Succinct Zero-Knowledge Batch Proofs for Set Accumulators

Matteo Campanelli
Aarhus University, Denmark
matteo@cs.au.dk

Dario Fiore
IMDEA Software Institute, Spain
dario.fiore@imdea.org

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

Jihye Kim
Kookmin University, Korea
jihyek@kookmin.ac.kr

Dimitris Kolonelos
IMDEA Software Institute, Spain
Universidad Politecnica de Madrid, Spain
dimitris.kolonels@imdea.org

Hyunok Oh
Hanyang University, Korea
hoh@hanyang.ac.kr

***Abstract*—Cryptographic accumulators are a common solution to proving information about a large set S. They allow to compute a short digest of S and short certificates of some of its basic properties, notably membership of an element. Accumulators also allow to track set updates: a new accumulator is obtained by inserting/deleting a given element. In this work we consider the problem of generating membership and update proofs for *batches* of elements so that we can succinctly prove additional properties of the elements (i.e., proofs are of constant size regardless of the batch size), and we can preserve privacy. Solving this problem would allow to obtain blockchain systems with improved privacy and scalability.**

The state-of-the-art approach to achieve this goal is to combine accumulators (typically Merkle trees) with zkSNARKs. This solution is however expensive for provers and does not scale for large batches of elements. In particular, there is no scalable solution for proving batch membership proofs when we require zero-knowledge (a standard definition of privacy-preserving protocols).

In this work we propose new techniques to efficiently use zkSNARKs with RSA accumulators. We design and implement two main schemes: 1) HARiSA, which proves batch membership in zero-knowledge; 2) B-INS-ARiSA, which proves batch updates. For batch membership, the prover in HARiSA is orders of magnitude faster than existing approaches based on Merkle trees (depending on the hash function). For batch updates we get similar cost savings compared to approaches based on Merkle tree; we also improve over the recent solution of Ozdemir et al. [USENIX'20].

I. Introduction

Blockchains are decentralized and distributed systems in which a vast network of nodes maintain, distributed and replicated, a digital ledger. Core to blockchains is the maintenance of the global state of the system across its nodes. This state is usually large and is encoded in data structures such as an UTXO set (unspent transaction outputs, intuitively the unspent coins) in Bitcoin and Zcash [1], [2], the set of account-balances in Ethereum, or the set of identities in Decentralized Identity (DID) systems (e.g., Iden3, Sovrin, Hyperledger Indy) [3]–[5]. In these systems executing a transaction typically involves two steps, one "local" and one "global": *(i)* checking a given property with respect to the current state (e.g., the transaction is properly signed, some coins are spendable, some credentials exist), and *(ii)* modifying the global state (e.g., updating balances, adding a credential) and checking its correct update. The validity checks that are local to the transaction can for example involve checking a digital signature. Checking against the global state typically translate into *set-membership* ($x \in S$) or *set-update* ($S' \overset{?}{=} S \setminus \{x\} \cup \{x'\}$).

Blockchain systems grow through time and so do their global states (at the time of writing the UTXO set in Bitcoin is 4.6 GB). Verifying this state *at scale* is a challenging problem: every user, even one that only "passively" looks at the history of transactions, must re-execute them and store them to verify future ones.

A common approach to address this problem is the use of authenticated data structures (ADS) [6]—Merkle trees [7] as their most popular and deployed incarnation, or RSA accumulators [8]–[11]. This idea [12]–[14] splits users into two groups. More "passive" users (aka verifiers) store only a *succinct digest* of the large set. A user proposing a transaction, on the other hand, has more information on the state (e.g., their account information) that it can use to *prove* either the membership of some elements, or the correctness of an update, with respect to the digest. This approach achieves scalable verification because ADS proofs are *succinct*, i.e., they are short and the time to verify is sublinear in the size of the set, e.g., it is logarithmic in Merkle trees, or constant in RSA accumulators.

While the efficiency benefits of this approach are clear, there are two additional challenges emerging in this space. They are the focus of our work.

1) *how to obtain privacy?* This is paramount whenever transactions data cannot be publicly exposed (e.g., to preserve anonymity or prevent front-running).

2) *how to improve throughput?* That is, the number of transactions we can process per unit of time.

The ADS-based approach above falls short on both issues. First, it generally requires that the global state is public. Second, it scales poorly when *proving many transactions*. Assume we want to batch prove m transactions at once, ADS either allow membership/update proofs for a single element only [7], or they have succinct batch proofs but the verifier must know and receive the elements [15]. This entails at least an $O(m)$ communication and verification time, affecting on-chain storage and work.

Both these problems can be solved via the use of *zkSNARKs* [16], [17], cryptographic proof systems that enable a prover to convince a verifier about the veracity of statements of the form "given a function F and a public input x, there is a secret w such that $F(x, w)$ is true". In particular, zkSNARKs proofs are *zero-knowledge* and *succinct*. The former means that proofs do not reveal any information about the secret w and give solutions to the privacy challenge. For instance, in Zcash one proves the existence of a coin that is valid and part of the UTXO set, without revealing which is the coin so as to guarantee anonymity of a spend transaction. The other property, succinctness, means that proofs are short and efficient to verify, regardless of how much time it takes to execute F. This gives a solution to the throughput question above. The idea (known as zk-Rollup [18]) is that an aggregator can: collect a batch of m transactions; prove that they are valid; compute the updated global set and the corresponding digest; and finally broadcast the new digest with a succinct proof that its update is correct.

If zkSNARKs make verifiers' life easier, the same cannot be said for provers. In these applications, the function to be proven includes the verification algorithm of an ADS which makes zkSNARK proving extremely expensive in terms of both computing time and RAM. As of today the most deployed option is based on Merkle trees [7]. Proving their verification for a set of size n and a batch of m elements requires to encode about $\approx m \log n$ hash computations in the zkSNARK constraint system. Even by using hash functions that are optimized for zkSNARKs [19], [20], the proving time degrades very quickly (see section VI). In part, a reason of this cost is that Merkle trees do not allow batch openings. RSA accumulators are a promising alternative as they enjoy constant-size batch proofs for membership and updates [15]. Yet, naively encoding their verification—$O(m)$ RSA group operations—in zkSNARK constraints is concretely prohibitive too ($\approx 1.8 \cdot m$ millions constraints).

To summarize, to the best of our knowledge in the state-of-the-art there is no scalable solution for succinct zero-knowledge proofs of batch membership.

The closest works in this area are two recent papers [21], [22] that efficiently combine RSA accumulators with zkSNARKs avoiding expensive encodings. Benarroch et al. [22] propose SNARKs for set-membership (proving that a committed element is in a large set, of which the verifier knows an RSA accumulator) but only support membership of a *single* element. Ozdemir et al. [21] propose a verifiable computation for batch *updates* of sets succinctly represented by RSA accumulators. Since in RSA accumulators, membership and updates are expressed through the same algebraic property, their techniques could be extended to support zero-knowledge *membership*. But this would entail a very high cost related to encoding RSA group operations as constraints. Finally, their improvements with respect to proving Merkle tree openings hold beyond a fairly large threshold: 2^{20}-elements sets and batches of $\approx 1,300$ values.

A. Our work

We advance this research line by proposing new techniques to efficiently use zkSNARKs with RSA accumulators.

Succinct proofs of batch membership. Our first result is a commit-and-prove [23] zkSNARK for batch membership, that is: given an RSA accumulator acc to a set $S = \{x_1, \ldots, x_n\}$ and a succinct Pedersen commitment c_u to a vector of values $(u_1, \ldots, u_m)$, it holds $u_i \in S$ for every $i = 1, \ldots, m$. Thanks to the commit-and-prove feature, our scheme can be efficiently and modularly composed with other commit-and-prove[1] zkSNARKs [24] in order to prove further properties of the committed elements, e.g., $\forall i : u_i \in S \wedge P(u_1, \ldots, u_m) = \mathsf{true}$ (P could be for example a numerical range check; see also our DID application in section VI-B2). We dub our construction HARiSA[2].

Our technical contributions include: a new randomization method for RSA accumulators witnesses (needed to obtain zero-knowledge) and a new way to prove the accumulator verification in zero-knowledge in a SNARK *without* encoding RSA group operations in the constraint system. The latter is based on a novel combination of (non-succinct) sigma protocols, succinct proof of knowledge of exponent [15], and zkSNARKs for integer arithmetic.

