emphasize that the butterfly gates within one step of step-DFT can be fully parallelized as long as $b \geq \log n$, so that $\frac{n \log n}{b} \leq n$ and a step never spans two rounds, which is the case in our protocol. Thus the de-amortization does not harm the parallelization of the computations as shown in Section 6.

- Block offset. To utilize the step-by-step update above, instead of computing and using the preprocessed data D_{B_i} of the first witness z_{B_i} for every block B_i, the updater uses the preprocessed data $D_{B_{i-1}}$ of the first witness $z_{B_{i-1}}$ of the previous block B_{i-1} to update the proofs for all witnesses in the current block B_i. At the same time, the updater computes the preprocessed data D_{B_i} for z_{B_i} step by step in the current block B_i. In this way, the updater *buys a block of time with a one block size distance offset*, within which it can incrementally compute the preprocessed data D_{B_i} for z_{B_i}. Once the updater completes the updates in the current block B_i, it also obtains the preprocessed data D_{B_i} of the witness z_{B_i}, which can be used to update the proofs for all witnesses in the next block B_{i+1}.

  Because the distance between $z_{B_{i-1}}$ and z_{B_i} is at most b, the distance between any witness in block B_i and $z_{B_{i-1}}$ is at most $2b$. Therefore, the update time remains $O(b)\mathbb{G} + O(b \log^2 b)\mathbb{F}$ using the preprocessed data $D_{B_{i-1}}$.

Combining the two algorithms, we obtain a fully-dynamic zkSNARK in the de-amortized setting with an update time of $O(\frac{dn \log n}{b} + b)\mathbb{G} + O(b \log^2 b)\mathbb{F}$, which is the same as the amortized cost in the naive approach. Note that the sequence of witnesses divided into blocks is merely to illustrate the idea of our construction. Our update algorithm only needs to know the new witness, the existing preprocessed data, the pre-defined parameter b and the state of the stepwise preprocessing, but not any future witnesses.

Partially updating preprocessed data. Finally, we introduce one more optimization to improve the complexity of the update time. We observe that although the size of the preprocessed data for any witness is $O(n)$, the updater only uses part of it to update the proof. That is, only $O(d)$ elements in the preprocessed data D_z of witness z are used by the updater to update the proof from z to z' within a block B_i. This is exactly how we get $O(d)$ group operations in the update time of our semi-dynamic zkSNARK. This observation suggests that although we cannot compute the full preprocessed data D_z for every z, it is possible to compute the part of D_z that will be used by the updater to update the proof for z'. Performing this partial updates from $D_{B_{i-1}}$ and then using it to generate the proof of z' turns out to be more efficient than generating the proof of z' using $D_{B_{i-1}}$ directly.

Protocol 3 (Stepwise DFT and Stepwise Preprocessing)**.**

- *DFT*($\mathbf{x}$)*:*
  1. *All indices are in reverse-bit form*
  2. *For all* $i \in [n]$*: if* $i < rev(i)$*, swap* x_i *and* $x_{rev(i)}$
  3. *For* $i \in [\log n]$*:*
     for $j \in [n]$*: compute Butterfly*$(\mathbf{x}, i, j)$

- *step-DFT*$_b(t, \mathbf{x})$*:*
  1. *All indices are in reverse-bit form*
  2. *Parse* $t = i \cdot n + j$ *where* $i \in [\log n], j \in [n]$
  3. *For* $k \in [\frac{n \log n}{b}]$*: compute Butterfly*$(\mathbf{x}, i, j + k)$
  4. *increase* t *by* 1

- *Butterfly*$(\mathbf{x}, i, j)$*:*
  1. *let* $k = j + 2^{i-1}$
  2. *compute* $\mathbf{x}'_j = \omega^{\frac{jn}{2^i}} \cdot \mathbf{x}_j + \mathbf{x}_k$ *and* $\mathbf{x}'_k = \omega^{\frac{jn}{2^i}} \cdot \mathbf{x}_j - \mathbf{x}_k$
  3. *let* $\mathbf{x}_j = \mathbf{x}'_j$ *and* $\mathbf{x}_k = \mathbf{x}'_k$

- *SemiDynark.step-*$\mathcal{K}(\mathbf{pk}, \mathbf{state}, d) \to \mathbf{state}'$*:*
  1. *Set* b_0, b_1, b_2, b_3 *accordingly to average the cost of each step*
  2. *If* $t \in [0, t_0]$ *where* $t_0 = b_0$*:*
     compute $[\mathbf{al}] = \mathbf{a} \circ [\boldsymbol{\ell}]$*,* $[\mathbf{bl}] = \mathbf{b} \circ [\boldsymbol{\ell}]$*,* $[\mathbf{ap}] = \mathbf{a} \circ [\mathbf{p}]$ *and* $[\mathbf{bp}] = \mathbf{b} \circ [\mathbf{p}]$ *in a* b_0*-step manner for* d *times with time* t*.*
  3. *If* $t \in [t_0, t_1]$ *where* $t_1 = t_0 + b_1$*:*
     compute step-DFT$_{b_1}$ *over* $[\mathbf{al}], [\mathbf{bl}] \in \mathbb{G}^n$ *and* $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ *for* d *times with time* t
  4. *If* $t \in [t_1, t_2]$ *where* $t_2 = t_1 + b_2$*:*
     compute $[\mathbf{al}] = \widehat{\boldsymbol{\eta}} \circ [\mathbf{al}]$*,* $[\mathbf{bl}] = \widehat{\boldsymbol{\eta}} \circ [\mathbf{bl}]$*,* $\mathbf{b} = \widehat{\boldsymbol{\eta}}' \circ \mathbf{a}$ *and* $\mathbf{b} = \widehat{\boldsymbol{\eta}}' \circ \mathbf{b}$ *in a* b_2*-step manner for* d *times with time* t*.*
  5. *If* $t \in [t_2, t_3]$ *where* $t_3 = t_2 + b_1$*:*
     compute step-iDFT$_{b_1}$ *over* $[\mathbf{al}], [\mathbf{bl}] \in \mathbb{G}^n$ *and* $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ *for* d *times with time* t
  6. *If* $t \in [t_3, t_4]$ *where* $t_4 = t_3 + b_3$*:*
     compute $[\mathbf{q}^a] = [\mathbf{al}] + \mathbf{a} \circ [\boldsymbol{\ell}] + [\mathbf{ap}]$ *and* $[\mathbf{q}^b] = [\mathbf{bl}] + \mathbf{b} \circ [\boldsymbol{\ell}] + [\mathbf{bp}]$ *in a* b_3*-step manner with time* t
  7. *Output* $\mathbf{state}'$

