DUPLEX: Scalable Zero-Knowledge Lookup Arguments over RSA Group

Semin Han
seminhan@hanyang.ac.kr
Hanyang University
Seoul, Republic of Korea

Geonho Yoon
geonho@hanyang.ac.kr
Hanyang University
Seoul, Republic of Korea

Hyunok Oh*
hoh@hanyang.ac.kr
Hanyang University, Zkrypto Inc.
Seoul, Republic of Korea

Jihye Kim*
jihyek@kookmin.ac.kr
Kookmin University, Zkrypto Inc.
Seoul, Republic of Korea

Abstract

Lookup arguments enable a prover to convince a verifier that a committed vector of lookup elements $\vec{f} \in \mathbb{F}^m$ is contained within a predefined table $T \in \mathbb{F}^N$. These arguments are particularly beneficial for enhancing the performance of SNARKs in handling non-arithmetic operations, such as batched range checks or bitwise operations. While existing works have achieved efficient and succinct lookup arguments, challenges remain, particularly when dealing with large vectors of lookup elements in privacy-sensitive applications.

In this paper, we introduce DUPLEX, a scalable zero-knowledge lookup argument scheme that offers significant improvements over previous approaches. Notably, we present the first lookup argument designed to operate over the RSA group. Our core technique allows for the transformation of elements into prime numbers to ensure compatibility with the RSA group, all without imposing substantial computational costs on the prover. Given m lookup elements, DUPLEX achieves an asymptotic proving time of $O(m \log m)$, with constant-sized proofs, constant-time verification, and a public parameter size independent of the table size N. Additionally, DUPLEX ensures the privacy of lookup elements and is robust against dynamic table updates, making it highly suitable for scalable verifiable computation in real-world applications.

We implemented and empirically evaluated DUPLEX, comparing it with the state-of-the-art zero-knowledge lookup argument Caulk [CCS'22]. Our experimental results demonstrate that DUPLEX significantly outperforms Caulk in proving time for both single and batched lookup arguments, while maintaining practical proof size and verification time.

1 Introduction

A Succinct Non-interactive Arguments of Knowledge (SNARK), which allows a verifier to efficiently validate the correctness of a statement using a short proof, has gained prominence with the growing demand for delegating computation while ensuring the integrity of the results. Applications like blockchain, artificial intelligence (AI), and cloud computing increasingly rely on offloading complex computations to external provers, while still requiring a reliable method to verify the results with minimal overhead. For example, rollups outsource transaction processing to external servers to reduce expensive on-chain operations [36]. In such systems, batches of transactions are processed off-chain, and only the updated states and succinct proofs are recorded on-chain. Instead of each node in the blockchain network performing the transaction computations themselves, they can simply verify the succinct proof, significantly improving the system's efficiency by reducing the overall computational load required to validate transactions.

Beyond improving computational efficiency, privacy in these outsourced computations has become a critical concern, especially when sensitive data is involved. For instance, in deep learning, weight values in a model are often proprietary and represent valuable intellectual property for providers [30, 33]. zero-knowledge SNARKs (zkSNARKs), which offer zero-knowledge guarantees, have emerged as a solution to ensure both privacy and correctness, allowing computations to be verified without revealing the underlying data. However, one of the most significant challenges for SNARKs remains the proving time, which has driven substantial research into optimizing this aspect. Most SNARK schemes require computations to be transformed into arithmetic circuits, which is relatively straightforward for some applications. Yet, many real-world computations, such as hash functions or range checks, involve non-arithmetic operations that do not easily fit into arithmetic circuit representations. These non-arithmetic operations often lead to inefficiencies, as they must be transformed into complex arithmetic constraints, significantly increasing both the proving time and the number of constraints. For example, proving that a value falls within a specific range (e.g., ensuring that x is a 256-bit number) requires breaking the value into binary components, introducing hundreds of additional constraints. As applications like blockchain [2, 4] and privacy-preserving machine learning [19, 30, 33] increasingly rely on such non-arithmetic operations, optimizing their proof generation is essential for making SNARKs more widely applicable.

Lookup arguments in the context of SNARKs. One of the most effective techniques to address the inefficiencies of proving non-arithmetic operations in SNARKs is the use of lookup arguments. Given a (predefined) table $T \in \mathbb{F}^N$ and a commitment $c_{\vec{f}}$ to a vector $\vec{f} \in \mathbb{F}^m$, where m represents the number of lookup elements, a prover can claim that all elements of $\vec{f}$ are contained within the table T with a short proof. Importantly, the verification of the

*Both authors are co-corresponding authors

lookup argument should be performed in time independent of N, as the table size N is typically much larger than the number of lookup elements m. Lookup arguments are particularly beneficial for handling non-arithmetic operations, such as range checks or bitwise operations. For instance, a range check that would otherwise require 256 constraints and additional field elements can be reduced to a single lookup operation when using a well-constructed table. Similarly, bitwise operations like XOR can be efficiently handled by organizing the lookup table with triples (a, b, c) for inputs and outputs, which simplifies the SNARK's arithmetic circuit.

Recognizing the potential of lookup arguments in SNARKs, extensive researches [8, 9, 15, 18, 21, 23–25, 37, 42, 48, 49] have focused on improving their efficiency and expanding their applications [2, 22, 43, 47]. While many lookup arguments are becoming increasingly efficient, achieving performance independent of the table size N is a critical goal, especially when N is much larger than m. Despite these advancements, challenges remain in terms of ensuring full compatibility between lookup arguments and different SNARK schemes, particularly those based on different polynomial structures. Additionally, privacy concerns are paramount when using lookup tables, especially in sensitive applications like AI, where lookup arguments can enhance non-arithmetic tensor operations [43] while protecting proprietary data [19]. Furthermore, verifiable computation systems, such as those involving Random Access Memory (RAM), benefit from lookup arguments that can adapt to dynamic environments where tables are frequently updated [22].

1.1 Our Works

We introduce DUPLEX[1], a scalable zero-knowledge lookup argument built over the RSA group[2]. DUPLEX achieves superior proving performance, constant-time verification, and small public parameters, all while maintaining constant-sized proofs. Importantly, DUPLEX ensures the privacy of lookup elements and is designed to handle dynamic tables, making it highly applicable to scenarios like verifiable RAM access or frequently updated databases. Moreover, it offers *flexibility*, meaning it can be applied to various SNARK frameworks (e.g., AIR, Plonkish, R1CS) without requiring specific transformations that incur additional overhead.

The foundation of DUPLEX lies in its extension of HARISA [16], a state-of-the-art set membership proof scheme based on the RSA group. HARISA already offers the key properties required for an efficient lookup argument, such as fast proving time, succinct verification, constant-sized proofs, and compact public parameters, all while preserving privacy. However, applying HARISA directly to lookup arguments is challenging due to two primary issues:

- *Element Uniqueness*: HARISA assumes the uniqueness of elements within its proofs, which is not compatible with lookup arguments where duplicate elements are common. For example, considering lookup argument for an AND operation, the lookup element vector $\vec{f} = (f_1 = 1111 \ \& \ 0000, f_2 = 1111 \ \& \ 0000, f_3 = 1100 \ \& \ 0101)$ could have identical elements (e.g., f_1 and f_2).
- *Compatibility with RSA Group*: RSA group-based accumulators require elements to be prime (or hashed-to-prime) to ensure security under the strong RSA and/or adaptive root assumptions. This is problematic for typical lookup arguments, where the elements may not naturally be prime numbers, as seen in many lookup tables for operations like AND. While encoding non-prime elements as primes is one approach, it introduces significant overhead in the SNARK circuit.

By resolving these challenges, DUPLEX delivers a lookup argument with competitive proving time, succinct verification, constant-sized proofs, and a public parameter size independent of the table size N. Additionally, DUPLEX is versatile across different SNARK frameworks and applicable to dynamic tables, all while upholding zero-knowledge guarantees essential for privacy-sensitive applications. Our contributions are summarized as follows:

- We introduce DUPLEX, a novel zero-knowledge lookup argument scheme built over the RSA group. It achieves competitive proving performance compared to existing schemes while maintaining succinct verification. This efficiency is realized through our innovative technique (we refer to it as *double lookup, less work*), which transforms the given table elements into prime numbers to ensure compatibility with the RSA group without imposing significant computational overhead on the prover. Asymptotically, for m lookup elements and an N-sized table, the proving time for DUPLEX operates independently of the table size, at $O(m \log m)$, with precomputation. Additionally, the verification time and the proof size is constant, and the public parameter size is $O(m)$, independent of the table size.
- DUPLEX provides zero-knowledge for the lookup elements, allowing a prover can convince the verifier that the lookup elements are indeed in the predefined table without revealing their values. This is particularly beneficial for privacy-sensitive applications, such as those in artificial intelligence.
- DUPLEX seamlessly integrates with SNARK circuits of any arithmetization (e.g., AIR, Plonkish, R1CS) without incurring additional costs. This flexibility is particularly advantageous in scenarios where consistency across computations from heterogeneous SNARKs is required, making it a versatile solution in diverse cryptographic settings.
- Our scheme supports zero-knowledge lookup arguments even for dynamic tables that can be continuously updated. For example, consider lookup elements $\vec{f} = (f_1 = t_1, f_2 = t_2)$, an initial table $T = \{t_1, t_2, \cdots, t_N\}$, and an updated table $T' = \{t_1, t_2, \cdots, t_N, t_{N+1}\}$. The prover does not need to regenerate the public parameters since the setup is independent of the table size. This feature is especially effective for real-world applications such as blockchain and verifiable computation environments involving RAM, where data is frequently updated.

[1]The name "Duplex"[46], referring to two living units within a single structure, reflects the dual scalability offered by the proposed lookup argument in terms of both performance and functionality, encompassing features such as zero-knowledge, flexibility, and adaptability to dynamic environments, all within a single construction. Furthermore, it aligns with our core technique, "double lookup, less work."

[2]While the RSA group is used in this work for clarity, DUPLEX can also be instantiated with other groups of unknown order, such as the class group of an imaginary quadratic field. The choice of group involves trade-offs, which are detailed in[11].

- We implemented DUPLEX in Rust and empirically validated its performance against the state-of-the-art zero-knowledge lookup argument Caulk [48]. For example, given 2^6 lookup elements, DUPLEX achieves a proving time of 207ms, making it 12.96× faster than Caulk of table of 2^{20} elements. While the verification time and proof size are slightly larger than those of Caulk—46ms and 1.17KB for DUPLEX compared to 36ms and 0.89KB for Caulk—these metrics remain within practical limits. This trade-off allows DUPLEX to provide significant improvements in proving efficiency, especially in applications requiring frequent updates to dynamic tables, such as blockchain or verifiable RAM access.

1.2 Related works

Lookup arguments in the literature. After Bootle et al. [8] introduced lookup argument, research on the lookup arguments has been amplified. One of the initial lookup arguments Plookup [25] proposed lookup arguments along with Plonk, with a proving cost dependent of table size N. Caulk [48] proposed the first solution that was sublinear in the size of the table, $O(m^2 + m \log N)$ for table size N and lookup elements size m, achieved by preprocessing elements in the table. Caulk is also the first protocol achieving zero-knowledge, that is, the lookup elements are unknown to the verifier. Caulk+ [37] attains a proving time of $O(m^2)$ by eliminating N factor, rendering Caulk lookup argument independent of the size of table. Ba*loo* [49] introduces quasilinear to the size of lookups and independent of the size of the table. It is based on commit-and-prove checkable subspace argument that for the table $\vec{t}$ and the lookup $\vec{a}$, a prover convinces verifier that they know a matrix M such that $M \cdot \vec{t} = \vec{a}$. cq [23] made $O(m \log m)$ prover works first time based on logarithmic derivative of [28]. It has proof aggregation by giving elements as the sum of derivatives. cq+ [14] improves cq by reducing Aurora's sumcheck [5] into checking one polynomial. zkcq [14] propose the up-to-date lookup argument achieving zero-knowledge by combining cq+ with the CP-SNARK.