Succinct proofs of batch insertion. Our second result concerns succinct proofs for batch insertion, that is: given two RSA accumulators $\mathsf{acc}, \mathsf{acc}'$ to sets S and S' respectively and a succinct Pedersen commitment c_u to $(u_1, \ldots, u_m)$, it holds that $S' = S \cup \{u_1, \ldots, u_m\}$.[3] We build our scheme for set insertions—dubbed B-INS-ARiSA[4]—by "scaling down" our techniques for set-membership, removing zero-knowledge and simplifying, finally obtaining a solution that is simpler and faster than our batch-membership scheme. Furthermore, inspired by [21], we show how to use this scheme to obtain one for proving MultiSwaps, which in a nutshell means checking if S' is obtained by applying a sequence of "swaps" $\{(x_1, x'_1), \ldots, (x_m, x'_m)\}$ (i.e., add x'_i, remove x_i) to S—essentially what we informally referred as set update. As shown in [21], proving MultiSwaps for accumulated sets has applications to verifiable outsourcing of state updates, applicable to Rollups [18] and efficient persistent RAM [25].

Implementation and evaluation. We implement our protocols and evaluate them experimentally, comparing with the state of the art. For zero-knowledge batch membership, we compare

[1]Roughly, the verification algorithm of the zkSNARK takes as input short commitments to a long (potentially private) input. This property is useful as the elements for which we prove set membership need to stay private, but still "referred to", e.g. for proving additional properties on them.

[2]HARiSA stands for "elements-Hiding Argument for RSA accumulators".

[3]More precisely, our schemes work with multisets.

[4]For Batch INSertion. It is pronounced as in the word *beans*.

our solution with Merkle trees on two benchmarks: one that considers generic membership operations, and one that implements a DID application. For batch updates, we compare our MultiSwap solution with that of Ozdemir et al. [21] and with Merkle trees. We do the latter comparison via a cost model analysis based on number of SNARK constraints. We also validate the case of set membership on a realistic application scenario (Decentralized Identity in section VI-B2.

For batch membership our construction saves at least *an order of magnitude in proving time* (depending on which hash function we use for Merkle tree in the comparison). As an example, proving a batch membership with SHA256 Merkle trees requires more than an hour, while it requires less than 20 seconds with our scheme. Its public parameters are 5x smaller than solutions based on Merkle trees. Our verification times and proof size are also competitive (close to one kilobyte for any batch/set size).

For MultiSwaps our solution surpasses Merkle-based swaps earlier than [21] (140 operations for 2^{20}-large sets).

II. Technical Overview

We now present a high-level overview of our main technical contributions.

Our core protocol is a succinct zero-knowledge proof of set membership for a batch of set elements. Given a (public) set $S = \{x_1, \dots, x_n\}$ and commitments to $u_1, \dots, u_m$, we aim at proving that $u_1, \dots, u_m \in S$. We require for privacy that the u_i-s remain hidden (and thus we provide them only as commitments in the public input). We also require for efficiency that proof size and verification time should not depend either on the batch size m or the set size n.

We start from applying RSA accumulators [9] to compress the set into a succinct digest. Given random group element g in a group of unknown order (e.g. an RSA or class group [26]), one can produce a compressed representation of the set as $\mathsf{acc} = g^{x_1 \cdot x_2 \cdots x_n}$. RSA accumulators enjoy succinct batch-membership proofs: to prove that $u_1, \dots, u_m \in S$ it suffices to provide a single group element (a *witness*) $W = g^{\prod_{i \in [n]} x_i / \prod_{i \in [m]} u_j}$, which the verifier can check as $W^{u_1 \cdots u_m} = \mathsf{acc}$.

Though succinct, the batch-membership proofs of RSA accumulators do not hide the u_i elements, as the verifier should know them in order to perform the exponentiation. To address this problem, we could use a non-interactive zero-knowledge proof of exponentiation, which can be obtained using a Σ-protocol [27] (a three-message zero-knowledge scheme) made non-interactive through the Fiat-Shamir transform [28]. In it the prover computes: $R \leftarrow W^r$ for a sufficiently large random r; a random oracle challenge $h \leftarrow H(\mathsf{acc}||g||W||R)$, an integer $k \leftarrow r + h \cdot \prod_{i \in [m]} u_i$. The verifier accepts this zk-proof (R, h, k) if $h = H(\mathsf{acc}||g||W||R)$ and $R \cdot \mathsf{acc}^h = W^k$.

This protocol however does not yet achieve our goal, which is to generate a zero-knowledge batch-membership proof for committed u_i's. Towards this goal, we need to solve the following technical challenges. (A) The verifier needs to know the witness W, which can itself leak information about the elements it proves membership of. For example for $m = 1$ one can efficiently find the element u_1 by brute-force testing all elements of the set S, $W^{x_i} \stackrel{?}{=} \mathsf{acc}$ (recall that the set is public). The x_{i^*} for which the test passes will be u_1. (B) The proof (R, h, k) above simply shows existence of an exponent u such that $W^u = \mathsf{acc}$, in particular it does not link this statement to committed $(u_1, \dots, u_m)$ such that $u = u_1 \cdots u_m$. (C) the proof is not succinct since the integer k is $O(m)$-bits long.

Our key contribution is an efficient technique to efficiently prove the verification of this Σ-protocol using a SNARK. Notably, we do not need encode any RSA group operations in the SNARK constraint system[5]. To obtain this result we combine three main ideas:

1) We introduce a novel randomization method for an RSA accumulator witness, $W \mapsto \hat{W}$, so that $\hat{W}$ provably doesn't leak any information about the u_i's.
   Our hiding-witness transformation works as follows: let $p_1, \dots, p_{2\lambda}$ be the first 2λ prime numbers. These primes are always (artificially) added to the accumulator, i.e., the accumulator of a set S is an RSA accumulator $\hat{\mathsf{acc}}$ of $\hat{S} \leftarrow S \cup \{p_1, \dots, p_{2\lambda}\}$: $\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{p_1 \cdots p_{2\lambda}}$ (we assume that S does not contain any of the p_i's). Then to produce the hiding witness $\hat{W}$ we raise to the exponent each prime p_i with probability $1/2$. A bit more formally, we sample $b_i \leftarrow\$ \{0, 1\}$ and set $\hat{W} \leftarrow W^{\prod_{i \in [2\lambda]} p_i^{1 - b_i}}$, b_i's should remain hidden. We formally prove that under a cryptographic assumption (DDH-II, a variant of DDH [29]) $\hat{W}$ is computationally indistinguishable from random and thus $\hat{W}$, alone, hides u_i's (see section IV-A). Notice that $\hat{W}$ can be verified through the equality $\hat{W}^{\prod_{i \in [m]} u_i \cdot \prod_{i \in [2\lambda]} p_i^{b_i}} = \hat{\mathsf{acc}}$. Therefore we can use the NIZK for exponentiation described above, but for base $\hat{W}$ and exponent $e := \prod_{i \in [m]} u_i \cdot \prod_{i \in [2\lambda]} p_i^{b_i}$.
   This technique solves the challenge (A) as it turns an RSA accumulator verification into a ZK verification. Yet challenges (B) and (C) remain: k is not short and the protocol only proves the existence of e such that $\hat{W}^e = \hat{\mathsf{acc}}$—which says nothing about membership of *legitimate* elements from S. For example e can contain only elements of $\{p_1, \dots, p_{2\lambda}\}$ and no element from S.
2) To solve (B) we "link" the Σ-protocol to $c_{\vec{u}}$, a commitment to all u_i's, by using a zkSNARK that proves the correct computation of k from the committed legitimate u_i's. Namely it proves that, for $c_{\vec{u}}$ commitment to $\vec{u}$ and $c_{r,s}$ commitment to integers $s = \prod_{i \in [2\lambda]} p_i^{b_i}$ and r, $k = r + h \cdot s \cdot \prod_{i \in [m]} u_i$ holds over the integers and $u_i > p_{2\lambda}$ for each $i \in [m]$. Recall that $p_{2\lambda}$ is the largest of all p_i's, so $u_i > p_{2\lambda}$ translates to $u_i \neq p_j$ for all $j \in [2\lambda]$. This means that the exponent of e contains elements u_i's committed a-priori and that they are *legitimate* (not one of the artificially added p_i's).

[5]A "constraint system" is an encoding of the property proved by the SNARK. Its size, the number of constraints, is a key efficiency metric when evaluating proof schemes.