Protocol 4 (Partial Update for Preprocessing)**.**

- *SemiDynark.partial-*$\mathcal{K}(\mathbf{pk}, semiD, z, z', I) \to semiD'$
  1. *Compute* $\Delta^a = A \cdot (z' - z)$ *and* $\Delta^b = B \cdot (z' - z)$
  2. *Compute* $I_{\delta^a} = A \cdot I$ *and* $I_{\delta^b} = B \cdot I$ *by parsing indices as vectors.*
  3. $\forall i \in I_{\delta^b}, [\Delta_i^{q^a}]_1 = \sum_{j \in I_{\Delta^a}} \Delta_j^a [\bar{\boldsymbol{\ell}}_j]_1 \eta_{i-j} + \Delta_j^a \eta_{i-j} [\bar{\boldsymbol{\ell}}_i]_1 + \mathbb{1}(j \in I_{\delta^b}) \Delta_i^a [\mathbf{p}_i]_1$
  4. $\forall i \in I_{\delta^a}, [\Delta_i^{q^b}]_1 = \sum_{j \in I_{\Delta^b}} \Delta_j^b [\bar{\boldsymbol{\ell}}_j]_1 \eta_{i-j} + \Delta_j^b \eta_{i-j} [\bar{\boldsymbol{\ell}}_i]_1 + \mathbb{1}(j \in I_{\delta^a}) \Delta_i^b [\mathbf{p}_i]_1$
  5. *Compute* $[\mathbf{q}^{a\prime}]_1 = [\mathbf{q}^a]_1 + [\Delta^{q^a}]_1$ *and* $[\mathbf{q}^{b\prime}]_1 = [\mathbf{q}^b]_1 + [\Delta^{q^b}]_1$
  6. *Output* $D' = ([\mathbf{q}^{a\prime}]_1, [\mathbf{q}^{b\prime}]_1)$

To see this, recall that the D consists of projection quotients $\mathbf{q}^a$ and $\mathbf{q}^b$, where the i-th element can be written as

$$[\mathbf{q}_i^a]_1 = \sum\nolimits_{j \in [n]} \left(\mathbf{a}_j [\bar{\boldsymbol{\ell}}_j]_1 \eta_{i-j}\right) + \left(\sum\nolimits_{j \in [n]} \mathbf{a}_j \eta'_{i-j}\right) [\bar{\boldsymbol{\ell}}_i]_1 + \mathbf{a}_i [\mathbf{p}_i]_1 \,,$$

which is a linear combination of elements in $\mathbf{a}$. It means that for each element of $[\mathbf{q}^a]_1$ used by the updater, the contribution of each change in $\mathbf{a}$ can be computed in constant time. The same applies to $[\mathbf{q}^b]_1$. We call this property the locality of the preprocessing. Based on this locality, we can efficiently compute the partial preprocessed data D_z from $D_{B_{i-1}}$ using Protocol 4. Since we only need to update $[\mathbf{q}^a]_1$ on $I_{z'-z}$ indices, the complexity is $O(db)\mathbb{G}$.

Using the partial updates, the update time becomes $O(\frac{dn \log n}{b} + db)\mathbb{G} + O(d \log^2 d)\mathbb{F}$. This is better than the complexity above as in our case that b is much larger than d. For example, when setting $b = \sqrt{n \log n}$, the complexity is reduced to $O(d\sqrt{n \log n})\mathbb{G} + O(d \log^2 d)\mathbb{F}$, from $O(d\sqrt{n \log n})\mathbb{G} + O(\sqrt{n \log n} \log^2 n)\mathbb{F}$.

Our fully-dynamic zkSNARK is presented formally in Protocol 5. Algorithms $\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}$ and $\mathbf{pk}, \mathbf{vk}$ of our fully-dynamic zkSNARK are exactly the same as our semi-dynamic zkSNARK in Protocol 2.

Theorem 5.2. *Protocol 5 is a fully-dynamic zkSNARK by Definition 5.1 under the q-DLOG assumption in the AGM model.*

Proof sketch. The protocol is essentially an algorithm to periodically compute the preprocessed data for a new witness in the de-amortized setting. The completeness follows from the correctness of step-$\mathcal{K}$, partial-$\mathcal{K}$ and the completeness of the semi-dynamic scheme as explained above. The public key, proof and verifier algorithm remains exactly the same as *SemiDynark*. Therefore, both the knowledge soundness and the zero-knowledge follow the same argument as the proof of Theorem 4.3.

Compiling semi-dynamic zkSNARK to full-dynamic zkSNARK. Our protocol also suggests a way to generically compile a semi-dynamic zkSNARK to a fully-dynamic zkSNARK in the de-amortized setting. Suppose that the running time of *SemiDynark*.$\mathcal{K}$ is $T(\mathcal{K})$ and can be computed stepwise. Let $T(\mathcal{U})$ be the running time of *SemiDynark*.$\mathcal{U}$ per unit distance. By running Protocol 5 (excluding the partial update optimization in Step 2), we are able to construct a fully dynamic zkSNARK with update time $O(d(\frac{T(\mathcal{K})}{b} + b \cdot T(\mathcal{U})))$. By setting $b = \sqrt{\frac{T(\mathcal{K})}{T(\mathcal{U})}}$, the update time is $O(d\sqrt{T(\mathcal{K})T(\mathcal{U})})$.

Protocol 5 (Fully-Dynamic zkSNARK)**.**

- $\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}$ *of FullyDynark are exactly the same as SemiDynark in Protocol 2.*
- *FullyDynark*.$\mathcal{U}(\mathbf{pk}, D, \mathbb{x}, \mathbb{w}, \mathbb{x}', \mathbb{w}', \pi, \mathsf{rand}) \to (\pi', \mathsf{rand}', D')$:
  It maintains the following data in D*:*
  - t*: index of steps*
  - $z_{B_{i-1}}$*: the first witness of the previous block* B_{i-1}
  - z_{B_i}*: the first witness of the current block* B_i
  - $semiD$*: the preprocessed data for* $z_{B_{i-1}}$
  - **state***: the intermediate result of preprocessing for* z_{B_i}

  The updater works as follows:

  1. *let* $z = (\mathbb{x}, \mathbb{w})$ *and* $z' = (\mathbb{x}', \mathbb{w}')$
  2. $semiD' \leftarrow SemiDynark.partial\text{-}\mathcal{K}(\mathbf{pk}, semiD, z_{B_{i-1}}, z, I_{z'-z})$
  3. $(\pi', \mathsf{rand}') \leftarrow SemiDynark.\mathcal{U}(\mathbf{pk}, semiD', z, z', \pi, \mathsf{rand})$
  4. $\mathbf{state}' \leftarrow SemiDynark.step\text{-}\mathcal{K}(\mathbf{pk}, \mathbf{state}, |I_{z'-z}|)$
  5. *Increment* t *by* $|I_{z'-z}|$ *to get* t'*.*
     When $t' = b$*, the updater moves from* B_i *to* B_{i+1} *by setting*
     - $t' \leftarrow 0$
     - $z_{B_{i-1}} \leftarrow z_{B_i}$ *and* $z_{B_i} \leftarrow z'$
     - *Extract* $semiD'$ *from completed* $\mathbf{state}'$
     - *Refresh* $\mathbf{state}'$ *to be* z' *which is the input for the future stepwise preprocessing*
  6. *Output* $(\pi', \mathsf{rand}', D')$

6 Evaluations

We have fully implemented our dynamic zkSNARK scheme, Dynark, and we evaluate its performance in this section.

Software and hardware. Dynark is implemented in Rust on top of the Groth16 implementation from the arkworks library [ac22a]. Our implementation consists of around 5500 lines of code. The bilinear pairing is instantiated on the BLS12-381 elliptic curve in all of our benchmarks.

All experiments were executed on an AWS c7i.48xlarge machine with 192 vCPUs. Our implementation supports full parallelization utilizing all vCPUs, and we report the performance of both parallelized and serialized implementations in this section. Numbers for setup and preprocessing are averaged over 3 executions, all other numbers are averaged over 10 executions.

6.1 Performance of Dynark and Comparison with Gorth16

Generation of R1CS. In our experiments, we set the size of the (extended) witness to be the same as the number of R1CS constraints $m = n$. For the convenience of measuring the concrete running time, we set $|\Delta_Z| = 2d, |\Delta_A| = |\Delta_B| = d$, and generate random R1CS instances and updates with these constraints[4]. We also provide the calculation of the update time in Table 2 when $|\Delta_Z|, |\Delta_A|, |\Delta_B|$ are not necessarily the same.

[4]Setting $|\Delta_Z| = |\Delta_A| = |\Delta_B| = d$ does not allow any meaningful update of the R1CS witness.

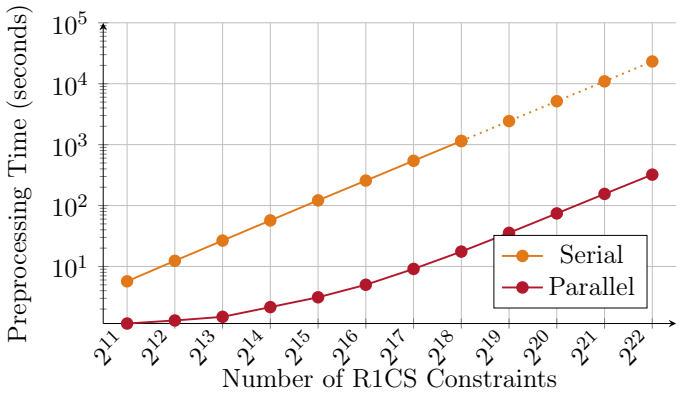

Figure 1: Preprocessing Time

Scheme	**Group Operation**
Semi-Dynamic	$(4\lvert\Delta_Z\rvert + 2\lvert\Delta_A\rvert + 2\lvert\Delta_B\rvert)\mathbb{G}$
Fully-Dynamic	$(4\lvert\Delta_Z\rvert + b + 3\lvert\Delta_A\rvert + 3\lvert\Delta_B\rvert + \frac{n(6+2\log n)}{b}\lvert\Delta_Z\rvert)\mathbb{G}$

Table 2: Number of group operations to update a proof in Dynark

Preprocessing time. Dynark has a pre-processing $\mathcal{K}$ on the witness and its running time is presented in Figure 1. As shown in the figure, for $n = 2^{20}$ constraints, it takes 74.3 seconds to compute the preprocessed data with full parallelization. For the serialized version, we only run it up to $n = 2^{18}$ due to the long running time and the dotted line denotes estimation. For $n = 2^{20}$, the serial time is estimated to be 5,158 seconds. This high overhead comes from the $O(n \log n)$ group operations and is consistent with the numbers reported in prior work (e.g.,[DGP+24, Table 3] and the reference implementation of cq [cq-22]). Fortunately, the preprocessing can be significantly accelerated with parallelization using the parallelized DFT and iDFT of group elements implemented in arkworks [ac22b]. We believe the parallel time is reasonable for one-time preprocessing in practice.

Before preprocessing, Dynark also has a trusted setup which generates a few additional elements compared to Groth16 (see $\mathit{SemiDynark}.\mathcal{G}$ in Protocol 2). For $n = 2^{20}$, the trusted setup of Groth16 takes 1.535 seconds, and Dynark is only slightly slower than Groth16 with access to the trapdoor τ. Alternatively, Dynark can take the prover key of any existing Groth16 scheme and compute the additional elements without knowing the trapdoor, which incurs a higher overhead of 59.4 seconds.

Update time. We first give the accurate calculation of the number of group operations per update in Table 2. $|\Delta_Z|, |\Delta_A|, |\Delta_B|$ are the number of updated elements in $z, A \cdot z, B \cdot z$, and b is the number of steps in the fully-dynamic scheme. Table 2 provides a way to precisely calculate the update time when $|\Delta_Z|, |\Delta_A|, |\Delta_B|$ are not necessarily the same in an update. The update time does not depend on $|\Delta_C|$ at all because Δ_C is fully determined by Δ_A, Δ_B to keep the witness valid, and does not appear in the update algorithm at all.

We only report the number of group operations in the table because they are the dominating cost. In our implementation, we use the naive algorithm with $O(d^2)$ field operations to compute the constants of the cross term. This is because for the size of updates we consider in the experiments, the naive algorithm is faster than the sparse convolution with complexity $O(d \log^2 d)$, and neither of them is the dominating cost compared to the group operations.