Lasso [42] convinces the verifier by proving the relation $M \cdot \vec{t} = \vec{a}$ in another way. It proves the relation using Surge, a generalization of sparse polynomial commitment Spark by modifying decomposable structures into subtables, called spark-only structure (SOS). The advantage of Lasso is that it allows the prover to commit to only "small" field element, whereas a random field element is required to committed, even if the witness is small. It can be applied to any arithmetization—AIR, Plonkish, or R1CS—by converting into customizable constraint systems introduced in [41].

Very recently, Campanelli et al. [15] introduce the super-efficient lookup argument, called μ-seek. They define a super-efficient lookup argument as lookup argument in which the prover runs independent of table size. μ-seek is therefore super-efficient since it has a linear proving time $O(m)$. On the other hand, both the verification and proof size are logarithmic on m (yet still succinct) due to the sumcheck protocol and CP-SNARK with a multilinear commitment scheme. Moreover, the public parameter depends on the table size since it precomputes all possible polynomial.

Lookup arguments and its applications. Driven by the merits of lookup arguments, research on their application has recently been activated. Jolt [2] introduces a new SNARK scheme for executions of virtual machines by integrating with its companion work, Lasso. Notus [47] introduces a lookup argument to the Dynamic Proof of Liabilities (DPoL) with an RSA accumulator and a hashchain. Each transaction is combined with a DI-hash and its membership is verified using the hashchain. Dutta et al. [22] present a batching-efficient RAM built upon lookup argument, which can be adapted for updatable table. zkLLM [43] devises a parallelizable lookup argument to leverage parallel computing resources, such as GPUs.

Set membership proof. Another relevant approach to prove $\vec{f} \in T$ is a set membership proof [3, 6, 12, 13, 16, 20, 31, 32, 34, 35, 44, 50]. It is analogous to lookup arguments in that the set membership proof aims to prove some elements x belongs to some public set S. Recent works on set membership proofs carry efficient prover and succinct verifier (and proof). Nevertheless, as it has obvious distinctions with lookup arguments, it is difficult to directly use a set membership proof as a lookup argument. Since the elements in the set membership proof are generally assumed to be unique, the duplication does not occur. However, in the lookup arguments, the duplication is likely scenario. For instance, the result of a bitwise operation can appear twice or more. For the reason, a set membership proof is not directly compatible with lookup argument while it has good properties. Furthermore, there is one more constraint that the elements should be prime numbers among the works based on RSA accumulator [3, 6, 13, 16, 31, 32]. Regarding that the given table elements[3] are potentially not prime number, such constraint charges additional cost incurred by transformation of elements into prime numbers.

1.3 Structure of the paper

We introduce notations and (informal) definition of cryptographic primitives used in this paper in Section 2. The technical intuition of our work is outlined in Section 3 and the construction is depicted in Section 4. In Section 5, implementation and evaluation for our construction is presented.

2 Preliminaries

We provide informal definitions of the primary cryptographic primitives and some notations used in our constructions.

2.1 Notations

Before we dive into our construction overview, we show some notations used throughout the paper. Most of the notations related to RSA accumulator are from [16]. Additionally, for the set S, $\prod_S$ denotes the product of all elements in the set S, i.e., $\prod_S = \prod_{u_i \in S} u_i$. This holds for the vector as well, e.g., $\prod_s$ denotes the product of all elements in vector $\vec{s}$. The left subset S_L denotes the first half of the elements in set S, and S_R denotes the latter half elements. For example, consider the set $S = \{u_1, \cdots, u_N\}$ where N is even; $S_L = \{u_1, \cdots, u_{\frac{N}{2}}\}$ and $S_R = \{u_{\frac{N}{2}+1}, \cdots, u_N\}$ respectively. For brevity, vectors are sometimes used as data structure to represent sets, and vice versa. For example, the table $T = \{t_1, \ldots, t_N\}$ is sometimes denoted as $\vec{t} = (t_1, \ldots, t_N)$. The concatenation operation is denoted as $\|$. Unless otherwise specified, the vector $\vec{f}$ denotes

[3] Suppose the table contains all possible results of bitwise operation. The results are probably not prime number.

Scheme	Zero-knowledge	Precomputation	Proof size	Prover work		Verifier work
				group	field	
plookup [25]	–	–	$5\mathbb{G}_1, 9\mathbb{F}$	$O(N)$	$O(N \log N)$	$2P$
Caulk [48]	✓	$O(N \log N)$	$14\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$15m$	$O(m^2 + m \log N)$	$4P$
Caulk+ [37]	✓	$O(N \log N)$	$7\mathbb{G}_1, 1\mathbb{G}_2, 2\mathbb{F}$	$8m$	$O(m^2)$	$3P$
flookup [24]	–	$O(N \log^2 N)$	$7\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$O(m)$	$O(m \log^2 m)$	$3P$
Ba*loo* [49]	–	$O(N \log N)$	$12\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$14m$	$O(m \log^2 m)$	$5P$
cq [23]	–	$O(N \log N)$	$8\mathbb{G}_1, 3\mathbb{F}$	$7m + O(m)$	$O(m \log m)$	$5P$
cq+ [14]	–	$O(N \log N)$	$8\mathbb{G}_1, 1\mathbb{F}$	$8m$	$O(m \log m)$	$5P$
zkcq+ [14]	✓	$O(N \log N)$	$9\mathbb{G}_1, 1\mathbb{F}$	$8m$	$O(m \log m)$	$6P$
Lasso w/ KZG+Gemini (unstructured table) [42]	–	–	$O(\log m)\mathbb{G}_1$ $\tilde{O}(\log m)\mathbb{F}$	$(c+1)m + cN^{\frac{1}{c}}$	$O(m+N)$	$2P$ $\tilde{O}(\log m)\mathbb{F}$
μ-seek [15] w/ KZG+Gemini	–	$O(N \log N)$	$2(\log m + 3)\mathbb{G}_1$ $6(\log m + 1)\mathbb{F}$	$O(m)$	$O(m)$	$2P$ $O(\log m)\mathbb{G}_1$
our scheme w/ cpGro16	✓	$O(N \log N)$	$13\mathbb{G}_1, 3\mathbb{G}_2, 4\mathbb{G}_?$	$O(m \log m)$	–	$3\mathbb{G}_?, 11P$

Table 1: Costs comparison of prior lookup works and our work. N is the size of table, m is the number of lookups (the size of lookup element vector). We assume $N > m$ for simplicity. $\mathbb{G}_1$ and $\mathbb{G}_2$ are pairing-friendly group and $\mathbb{G}_?$ is group of unknown order. In this paper, we assume RSA group for $\mathbb{G}_?$. The complexity in precomputation is the number of group operation. For the prover and verifier work, c denotes the decomposition factor of Lasso, and P refers to pairing operation. If the cell is marked as –, it means that it has nothing corresponding.

the vector of lookup elements f_i, and $\vec{t}$ generally denotes the vector of table elements t_i in table T.

2.2 Commitments

A commitment scheme allows one to commit to a value in a manner that is both hiding and binding. Specifically, the hiding property ensures that the committed value remains secret, while the binding property guarantees that the commitment can only be opened to the originally committed value. A commitment scheme consists of two algorithms (Setup, Comm) where Setup takes security parameter 1^λ as input and outputs commitment key ck, and Comm returns a commitment c for the input of ck, message m, opening randomness o. A commitment scheme is additively homomorphic if it satisfies the following conditions: for any messages m_i and m_j such that $m_i \neq m_j$ and any randomnesses o_i and o_j such that $o_i \neq o_j$, $\mathsf{Comm}(m_i; o_i) + \mathsf{Comm}(m_j; o_j) = \mathsf{Comm}(m_i + m_j; o_i + o_j)$ holds.

2.3 SNARKs

A SNARK for a relation R consists of three algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy})$ as follows:

- $\mathsf{Setup}(1^\lambda, R) \rightarrow \mathsf{crs}$ takes a security parameter 1^λ and a relation R as inputs and outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x; w) \rightarrow \pi$ returns a proof π on crs with a statement x and a witness w
- $\mathsf{Vfy}(\mathsf{crs}, x, \pi) \rightarrow \{0, 1\}$ on input crs, a statement x, and a proof π returns 1 if the proof is correct and 0 otherwise.

A SNARK needs to satisfy completeness, knowledge-soundness, and succinctness. Completeness means that the honest verifier always accepts the proof for any pair (x, w) satisfying the relation. Formally, a SNARK is complete if it holds with overwhelming probability that $\mathsf{Vfy}(\mathsf{crs}, x, \pi) = 1$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, R)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x; w)$ for $(x; w) \in R$. Knowledge soundness says that a valid witness can be extracted from a proof that passes verification. We can say that a SNARK is succinct if it has a proof of small size and fast verification time, formally both are poly-logarithmic in the witness size. A SNARK may also be zero-knowledge if the proof reveals nothing about the witness, and we refer it to zkSNARK. We employ commit-and-prove SNARKs (CP-SNARKs) [17] as a framework in this work. CP-SNARK is a SNARK that the prover can prove properties of committed inputs efficiently with some existing commitment scheme Comm, e.g., Pedersen Commitment. In this paper, $c_?$ denotes the committed value (of subscript) and its opening is denoted as $o_?$. For instance, c_u is the commitment to the value u and its opening is o_u. As discussed in [17], CP-SNARK has the modular composition of SNARKs with the committed witnesses. For example, $R_1(c_u; w_1)$ and $R_2(c_u; w_2)$ can be proven with CP-SNARK relation $\tilde{R}(c_u; w_1, w_2, u, o_u)$: $\tilde{R}(c_u; w_1, w_2, u, o_u) = 1 \Leftrightarrow R_1(c_u; w_1) = 1 \wedge R_2(c_u; w_2) = 1$. A CP-SNARK for the relation $\tilde{R}$ is denoted as cpΠ.

2.4 RSA Accumulator

An RSA accumulator is a data structure that transforms a large set S into a compact digest, based on RSA group. Built upon the RSA accumulator, the membership proof can prove that u_i is a valid element of the set S for any set element $u_i \in S$, even in privacy-preserving manner. The instantiation of the RSA accumulator is secure under strong RSA assumption [3] and adaptive root assumption [45]. The elements for the RSA accumulator must be primes (or the ones hashed-to-prime). The syntax of the accumulator scheme we use in this paper is as follows:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp} := (\mathbb{G}_?, g_?)$ takes a security parameter 1^λ as input and outputs a public parameter pp.

- $\mathsf{Acc}(\mathsf{pp}, S) \rightarrow \mathsf{acc} := g_?^{\prod_S}$ returns an accumulator where all of the elements in set S are accumulated by exponentiating the product of S.
- $\mathsf{MemPrv}(\mathsf{pp}, S, U) \rightarrow W := g_?^{\prod_S / \prod_U}$ computes a witness W by exponentiating all elements in set S except those in U, which is the subset of S that includes the elements to be proven.
- $\mathsf{MemVfy}(\mathsf{pp}, W, U, \mathsf{acc}) \rightarrow \{0,1\}$ accepts if and only if $W^{\prod_U} = \mathsf{acc}$, rejects otherwise.