3) Although the above careful interplay between RSA Accumulators, Σ-protocols, zkSNARKs and our hiding technique for RSA accumulators witnesses gives a secure zero-knowledge proof of set membership, it is not yet succinct, as the verifier needs to receive the $O(m)$-long integer k. To solve this technical challenge, we apply a succinct proof of knowledge of exponent PoKE [15]. Instead of sending k, the prover sends $B = \hat{W}^k$ accompanied with a succinct proof that there is an integer k such that $B = \hat{W}^k$. Adding the PoKE proof however breaks the link between the Σ-protocol and the zkSNARK as the latter is supposed to generate a proof for a public k. To solve this last challenge, we "open the box" of PoKE verification and observe that the verifier receives the short integer $\hat{k} = k \mod \ell$, where ℓ is a random prime challenge of 2λ bits. Therefore, the last idea of our protocol is to let the zkSNARK prove the same statement as above but for $\hat{k}$, namely that $\hat{k} = r + h \cdot s \cdot \prod_{i\in[m]} u_i \mod \ell$.

III. Background

A. Commitments

Commitment schemes allow one to commit to a value, or a collection of values (e.g., a vector), in a way that is *binding*—a commitment cannot be opened to two different values—and *hiding*—commitment leaks nothing about the value it opens to. In our work we also consider commitment schemes that are *succinct*, meaning informally that the commitment size is fixed and shorter than the committed value. Here is a brief description of the syntax we use in our work: $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ returns a commitment key ck; $\mathsf{Comm}(\mathsf{ck}, x; o) \to c$ produces a commitment c on input a value x and randomness o (which is also the opening).

B. (Commit-and-Prove) SNARKs

Definition 1 (SNARK). *A SNARK for a relation family $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:*

- $\Pi.\mathsf{Setup}(1^\lambda, R) \to \mathsf{crs}$ *outputs a relation-specific common reference string* crs.
- $\Pi.\mathsf{Prove}(\mathsf{crs}, \mathbb{x}, \mathbb{w}) \to \pi$ *on input* crs*, a statement $\mathbb{x}$ and a witness $\mathbb{w}$ such that $R(\mathbb{x}, \mathbb{w})$, it returns a proof π.*
- $\Pi.\mathsf{Verify}(\mathsf{crs}, \mathbb{x}, \pi) \to b \in \{0,1\}$ *on input* crs*, a statement $\mathbb{x}$ and a proof π, it accepts or rejects the proof.*

We require a SNARK to be complete, knowledge-sound and succinct. Completeness means that for any $\lambda \in \mathbb{N}, R \in \mathcal{R}_\lambda$ and $(\mathbb{x}, \mathbb{w}) \in R$ it holds with overwhelming probability that $\mathsf{Verify}(\mathsf{crs}, \mathbb{x}, \pi) = 1$ *where* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, R)$ *and proof* $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, R, \mathbb{x}, \mathbb{w})$*. Knowledge soundness informally states we can efficiently "extract" a valid witness from a proof that passes verification. Succinctness means that proofs are of size* $\mathsf{poly}(\lambda)$ *(or sometimes* $\mathsf{poly}(\lambda, \log|\mathbb{w}|)$*) and can be verified in time* $\mathsf{poly}(\lambda)\mathsf{poly}(|\mathbb{x}| + \log|\mathbb{w}|)$*. A SNARK may also satisfy zero-knowledge, that is the proof leaks nothing about the witness (this is modeled through a simulator that can output a valid proof for an input in the language without knowing the witness). In this case we call it a zkSNARK. Whenever the relation family is obviously defined, we talk about a "SNARK for a relation R".*

1) Commit-and-Prove SNARKs (CP-SNARKs): We use the framework for black-box modular composition of commit-and-prove SNARKs (or CP-SNARKs) in [24] and [22]. Informally a CP-SNARK is a SNARK that can efficiently prove properties of inputs committed through some commitment scheme C. More in detail, a CP-SNARK for a relation $R_{\mathsf{inner}}(\mathbb{x}; u, \omega)$ is a SNARK that for a given commitment c can prove knowledge of $\mathbb{w} := (u, \omega, o)$ such that $c = \mathsf{Comm}(u; o)$ and $R_{\mathsf{inner}}(\mathbb{x}; u, \omega)$ holds. We can think of ω as a non-committed part of the witness. In a CP-SNARK, besides the proof, the verifier's inputs are $\mathbb{x}$ and c.

Remark 1 (Syntactic Sugar for SNARKs/CP-SNARKs). *For convenience we will use the following notational shortcuts. We make explicit what the private input of the prover is by adding semicolon in a relation and in a prover's algorithm (e.g., $R(\mathbb{x}; \mathbb{w})$). We explicitly mark relations as "commit-and-prove" by a tilde. We let the assumed commitment scheme implicit when it's obvious from the context. Occasionally, we will also explicitly mark the commitment inputs by squared box around them (e.g. $\boxed{c_u}$) and we will assume implicitly that the relation includes checking the opening of these commitments (and we will not make explicit the openings). We assume that in the commitment $\boxed{c_u}$ the subscript u defines the variable u the commitment opens to. Analogously the opening for c_u is automatically defined as o_u. Example: $\tilde{R}_{\mathsf{ck}}(\boxed{c_u}, h; r) = 1 \Leftrightarrow h = \mathsf{SHA256}(u\|r)$ is a shortcut for $\tilde{R}_{\mathsf{ck}}(c_u, h; r, u, o_u) = 1 \Leftrightarrow h = \mathsf{SHA256}(u\|r) \wedge c_u = \mathsf{Comm}(\mathsf{ck}, u; o_u)$.*

2) Modular SNARKs through CP-SNARKs: We make use of the following folklore composition of (zero-knowledge) CP-SNARKs (cf. [24, Theorem 3.1]). Fixed a commitment scheme and given two CP-SNARKs $\mathsf{cp}\Pi_1, \mathsf{cp}\Pi_2$ respectively for two "inner" relations $\tilde{R}_1$ and $\tilde{R}_2$, we can build a (CP) SNARK for their conjunction (for a shared witness u) $\tilde{R}^*(\boxed{c_u}, \mathbb{x}_1, \mathbb{x}_2; \omega_1, \omega_2) = R_1(\boxed{c_u}, \mathbb{x}_1; \omega_1) \wedge R_2(\boxed{c_u}, \mathbb{x}_2; \omega_2)$ like this: the prover commits to u as $c_u \leftarrow \mathsf{Comm}(u, o)$; generates proofs π_1 and π_2 from the respective schemes; outputs combined proof $\pi^* := (c_u, \pi_1, \pi_2)$. The verifier checks each proof over respective inputs $(\mathbb{x}_1, c_u)$ and $(\mathbb{x}_2, c_u)$, with shared commitment c_u.

C. Accumulators to Multisets

A multiset is an unordered collection of values in which the same value may appear more than once. We denote by $S_1 \uplus S_2$ the union of multisets S_1 and S_2, i.e., the multiset S_3 where the multiplicity of any $x \in S_3$ is the sum of its multiplicity in S_1 and S_2. For two multisets $S_2 \subset S_1$, $S_1 \boxminus S_2$ denotes the multiset difference of S_1 and S_2, i.e., the multiset S_3 where the multiplicity of any $x \in S_3$ is the multiplicity of x in S_1 minus the multiplicity of x in S_2.

Cryptographic accumulators [8] are succinct commitments to sets that also allow one to generate succinct proofs of membership (and sometimes also non-membership). In our work we use accumulators that enjoy three additional properties. First, they support multisets. Second, they are *dynamic*, meaning that from an accumulator to S_1 one can publicly compute an accumulator to $S_1 \uplus S_2$ without knowing S_1. Third, one can create succinct membership proofs for batches of elements, i.e., to prove that $X \subset S$. More in detail, such an accumulator is a tuple of algorithms $\mathsf{Acc} = (\mathsf{Setup}, \mathsf{Accum}, \mathsf{PrvMem}, \mathsf{VfyMem}, \mathsf{Ins})$ such that:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ generates public parameters;
- $\mathsf{Accum}(\mathsf{pp}, S) \to \mathsf{acc}$ generates the accumulator acc for a multiset S;
- $\mathsf{PrvMem}(\mathsf{pp}, S, X) \to W_X$ generates a membership proof for $X \subset S$;
- $\mathsf{VfyMem}(\mathsf{pp}, \mathsf{acc}, X, W_X) \to 0/1$ accepts or rejects a membership proof W_X;
- $\mathsf{Ins}(\mathsf{pp}, \mathsf{acc}, S') \to \mathsf{acc}'$ computes accumulator to $S \uplus S'$.