For the concrete running time, we first fix $d = 1$ and vary the number of R1CS constraints. Figure 2 shows the performance of Dynark and the comparison with Groth16, which computes new proofs from scratch. As shown in the figure, our update time is significantly faster than

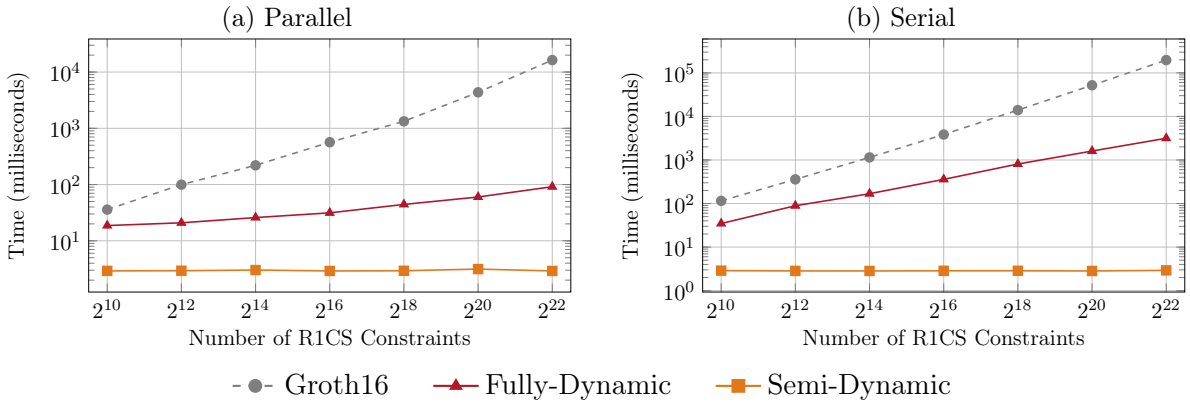

Figure 2: Update time of Dynark with $d = 1$.

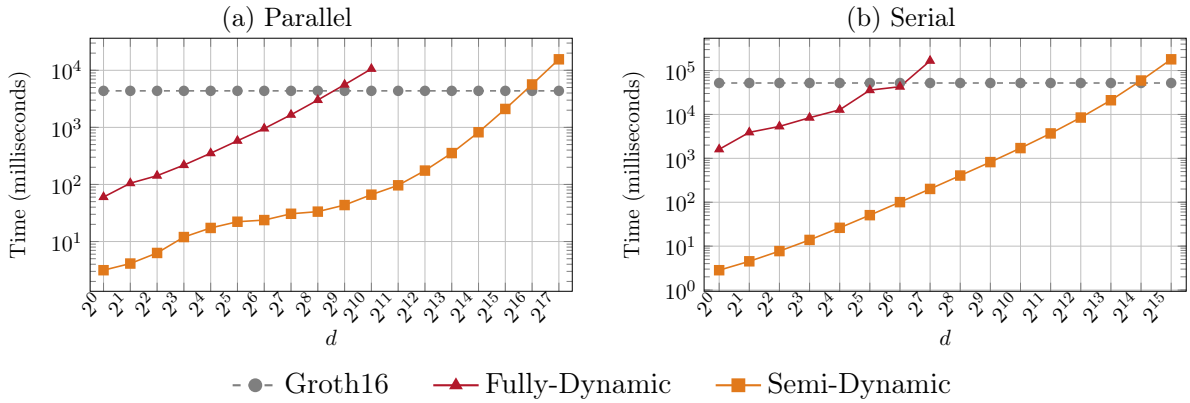

Figure 3: Update time of Dynark with different d and $n = 2^{20}$.

Groth16. With parallelization, for $n = 2^{20}$, it merely takes 3 milliseconds to update the proof in our semi-dynamic zkSNARK, and 60 milliseconds in our fully-dynamic zkSNARK. These are 1433× and 73× faster than Groth16, which takes 4.4 seconds to compute the proof from scratch. We also report the serial time in Figure 2b. For $n = 2^{20}$, it takes 3 milliseconds for our semi-dynamic zkSNARK to update the proof, and 1.6 seconds for our fully-dynamic zkSNARK, while it takes 52 seconds for Groth16 to generate the proof. The speedups are 17334× and 33×, respectively. It demonstrates that our speedup does not only come from the parallelization. In fact, our semi-dynamic zkSNARK is so fast that the parallelization does not help at all.

To better understand the update time of Dynark with respect to different size of the updates, we fix the size of $n = 2^{20}$ and vary d. Figure 3 shows the performance of Dynark, with the fixed prover time of Groth16 for $n = 2^{20}$ as a reference. As shown in the figure, our semi-dynamic zkSNARK is extremely fast. With parallelization, it merely takes 2.1 seconds to update a proof with $d = 2^{15}$, which is still more than 2× faster than Groth16. The break-even point is between 2^{15} and 2^{16}. The update time of our fully-dynamic zkSNARK grows linearly with d, which is consistent with our complexity analysis. It takes 3.0 seconds to update the proof with $d = 2^8$, which is 1.4× faster than Groth16. The break-even point is between 2^8 and 2^9. Figure 3b shows similar behavior for the serial time, where the break-even points are between 2^{13} and 2^{14} for our semi-dynamic zkSNARK and between 2^6 and 2^7 for our fully-dynamic zkSNARK.

Proof size and verifier time. The proof size of our schemes is 192 bytes and the verifier time is 4.4 milliseconds, which are exactly the same as Groth16.

Scheme	$\log_2 n$	Update Time(ms)	Verifier Time(ms)	Proof Size
Dynaverse	18	70	95	96 KB
	20	80	125	192 KB
	22	120	179	384 KB
Semi-Dynamic	18	2.9	4.4	192 B
	20	3.1		
	22	3.0		
Fully-Dynamic	18	44.3		
	20	60.1		
	22	91.4		

Table 3: Update time of Dynark and [WPSP25a], $d = 1$.

6.2 Comparison with Prior Work

We further compare the performance of Dynark with the dynamic zkSNARK scheme in [WPSP25b]. As the implementation of [WPSP25b] is not available, we compare with the numbers reported in an earlier version of the paper [WPSP25a, Table 3], which were obtained on an AWS machine with the same configuration as our machine and with full parallelization.

Table 3 shows the comparison with $d = 1$. Note that Dynark works for R1CS and the scheme in [WPSP25b] is for Plonkish constraints. R1CS is usually more compact than Plonkish constraints for the same computation. However, we still use the same n to compare with [WPSP25b]. As shown in the table, the major advantage of Dynark is that the proof size is significantly smaller than [WPSP25b]. It is merely 192 bytes regardless of n. This is 5,000–20,000× smaller than [WPSP25b]. The verifier time is also faster than [WPSP25b] for all n. In terms of the update time, our semi-dynamic zkSNARK is 23–40× faster than [WPSP25b], while our fully-dynamic zkSNARK is roughly 1.3× faster than [WPSP25b].