We can notice that calculating W from scratch impedes proving performance since it accompanies $O(|S|)$ exponentiations over RSA group. Precomputation based on divide-and-conquer approach [38] for witness generation can reduce the cost of prover whereas it requires additional storage to store the precomputed results, covering all possible witnesses. In this paper, the RSA accumulator and the membership proof scheme follow HARiSA (Figure 1), which is introduced in [16], and we further enhance it with precomputation.

2.4.1 Revisiting HARiSA Since DUPLEX employs HARiSA as a building block, we can ride the wave of the benefits of HARiSA. First of all, HARiSA takes out RSA operation from the SNARK circuit to reduce the proving overhead by combining sigma protocol, proof of knowledge exponent (PoKE, [7]) and zkSNARKs. Briefly, they hide membership elements through sigma protocol and it is proven under $\mathsf{cp}\Pi^{\mathsf{modarithm}}$ for the relation as follows:

$$\tilde{R}_{\mathsf{ck}}^{\mathsf{modarithm}}(c_{\vec{u}}, c_{s,r}, h, l, \hat{k}) = 1 \Leftrightarrow \hat{k} = s \cdot h \cdot \prod_{i \in [m]} u_i + r \bmod l$$

What's more, it provides the privacy of the witness by using a technique where witness W is hidden by randomly exponentiating among prime numbers $p_i \in \mathbb{P}_{2\lambda}$. The randomization of W helps prevent leakage of elements (to be proven) from brute-force testing for all elements in the set S: $W^{u_i} \stackrel{?}{=} \mathsf{acc}$. Also, they get succinctness by adopting PoKE technique.

In summary, they construct zero-knowledge, succinct, and efficient membership proof scheme by combining the aforementioned techniques in commit-and-prove way. However, there exists difficulties to use HARiSA as lookup argument directly since it is built for the membership proof. As other membership proof schemes, each of the membership elements to be proven are unique in HARiSA with being prime numbers. Thus, it does not offer an efficient way to transform the given table and lookup elements into RSA-compatible form when it comes to lookup arguments.

3 Technical overview

We now present a high-level overview of our main technical contributions. The core protocol in this work extends the membership proof to the lookup argument. In particular, HARiSA, state-of-the-art zero-knowledge membership proof protocol based on RSA accumulator, has the advantages in terms of both performance and functionality. It has optimal proving time, succinct verification and constant-sized proof. Moreover, it satisfies zero-knowledge while being compatible with SNARKs of any arithmetization. The noteworthy feature is that it takes RSA operation out from SNARK circuit, and only the connectivity between the membership proof and the circuit through a commit-and-prove way[4]. However, directly applying HARiSA to the lookup argument has following restrictions: (A) Basically, HARiSA is the membership proof scheme where it is assumed that the duplication among elements to be proven does not occur, while it is likely in lookup arguments, i.e., there can exist $f_i = f_j$ for $i \neq j$. (B) The given elements (both table elements and lookup elements) can be (and it is quite probable) composite numbers. However, since HARiSA plays over RSA groups, each element should be transformed into the proper form, such as prime numbers or ones hashed to prime, and such transformation can incur additional cost for prover. Our primary contribution is to provide an efficient and scalable

$\underline{\mathsf{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}\right):}$

$\mathsf{crs}_1 \leftarrow \mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}_{\mathsf{ck}}^{\mathsf{modarithm}})$

return $\mathsf{crs} := (\mathsf{ck}, \mathsf{pp}, \mathsf{crs}_2)$

$\underline{\mathsf{Prove}\ (\mathsf{crs}, \mathsf{acc}, c_{\vec{u}}; W_{\vec{u}}, \vec{u}, o_{\vec{u}}):}$

$\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i}$

Let $u^* = \prod_i u_i$, $p^* = \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i$

Sample $b_1, \ldots, b_{2\lambda} \leftarrow\$ \{0,1\}$

Let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i}$, $\bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$ $\quad \hat{W}_{\vec{u}} \leftarrow W_{\vec{u}}^{\bar{s}}$

Sample $r \leftarrow\$ \{0,1\}^{\|p^*\|+\|u^*\|+2\lambda}$

$c_{s,r} \leftarrow \mathsf{Comm}_{\mathsf{ck}}(s, r; o_{s,r})$

$R \leftarrow \hat{W}_{\vec{u}}^r$

$h \leftarrow H(\mathsf{crs}||\mathsf{acc}||c_{\vec{u}}||c_{s,r}||\hat{W}_{\vec{u}}||R)$

$k \leftarrow r + (u^* s)h$

$\pi_1 \leftarrow \Pi^{\mathsf{PoKE}}.\mathsf{Prv}\big((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R; k\big)$

Parse π_1 as $(Q, \hat{k})$

$\ell \leftarrow H_{\mathsf{prime}}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R)$

$\pi_2 \leftarrow \mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, \hat{k}; \vec{u}, o_{\vec{u}}, r, s, o_{s,r})$

return $\pi = \big(\hat{W}_{\vec{u}}, R, c_{s,r}, \pi_1, \pi_2\big)$

$\underline{\mathsf{Verify}\ (\mathsf{crs}, \mathsf{acc}, c_{\vec{u}}, \pi):}$

$\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i}$

Parse π as $(\hat{W}_{\vec{u}}, R, c_{s,r}, \pi_1, \pi_2)$ and π_1 as $(Q, \hat{k})$

$\ell \leftarrow H_{\mathsf{prime}}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R)$

$h \leftarrow H(\mathsf{crs}||\mathsf{acc}||c_{\vec{u}}||c_{s,r}||\hat{W}_{\vec{u}}||R)$

Reject if $\Pi^{\mathsf{PoKE}}.\mathsf{Vfy}(\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R, \pi_1) \neq 1$

Reject if $\mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, \hat{k}, \pi_2) \neq 1$

Figure 1: HARiSA scheme we will use throughout this work

[4] Here we provide an intuition for the better delivery. Further detail is referred to [16].

construction for lookup arguments by resolving above obstacles. To achieve this, we combine below main techniques.

(1) Apart from applicability of the element, the likelihood of duplication among elements is hindrance to use HARiSA as lookup arguments. In order to handle duplication, we first sort both the table and lookup elements vectors. Then, create a temporary vector where each element is copied to the one in lookup elements vector at same index until the duplication occurs. Whenever the duplication occurs in lookup elements vector, the element of the temporary vector at that index is copied with another table elements which is not in the lookup elements vector. For instance, for the temporary vector $\vec{e}$, lookup elements $\vec{f}$, and table elements $\vec{t}$ where $\vec{f}$ and $\vec{t}$ are sorted in ascending order, $e_i = f_i$ if i-th element has no duplication. Otherwise the duplication exists such that $f_j = f_{j+1}$, $e_{j+1} = t_k$ such that $t_k \notin \vec{f} \wedge t_k \in \vec{t}$. Then, check that $\forall i \in [1, m] : (f_i - e_i)(f_i - f_{i-1}) = 0$. This confirms that each element is either the original or a duplication. Combining it with the membership proof, we can prove that all elements, including duplication elements, are in the lookup table. Namely, the membership proof for $\vec{e}$ is able to that all the elements in $\vec{e}$ are in the table. Accordingly, the equation is finally able to ensure that all the elements in $\vec{f}$ are either original elements in table or duplicated ones.
(2) Including HARiSA, using RSA accumulator for lookup arguments has significant obstacle, even though other challenges are resolved. Directly putting elements into an RSA accumulator is difficult in that the elements—both table elements and lookup elements—may not have proper form, such as the prime numbers or ones hashed-to-prime. Simply transforming elements into prime numbers using hash-to-prime is not a suitable solution, as it requires hash computations within the SNARK circuit that scale linearly with the number of elements to be proven, leading to significant computational overhead for the prover. So as to construct lookup argument using HARiSA with affordable proving overhead of well-transformation, we devise *double lookup, less work* technique. For an element t_i, let $\hat{t}_i \leftarrow t_i \| z_i$ be the prime number where z_i is the random prime number and make a table for z_i. From the perspective of the prover, the prover proves that $\hat{f}_i = f_i \| z_i$ and both $\hat{f}_i$ and z_i are in the table using HARiSA. It brings only one additional lookup for z_i to the original one lookup for f_i, which does not harm the asymptotic proving complexity and has little impact on practicality. Owing to the advantage of HARiSA, this additional lookup can be efficiently proven in batch manner. Further scrutiny can be seen in Section 4.2.

Overcoming the aforementioned hindrances, our lookup argument attains two benefits. The first one is flexibility, which means that the lookup argument is applicable to SNARKs of any arithmetization. Existing works represent argument as polynomial relation, such as KZG polynomial commitment [29] or a multilinear extension of sparse matrix [42], whereas our construction deals with the lookup element as its own value. After transformation with *double lookup, less work*, we can utilize HARiSA membership proof. As a result, the (committed) values of lookup elements are directly passed to SNARKs of any arithmetization through commit-and-prove way which is identical technique to HARiSA without additional costs. Namely, DUPLEX does not need to manipulate to tune into specific arithmetization. The another one is that DUPLEX can be extended to dynamic environments where the table is updated over time. By combining B-INS-ARiSA (introduced in [16, Section 5]) with DUPLEX, we can prove lookup argument for the updated table with preserving privacy. Similar to the core idea of DUPLEX, DUPLEX for dynamic table begins by proving well-transformation of updated elements in outline. It guarantees that the table is correctly updated. Then, run DUPLEX for the lookup elements in succession. Consequently, DUPLEX for the dynamic table can be constructed proving the update is done correctly and the lookup argument itself. Note that the DUPLEX is not affected by the update of table (and proving well-transformation of updated elements), zero-knowledge can be maintained even the update take places.

4 Zero-knowledge Lookup Arguments over RSA Group

In this section, we describe the construction of our scalable zero-knowledge lookup arguments, DUPLEX, over RSA group. We start from extending HARiSA, zero-knowledge set membership proof over RSA group. When using HARiSA for lookup arguments, two primary obstacles must be addressed to harness its full potential. The first one is that a lookup element may be a duplicate of another element which does not occur in general membership proof protocol. The second one is all the elements including table elements and lookup elements, must be transformed into an appropriate form to be compatible with RSA group (prime numbers or ones hashed-to-prime). We demonstrate each of the challenges and introduce the corresponding solutions accordingly. Then, we show the construction of the proposed lookup arguments, DUPLEX in Figure 3 where HARiSA is used as a building block while handling duplication and well-transformation. In Figure 2, we provide high-level structure of our proposed scheme.

4.1 Duplication handling

As aforementioned, one of the main hindrances is duplication among the lookup elements. It arises from the fundamental difference in the uniqueness of elements within membership proofs compared to the lookup arguments. In the membership proof protocols, each element to be proven is assumed to be unique. However, lookup arguments do not inherently guarantee such uniqueness, leading to possible duplication among the lookup elements. Specifically, it is conceivable for two distinct lookup elements denoted as u_i and u_j to be equivalent ($u_i = u_j$) for $i \neq j$. For clarity, let all the elements be prime numbers[5]. Unless duplication exists, HARiSA can be used as a lookup argument in itself due to the uniqueness of the elements. However, if duplication occurs (which is quite probable), the membership proof cannot be used as the lookup argument. To illustrate, consider the table $T = \{t_1, t_2, \ldots, t_N\}$ and the lookup elements $\vec{f} = (f_1 =$

[5]Even if this is a strong assumption, we assume that all the elements have an applicable form to be used in HARiSA to concentrate on the duplication. The inspection of the suitability of the elements is in Section 4.2.