A multiset accumulator is secure if any PPT adversary has negligible probability of creating a valid membership proof for a multiset $X \not\subset S$, namely to output a tuple (S, X, W) such that there is an $x \in X$ such that $x \notin S$ and $\mathsf{VfyMem}(\mathsf{pp}, \mathsf{Accum}(\mathsf{pp}, S), X, W_X) = 1$.

We note that the popular RSA accumulator [9]–[11], [15] enjoys all the properties mentioned above.

D. Relations for batch set-membership and set-insertion

Our focus in this work is on building efficient CP-SNARKs for the following two relations parametrized by an accumulator scheme Acc and parameters $\mathsf{pp}_{\mathsf{Acc}}$:

$$\tilde{R}^{\text{mem}}_{\mathsf{ck}}(\boxed{c_U}, \mathsf{acc}; W) = 1 \Leftrightarrow \mathsf{Acc.VfyMem}(\mathsf{pp}, \mathsf{acc}, U, W) = 1$$

$$\tilde{R}^{\text{ins}}_{\mathsf{ck}}(\boxed{c_U}, \mathsf{acc}, \mathsf{acc}') = 1 \Leftrightarrow \mathsf{Acc.Ins}(\mathsf{pp}_{\mathsf{Acc}}, \mathsf{acc}, U) = \mathsf{acc}'$$

In a nutshell, a CP-SNARK for $\tilde{R}^{\text{mem}}_{\mathsf{ck}}$ can prove that c_U is a commitment to a vector of values such that each of them is in the multiset accumulated in acc. A CP-SNARK for $\tilde{R}^{\text{ins}}_{\mathsf{ck}}$ can instead prove that acc' is a correct update of the accumulator acc obtained by inserting the elements committed in c_U. For the relation $\tilde{R}^{\text{ins}}_{\mathsf{ck}}$ we are not interested in obtaining proofs that are zero-knowledge (i.e., so as to hide U), as the Ins algorithm is deterministic and thus simply having public accumulators $\mathsf{acc}, \mathsf{acc}'$ may leak information on the added elements.

The specific notion of knowledge soundness we assume for CP-SNARKs for these relations is the one where the malicious prover is allowed to select an arbitrary set S to be accumulated but the accumulator acc is computed honestly from S. Given an accumulator scheme Acc, we informally talk about this specific notion as "security under the Trusted Accumulator-Model for Acc". We do not provide formal details since this model corresponds to the notion of partial-extractable soundness in Section 5.2 in [22][6]; we refer the reader to this work for further details.

This trusted accumulator model fits several applications where the accumulator is maintained by the network.

[6]We notice that their model uses a slightly different language and formalizes accumulators as (binding-only) commitments for commit-and-prove NIZKs.

On the other hand, we stress that in the R^{ins} relation, the trusted accumulator assumption is assumed only for acc but *not* for acc'. The interesting implication of this is that one can view a CP-SNARK for R^{ins} as a means to move from a trusted accumulator acc to a trustworthy one acc'. Indeed, thinking of acc_0 as the accumulator to the empty set that everyone knows and can efficiently compute, R^{ins} then allows one to certify the generation of an accumulator to any multiset.

In the following two sections we show two interesting byproducts of having modular commit-and-prove SNARKs for the relations $\tilde{R}^{\text{mem}}_{\mathsf{ck}}$ and $\tilde{R}^{\text{ins}}_{\mathsf{ck}}$.

1) Composing (commit-and-prove) set-membership relations: The advantage of having CP-SNARKs for the set-membership relation (rather than just SNARKs) is that one can use the composition of section III-B2 to obtain *efficient* zkSNARKs for proving properties of elements in an accumulated set, e.g., to show that $\exists U = \{u_1, \ldots, u_n\}$ such that a property P holds for U (say, every u_i is properly signed) and $U \subset S$, where S is accumulated in some acc. In particular, such a zkSNARK can be obtained via the simple and efficient composition of a CP-SNARK for $\tilde{R}^{\text{mem}}_{\mathsf{ck}}$ (like the ones we construct in our work) and any other CP-SNARK for P.

2) From set-insertion to $\mathsf{MultiSwap}$*:* Ozdemir et al. [21] introduce an operation over (RSA) accumulators called $\mathsf{MultiSwap}$. Consider two multisets S and S' and a sequence of pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where each pair represents *in order* a "swap", namely removal of x_i and insertion of y_i. Verifying a $\mathsf{MultiSwap}$ means checking that $S' = S_n$ where $S_0 = S$ and $S_i = S_{i-1} \boxminus x_i \uplus y_i$. [21] shows that this check can be reduced to

$$\exists S_{mid} : S_{mid} = S \uplus \{y_i\}_i \wedge S_{mid} = S' \uplus \{x_i\}_i$$

So, when using accumulators, we can represent $\mathsf{MultiSwap}$ via the following relation:

$$R^{\text{mswap}}(\mathsf{acc}, \mathsf{acc}'; X, Y) = 1 \Leftrightarrow$$

$$\exists \mathsf{acc}_{\mathsf{mid}} : R^{\text{ins}}(\mathsf{acc}, \mathsf{acc}_{\mathsf{mid}}; Y) \wedge R^{\text{ins}}(\mathsf{acc}_{\mathsf{mid}}, \mathsf{acc}'; X)$$

Thus, a CP-SNARK for R^{mswap} can be obtained via the (self)composition of a CP-SNARK for R^{ins}.

3) Chaining $\mathsf{MultiSwap}$*:* Consider a scenario where an accumulator evolves in time, namely at time i a user returns a new accumulator acc_i along with a proof π_i that $(\mathsf{acc}_{i-1}, \mathsf{acc}_i) \in R^{\text{mswap}}$ (and possibly additional proof that the elements added/removed satisfy a certain property, e.g., in Rollup they are valid transactions). It is easy to see that the concatenation $(\pi_1, \mathsf{acc}_1, \ldots, \mathsf{acc}_{n-1}, \pi_n)$ constitutes a proof for $(\mathsf{acc}_0, \mathsf{acc}_n) \in R^{\text{mswap}}$.

E. Building blocks

1) Pedersen Commitments of Integer values: The CP-SNARKs we construct are defined for commitments generated using the classical extension of Pedersen commitments to vectors. In particular, we sometimes use a variant of this scheme for committing to integers (instead of field elements);

we describe it in fig. 1b. We assume a prime p and an algorithm $\mathcal{G}_p$ that generates appropriate parameters for groups of order p. Since we commit to an integer x whose size is potentially larger than p we split the integer into several "chunks", of size $\mathsf{ChkSz} \leq p$ specified in the parameters, and then we apply the standard vector-Pedersen on this split representation. We let the setup algorithm take as input a bound B denoting the max integer that we can commit to. The construction is perfectly hiding, and computationally binding secure under the discrete logarithm assumption.

2) RSA Accumulators: Another crucial component of our CP-SNARKs are RSA accumulators to multisets [9], [15], that we recall in fig. 1a. In particular, we assume their instantiation over any group of unknown order (including, e.g., classical RSA groups or class groups [26]) whose parameters are generated by an algorithm $\mathcal{G}_?$ and over which the Strong RSA [9] and the Adaptive Root [30] assumptions hold.

IV. HARISA: ZERO-KNOWLEDGE CP-SNARK FOR BATCH SET-MEMBERSHIP

In this section we show the construction of a CP-SNARK for the relation $\tilde{R}^{\mathsf{mem}}_{\mathsf{ck}}$ defined in *section* III-D1, where: the accumulator is the classical RSA accumulator from Figure 1a where the accumulated elements are prime numbers larger than the 2λ-th prime (1619 for $\lambda = 128$), and the commitment scheme for the commit-and-prove functionality is the Pedersen scheme of Fig. 1b. In appendix C we discuss how this construction can be easily extended to accumulate arbitrary elements via an efficient hash-to-prime function.

A. RSA Accumulators with hiding witnesses

We describe a method to turn a witness W of an RSA accumulator into another witness that computationally hides all the elements u_i it proves membership of. As discussed in Section II this constitutes the first building block towards achieving a zero-knowledge membership proof for committed elements.

Let $\mathbb{P}_n = \{2, 3, 5, 7, \ldots, p_n\}$ be the set of the first n prime numbers. Our method relies on two main ideas.