Acknowledgments

We thank the authors of [WPSP25b] for helpful discussions on dynamic zkSNARKs. We thank Zhongtang Luo for pointing out the dynamic algorithms and lower bounds for convolutions in [FHM01]. This work is supported by the National Science Foundation (NSF) under Grant No. 2401481 and a Google Research Scholar Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these institutes.

References

[ac22a] arkworks contributors. ark-groth16. `https://github.com/arkworks-rs/groth16`, 2022.

[ac22b] arkworks contributors. `arkworks` zksnark ecosystem. `https://arkworks.rs`, 2022.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104, 2017.

[APKP24] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4316–4330, 2024.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[BC24] Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. *Cryptology ePrint Archive*, 2024.

[BFK+24] Alexander R Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In *Annual International Cryptology Conference*, pages 276–307. Springer, 2024.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 677–706. Springer, 2020.

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Advances in Cryptology – ASIACRYPT 2021*, pages 65–97, 2021.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*, 2018.

[BSCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019*, pages 103–128, 2019.

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. *Cryptology ePrint Archive*, 2022.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768, 2020.

[CJL+24] Kostas Kryptos Chalkias, Charanjit Jutla, Jonas Lindstrom, Varun Madathil, and Arnab Roy. Improved polynomial division in cryptography. Cryptology ePrint Archive, Paper 2024/1279, 2024.

[cq-22] cq reference implementation. https://github.com/geometryxyz/cq, 2022.

[DCB25] Trisha Datta, Binyi Chen, and Dan Boneh. Veritas: Verifying image transformations at scale. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 4606–4623. IEEE, 2025.

[DGP+24] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. Batching-efficient ram using updatable lookup arguments. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4077–4091, 2024.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763, 2022.

[FHM01] Gudmund Skovbjerg Frandsen, Johan P Hansen, and Peter Bro Miltersen. Lower bounds for dynamic algebraic problems. *Information and Computation*, 171(2):333–349, 2001.

[FK23] Dankrad Feist and Dmitry Khovratovich. Fast amortized kzg proofs. *Cryptology ePrint Archive*, 2023.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, 1987.

[GAZ+22] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. {Zero-Knowledge} middleboxes. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4255–4272, 2022.

[GGJ+23] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1880–1894, 2023.

[GGK+25] Sanjam Garg, Aarushi Goel, Dimitris Kolonelos, Sina Shiehian, and Rohit Sinha. Split prover zero-knowledge snarks. In *Public-Key Cryptography – PKC 2025*, pages 353–380, 2025.

[GGS87] Apostolos Gerasoulis, Michael D Grigoriadis, and Liping Sun. A fast algorithm for trummer's problem. *SIAM journal on Scientific and Statistical Computing*, 8(1):s135–s138, 1987.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In *Annual International Cryptology Conference*, pages 193–226. Springer, 2023.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, 2016.

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing*, 1992.

[LJGK25] Zhongtang Luo, Yanxue Jia, Alejandra Victoria Ospina Gracia, and Aniket Kate. Cauchyproofs: Batch-updatable vector commitment with easy aggregation and application to stateless blockchains. pages 1947–1963, 2025.

[LXZ21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.

[Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, 1987.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 2000.

[NT16] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 255–271. IEEE, 2016.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. pages 238–252, 2013.

[QSL+25] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkgpt: An efficient non-interactive zero-knowledge proof framework for llm inference. In *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

[RIS23] RISC Zero. Scalable, Transparent Arguments of RISC-V Integrity. Technical report, RISC Zero, 2023.

[RT97] John H Reif and Stephen R Tate. On dynamic algorithms for algebraic problems. *Journal of Algorithms*, 22(2):347–371, 1997.

[sc] snarkjs contributors. Snarkjs. https://github.com/iden3/snarkjs.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology – CRYPTO 2020*, pages 704–737, 2020.

[SLZ24] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. *arXiv preprint arXiv:2404.16109*, 2024.

[TAB+20] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In *Security and Cryptography for Networks*, 2020.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.

[vit] An Incomplete Guide to Rollups. https://vitalik.ca/general/2021/01/05/rollup.html.

[WPSP25a] Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs. https://eprint.iacr.org/archive/2024/1566/20250214:152438, Version 2025-02-14, 2025.

[WPSP25b] Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs. https://eprint.iacr.org/archive/2024/1566/20250716:152311, Version 2025-07-16, 2025.

[WPSP25c] Weijie Wang, Charalampos Papamanthou, Shravan Srinivasan, and Dimitrios Papadopoulos. Dynamic zk-SNARKs. https://eprint.iacr.org/archive/2024/1566/20251013:145022, Version 2025-10-13, 2025.

[WTS+18] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018.

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.

[XZC+22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. pages 3003–3017, 2022.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Annual International Cryptology Conference*, pages 299–328. Springer, 2022.

[XZZ+19] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Annual International Cryptology Conference*, pages 733–764. Springer, 2019.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: efficient field-agnostic polynomial commitment schemes from foldable codes. In *Annual International Cryptology Conference*, pages 138–169. Springer, 2024.

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2039–2053, 2020.

A Faster Groth16 Prover

The proof quotient. The prover of Groth16 in Protocol 1 spends most of its time in Step 2: computing the coefficients of the quotient polynomial $q(X)$ defined by $A(X) \cdot B(X) - C(X) = q(X)\, v_{\mathbb{H}}(X)$ takes $O(n \log n)\mathbb{F}$, and committing to them takes $O(n)\mathbb{G}$, whereas all other steps are linear in the witness. We call $q(X)$ the *proof quotient*, as dividing by the vanishing polynomial is the central idea of how most zkSNARKs based on univariate polynomials prove the multiplication check. A faster Groth16 prover outputs the same proof for the same prover key, but computes the proof quotient with fewer DFTs.

Existing algorithms. Since $A(X) \cdot B(X) - C(X)$ vanishes on $\mathbb{H}_n$, its quotient cannot be obtained from evaluations on $\mathbb{H}_n$. Implementations such as [ac22a] and [sc] therefore evaluate on a coset of $\mathbb{H}_n$, which takes 7 DFTs of size n without preprocessing, or 6 with preprocessing. The coset must exist, which prevents instantiating $\mathbb{H}_n$ as the largest multiplicative subgroup of the field. The recent work of [CJL+24] removes this restriction by computing the quotient with polynomial derivatives, using 6 DFTs directly over $\mathbb{H}_n$, but does not reduce the number of DFTs.