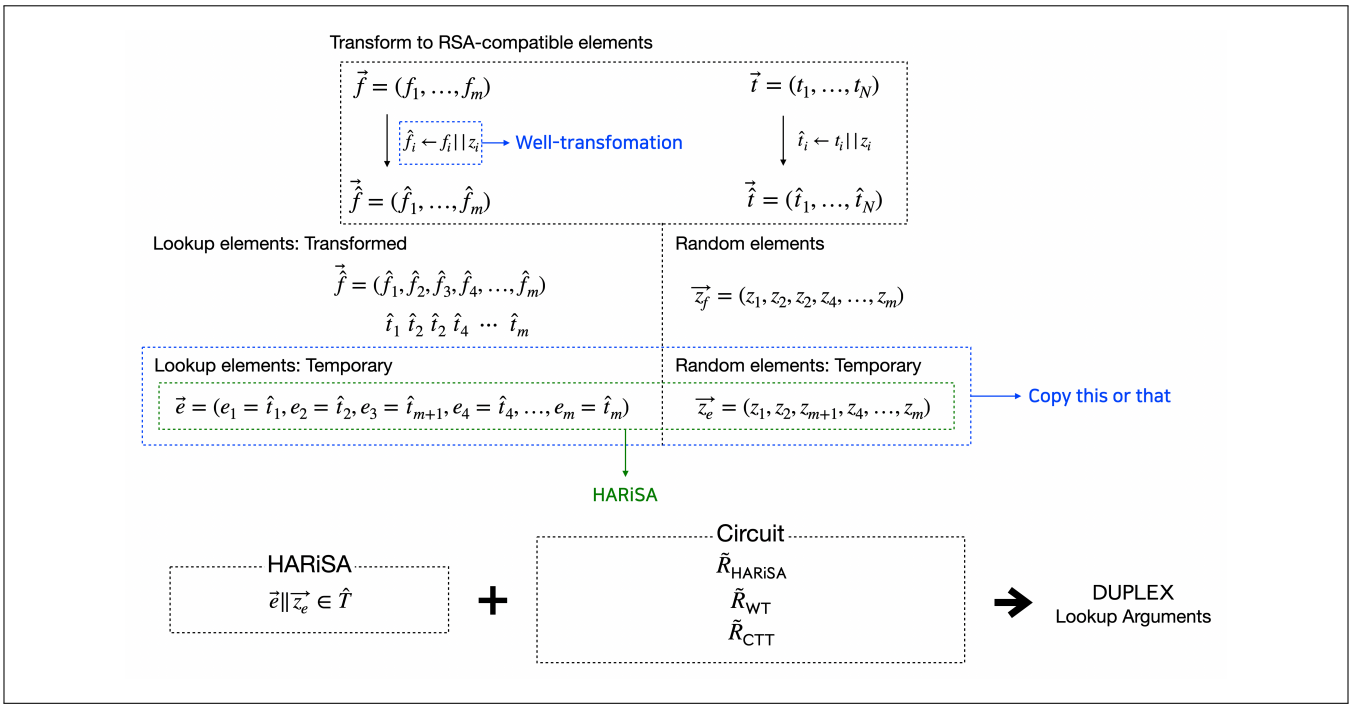


Figure 2: Bird's eye view of our protocol. The initial elements are assumed to be sorted in ascending order. $\tilde{R}_{\text{HARiSA}}$ denotes the relation proven under SNARK circuit in HARiSA. $\tilde{R}_{\text{ck}}^{\text{WT}}$ and $\text{cp}\Pi^{\text{CTT}}$ denotes the relations for well-transformation (double lookup, less work in Section 4.2) and duplication handling (copy-this-or-that in Section 4.1) respectively.

$t_1, f_2 = t_2, \ldots, f_i = t_i, \ldots, f_j = t_i, \ldots, f_m = t_m$) for $N > m$. As shown, owing to $f_i = f_j$ for $i \neq j$, generating a witness becomes problematic. In this case, for an accumulator $\text{acc} \leftarrow g^{\prod_T}$, the witness is generated as $W \leftarrow g^{\prod_T / \prod_f}$. In order to prove membership, a prover should pass W, and lookup elements $\vec{f}$[6]. However, t_i appears twice in $\vec{f}$ (at f_i and f_j) while acc contains only one t_i. Hence, it is hard to prove that t_i appears twice through a membership proof. If we update the accumulator reflecting the duplication, $\text{acc} \leftarrow g^{\prod_{T'} s.t. T' = T \cup \{t_i\}}$, to get two t_i in the accumulator. Moreover, since the accumulation phase precedes the accumulation phase, and the number and identity of duplicated elements vary with each invocation, it is impractical to reflect duplications in the accumulator. For simplicity, if $\vec{f}$ only consists of m copies of t_i, the accumulator must handle $m-1$ duplicated elements by multiplying $\text{acc}' \leftarrow \text{acc}^{\prod_{k=1}^{m-1} t_i}$. Besides, this can affect the security in that the elements are not coprime if the duplication is reflected. Thus, duplication has to be managed carefully to avoid harming security and practicality.

To handle such duplication, we combine the *copy-this-or-that* technique introduced in Halo2 [9] with HARiSA. The copy-this-or-that proves that all the duplicated elements are indeed the same. For example, assume the vector of lookup elements $\vec{f} = (f_1, f_2, f_3, f_4, \ldots, f_m)$ and f_3 and f_4 are same. To shed light on, we suppose that the vector of table elements $\vec{t}$ and the vector of lookup elements $\vec{f}$ are sorted by ascending order without loss of generality. At this point, the copy-this-or-that proves that f_3 and f_4 are the indeed in the set even though they have same value. For T and $\vec{f}$, generate a temporary vector $\vec{e} = (e_1 = f_1, e_2 = f_2, e_3 = f_3, e_4 = t_{m+1}, e_5 = f_5, \ldots, e_m = f_m)$. Note that t_{m+1} is also a legitimate element in the table T. Proving $\forall i \in [2, m], (f_i - e_i)(f_i - f_{i-1}) = 0 \wedge f_1 = e_1$ ensures that all the elements in $\vec{e}$ contain the elements of $\vec{f}$ at the corresponding index, or valid other elements which exist in the range $[m+1, N]$. Finally, if $\vec{e}$ passes verification of the membership proof, then it implies that all of the elements in $\vec{f}$ are in the table subsequently[7]. Formally, for a vector of lookup elements $\vec{f}$, and the temporary (sorted) vector of lookup element $\vec{e} \in \mathbb{F}^m$, the relation for copy-this-or-that is as follows:

$$\tilde{R}_{\text{ck}}^{\text{CTT}}(c_{\vec{f}}, c_{\vec{e}}) = 1 \Leftrightarrow \forall i \in [2, m] : (f_i - e_i)(f_i - f_{i-1}) = 0 \wedge f_1 = e_1$$

For a vector of lookup elements $\vec{f} \in \mathbb{F}^m$ and the temporary vector $\vec{e} \in \mathbb{F}^m$ to handle duplication in $\vec{f}$, $\tilde{R}_{\text{ck}}^{\text{CTT}}$ proves that the element $f_i \in \vec{f}$ is equal to either f_{i-1} or $e_i \in \vec{e}$, and e_1 is a copy of f_1. $\tilde{R}_{\text{ck}}^{\text{CTT}}$ is proven with HARiSA to assure that $\vec{e}$ consists of only valid elements in table T. If both the membership proof for $\vec{e}$ and SNARK proof for $\tilde{R}_{\text{ck}}^{\text{CTT}}$ are verified, it implies that $\vec{e}$ contains only

[6] Rigorously, lookup elements are passed in hidden through masking with some randomness ($k \leftarrow r + f_i sh$) in HARiSA.

[7] In general, since the number of lookup elements is smaller than the table size, we assume that $N > m$

valid elements in T, and e_i is a copy of f_i if f_i is a unique element, or a copy of some t_k which does not appear in $\vec{f}$ but in $\vec{t}$ otherwise.

4.2 Double Lookup, Less Work: Proving well-transformation of elements

The second challenge pertains to the requirement that all elements within the table must prime numbers or hashed ones through a collision-resistant hash-to-prime function to construct lookup argument from HARiSA, which plays over RSA group. To accommodate this, the lookup elements f must undergo a transformation into elements $\hat{f}$ that are compatible with the RSA group. This transformation is not trivial; it necessitates the validation that the transformed elements $\hat{f}$ correctly reflect its original counterparts f. As the lookup elements are witnesses, the integrity of the transformation must be proven in the SNARK circuit. However, ensuring such well-transformation is computationally intensive, incurring $O(m)$ of heavy computation such as hash functions where m denotes the number of lookups. This intensive computation in the circuit significantly impacts the scalability of HARiSA when applied to lookup arguments, even if solution to the elements duplication exists.

For instance, suppose that the table elements are $T = \{t_1, \ldots, t_N\}$, and lookup elements are in $\vec{f}$. Naively, we can use hash-to-prime where we can map each element of table into a unique prime. Then, the converted table $\hat{T} = \{\hat{t}_1, \ldots, \hat{t}_N\}$ where $\hat{t}_i \leftarrow H_{prime}(t_i)$. In this case, the prover computes $\hat{f}_i \leftarrow H_{prime}(f_i)$. The prover should prove that 1) $\hat{f}_i$ is in the table $\hat{T}$ and 2) $\hat{f}_i$ is indeed the output of hash-to-prime of f_i. However, the latter incurs m-hash check in the circuit, and this lets program computationally heavy. Another potential solution is to compute a random number z_i making $\hat{f}_i = f_i \| z_i$. This approach leads to checking $\hat{f}_i = f_i \| z_i$ in the circuit. Additionally, to guarantee that there is no f_j such that $\hat{f}_i = f_j \| z_i$ where $i \neq j$, we should check the range of z_i. If z_i exceeds (or close to) the size of f_i, there can be f_j which makes $\hat{f}_i = f_j \| z_i$ for $i \neq j$[8]. Although it mitigates prover's computation compared to the previous method, it still requires excessive computation than necessary.

We propose a novel technique for the efficient lookup arguments over RSA group. The sketch of our technique is as follows: 1) Prover computes random prime number z_i such that $\hat{f}_i = f_i \| z_i$ letting $\hat{f}_i$ be a prime number. The difference with aforesaid approach is that the z_i should be a *prime number* as well as random. 2) Make the table for z_i. This table can exist independently or as together with a table of $\hat{t}_i$. 3) Prover proves that $\hat{f}_i = f_i \| z_i$ and $\hat{f}_i$ and z_i are in the table respectively using HARiSA. Since both $\hat{f}_i$ and z_i are prime number, there is no restriction proving membership with HARiSA. However, since f_i is witness, we should encode that the transformation is correctly done (we refer to it as well-transformation) in the SNARK circuit. If the verification passes, it is ensured that the f_i is a valid member and $\hat{f}_i$ is well-transformed from f_i simultaneously. This solution yields only one additional lookup (for z_i) per existing lookup (for $\hat{f}_i$) which doubles the prover's work but remains manageable. Furthermore, it does not affect asymptotic complexity of the prover's work, $O(m \log m)$. The relation for well-transformation is as follows:

$$\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}(c_{\vec{\hat{f}}}, c_{\vec{z}}, c_{\vec{f}}) = 1 \Leftrightarrow \forall i \in [m], \hat{f}_i = f_i \| z_i \Leftrightarrow \hat{f}_i = f_i \cdot 2^n + z_i$$

$\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$ proves that $\hat{f}_i$ is well-transformed from f_i and z_i of n bits. Note that since z_i is publicly opened as a table element with t_i, it doesn't need to be proven whether z_i is prime or not. With HARiSA, if $\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$ is satisfied and the membership proof for $\hat{f}_i$ and z_i is verified, then it implies the well-transformation.