First, prover and verifier modify the accumulator acc so as to contain the first 2λ primes by computing $\hat{\mathsf{acc}} \leftarrow \mathsf{acc}^{\left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i\right)}$. Note, $\hat{\mathsf{acc}} = g_?^{\left(\prod_{x_i \in S} x_i\right) \cdot \left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i\right)} = \mathsf{Accum}(\mathsf{pp}, S \cup \mathbb{P}_{2\lambda})$.

Second, we build a randomized witness for $X \subset S$ as the witness for $(X \cup P) \subset (S \cup \mathbb{P}_{2\lambda})$ where P is a randomly chosen subset of $\mathbb{P}_{2\lambda}$. More in detail, given W, the prover compute $\hat{W}$ as follows:

- choose at random 2λ bits $b_1, \ldots, b_{2\lambda} \leftarrow\$ \{0, 1\}$ and let $s := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{b_i}$ and $\bar{s} := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}$.
- $\hat{W} \leftarrow W^{\bar{s}} = g_?^{\left(\prod_{x_i \in S \setminus X} x_i\right) \cdot \left(\prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-b_i}\right)}$.

Essentially, we have s as the product of the randomly chosen primes, $\bar{s}$ as the product of the primes not chosen, and we denote with $p^* := \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i$ the product of all the first 2λ primes. We denote the bit-length of p^* as $\rho = \lceil \log(p^*) \rceil$. Finally, by $\mathcal{D}_{2\lambda}$ we denote the distribution of $\bar{s}$, according to the sampling method described above. Note that $s\bar{s} = p^*$. Also, the new witness $\hat{W}$ could be verified by checking $\hat{W}^{s \prod_{x_i \in X} x_i} = \hat{\mathsf{acc}}$.

Our first technical contribution is proving that this randomization is sufficient. More precisely, we use a computational assumption over groups of unknown order, called DDH-II, and we show that under DDH-II $\hat{W}$ is computationally indistinguishable from a random $R \leftarrow\$ \mathbb{G}_?$. We stress that this hiding property holds only for the value $\hat{W}$ *alone*, i.e., when the random subset of $\mathbb{P}_{2\lambda}$ is not revealed.

```
Setup(1^λ):
    $(\mathbb{G}_?, g_?) \leftarrow \mathcal{G}_?(1^\lambda)$
    return $\mathsf{pp} := (\mathbb{G}_?, g_?)$

Accum(pp, S):
    $\mathsf{prd} \leftarrow \mathsf{Prod}(S)$
    return $\mathsf{acc} := g_?^{\mathsf{prd}}$

Ins(pp, acc, S'):
    $\mathsf{prd}' \leftarrow \mathsf{Prod}(S')$
    return $\mathsf{acc}' := \mathsf{acc}^{\mathsf{prd}'}$

PrvMem(pp, S, X):
    $\mathsf{prd} \leftarrow \mathsf{Prod}(S)$
    $\mathsf{prd}_X \leftarrow \mathsf{Prod}(X)$
    return $W := g_?^{\mathsf{prd}/\mathsf{prd}_X}$

VfyMem(pp, acc, X, W):
    $\mathsf{prd}_X \leftarrow \mathsf{Prod}(X)$
    Accept iff $W^{\mathsf{prd}_X} = \mathsf{acc}$
```

(a) RSA Accumulator for multisets of prime numbers. Above $\mathsf{Prod}(S)$ denotes the integer product of the elements in S.

```
Setup($1^\lambda, \mathsf{B} \in \mathbb{N}, \mathsf{ChkSz} \in \mathbb{N}, n \in \mathbb{N}$):
    $(\mathbb{G}_p, f) \leftarrow \mathcal{G}_p(1^\lambda)$
    If $\mathsf{ChkSz} > p$ then output $\perp$
    Let $N := n \cdot \left\lceil \frac{\mathsf{B}}{\mathsf{ChkSz}} \right\rceil$
    Sample $g_1, \ldots, g_N, h \leftarrow\$ \mathbb{G}_p$
    return $\mathsf{ck} := (\mathbb{G}, \mathsf{B}, \mathsf{ChkSz}, n, g_1, \ldots, g_N, h)$

Comm($\mathsf{ck}, \vec{x} \in \mathbb{Z}^n; r \in \mathbb{Z}_p$):
    If $\exists i : x_i > \mathsf{B}$ then output $\perp$
    Let $\left(x_1^{(i)}, \ldots, x_m^{(i)}\right)$ be the representation of $x_i$ in base ChkSz for $i \in [n]$
    $\vec{y} := \left(x_1^{(1)}, \ldots, x_m^{(1)}, \ldots, x_1^{(n)}, \ldots, x_m^{(n)}\right)$
    return $h^r \prod_{i=1}^{N} g_i^{y_i}$
```

(b) Pedersen Commitments for vectors of integers. B is an upper bound over the integers we can commit to. ChkSz is the size of the chunks in which we divide each integer. n is the number of integers we can commit at the same time. $m = \left\lceil \frac{\mathsf{B}}{\mathsf{ChkSz}} \right\rceil$ is the number of chunks needed for each integer.

Fig. 1: Accumulator and commitment schemes we will use throughout this work

As we show later, this is sufficient for our purpose as we can hide the integer s in the same way as we hide the elements we prove membership of.

In the following section we state and explain the DDH-II assumption. In brief, this is a variant of the classical DDH assumption where the random exponents follow specific, not uniform, distributions. Next, we prove that under DDH-II $\hat{W}$ is computationally indistinguishable from random.

1) The DDH-II assumption: First, we state the DDH-II assumption, which is parametrized by a generator $\mathcal{G}_?(1^\lambda)$ of a group (of unknown order in our case) and by a well-spread distribution $\mathcal{WS}_{2\lambda}$ (in our case $\mathcal{D}_{2\lambda}$). A distribution $\mathcal{WS}_{2\lambda}$ with domain $\mathcal{X}_{2\lambda}$ is called *well-spread* if $\Pr[x \leftarrow\!\!\$\ \mathcal{WS}_{2\lambda}] \leq 2^{-2\lambda}$ for each $x \in \mathcal{X}_{2\lambda}$. Intuitively, this means that the elements sampled from this distribution are 'sufficiently random'.

Assumption 1 (DDH-II). *Let $\mathbb{G}_? \leftarrow \mathcal{G}_?(1^\lambda)$ and $g_? \leftarrow\!\!\$\ \mathbb{G}_?$. Let $\mathcal{WS}_{2\lambda}$ be a well-spread distribution with domain $\mathcal{X}_{2\lambda} \subseteq [1, \mathsf{minord}(\mathbb{G}_?)]$. Then for any PPT $\mathcal{A}$:*

$$\left|\Pr[\mathcal{A}(g_?^x, g_?^y, g_?^{xy}) = 0] - \Pr[\mathcal{A}(g_?^x, g_?^y, g_?^t) = 0]\right| = \mathsf{negl}(\lambda)$$

where $x \leftarrow\!\!\$\ \mathcal{WS}_{2\lambda}$ and $y, t \leftarrow\!\!\$\ [1, \mathsf{maxord}(\mathbb{G}_?)2^\lambda]$.[7]

Our distribution of interest $\mathcal{D}_{2\lambda}$ can be shown well-spread: there are $2^{2\lambda}$ outcomes and are all distinct, $\bar{s} = \prod_{p_i \in \mathbb{P}_{2\lambda}} p_i^{1-bi}$ are distinct since they are different products of the same primes (no p_i can be used twice). It follows that $\Pr[\bar{s} \leftarrow\!\!\$\ \mathcal{D}_\lambda] = 1/2^{2\lambda}$ for every $\bar{s}$.

Remark 2. *The constraint that the domain should be in $[1, \mathsf{minord}(\mathbb{G}_?)]$ is for the following reason: If a sampled x is larger than $\mathsf{ord}(g_?)$ then in the exponent of $g_?^x$ a reduction modulo $\mathsf{ord}(g_?)$ will implicitly happen leading to a $g_?^x = g_?^{x'}$ for some $x' \neq x$. This can turn $g_?^x$ more frequently sampled, which can potentially help the adversary distinguish between $(g_?^x)^y$ and $g_?^t$.*

Different variants of DDH-II have been proven secure in the generic group model [31], [32] for prime order groups [33], [34]. Similarly, we can prove it secure for groups of unknown order, with minor technical modifications related to GGM proofs for hidden order groups [35].