Our algorithm. Quotient decomposition yields a closed form of the proof quotient that avoids the division altogether. Applying Theorem 3.1 to every product $\ell_i(X)\ell_j(X)$ in $A(X) \cdot B(X)$, the quotient is a linear combination of the Lagrange polynomials and of the diagonal terms $\phi_{i,i}(X)$, and by shift invariance its coefficients are two circular convolutions with the fixed vector $\boldsymbol{\eta}'$. This is the dense $\times$ dense case of Section 3.

Theorem A.1 (Proof Quotient). *Let* $A(X) = \sum_{i\in[n]} a_i\ell_i(X)$, $B(X) = \sum_{i\in[n]} b_i\ell_i(X)$ *and* $C(X) = \sum_{i\in[n]} c_i\ell_i(X)$ *with* $\mathbf{a}\circ\mathbf{b} = \mathbf{c}$. *The proof quotient* $q(X)$ *defined by* $A(X)\cdot B(X) = q(X)\,v_{\mathbb{H}}(X) + C(X)$ *is*

$$q(X) = \sum_{i\in[n]} \big(\tilde{\mathbf{q}}_i \cdot \ell_i(X) + c_i \cdot \phi_{i,i}(X)\big), \qquad \tilde{\mathbf{q}} = (\boldsymbol{\eta}' * \mathbf{b}) \circ \mathbf{a} + (\boldsymbol{\eta}' * \mathbf{a}) \circ \mathbf{b} \in \mathbb{F}^n,$$

where $\phi_{i,i}(X)$ *is defined in Theorem 3.1 and* $\boldsymbol{\eta}'$ *in Section 3.*

Proof. Expanding the product and applying Theorem 3.1 to each $\ell_i(X)\ell_j(X)$ gives

$$\begin{aligned} A(X)\cdot B(X) &= \sum_{i,j\in[n]} a_i b_j\, \ell_i(X)\ell_j(X) \\ &= \Big(\sum_{i\neq j} a_i b_j\, \phi_{i,j}(X) + \sum_{i\in[n]} a_i b_i\, \phi_{i,i}(X)\Big) v_{\mathbb{H}}(X) + \sum_{i\in[n]} a_i b_i\, \ell_i(X). \end{aligned}$$

The last sum is $C(X)$ since $a_ib_i = c_i$, so $q(X) = \sum_{i\neq j} a_ib_j\phi_{i,j}(X) + \sum_{i\in[n]} c_i\phi_{i,i}(X)$. It remains to collect, in the first sum, the coefficient of each $\ell_i(X)$. By Equation (5), and since $\eta_0 = 0$ lets the inner sums run over all of $[n]$,

$$\begin{aligned} \sum_{i\neq j} a_i b_j \big(\eta_{j-i}\,\ell_i(X) + \eta_{i-j}\,\ell_j(X)\big) &= \sum_{i\in[n]} \Big(a_i \sum_{j\in[n]} \eta_{j-i}\, b_j + b_i \sum_{j\in[n]} \eta_{j-i}\, a_j\Big)\, \ell_i(X) \\ &= \sum_{i\in[n]} \big(a_i\,(\boldsymbol{\eta}' * \mathbf{b})_i + b_i\,(\boldsymbol{\eta}' * \mathbf{a})_i\big)\,\ell_i(X) = \sum_{i\in[n]} \tilde{\mathbf{q}}_i\,\ell_i(X), \end{aligned}$$

where the second step uses $\eta_{j-i} = \eta'_{i-j}$. □

With the preprocessed $[\bar{\boldsymbol{\ell}}]_1$ and $[\mathbf{p}]_1$ of Section 4.3, which are computed once from $Groth16.\mathbf{pk}$ without the trapdoor, Theorem A.1 gives

$$[Q]_1 = \sum_{i\in[n]} \big(\tilde{\mathbf{q}}_i \cdot [\bar{\boldsymbol{\ell}}_i]_1 + c_i \cdot [\mathbf{p}_i]_1\big), \qquad \tilde{\mathbf{q}} = \mathrm{iDFT}\big(\widehat{\boldsymbol{\eta}}' \circ \mathrm{DFT}(\mathbf{b})\big) \circ \mathbf{a} + \mathrm{iDFT}\big(\widehat{\boldsymbol{\eta}}' \circ \mathrm{DFT}(\mathbf{a})\big) \circ \mathbf{b},$$

where $\widehat{\boldsymbol{\eta}}' = \mathrm{DFT}(\boldsymbol{\eta}')$ does not depend on the witness. Protocol 6 presents the resulting prover. Its public key is that of Protocol 2, and its proofs and its verifier are identical to those of Groth16, so the security of Groth16 carries over as in the proof of Theorem 4.3.

Complexity analysis. Computing $\tilde{\mathbf{q}}$ takes two DFTs and two iDFTs of size n over $\mathbb{H}_n$ and $O(n)$ field multiplications, and committing to $[Q]_1$ takes $2n$ group operations, n more than Step 2 of Groth16. All other steps are unchanged. The prover therefore still takes $O(n\log n)\mathbb{F} + O(n)\mathbb{G}$, with 4 DFTs instead of 6, no coset, and n additional group operations.

B Sparse Groth16 Prover

Sparse witnesses. The running time of the Groth16 prover is $O(n\log n)\mathbb{F} + O(n)\mathbb{G}$ regardless of the witness. However, the linear steps Steps 3 to 5 only involve the non-zero entries of z, and the dynamic setting of this paper produces exactly such inputs: the cross term in Section 4.3 is the proof quotient of the sparse vectors Δ^A and Δ^B. A *sparse Groth16 prover* outputs the same proof as Groth16, for the same prover key, in time that depends only on the number of non-zero entries of z, $\mathbf{a} = A\cdot z$ and $\mathbf{b} = B\cdot z$. We denote their sets of non-zero indices by I_z, I_a and I_b, and let $m = |I_z| + |I_a| + |I_b|$. This application may be of independent interest.