Remark 1. *Note that since the probability that a prime number x exists is approximately $\frac{1}{\log x}$ according to the Prime Number Theorem, the size of z_i is sufficient for $(\log a + \log\log a)$-bit to make $\hat{f}_i$ and z_i themselves prime numbers where f_i is a bits in size. Considering z_i is the size of b-bit, $\hat{f}_i$ is the size of $(a+b)$-bit. Since $z_i \simeq 2^b$ and the probability that $\hat{f}_i$ is prime is $\frac{1}{a+b}$, there exists at least one prime number $\hat{f}_i \in \{0,1\}^{a+b}$ if $2^b \geq a+b$. Accordingly, $b \simeq \log(a+b)$ to $\hat{f}_i$ be prime number. Similarly, the probability that z_i is a prime number is $\frac{1}{b}$ in this case. Consequently, the inequality $z_i \geq a \log a$ ensures $\hat{f}_i$ and z_i are prime number. In other words, z_i is sufficient for $(\log a + \log\log a) - bit$.*

For example, if f is size of 16 bits, then, z is sufficient for 6 bits. However, we can set the size of z greater as necessary. In our construction, since the domain of the elements should be greater than the size of $\mathbb{P}_{2\lambda}$, i.e., z should be greater than the 2λ-th prime[9]. As it is shown, the size of the auxiliary z remains small and does not increase the overall size significantly.

Remark 2. *Unlike HARiSA, our proposed scheme does not need to prove the bound of the element R^{bound} separately due to the existence of $\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$. In HARiSA, R^{bound} proves that the elements are in the specific domain, i.e., all u_i are greater than public integer B. In detail, the elements are greater than the blinding factors $p_i \in \mathbb{P}_{2\lambda}$, which hide the witness W. Through transformation, the lookup elements are transformed as $\hat{f}_i \leftarrow f_i || z_i$ and these transformations are proven with $\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$. Note that since the smallest element among $\vec{z}$ is greater than $\mathbb{P}_{2\lambda}$, all the elements in $\vec{\hat{f}}$ are prime numbers. Consequently, $\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$ implies R^{bound}, we can therefore omit R^{bound} in DUPLEX.*

4.3 Our construction for lookup argument DUPLEX

Corresponding with the above description, our construction is portrayed in Figure 3. In the setup phase, the initial table T is transformed into vector of prime numbers $\vec{t} \leftarrow t_i \| z_i$ such that z_i is a prime number as described in Section 4.2. Then, the prover argues largely two parts. The first one is that the transformation of lookup elements is correctly done. The second is that the duplicated elements are also in the table as well as the original ones.

[8] Generally, the concatenation check evokes bitwise operation. For example, $\hat{f}_i = f_i \| z_i$ can be checked through $\hat{f}_i = f_i \cdot 2^n + z_i$

[9] It can be different depending on security parameter setting. Under our experiment setting, we choose $2\lambda = 256$ primes, z should be greater than 1621, the 2λ-th prime number.

$\underline{\text{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}, T\right):}$

$\forall i \in [N]:$ Compute a random prime z_i such that $\hat{t}_i \leftarrow t_i || z_i$ and $\hat{t}_i$ is a prime number

Update the set $\hat{T} = \{\hat{t}_1, \ldots, \hat{t}_N, z_1, \cdots, z_N\}$

$\mathsf{crs}_1 \leftarrow \mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\mathsf{WT}})$

$\mathsf{crs}_2 \leftarrow \mathsf{cp}\Pi^{\mathsf{CTT}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\mathsf{CTT}})$

$\mathsf{crs}_3 \leftarrow \textsc{harisa}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \mathsf{pp})$

$\mathsf{crs} = (\mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$

return $(\mathsf{crs}, \hat{T})$

$\underline{\mathsf{LookupPrv}\left(\mathsf{crs}, \mathsf{acc}, \hat{T}, \vec{\hat{f}}, \vec{f}, \vec{z_f}\right):}$

Create a temporary vector $\vec{e}$ such that $e_i = \hat{f_i}$

if $\hat{f_i}$ has no duplication, $e_i = \hat{t}_k$ for $k \leftarrow\!\!\$\ [m+1, N]$ otherwise.

$\vec{u} \leftarrow \vec{e}.\mathsf{append}(\vec{z_e})$

$\pi_1 \leftarrow \mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Prv}(\mathsf{crs}_1, c_{\vec{\hat{f}}}, c_{\vec{f}}, c_{\vec{z_f}}; \vec{\hat{f}}, o_{\vec{\hat{f}}}, \vec{f}, o_{\vec{f}}, \vec{z_f}, o_{\vec{z_f}})$

$\pi_2 \leftarrow \mathsf{cp}\Pi^{\mathsf{CTT}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\vec{u}}, c_{\vec{e}}; \vec{u}, o_{\vec{u}}, \vec{e}, o_{\vec{e}})$

$\pi_3 \leftarrow \textsc{harisa}.\mathsf{OptPrv}(\mathsf{crs}_3, \mathsf{acc}, c_{\vec{u}}; \vec{u}, o_{\vec{u}})$

$\pi = (\pi_1, \pi_2, \pi_3)$

return π

$\underline{\mathsf{LookupVfy}\left(\mathsf{crs}, \mathsf{acc}, c_{\vec{u}}, c_{\vec{\hat{f}}}, c_{\vec{f}}, c_{\vec{z_f}}, c_{\vec{e}}, \pi\right):}$

Reject if $\mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Vfy}(\mathsf{crs}_1, c_{\vec{\hat{f}}}, c_{\vec{f}}, c_{\vec{z_f}}, \pi_1) \neq 1$

Reject if $\mathsf{cp}\Pi^{\mathsf{CTT}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{u}}, c_{\vec{e}}, \pi_2) \neq 1$

Reject if $\textsc{harisa}.\mathsf{Vfy}(\mathsf{crs}_3, \mathsf{acc}, c_{\vec{u}}, \pi_3) \neq 1$

Figure 3: Our proposed lookup argument DUPLEX

The former one can be proven under $\tilde{R}^{\mathsf{WT}}_{\mathsf{ck}}$. $\mathsf{cp}\Pi^{\mathsf{WT}}$ generates the proof π_1 which proves that $\hat{f_i} = f_i || z_{f_i}$ for all $i \in [1, m]$. Note that z_{f_i} is an auxiliary prime z corresponding to f_i. The latter one can be viewed as two steps again: 1) Ensuing that the duplication of $\vec{\hat{f}}$ and $\vec{z_f}$ are accurately reflected to temporary vector $\vec{e}$. 2) Verifying all the elements, including duplicated ones, are legitimate elements from table. The first one can be proven under $\tilde{R}^{\mathsf{CTT}}_{\mathsf{ck}}$. π_2 checks that $(u_i - e_i)(u_i - u_{i-1}) = 0$ which implies that all $\hat{f_i}$ and z_{f_i} are copy of either the previous element or another valid element that is in $\hat{T}$, but not in $\vec{u}$. Sequentially, HARISA is evoked to generate π_3 which proves the membership proof of $\vec{u}$. Note that $\vec{u}$ is merging of temporary vector $\vec{e}$ and its corresponding auxiliary primes $\vec{z_e}$. If all verification pass, it means that $\vec{\hat{f}}$ is well-transformed (with π_1), $\vec{\hat{f}}$ contains only valid duplication (with π_2) and $\vec{\hat{f}}$ comprises only the elements from table (with π_3)[10].

4.4 Precomputation for Optimizing Witness Generation

During invoking HARISA in DUPLEX, a prover may generate witness from scratch. Observing HARISA closely, a prover has to generate witness W as $g^{\prod_{f_i \in T \setminus \{f_j\}} f_i}$ to prove inclusion proof for f_j[11]. Computing W from scratch costs $O(N)$-exponentiation over RSA group. In order to optimize witness generation, we adopt precomputation in [38] as follows: For example, let $g \leftarrow \mathbb{G}_?$ and the set $T = \{t_1, t_2, t_3, t_4\}$. Using a divide-and-conquer approach, we divide the set into $T_L = \{t_1, t_2\}$ and $T_R = \{t_3, t_4\}$ and calculate the exponentiation of all elements in T_R and T_L to get $g_{1,1} = g^{\prod T_R}, g_{1,2} = g^{\prod T_L}$. For $g_{1,1}$ and $g_{1,2}$, repeating the previous step, it is possible to calculate ingredients of the witness as $g_{2,1} = g_{1,1}^{t_2} = g^{t_2 t_3 t_4}$, $g_{2,2} = g_{1,1}^{t_1} = g^{t_1 t_3 t_4}$, $g_{2,3} = g_{1,2}^{t_4} = g^{t_1 t_2 t_4}$, and $g_{2,4} = g_{1,2}^{t_3} = g^{t_1 t_2 t_3}$. As a result, precomputation takes $O(N \log N)$.

Precomputation allows a prover to compute witness without exponentiating all elements. With ingredients, a prover requires only $O(m \log m)$ multiplication to compute W. With extended Euclidean algorithm, a prover can compute Bezout coefficient easily. Using the extended Euclidean algorithm, a prover is able to compute Bezout coefficient easily, i.e., a prover can get a and b such that $ax + by = K$ for $gcd(x, y) = K$. Based on this fact, a prover can compute the witness for f_i, and f_j easily with the so-called ingredients. If the prover tries to compute W for $f_1 = t_1$ and $f_3 = t_3$ in the above example, it can compute a and b such that $a \cdot f_1 + b \cdot f_3 = 1$. For the ingredients $g_{2,1}$ and $g_{2,3}$, a prover computes $g_{2,1}^b \cdot g_{2,3}^a = g^{a \cdot \frac{\pi_T}{f_3} + b \cdot \frac{\pi_T}{f_1}} = g^{\frac{\pi_T (a \cdot f_1 + b \cdot f_3)}{f_1 \cdot f_3}} = g^{\frac{\pi_T}{f_1 \cdot f_3}}$. In general, by recursively performing this procedure, we can compute the witness in $O(m \log m)$ time.

Investigating the trade-off of the precomputation, it requires storage for the ingredients while it can enhance the proving time. Viewing it as a tree structure (in Figure 7, Appendix A), a prover has to store all N leaves in the tree. For example, a prover is in need of 256MB storage for table of $N = 2^{20}$. Yet, it is still attractive in that it necessitates computation only once during the initial phase while it can significantly reduce the proving time. The details are deferred to Appendix A.