Remark 3. *The need of an at least $2^{2\lambda}$-large domain $\mathcal{X}_{2\lambda}$ (and at most $2^{-2\lambda}$ probability) for λ security parameter comes from well-known subexponential attacks on DLOG.*

2) Security Proof of our hiding witnesses:

Theorem 1. *For any parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, set S (where $S \cap \mathbb{P}_{2\lambda} = \emptyset$), $R \leftarrow\!\!\$\ \mathbb{G}_?$ and $\hat{W}$ computed as described above it holds that:*

$$\left|\Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0] - \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]\right| = \mathsf{negl}(\lambda)$$

for any PPT adversary $\mathcal{A}$, assuming that DDH-II assumption holds for $\mathbb{G}_?$ and $\mathcal{D}_{2\lambda}$.

Proof. Let $\mathcal{A}$ be a PPT adversary that achieves a non-negligible advantage ϵ on the above, i.e.

$$\left|\Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0] - \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]\right| = \epsilon$$

We construct an adversary $\mathcal{B}$ against DDH-II that, using adversary $\mathcal{A}$, gains the same advantage.

$\mathcal{B}$ receives $(\mathbb{G}_?, g_?, g_?^{\bar{s}}, g_?^r, g_?^{b\bar{s}r+(1-b)t})$, where $\bar{s} \leftarrow\!\!\$\ \mathcal{D}_{2\lambda}$ and $r, t \leftarrow\!\!\$\ [1, \mathsf{maxord}(\mathbb{G}_?)2^\lambda]$. Then it chooses arbitrarily an element u and sets $S = \{u\}$, $\mathsf{pp} \leftarrow (\mathbb{G}_?, g_?^r)$ and $V = g_?^{b\bar{s}r+(1-b)t}$. $\mathcal{B}$ sends (pp, S, V) to the adversary $\mathcal{A}$, who outputs a bit b^*. Finally, $\mathcal{B}$ outputs b^*.

First, notice that $g_?^r$ is statistically close to a random group element of $\mathbb{G}_?$, meaning that $\mathcal{A}$ cannot distinguish pp from public parameters generated by $\mathsf{Acc.Setup}(1^\lambda)$. Furthermore if $b = 0$ then V is again a (statistically indistinguishable element from a) uniformly random group element of $\mathbb{G}_?$ so $\Pr[\mathcal{B} = 0|b = 0] = \Pr[\mathcal{A}(\mathsf{pp}, S, R) = 0]$. On the other hand, if $b = 1$ then $V = g_?^{r \cdot \bar{s}} = \hat{W}_u$ is a witness of u so $\Pr[\mathcal{B} = 0|b = 1] = \Pr[\mathcal{A}(\mathsf{pp}, S, \hat{W}) = 0]$. Therefore we conclude that the probability of $\mathcal{B}$ to win the DDH-II is ϵ. □

B. Building Blocks

1) Succinct proofs of knowledge of exponent (PoKE*):* We recall the succinct proofs of knowledge of a DLOG for hidden order groups, introduced by Boneh et al. [15]. More formally, PoKE is a protocol for the relation

$$R^{\mathsf{PoKE}}(A, B; x) = 1 \Leftrightarrow A^x = B$$

parametrized by a group of unknown order $\mathbb{G}_?$ and a random group element $g_? \in \mathbb{G}_?$. The statement consists of group elements $A, B \in \mathbb{G}_?$ while the witness is an arbitrarily large $x \in \mathbb{Z}$.

In Figure 8 (appendix A) is the description of the protocol. For simplicity we directly expose its non-interactive version (after Fiat-Shamir). This protocol is succinct: proof size and verifier's work are independent of the size of x, $O(1)$ and $O(\|\ell\|) = O(\lambda)$ respectively.

2) CP-SNARK for integer arithmetic relations: We assume an efficient CP-SNARK $\mathsf{cp\Pi}^{\mathsf{modarithm}}$ for the following relation:

$$\tilde{R}_{\mathsf{ck}}^{\mathsf{modarithm}}(\boxed{c_{\vec{u}}}, \boxed{c_{s,r}}, h, \ell, k') = 1 \Leftrightarrow$$

$$k' = s \cdot h \cdot \prod_{i \in [m]} u_i + r \mod \ell$$

Above, $\vec{u} = (u_1, \ldots, u_m) \in \mathbb{Z}^m$ is a vector of integers with a corresponding multi-integer commitment $c_{\vec{u}}$; $r, s \in \mathbb{Z}$ are integers committed with a corresponding multi-integer commitment $c_{s,r}$ and $\ell, h \in \mathbb{Z}$, $k' \in [0, \ell - 1]$ are (small) integers known as public inputs by both prover and verifier.

The above relation is equivalent to the integer relation:

$$\tilde{R}_{\mathsf{ck}}^{\mathsf{arithm}}(\boxed{c_{\vec{u}}}, \boxed{c_{s,r}}, h, \ell, k'; q) = 1 \Leftrightarrow q\ell + k' = s \cdot h \prod_i u_i + r$$

[7]Since the order of the group is unknown, we cannot efficiently produce uniformly random elements with $y, t \leftarrow\!\!\$\ [1, \mathsf{ord}(g_?)]$. However, $y, t \leftarrow\!\!\$\ [1, \mathsf{maxord}(\mathbb{G})2^\lambda]$ still produces statistically close to uniform elements.

In fact this is how a modulo operation is encoded in a SNARK circuit. q here is a witness given to the SNARK.[8]

3) CP-SNARK for inequalities: We need a CP-SNARK $\mathsf{cp\Pi}^{\mathsf{bound}}$ for the relation (where B is a public integer):

$$\tilde{R}^{\mathsf{bound}}_{\mathsf{ck}}(\boxed{c_{\vec{u}}}, B) = 1 \Leftrightarrow u_i > B$$

*C. Our Construction for Batched Set Membership (*HAR*i*SA*)*

Here we describe our CP-SNARK for the relation $\tilde{R}^{\mathsf{mem}}_{\mathsf{ck}}$ for RSA accumulators and Pedersen commitments to vectors of integers. Let us recall the setting in more detail.

Prover and verifier hold an accumulator acc to a set S and a commitment $c_{\vec{u}}$. The set's domain are natural primes greater than $p_{2\lambda}$, the 2λ-th prime. The protocol works in the "trusted accumulator model" (section III-D), which means the set is assumed to be public but the verifier does not take it as an input, it only uses acc, for efficiency reasons.[9]

The prover knows a batch of set elements $\vec{u} = (u_1, \ldots, u_m)$ that are an opening of the commitment $c_{\vec{u}}$, and its goal is to convince the verifier that all the u_i's are in S. To this end, we assume that the prover has an accumulator witness $W_{\vec{u}}$ as an input, either precomputed or given by a witness-providing entity. In this sense, the prover's goal translates into convincing the verifier that it has $W_{\vec{u}}$ such that $W_{\vec{u}}^{\prod_i u_i} = \mathsf{acc}$.

We give a full description of the CP-SNARK in Figure 2. We refer to the technical overview (sec. II) for an high-level explanation. Below we provide additional comments.

To begin with, both prover and verifier transform the accumulator acc into $\hat{\mathsf{acc}}$, the one corresponding to the same set with the additional small prime numbers from $\mathbb{P}_{2\lambda}$.[10] Next, the prover transforms W into a hiding witness as $\hat{W} = W^{\bar{s}}$ via our masking method of section IV-A, and then computes a (Fiat-Shamir-transformed) zero-knowledge Σ-protocol for the accumulator's verification $\hat{W}^{su^*} = \hat{\mathsf{acc}}$. However, since the last message k of the protocol is not succinct, it computes a PoKE for the relation $(\hat{\mathsf{acc}}^h R) = (\hat{W}_{\vec{u}})^k$ (exponent k), which is the verification equation of the Σ-protocol. The PoKE verification requires a check $Q^\ell \hat{W}_{\vec{u}}^{res}$ where res is supposed to be $k \mod \ell$. The last step of the proof is to show that res is not just "some exponent" but it is exactly $r + hsu^* \mod \ell$ with u^* being the product of all the u_i's committed in $c_{\vec{u}}$. To do so, the prover generates a proof with the $\mathsf{cp\Pi}^{\mathsf{arithm}}$ CP-SNARK over the commitments $c_{\vec{u}}$, $c_{s,r}$ (r is the masking randomness of the Σ-protocol sampled in the first move). Also, for a technicality related to the soundness of the protocol, we require that s and r are committed *before* receiving the random oracle challenge h. Finally, the prover generates a proof with $\mathsf{cp\Pi}^{\mathsf{bound}}$ over the commitment $c_{\vec{u}}$ to ensure that the elements are in the right domain.[11]