Protocol 6 (Faster Groth16 Prover)**.**

- *FastGroth16.*$\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$*: the same as SemiDynark.*$\mathcal{G}$ *in Protocol 2, with* $\mathbf{pk} = (Groth16.\mathbf{pk}, [\bar{\boldsymbol{\ell}}]_1, [\mathbf{p}]_1)$ *and* $\mathbf{vk} = Groth16.\mathbf{vk}$*. Let* $\widehat{\boldsymbol{\eta}}' = DFT(\boldsymbol{\eta}')$*.*
- *FastGroth16.*$\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to \pi$
  1. *Let* $z = (\mathbb{x}, \mathbb{w})$ *and sample* $r, s \leftarrow \mathbb{Z}_p$*.*
  2. *Compute* $\mathbf{a} = A \cdot z, \mathbf{b} = B \cdot z, \mathbf{c} = C \cdot z \in \mathbb{F}^n$*.*
  3. *Compute* $\tilde{\mathbf{q}} = iDFT\big(\widehat{\boldsymbol{\eta}}' \circ DFT(\mathbf{b})\big) \circ \mathbf{a} + iDFT\big(\widehat{\boldsymbol{\eta}}' \circ DFT(\mathbf{a})\big) \circ \mathbf{b}$*.*
  4. *Compute* $[Q]_1 = \sum_{i\in[n]} \tilde{\mathbf{q}}_i[\bar{\boldsymbol{\ell}}_i]_1 + \sum_{i\in[n]} \mathbf{c}_i[\mathbf{p}_i]_1$*.*
  5. *Compute* $[A]_1, [B]_2, [C]_1$ *as in Steps 3 to 5 of Protocol 1, using this* $[Q]_1$*.*
  6. *Output* $\pi = ([A]_1, [C]_1, [B]_2)$*.*
- *FastGroth16.*$\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$*: output* $Groth16.\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi)$*.*

Sparse proof quotient. The obstacle is again the proof quotient, which the standard prover computes with DFTs of size n regardless of sparsity. By Theorem A.1, for $I = I_a \cup I_b$,

$$q(X) = \sum_{i\in I} \tilde{\mathbf{q}}_i \cdot \ell_i(X) + \sum_{i\in I_a\cap I_b} c_i \cdot \phi_{i,i}(X), \qquad \tilde{\mathbf{q}} = (\boldsymbol{\eta}' * \mathbf{b}) \circ \mathbf{a} + (\boldsymbol{\eta}' * \mathbf{a}) \circ \mathbf{b},$$

since $\tilde{\mathbf{q}}_i = 0$ and $c_i = a_i b_i = 0$ outside these index sets. This is the sparse $\times$ sparse case of Section 3, and coincides with the cross term in Equation (8) with Δ^A, Δ^B replaced by $\mathbf{a}, \mathbf{b}$. Committing to it with $[\bar{\boldsymbol{\ell}}]_1$ and $[\mathbf{p}]_1$ takes $O(|I|)\mathbb{G}$. The remaining cost is the two convolutions in $\tilde{\mathbf{q}}$: only the entries of $\boldsymbol{\eta}' * \mathbf{a}$ and $\boldsymbol{\eta}' * \mathbf{b}$ on I are needed and the inputs are non-zero only on I, so they take $O(|I|^2)\mathbb{F}$ directly, or $O(n \log n)\mathbb{F}$ by DFT. We call such an operation a *sparse convolution*, and give an $O(|I| \log^2 |I|)$ algorithm for it, a specialized and simplified version of [GGS87], based on the following identity.

Theorem B.1 (Sparse Convolution)**.** *Let* $\mathbf{a} \in \mathbb{F}^n$ *be non-zero only on* $I \subseteq [n]$*, and define*

$$v_I(X) := \prod_{i\in I}(X - \omega^i), \qquad \ell_i^I(X) := \frac{\omega^i}{n} \cdot \frac{v_I(X)}{X - \omega^i}, \qquad a_I(X) := \sum_{i\in I} a_i\, \ell_i^I(X).$$

Then, for every $i \in I$*,*

$$(\boldsymbol{\eta}' * \mathbf{a})_i = \frac{1}{v_I'(\omega^i)}\Big(a_I'(\omega^i) - \frac{\omega^i}{2n}\, v_I''(\omega^i)\, a_i\Big).$$

Proof. Fix $i \in I$. Since $\eta_0' = 0$ and $a_j = 0$ for $j \notin I$,

$$(\boldsymbol{\eta}' * \mathbf{a})_i = \sum_{j\in I\setminus\{i\}} \eta'_{i-j}\, a_j = \sum_{j\in I\setminus\{i\}} \frac{\omega^j}{n(\omega^i - \omega^j)}\, a_j.$$

We show that $a_I'(\omega^i)$ equals $v_I'(\omega^i)$ times this sum, plus a term for $j = i$. For $j \neq i$, differentiating $\ell_j^I(X)$ and using $v_I(\omega^i) = 0$,

$$(\ell_j^I)'(\omega^i) = \frac{\omega^j}{n} \cdot \frac{v_I'(X)\,(X - \omega^j) - v_I(X)}{(X - \omega^j)^2}\Big|_{X=\omega^i} = \frac{\omega^j}{n(\omega^i - \omega^j)}\, v_I'(\omega^i).$$

Protocol 7 (Sparse Convolution)**.**

- $\mathit{SparseConv}_{\boldsymbol{\eta}'}(\mathbf{a}, I) \to \tilde{\mathbf{a}}$
  1. *Compute the coefficient form of* $v_I(X) = \prod_{i\in I}(X - \omega^i)$*, the vanishing polynomial over* $\{\omega^i\}_{i\in I}$*, using divide-and-conquer*
  2. *Compute the coefficient forms of the derivatives* $v_I'(X)$ *and* $v_I''(X)$
  3. *Compute the evaluations of* $v_I'(X)$ *and* $v_I''(X)$ *over* $\{\omega^i\}_{i\in I}$ *using multipoint evaluation*
  4. *Let* $a_I(X) = \sum_{i\in I} a_i \frac{\omega^i}{n} \frac{v_I(X)}{X-\omega^i}$*; its evaluations are* $a_I(\omega^i) = a_i \frac{\omega^i}{n} v_I'(\omega^i)$ *for* $i \in I$
  5. *Compute the coefficient form of* $a_I(X)$ *by interpolation*
  6. *Compute the coefficient form of the derivative* $a_I'(X)$
  7. *Compute the evaluations of* $a_I'(X)$ *over* $\{\omega^i\}_{i\in I}$ *using multipoint evaluation*
  8. *Compute* $\tilde{\mathbf{a}}_i = \frac{1}{v_I'(\omega^i)} \left(a_I'(\omega^i) - \frac{\omega^i}{2n} v_I''(\omega^i) a_i \right)$ *for all* $i \in I$
  9. *Output* $\tilde{\mathbf{a}}$