4.5 Security Analysis

We present the intuition of the security for our proposed scheme and provide a sketch proof. As DUPLEX is built upon HARISA, the overall security relies on the security of HARISA. The major differences from HARISA are that DUPLEX includes two additional components: duplication handling and element transformation. The former, as shown in Section 4.1, just brings multiplication check in the SNARK circuit on $\tilde{R}^{\mathsf{CTT}}_{\mathsf{ck}}$ and nothing else. Thus, duplication handling does not harm the existing security of HARISA. Inspecting

[10] For better understanding, we depict each relation producing distinct proofs as π_1 π_2, and π_3. In practice, all the relations can be proven with a single proof using the same SNARK.

[11] For the better delivery, we focus more on HARISA. The explanation is on the context of membership proof.

well-transformation, it renders whole elements to be fit to RSA group without hash-to-prime which makes a significant overhead to prover. To achieve transformation, we introduce a subsidiary random element prime number z. In other words, the element itself is different from those in HARiSA. As the transformation itself is proven in the SNARK circuit on $\tilde{R}_{\mathsf{ck}}^{\mathsf{WT}}$, we need to consider the modified form of the elements affects the security of HARiSA. Recall that z_i and $\hat{f}_i$ are prime numbers which are not in $\mathbb{P}_{2\lambda}$. Then, every accumulated elements in $\vec{t}$ and $\vec{z}$ is distinct, thus it can be considered as a general membership proof based on RSA accumulator.

Theorem 1. *Let H and H_{prime} be modeled as random oracles, HARiSA in Figure 1 be secure, and $\mathsf{cp}\Pi^{\mathsf{WT}}$ and $\mathsf{cp}\Pi^{\mathsf{CTT}}$ be secure CP-SNARKs. The construction in Figure 3 is a secure CP-SNARKs: knowledge-sound under the adaptive root assumption, zero-knowledge under the DDH-II assumption, and succinct.*

Proof. For knowledge-soundness, it is clear that the transformed elements and duplication handling do not affect on the knowledge-soundness of HARiSA. As described previously, two relations $\tilde{R}_{\mathsf{ck}}^{\mathsf{CTT}}$ and $\tilde{R}_{\mathsf{ck}}^{\mathsf{WT}}$ proven in the SNARK circuit arise, and the lookup argument along with the both the table and lookup elements can be considered as a general membership proof after transformation. Likewise, the zero-knowledge for witness W can be maintained since the transformed elements are prime numbers greater than $\mathbb{P}_{2\lambda}$. As a result, it is secure under the DDH-II assumption.

DUPLEX involves HARiSA and two additional components, the duplication handling and the element transformation. In terms of verification, these two things only produce two more relations $\tilde{R}_{\mathsf{ck}}^{\mathsf{CTT}}$ and $\tilde{R}_{\mathsf{ck}}^{\mathsf{WT}}$, which are proven in the SNARK circuit. Consequently, verification time is identical to the one in HARiSA. Rigorously, the element transformation leads to twice membership proof (HARiSA) for a single lookup element. However, since HARiSA is able to be proven in a batch manner via PoKE [7], DUPLEX can reap benefits from batch proving, resulting in constant verification and constant-sized proof. DUPLEX eventually satisfies the succinctness. □

4.6 Observation on Flexibility

In this section, we sift the *flexibility* where the lookup arguments can be applied to SNARKs of any arithmetization (e.g., AIR, Plonkish, or R1CS) without incurring additional costs by comparing DUPLEX with existing works. The main difference between DUPLEX and other works stems from perspective in which we approach the lookup argument. Most of existing works represent the lookup argument in polynomial, whereas DUPLEX expresses it as a type of membership proof after modifications described in Section 4.1 and 4.2. Plookup [25] proves the lookup argument through the divisibility of the product of $\vec{f}$ and $\vec{t}$ where $\vec{f}$ is the lookup element, and the $\vec{t}$ is the table element. Caulk [48], Ba*loo* [49], and cq [23] employs a similar method to Plookup except that the elements are handled through KZG polynomial commitment [29] and some polynomial protocol terminology from [26]. That is, the above arguments are confined to the Plonkish arithmetization.

Lasso [42] proposes a distinct approach by showing that there exists a sparse matrix M such that $M \times \vec{t} = \vec{a}$ for the table elements $\vec{t}$ and lookup elements $\vec{a}$. In Lasso, the table is decomposed into subtables $\{\vec{t}_1, \ldots, \vec{t}_\alpha\}$, where the tensor product of subtables $\bigotimes_{i\in[\alpha]} \vec{t}_i = \vec{t}$. Lasso shows flexibility by converting it into customizable constraint systems (CCS), as introduced in [41]. However, it handles elements as sparse polynomials, incurring additional cost proportional to the number of non-zero entries in the CCS matrices in order to convert the arithmetization into CCS form. That is, the cost of the prover is proportional to the number of non-zero elements in the CCS matrix, which means that the number of inputs affects. With such matrix-vector product based approach, the lookup arguments are proven by matching values in the decomposed matrix and offline memory checking[12], which are reduced to multilinear polynomials to facilitate the validation of matrix-vector multiplication. In practice, Diamond and Posen [21] applies Lasso to Hyperplonk [18] and Babyspartan [40] applies Lasso to SuperSpartan [41], extending Plonkish constraint system to a CCS.

On the other hand, DUPLEX manages each lookup element as a (committed) value itself. As demonstrated previously, the table elements and lookup elements can be regarded as set elements and membership elements (to be proven) respectively in the context of membership proof scheme after transformation into prime numbers (see double lookup, less work in Section 4.2). As HARiSA does, each element is handled as a commitment in DUPLEX, and the commitment is taken as input to the SNARK circuit, which validates the relations $\tilde{R}_{\mathsf{ck}}^{\mathsf{CTT}}$ and $\tilde{R}_{\mathsf{ck}}^{\mathsf{WT}}$. Namely, DUPLEX can be used along with SNARK circuit of any arithmetization through commit-and-prove way. The flexibility of arithmetizations provides advantages especially when the consistency in computations across heterogeneous SNARK circuits is needed to be validated. In practice, for example, DUPLEX can ameliorate the interoperability of the blockchain by efficiently validating computations from different blockchains where each blockchain utilizes SNARKs of different arithmetization.

4.7 Extension to Dynamic Table

DUPLEX can be extended to the dynamic tables where the table elements are updated over time. The update can appear verifiable outsourcing of state update [10] which is developed recently for blockchain environments or RAM construction [16, 22, 36]. Briefly, the states are stored in a public table (set) T, and a short digest of T is stored (or published) as an accumulator acc. When some states are updated, the updated table T' is produced and the accumulator acc$'$ is computed corresponding to T'. By proving insertion and deletion between T and T' with acc and acc$'$, a prover claims that the states are updated correctly[13].

An RSA accumulator can be used to manage the dynamic table. B-INS-ARiSA which is introduced in [16, Section 5] suggests the way to prove correctness of update of the table. By combining it with DUPLEX, we can construct a lookup argument for dynamic environments. Concisely, B-INS-ARiSA is called when an update occurs to prove that the updated accumulator acc$'$ comprises valid elements. It is essential to consider that the table is transformed in DUPLEX. Therefore, DUPLEX for a dynamic table operates as follows.

[12] The offline memory checking is independent of our interest. We refer to [39, 42] for more detail.

[13] Such insertion and deletion are handled all together, which is referred to as MultiSwap in [36]

Given the existing table T, updated table T', and the accumulator acc for T, the vector for updated elements is denoted as $\vec{u}$, i.e., $T' \leftarrow T \cup \{u_i\}_{u_i \in \vec{u}}$. The transformed elements are denoted with a hat mark, that is, $\hat{T}$, $\hat{T}'$, and $\vec{\hat{u}}$. Then, we prove that $\vec{\hat{u}}$ is well-transformation of $\vec{u}$ first, then run the B-INS-ARISA with $\hat{T}, \hat{T}'$, acc, and acc$'$. By proving the well-transformation for updated elements $\vec{u}$, it is ensured that the updated table $\hat{T}'$ correctly reflects the updates from the $\hat{T}$ with $\vec{\hat{u}}$, which is the vector of transformed elements. The lookup argument protocol for a dynamic environment can be constructed by proceeding DUPLEX in sequence. Since the set update is deterministic and both the existing table and the updated table (as well as their corresponding accumulators) are publicly accessible, proving the set updates as zero-knowledge is not of primary interest. However, preserving privacy for lookup argument is still possible since proving the set update does not affect the subsequent DUPLEX. Namely, the lookup elements are not correlated with the updated elements. Therefore, we can provide privacy for the lookup argument in a dynamic setting. Full description for our construction is in Appendix B (Figure 8).

5 Evaluation

5.1 Instantiations and Implementation

We consider the construction of the DUPLEX scheme as depicted in Figure 3 for an arbitrary table. Note that it is feasible because DUPLEX includes a transformation phase to make the table elements applicable to HARISA. Additionally, with the precomputation introduced in Figure 6, we can optimize the proving time. We instantiate the CP-SNARKs building block of the construction, $\mathsf{cp}\Pi^{\mathsf{arithm}}$, $\mathsf{cp}\Pi^{\mathsf{CTT}}$, and $\mathsf{cp}\Pi^{\mathsf{WT}}$, with LegoGroth16 from [17], an efficient commit-and-prove SNARK based on Gro16 [27]. Identically with Gro16, it works over the bilinear map. We use the curve BN254 for our instantiation. For the accumulator scheme, we use a 2048-bit RSA group. As HARISA does, we use $2\lambda = 256$ primes[14] to hide the RSA witness in our construction.

We implement DUPLEX using the Arkworks library [1] in Rust. The implementation consists of HARISA, LegoGroth16, and relations described above. All the benchmarks depicted in this section were obtained by running on a laptop with an Apple M1 Pro processor and 32GB of RAM.

5.2 Benchmarks for Lookup Arguments

We evaluate our proposed scheme DUPLEX by comparing it to Caulk [48], the state-of-the-art. Since Caulk presents a lookup argument satisfying zero-knowledge, and it is the only one that provides its implementation and evaluation in the literature, it is the optimal candidate for comparison to analyze the performance of DUPLEX. We also provide a comparison table (Table 1) for the asymptotic complexity with existing works.

5.2.1 Benchmarks for a Single Lookup Element In Figure 4, we compare DUPLEX to Caulk for single opening (i.e., $m = 1$) by varying the log-scale set size N on the x-axis. We observe that the DUPLEX remains constant for all values of N while Caulk's proving time grows gradually as set size increases. The difference arises because Caulk has set-related factor in its proving time as well as batch size ($O(m^2 + m \log N)$) while DUPLEX is only affected by batch size ($O(m \log m)$). In case of the smallest set size ($N = 2^6$ in this case), DUPLEX is nearly 5× faster than Caulk.

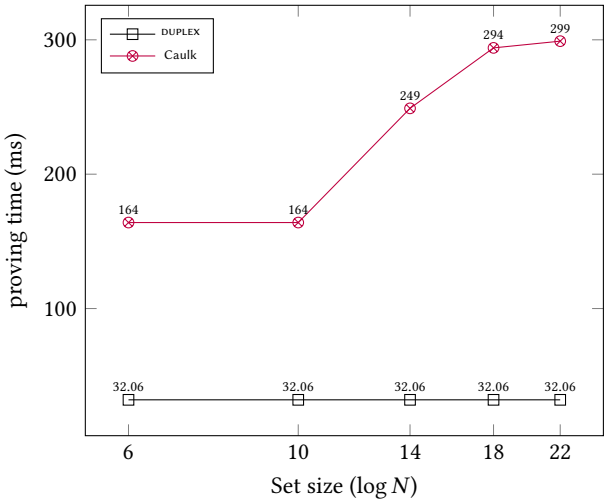


Figure 4: Comparison of lookup argument for single opening in zero-knowledge: Our work (DUPLEX) vs Caulk.