We present our construction in fig. 2. This construction is obtained by applying Fiat-Shamir in the random oracle model (ROM) and additional optimizations to its interactive counterpart which we describe in the appendix (fig. 9).

```
Setup $(1^\lambda, \mathsf{ck}, \mathsf{pp})$ :
    $\mathsf{crs}_2 \leftarrow \mathsf{cp\Pi}^{\mathsf{modarithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\mathsf{modarithm}}_{\mathsf{ck}})$
    $\mathsf{crs}_3 \leftarrow \mathsf{cp\Pi}^{\mathsf{bound}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, R^{\mathsf{bound}})$
    $\mathbf{return}\ \mathsf{crs} := (\mathsf{ck}, \mathsf{pp}, \mathsf{crs}_{\mathsf{arithm}}, \mathsf{crs}_{\mathsf{bound}})$
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 $u^* = \prod_i u_i$
    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 $r \leftarrow\!\$\ \{0,1\}^{\|p^*\| + \|u^*\| + 2\lambda}$
    $c_{s,r} \leftarrow \mathsf{Comm}_{\mathsf{ck}}(s, r; o_{s,r})$
    $R \leftarrow \hat{W}_{\vec{u}}^r$
    $h \leftarrow H(\mathsf{crs} || \mathsf{acc} || c_{\vec{u}} || c_{s,r} || \hat{W}_{\vec{u}} || R)$
    $k \leftarrow r + (u^* s) h$
    $\pi_1 \leftarrow \Pi^{\mathsf{PoKE}}.\mathsf{Prv}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R; k)$
    Parse $\pi_1$ as $(Q, res)$
    $\ell \leftarrow H_{\mathsf{prime}}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R)$
    $\pi_2 \leftarrow \mathsf{cp\Pi}^{\mathsf{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, res; \vec{u}, o_{\vec{u}}, r, s, o_{s,r})$
    $\pi_3 \leftarrow \mathsf{cp\Pi}^{\mathsf{bound}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\vec{u}}, p_{2\lambda}; \vec{u}, o_{\vec{u}})$
    $\mathbf{return}\ \pi = (\hat{W}_{\vec{u}}, R, c_{s,r}, \pi_1, \pi_2, \pi_3)$
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 $\pi$ as $(\hat{W}_{\vec{u}}, R, c_{s,r}, \pi_1, \pi_2, \pi_3)$ and $\pi_1$ as $(Q, res)$
    $\ell \leftarrow H_{\mathsf{prime}}((\mathbb{G}_?, g_?), \hat{W}_{\vec{u}}, \hat{\mathsf{acc}}^h R)$
    $h \leftarrow H(\mathsf{crs} || \mathsf{acc} || c_{\vec{u}} || c_{s,r} || \hat{W}_{\vec{u}} || R)$
    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 $\mathsf{cp\Pi}^{\mathsf{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, res, \pi_2) \neq 1$
    Reject if $\mathsf{cp\Pi}^{\mathsf{bound}}.\mathsf{Vfy}(\mathsf{crs}_3, c_{\vec{u}}, p_{2\lambda}, \pi_3) \neq 1$
```

Fig. 2: HAR*i*SA: our scheme for proving set membership of a committed element. We let H denote a cryptographic hash function modeled as a random oracle.

Theorem 2. *Let H, H_{prime} be modeled as random oracles and $\mathsf{cp\Pi}^{modarithm}$, $\mathsf{cp\Pi}^{bound}$ be secure CP-SNARKs. The construction in fig. 2 for the relation $\tilde{R}^{mem}_{\mathsf{ck}}$ is a secure CP-SNARK: succinct,*

[8] For the sake of our general protocol, it is not necessary that q remains hidden. It is only important that the proof is succinct w.r.t. its size. However, $\vec{u}$, s and r should remain hidden.

[9] This is a common consideration in scalable systems. The accumulator to the set is either computed once by the verifier or validated by an incentivized majority of parties that is supposed to maintain it.

[10] This operation can also be precomputed, we make it explicit only to show that they can both work with a classical RSA accumulator as an input.

[11] For the sake of generality we present π_2, π_3 as distinct proofs. In practice they can be proved by the same CP-SNARK and save one proof-size.

knowledge-sound under the adaptive root assumption, and zero-knowledge under the DDH-II assumption.

Proof. For succinctness, one can inspect that the proof size is proportional to the proof size of $\mathsf{cp}\Pi^{\text{arithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$ plus some small constant overhead. Similarly for the verifier's cost. So succinctness is inherited from succinctness of $\mathsf{cp}\Pi^{\text{arithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$.

The proof for its interactive version (fig. 9) is in the appendix, theorem 3. Then knowledge-soundness and zero-knowledge come directly from security of the Fiat-Shamir transformation in the random oracle model. □

Extensions. In Appendix B we show how to extend our CP-SNARK to support arbitrary—not necessarily prime—set elements, using an efficient hash-to-prime proof based on a single hash execution. We also discuss how to extend our protocol to prove (in zero-knowledge) batch non-membership.

V. B-INS-ARISA: CP-SNARK FOR SET-INSERTION

We show a CP-SNARK for the relation $\tilde{R}^{\text{ins}}_{\mathsf{ck}}$ (see sec. III-D) and consequently for the $\mathsf{MultiSwap}$ relation R^{mswap}, using RSA accumulators. We call this construction B-INS-ARiSA.

For set-insertion we need to prove that $\mathsf{Acc.Ins}(\mathsf{pp}, \mathsf{acc}, U) = \mathsf{acc}'$, where acc and acc' are public but the set of elements added U is not publicly provided,[12] but instead a succinct commitment of it c_U. The accumulator acc is assumed to be trusted, in the sense that it is computed correctly from a set of valid elements, however for acc' we do not make this assumption. In fact this is essentially the purpose of the protocol, to prove correctness of acc'.

A. Our construction for $\tilde{R}^{ins}_{\mathsf{ck}}$ (B-INS-AR*i*SA)

We begin with an high-level overview of the scheme. Proving correctness of set-insertion in RSA accumulators roughly consists of proving the following:

1) $\mathsf{acc}^{\prod_{u_i \in U} u_i} = \mathsf{acc}'$.
2) $u_i \in \mathcal{D}$ for each $u_i \in U$.

Clearly the first point ensures that the insertion of the elements has been done correctly. However we still need to prove that the elements of U are in the correct domain $\mathcal{D}$. A usual domain for secure RSA Accumulators is the prime numbers, $\mathcal{D} = \mathbb{P}$. We will discuss later alternative domains.

1) On the choice of set-membership protocol: Notice that the first point is in fact a set-membership verification for the set of acc' and acc is the corresponding witness of the membership. Therefore, we could in principle apply our batch set-membership protocol of sec. IV and already obtain a construction. However, that construction would carry an overhead, due to zero-knowledge, unnecessary for the the purposes of this section(for set-insertion we do not aim at zero-knowledge as discussed above). Therefore we use a simple PoKE proof for the exponentiation $\mathsf{acc}^{\prod_{u_i \in U} u_i} = \mathsf{acc}'$.

2) On the choice of the domain: For the second point, $u_i \in \mathcal{D}$, we need a domain that preserves the security of RSA accumulators but at the same time can be proven efficiently with a succinct protocol. Some examples of secure domains include: (1) prime numbers or prime numbers of a specific size, (2) outputs of a collision resistant hash-to-prime function, (3) outputs of a division-intractable hash function.

However, for the first two options there is no known efficient argument of knowledge; the only existing (succinct) solution is proving them with a general-purpose SNARK.[13] In particular, it is the primality check that is difficult to handle, and encoding it inside a SNARK circuit gets prohibitive as it usually requires many iterations.

Ozdemir et al. [21] observed that the division-intractable hash of Coron and Naccache [37] is (comparably) lightweight. Division intractability of a hash function H_{DI} with range in $\mathbb{Z}$ briefly means that it is hard for an adversary to find an element x^* and a set $\{x_i\}$ such that $x \notin \{x_i\}$ but $H_{\text{DI}}(x) \mid \prod_i H_{\text{DI}}(x_i)$. Essentially the function of [37] consists of a single hash computation and an addition (of 2048-bit integers). This function, denoted H_{DI}, works as follows: given a large public offset Δ of 2048-bits, the output of H_{DI} is

$$H_{\text{DI}}(x) = \Delta + H(x)$$

where H is any collision-resistant hash function with image $[0, 2^{2\lambda}]$. It can be shown that if we model H as a random oracle H_{DI} is collision-intractable [37]. That is any output has at least a unique large prime factor, with overwhelming probability.