For $j = i$, write $v_I(X) = (X - \omega^i)\, g(X)$, so that $\ell_i^I(X) = \frac{\omega^i}{n}\, g(X)$. Differentiating $v_I(X)$ twice gives $v_I'(\omega^i) = g(\omega^i)$ and $v_I''(\omega^i) = 2\, g'(\omega^i)$, hence $(\ell_i^I)'(\omega^i) = \frac{\omega^i}{2n}\, v_I''(\omega^i)$. Summing over $j \in I$ with weights a_j,

$$a_I'(\omega^i) = v_I'(\omega^i) \sum_{j\in I\setminus\{i\}} \frac{\omega^j}{n(\omega^i - \omega^j)}\, a_j + \frac{\omega^i}{2n}\, v_I''(\omega^i)\, a_i = v_I'(\omega^i)\, (\boldsymbol{\eta}' * \mathbf{a})_i + \frac{\omega^i}{2n}\, v_I''(\omega^i)\, a_i,$$

and solving for $(\boldsymbol{\eta}' * \mathbf{a})_i$ gives the claim. □

Sparse convolution algorithm. Protocol 7 computes $\mathrm{SparseConv}_{\boldsymbol{\eta}'}(\mathbf{a}, I)$, the entries of $\boldsymbol{\eta}' * \mathbf{a}$ on I, from Theorem B.1. Let $m' = |I|$. The coefficients of $v_I(X)$ are computed by divide-and-conquer: split I into two halves, recurse, and multiply the two results with a DFT in $O(m' \log m')$ time, so $T(m') = 2T(m'/2) + O(m' \log m') = O(m' \log^2 m')$. The derivatives $v_I'(X)$ and $v_I''(X)$ follow in $O(m')$ time, and their evaluations on $\{\omega^i\}_{i\in I}$ by multipoint evaluation in $O(m' \log^2 m')$ time. The polynomial $a_I(X)$ has degree $m' - 1$, and since $\ell_j^I(\omega^i) = 0$ for $j \neq i$ and $\ell_i^I(\omega^i) = \frac{\omega^i}{n}\, g(\omega^i) = \frac{\omega^i}{n}\, v_I'(\omega^i)$ as in the proof above, its evaluations are $a_I(\omega^i) = a_i \frac{\omega^i}{n}\, v_I'(\omega^i)$ for $i \in I$. Interpolating its coefficients, differentiating, and evaluating $a_I'(X)$ on $\{\omega^i\}_{i\in I}$ take $O(m' \log^2 m')$ time. In total, $\mathrm{SparseConv}_{\boldsymbol{\eta}'}$ takes $O(m' \log^2 m')$ field operations.

Protocol 8 presents the sparse Groth16 prover. It replaces Step 2 of Groth16 by the sparse proof quotient, computed with Protocol 7, and restricts the linear combinations in Steps 3 to 5 to I_z. Its public key is that of Protocol 2, and its proofs and its verifier are identical to those of Groth16, so the security of Groth16 carries over as in the proof of Theorem 4.3.

Complexity analysis. The two sparse convolutions take $O(|I| \log^2 |I|)\mathbb{F}$, committing to $[Q]_1$ takes $O(|I|)\mathbb{G}$, and the linear steps take $O(|I_z|)\mathbb{G}$. The prover therefore takes $O(m \log^2 m)\mathbb{F} + O(m)\mathbb{G}$ for $m = |I_z| + |I_a| + |I_b|$, independent of n, after the one-time computation of $[\bar{\boldsymbol{\ell}}]_1$ and $[\mathbf{p}]_1$ from $\mathit{Groth16}.\mathbf{pk}$ in $O(n \log n)\mathbb{G}$.

Protocol 8 (Sparse Groth16 Prover)**.**

- *SparseGroth16.*$\mathcal{G}(1^\lambda, \mathbb{i}) \to (\mathbf{pk}, \mathbf{vk})$*: the same as SemiDynark.*$\mathcal{G}$ *in Protocol 2, with* $\mathbf{pk} = (Groth16.\mathbf{pk}, [\bar{\ell}]_1, [\mathbf{p}]_1)$ *and* $\mathbf{vk} = Groth16.\mathbf{vk}$*.*
- *SparseGroth16.*$\mathcal{P}(\mathbf{pk}, \mathbb{x}, \mathbb{w}) \to \pi$
  1. *Let* $z = (\mathbb{x}, \mathbb{w})$ *and sample* $r, s \leftarrow \mathbb{Z}_p$*.*
  2. *Compute* $\mathbf{a} = A \cdot z, \mathbf{b} = B \cdot z, \mathbf{c} = C \cdot z \in \mathbb{F}^n$*. Let* I_z, I_a, I_b *be their sets of non-zero indices, and* $I = I_a \cup I_b$*.*
  3. *Compute* $\tilde{\mathbf{a}} = SparseConv_{\boldsymbol{\eta}'}(\mathbf{a}, I)$ *and* $\tilde{\mathbf{b}} = SparseConv_{\boldsymbol{\eta}'}(\mathbf{b}, I)$ *by Protocol 7.*
  4. *Compute* $[Q]_1 = \sum_{i \in I}(\mathbf{a}_i \tilde{\mathbf{b}}_i + \mathbf{b}_i \tilde{\mathbf{a}}_i)[\bar{\ell}_i]_1 + \sum_{i \in I_a \cap I_b} \mathbf{c}_i [\mathbf{p}_i]_1$*.*
  5. *Compute* $[A]_1 = [\alpha]_1 + r[\delta]_1 + \sum_{i \in I_z} z_i [\mathbf{u}_i]_1$*.*
  6. *Compute* $[B]_2 = [\beta]_2 + s[\delta]_2 + \sum_{i \in I_z} z_i [\mathbf{v}_i]_2$ *and* $[B]_1$ *similarly.*
  7. *Compute* $[C]_1 = s[A]_1 + r[B]_1 - rs[\delta]_1 + \sum_{i \in [k,m] \cap I_z} z_i [\mathbf{t}^{\mathsf{p}}_{i-k}]_1 + [Q]_1$*.*
  8. *Output* $\pi = ([A]_1, [C]_1, [B]_2)$*.*
- *SparseGroth16.*$\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi) \to 0/1$*: output* $Groth16.\mathcal{V}(\mathbf{vk}, \mathbb{x}, \pi)$*.*