5.2.2 Benchmarks for Batch Lookup Elements Figure 5 depicts the batch proving for the lookup argument comparing DUPLEX to Caulk. The Caulk-* represents Caulk's performance, where the asterisk symbol stands for the log-scale table size ($\log N$ in this paper). In Figure 5, we show the proving time in the y-axis in the log-scale varying the batch size m in the x axis. DUPLEX is faster than Caulk for all values of m, regardless of whether Caulk's set size is large ($N = 2^{20}$) or small ($N = 2^8$)[15]. For larger batch sizes, we expect that DUPLEX will have more benefit than Caulk since Caulk's proving time has quadratic component on m. Even though Caulk's verification time and proof size are slightly better than DUPLEX, we argue that HARISA is still practical since it also has constant asymptotic complexity and the estimation is quite small as well.

We provide more detailed performance metrics of DUPLEX in Table 2, including results for larger batch sizes. Since DUPLEX has the advantage of batch proving, we can see that it maintains practicality even as the batch size increases.

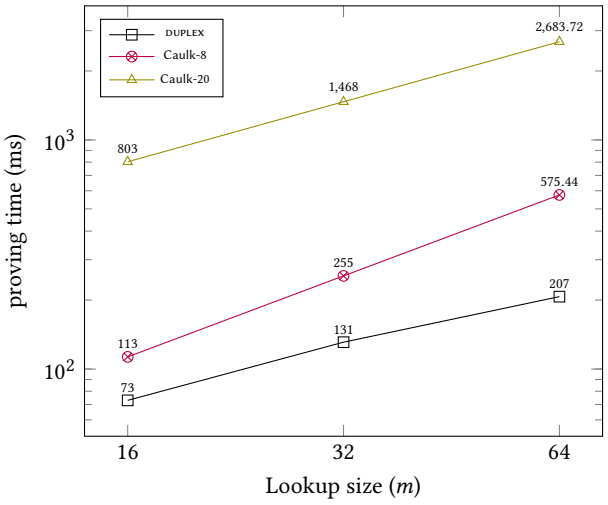
5.2.3 Analysis on CRS size and RAM consumption. Our proposed DUPLEX has benefits in terms of CRS size and memory consumption as well. Theoretically, we have a small-sized CRS independent of set size, while Caulk has $O(N)$-sized srs. For batch sizes $m = (1, 16, 64, 256)$, the CRS sizes are 14.4KB, 91.58KB, 433.65KB, and 1.71MB, respectively. DUPLEX occupies 1.81MB, 21.14MB, and 85.47MB of memory for $m = (1, 16, 64)$. For larger batch sizes $m = (256, 1024)$, DUPLEX consumes 337.01MB and 1.34GB of

[14] Strictly speaking, HARISA uses 232 primes. We use 256 primes, a power of 2, for computational convenience. Note that since we set λ greater than the minimum, security is maintained under DDH-II assumption.

[15] Note that since the table size N doesn't affect the proving time of DUPLEX, we experimented with $N = 2^{11}$ for convenience.

	Proving time (s)					Verification time (ms)					Proof size
$m =$	**16**	**32**	**64**	**256**	**1024**	**16**	**32**	**64**	**256**	**1024**	**(KB)**
DUPLEX	0.073	0.131	0.207	0.718	2.902	31.537	50.072	43.889	47.232	60.137	0.895

Table 2: Performance table of DUPLEX.



Scheme	**V time** (ms)	**Proof size** (KB)
Caulk-$*$	36	0.89
DUPLEX	46	1.17

Figure 5: Comparison of lookup arguments in zero-knowledge: Our work (DUPLEX) vs Caulk. The plot is in log-scale. The estimated verification time and proof size are average value of Table 2. Verification time and proof size of both schemes remain constant regardless of set/batch size.

memory. As the estimation shows, DUPLEX has feasibility even for large batch sizes. We expect that the low memory usage adds up to competitive power when employing the practical application demanding large size such as large language model (LLM) or ethereum virtual machine (EVM).

Acknowledgments

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2021-0-00518, Blockchain privacy preserving techniques based on data encryption) and by a grant from ZK Proof Research Center.

References

[1] arkworks contributors. 2022. `arkworks` *zkSNARK ecosystem.* https://arkworks.rs
[2] Arasu Arun, Srinath Setty, and Justin Thaler. 2023. Jolt: SNARKs for Virtual Machines via Lookups. Cryptology ePrint Archive, Paper 2023/1217. https://eprint.iacr.org/2023/1217 https://eprint.iacr.org/2023/1217.
[3] Niko Baric and Birgit Pfitzmann. 1997. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*. Springer, 480–494.
[4] barry WhiteHat. 2018. roll_up: Scale ethereum with SNARKs. https://github.com/barryWhiteHat/roll_up.
[5] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2018. Aurora: Transparent Succinct Arguments for R1CS. Cryptology ePrint Archive, Paper 2018/828. https://eprint.iacr.org/2018/828 https://eprint.iacr.org/2018/828.
[6] Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. 2019. Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. *IACR Cryptol. ePrint Arch.* 2019 (2019), 1255.
[7] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Annual International Cryptology Conference*. Springer, 561–586.
[8] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. 2018. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 595–626.
[9] Sean Bowe, Jack Grigg, and Daira Hopwood. [n. d.]. Halo2. https://github.com/zcash/halo2.
[10] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. 2013. Verifying computations with state. In *Proceedings of the twenty-fourth ACM Symposium on Operating Systems Principles*. 341–357.
[11] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 677–706.
[12] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *International workshop on public key cryptography*. Springer, 481–500.
[13] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual international cryptology conference*. Springer, 61–76.
[14] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. 2023. Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees. Cryptology ePrint Archive, Paper 2023/1518. https://eprint.iacr.org/2023/1518 https://eprint.iacr.org/2023/1518.
[15] Matteo Campanelli, Dario Fiore, and Rosario Gennaro. 2024. Natively Compatible Super-Efficient Lookup Arguments and How to Apply Them. Cryptology ePrint Archive, Paper 2024/1058. https://eprint.iacr.org/2024/1058 https://eprint.iacr.org/2024/1058.
[16] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. 2022. Succinct Zero-Knowledge Batch Proofs for Set Accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 455–469. https://doi.org/10.1145/3548606.3560677
[17] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2019/142. https://doi.org/10.1145/3319535.3339820 https://eprint.iacr.org/2019/142.
[18] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 499–530.
[19] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 560–574.
[20] Ivan Damgård and Nikos Triandopoulos. 2008. Supporting Non-membership Proofs with Bilinear-map Accumulators. *IACR Cryptol. ePrint Arch.* 2008 (2008), 538.
[21] Benjamin E Diamond and Jim Posen. 2023. Succinct arguments over towers of binary fields. *Cryptology ePrint Archive* (2023).
[22] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. 2024. Batching-Efficient RAM using Updatable Lookup Arguments. *Cryptology ePrint Archive* (2024).
[23] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763. https://eprint.iacr.org/2022/1763 https://eprint.iacr.org/2022/1763.

[24] Ariel Gabizon and Dmitry Khovratovich. 2022. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. *Cryptology ePrint Archive* (2022).
[25] Ariel Gabizon and Zachary J Williamson. 2020. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive* (2020).
[26] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).
[27] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.
[28] Ulrich Haböck. 2022. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530. https://eprint.iacr.org/2022/1530 https://eprint.iacr.org/2022/1530.
[29] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.
[30] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2024. vcnn: Verifiable convolutional neural network based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing* (2024).
[31] Jiangtao Li, Ninghui Li, and Rui Xue. 2007. Universal accumulators with efficient nonmembership proofs. In *International Conference on Applied Cryptography and Network Security*. Springer, 253–269.
[32] Helger Lipmaa. 2012. Secure accumulators from euclidean rings without trusted setup. In *International Conference on Applied Cryptography and Network Security*. Springer, 224–240.
[33] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.
[34] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.
[35] Lan Nguyen. 2005. Accumulators from bilinear pairings and applications. In *Cryptographers' track at the RSA conference*. Springer, 275–292.
[36] Alex Ozdemir, Riad Wahby, Barry Whitehat, and Dan Boneh. 2020. Scaling verifiable computation using efficient set accumulators. In *29th USENIX Security Symposium (USENIX Security 20)*. 2075–2092.
[37] Jim Posen and Assimakis A. Kattis. 2022. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957. https://eprint.iacr.org/2022/957 https://eprint.iacr.org/2022/957.
[38] Tomas Sander, Amnon Ta-Shma, and Moti Yung. 2001. Blind, auditable membership proofs. In *Financial Cryptography: 4th International Conference, FC 2000 Anguilla, British West Indies, February 20–24, 2000 Proceedings 4*. Springer, 53–71.
[39] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. 704–737. https://doi.org/10.1007/978-3-030-56877-1_25
[40] Srinath Setty and Justin Thaler. 2023. BabySpartan: Lasso-based SNARK for non-uniform computation. *Cryptology ePrint Archive* (2023).
[41] Srinath Setty, Justin Thaler, and Riad Wahby. 2023. Customizable constraint systems for succinct arguments. *Cryptology ePrint Archive* (2023).
[42] Srinath Setty, Justin Thaler, and Riad Wahby. 2024. Unlocking the lookup singularity with Lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 180–209.
[43] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkLLM: Zero Knowledge Proofs for Large Language Models. arXiv:2404.16109 [cs.LG] https://arxiv.org/abs/2404.16109
[44] Michael Walfish and Andrew J. Blumberg. 2015. Verifying Computations without Reexecuting Them. *Commun. ACM* 58, 2 (jan 2015), 74–84. https://doi.org/10.1145/2641562
[45] Benjamin Wesolowski. 2019. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*. Springer, 379–407.
[46] Wikipedia. 2024. Duplex (building). https://en.wikipedia.org/wiki/Duplex_(building).
[47] Jiajun Xin, Arman Haghighi, Xiangan Tian, and Dimitrios Papadopoulos. 2024. Notus: Dynamic Proofs of Liabilities from Zero-knowledge RSA Accumulators. Cryptology ePrint Archive, Paper 2024/395. https://eprint.iacr.org/2024/395 https://eprint.iacr.org/2024/395.
[48] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup Arguments in Sublinear Time. Cryptology ePrint Archive, Paper 2022/621. https://eprint.iacr.org/2022/621 https://eprint.iacr.org/2022/621.
[49] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. 2022. Baloo: Nearly Optimal Lookup Arguments. Cryptology ePrint Archive, Paper 2022/1565. https://eprint.iacr.org/2022/1565 https://eprint.iacr.org/2022/1565.
[50] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2017. An expressive (zero-knowledge) set accumulator. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 158–173.

A Precomputation for Witness Generation

In this section, we show the construction of precomputation and assembly for set accumulator. As foregoing, one of the prominent overhead in HARiSA is witness generation. It takes $O(N)$-exponentiation over RSA group. To shed light on, we depict our construction as binary tree structure. In figure 7, it illustrates the structuralization of Precompute algorithm. The root of the tree is the starting point with group generator g. Whenever Precompute is called, it exponentiates some set elements over previous outputs. At first step, the left child node is exponentiation of the right half of the set over parent, g. In contrast, the right child node is exponentiation of the left half of the set over g. Those are denoted $g_{1,1}$ and $g_{1,2}$ respectively. In the same way, the "left" child node exponentiates the "right" side of the remaining elements, and the "right" child node expoenentiates the "left" side of the remaining elements. The "remaining" in here means that non-used elements in its parent node. For example, the remaining elements for $g_{2,1}$ and $g_{2,2}$ are $u_1, \ldots, u_{\frac{N}{2}}$, and the remaining elements for $g_{2,3}$ and $g_{2,4}$ are $u_{\frac{N}{2}+1}, \ldots, u_N$. If the membership proof for u_i for i which is odd, the prover simply takes $g_{\log N,i}$ as W instead of computing $O(N)$ exponentiations. This is more effective in realistic application where the prover may prove (batch) membership proof several times. To make whole preprocess tree for the set of length N, it is required to compute $O(N \log N)$ times exponentiation. We present the algorithms for precomputation in Figure 6. Newly added functions are Precompute and Assemble which are precomputation and witness generation with precomputed ingredients respectively.

B DUPLEX for Dynamic Table

As described in Section 4.7, we can apply our lookup arguments against dynamic environment where the table is updatable. In Figure 8, the construction for proving table update of DUPLEX is depicted. We prove that a (possibly) batch of commitments to the updated elements $\vec{u} = (u_1, \ldots, u_n)$ is in right domain. Recall that all of the elements throughout our system have transformed form, that is, $\hat{t} \leftarrow t||z$. Therefore, the updated elements must be transformed into $\hat{u_i} \leftarrow u_i||z_i$. Then, performing lookup argument according to the updated table $\hat{T}' \leftarrow \hat{T} \cup \{u_1, \ldots, u_n\}$. Since proving update is required once until further update occurs, the advantages of our lookup argument are preserved in dynamic environments. In detail, the space/time complexity of DUPLEX is maintained since there is no dependency between proving update and the lookup argument. Since the lookup vector in DUPLEX is not revealed publicly, zero-knowledge for lookup vector is guaranteed independently of being able to know what elements are being updated[16].

Theorem 2. *Let H_{prime} be modeled as random oracles,* $\mathsf{cp}\Pi^{\mathsf{WT}}$*, and* $\mathsf{cp}\Pi^{modarithm}$ *be secure CP-SNARKs. The construction in Figure 8*

[16] Obviously, the lookup vector can contain updated element. However, it is not known whether the lookup vector actually contains updated elements.

Precompute (w, S) :

If $|S| = 1$:

return $w^{\prod S}$

Else :

Precompute$(w^{\prod S_R}, S_L)$

Precompute$(w^{\prod S_L}, S_R)$

Assemble $(pp, a, b, w_{i,j}, w_{k,l})$:

Compute x and y such that $ax + by = 1$

$c = (w_{i,j})^{y}(w_{k,l})^{x} = g^{\frac{y}{a}\prod S + \frac{x}{b}\prod S} = g^{\frac{\prod S(by+ax)}{ab}} = g^{\frac{\prod S}{ab}}$

return c

OptPrv $\left(\text{crs}, \text{acc}, \prod_S, c_{\vec{u}}; \vec{u}, o_{\vec{u}}\right)$:

Let $\vec{w} = \{\}$

For $i = 1$ to m:

$w_{0,i} \leftarrow g^{u[i-1]}$

For $i = 1$ to $\log m$:

For $j = 1$ to $m \cdot 2^{-i}$:

$w_{i,j} =$

Assemble$(pp, u[2j-2], u[2j-1], w_{i-1,2j-1}, w_{i-1,2j})$

$\vec{w}$.append$(w_{i,j})$

$W_{\vec{u}} \leftarrow w_{\log m,1}$

$\hat{\text{acc}} \leftarrow \text{acc}^{\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i}$

Let $u^* = \prod_i u_i$, $p^* = \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i$

Sample $b_1, \ldots, b_{2\lambda} \leftarrow\$ \{0,1\}$

Let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i}$, $\bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$ $\hat{W}_{\vec{u}} \leftarrow W_{\vec{u}}^{\bar{s}}$

Sample $r \leftarrow\$ \{0,1\}^{\|p^*\|+\|u^*\|+2\lambda}$

$c_{s,r} \leftarrow \text{Comm}_{\text{ck}}(s, r; o_{s,r})$

$R \leftarrow \hat{W}_{\vec{u}}^{r}$ $h \leftarrow H(\text{crs}||\text{acc}||c_{\vec{u}}||c_{s,r}||\hat{W}_{\vec{u}}||R)$ $k \leftarrow r + (u^* s)h$

$\pi_1 \leftarrow \Pi^{\text{PoKE}}.\text{Prv}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\text{acc}}^h R; k)$ Parse π_1 as $(Q, \hat{k})$

$\ell \leftarrow H_{\text{prime}}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\text{acc}}^h R)$

$\pi_2 \leftarrow \text{cp}\Pi^{\text{modarithm}}.\text{Prv}(\text{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, \hat{k}; \vec{u}, o_{\vec{u}}, r, s, o_{s,r})$

return $\pi = (\hat{W}_{\vec{u}}, R, c_{s,r}, \pi_1, \pi_2)$

Figure 6: Optimized construction for witness generation. The colored with blue is modified proving function of HARiSA consistent with precomputation.

is a secure CP-SNARKs: knowledge-sound under the adaptive root assumption and succinct.

Proof. As described in Theorem 1, we examine meticulously whether the differences added to HARiSA affects the security of HARiSA. Likewise Figure 3, the difference is that updated elements are transformed into prime numbers, i.e., as $\hat{u}_i \leftarrow u_i||z_{u_i}$ such that z_{u_i} is prime number. Therefore, the adaptive root assumption holds for the construction in Figure 8 where updated elements are transformed. Also, we observe that the succinctness is satisfied. The proof from $\text{cp}\Pi^{\text{WT}}$, $\text{cp}\Pi^{\text{modarithm}}$ and PoKE are constant-sized, and the verification runs in $O(1)$ in the same manner as the Figure 3. □

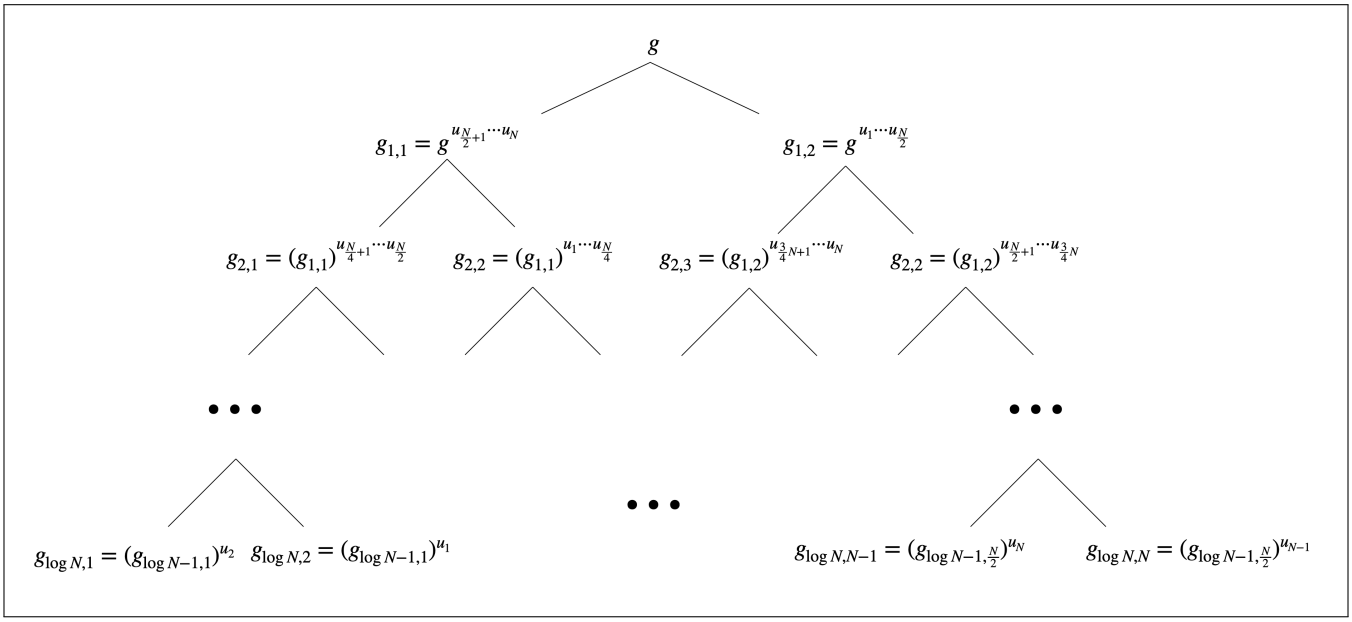


Figure 7: The Precompute **algorithm in tree view**

$\underline{\mathsf{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}\right):}$

$\mathsf{crs}_2 \leftarrow \mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\mathsf{modarithm}})$

$\mathsf{crs}_3 \leftarrow \mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\mathsf{WT}})$

return $(\mathsf{ck}, \mathsf{pp}, \mathsf{crs}_2, \mathsf{crs}_3)$

$\underline{\mathsf{Prove}\left(\mathsf{crs}, \mathsf{acc}, \mathsf{acc}', c_{\vec{u}}; \vec{\tilde{u}}, o_{\vec{u}}, \vec{u}, \vec{z}\right):}$

Let $u^* = \prod_{u_i \in \vec{u}} u_i$

$\pi_1 \leftarrow \Pi^{\mathsf{PoKE}}.\mathsf{Prv}\big((\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}'; u^*\big)$

Parse π_1 as $(Q, \hat{k})$

$\ell \leftarrow H_{\mathrm{prime}}((\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}')$

$\pi_2 \leftarrow \mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\vec{\tilde{u}}}, \ell, \hat{k}; \vec{\tilde{u}}, o_{\vec{\tilde{u}}})$

$\pi_3 \leftarrow \mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\vec{\tilde{u}}}, c_{\vec{u}}, c_{\vec{z}}; \vec{\tilde{u}}, o_{\vec{\tilde{u}}}, \vec{u}, o_{\vec{u}}, \vec{z}, o_{\vec{z}})$

return $\pi = (\pi_1, \pi_2, \pi_3)$

$\underline{\mathsf{Verify}\left(\mathsf{crs}, \mathsf{acc}, \mathsf{acc}', c_{\vec{\tilde{u}}}, c_{\vec{u}}, c_{\vec{z}}, \pi\right):}$

Parse π as (π_1, π_2, π_3) and π_1 as $(Q, \hat{k})$

Reject if $\Pi^{\mathsf{PoKE}}.\mathsf{Vfy}((\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}', \pi_1) \neq 1$

Reject if $\mathsf{cp}\Pi^{\mathsf{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{\tilde{u}}}, \ell, \hat{k}, \pi_2) \neq 1$

Reject if $\mathsf{cp}\Pi^{\mathsf{WT}}.\mathsf{Vfy}(c_{\vec{\tilde{u}}}, c_{\vec{u}}, c_{\vec{z}}, \pi_3) \neq 1$

Figure 8: Construction for dynamic table in DUPLEX