H can be any standard hash function as SHA256, or even a SNARK-friendly hash as Poseidon [20]. Proving a hash evaluation per element inserted inside a SNARK can be affordable in comparison to the rest of the solutions mentioned above that require primality checks. For this reason, we use division-intractable hashes as to produce accumulator elements. This technique, together with an implementation inside a SNARK, was introduced in [21].

The original elements of the set are arbitrary integers, $S \subset \mathbb{Z}$. Every element of the set x is mapped, through H_{DI}, to a division-intractable element $u = H_{\text{DI}}(x)$ that are next accumulated to produce acc. Proving that $\{x_1, \ldots, x_m\}$ were inserted in S is equivalent to proving that the accumulator was updated with the corresponding $\{u_1, \ldots, u_m\} = \{H_{\text{DI}}(x_1), \ldots, H_{\text{DI}}(x_m)\}$. We refer the reader to [37] for a security analysis.

For our protocol we assume a CP-SNARK for the above DI-hash function evaluation:

$$\tilde{R}^{H_{\text{DI}}}_{\mathsf{ck}}(\boxed{c_{\vec{u}}}; \vec{x}) = 1 \Leftrightarrow u_i = \Delta + H(x_i)$$

parameterized by a division-intractable hash, (H, Δ).

3) CP-SNARK for integer arithmetic relations: Again we assume an efficient CP-SNARK $\mathsf{cp}\Pi^{\text{modarithm}}$ for the relation: $\tilde{R}^{\text{modarithm}}_{\mathsf{ck}}(\boxed{c_{\vec{u}}}, \ell, k') = 1 \Leftrightarrow k' = \prod_{i \in [m]} u_i \mod \ell$ which is a simplification of the relation defined in section IV.

[12]As mentioned before, not giving U to the verifier is for the sake of succinctness. Hiding U is not in our scope.

[13]Specialized solutions based on Σ-protocols exist [36] but are both inefficient and for a single prime, thus not succinct.

4) Summary of the construction: Putting things together, for our construction we prove that: a batch $U = \{u_1, \ldots, u_m\}$ of committed elements is an output of H_{DI}, with $\mathsf{cp}\Pi^{H_{\text{DI}}}$; these elements are inserted in the accumulator, with a PoKE for $\mathsf{acc}^{\prod_i u_i} = \mathsf{acc}'$.

However, there should be a way to "link" the elements U in the two proofs. Essentially to show that the proofs are about the same batch of elements. In order to avoid encoding the RSA exponentiation acc^{u^*} inside the SNARK, which would be virtually infeasible,[14] we use an intermediate CP-SNARK that proves the following: the product u^* of the committed elements modulo the ℓ of the PoKE equals the res part of the PoKE proof, $u^* = res \mod \ell$. As we show in the next section, this guarantees that the u^* of the PoKE is the same as the u^* (implicitly) committed, in $c_{\vec{u}}$.

A full description of our scheme is in Figure 3.

```
$\underline{\mathsf{Setup}\left(1^\lambda, \mathsf{ck}, \mathsf{pp}\right):}$
    $\mathsf{crs}_2 \leftarrow \mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{\text{modarithm}}_{\mathsf{ck}})$
    $\mathsf{crs}_3 \leftarrow \mathsf{cp}\Pi^{H_{\text{DI}}}.\mathsf{Setup}(1^\lambda, \mathsf{ck}, \tilde{R}^{H_{\text{DI}}}_{\mathsf{ck}})$
    $\mathbf{return}\ \mathsf{crs} := (\mathsf{ck}, \mathsf{pp}_{\mathsf{Acc}}, \mathsf{crs}_2, \mathsf{crs}_3)$
$\underline{\mathsf{Prove}\left(\mathsf{crs}, \mathsf{acc}, \mathsf{acc}', c_{\vec{u}}; \vec{u}, o_{\vec{u}}, \vec{x}\right):}$
    Let $u^* = \prod_i u_i$
    $\pi_1 \leftarrow \Pi^{\mathsf{PoKE}}.\mathsf{Prv}\left((\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}'; u^*\right)$
    Parse $\pi_1$ as $(Q, res)$
    $\ell \leftarrow H_{\text{prime}}((\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}')$
    $\pi_2 \leftarrow \mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, c_{\vec{u}}, \ell, res; \vec{u}, o_{\vec{u}})$
    $\pi_3 \leftarrow \mathsf{cp}\Pi^{H_{\text{DI}}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\vec{u}}; \vec{u}, o_{\vec{u}}, \vec{x})$
    $\mathbf{return}\ \pi = (\pi_1, \pi_2, \pi_3)$
$\underline{\mathsf{Verify}\left(\mathsf{crs}, \mathsf{acc}, \mathsf{acc}', c_{\vec{u}}, \pi\right):}$
    Parse $\pi$ as $(\pi_1, \pi_2, \pi_3)$ and $\pi_1$ as $(Q, res)$
    Reject if $\Pi^{\mathsf{PoKE}}.\mathsf{Vfy}(\mathbb{G}_?, g_?), \mathsf{acc}, \mathsf{acc}', \pi_1) \neq 1$
    Reject if $\mathsf{cp}\Pi^{\text{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{u}}, \ell, res, \pi_2) \neq 1$
    Reject if $\mathsf{cp}\Pi^{H_{\text{DI}}}.\mathsf{Vfy}(\mathsf{crs}_3, c_{\vec{u}}, \pi_3) \neq 1$
```

Fig. 3: B-INS-ARiSA: our scheme for proving correct set insertion of a committed batch of elements.

B. Multiswaps

As argued in section III-D2 batch-insertion gives a succinct MultiSwap protocol: the relation R^{mswap} roughly consists of two set insertions.

$$R^{\text{mswap}}(\mathsf{acc}, \mathsf{acc}'; X, Y) = 1 \Leftrightarrow$$
$$\exists \mathsf{acc}_{\text{mid}} : R^{\text{ins}}(\mathsf{acc}, \mathsf{acc}_{\text{mid}}; Y) \wedge R^{\text{ins}}(\mathsf{acc}_{\text{mid}}, \mathsf{acc}'; X)$$

[14]An RSA exponentiation of this size would require nearly 2 millions constraints per element of the batch.

Given a set S, its corresponding (trusted) accumulator acc and a sequence of "swap" pairs $(x_1, y_1), \ldots, (x_m, y_m)$ the prover computes $\mathsf{acc}_{\text{mid}}$, acc' and two corresponding batch insertion proofs for $\mathsf{acc} \xrightarrow{\mathsf{Ins}} \mathsf{acc}_{\text{mid}}$, $\mathsf{acc}' \xrightarrow{\mathsf{Ins}} \mathsf{acc}_{\text{mid}}$. In short the prover publishes acc' and the proof for the multiswap is $\mathsf{acc} \xrightarrow{\mathsf{MultiSwap}} \mathsf{acc}'$ is:

$$\pi \leftarrow (\pi_1^{\text{ins}}, \pi_2^{\text{ins}}, \mathsf{acc}_{\text{mid}})$$

This proof can convince a verifier that the multiswap was done correctly (and that acc' is trusted).

1) Generating acc_{mid} and acc': Computationally speaking the bottleneck in the above is the generation of acc'. Nevertheless, the intermediate value $\mathsf{acc}_{\text{mid}}$ is the result of the batch insertion of all y_i's, hence it can be efficiently computed in time $O(m)$ (m the size of the batch) by m sequential Acc.Ins. On the other hand, the value acc' is the result of "batch deletion" of all x_i's, an operation that cannot be done efficiently (in $O(m)$-time) and the only manner is to compute acc' from scratch, i.e. accumulate all remaining values: $\mathsf{acc}' \leftarrow \mathsf{Acc.Accum}(\mathsf{pp}, S')$, where $S' = S \uplus \{y_i\}_i \boxminus \{x_i\}$. This requires time proportional to the size of the set, $O(n + m)$.

To this end, one can use a precomputation technique to speed-up the online computational cost. As shown by Boneh et al. [15], if one has precomputed a witness W_{x_1} then already $\mathsf{acc}' = W_{x_1}$ is an accumulator for $S \boxminus \{x_1\}$. If one has precomputed witnesses W_{x_1} and W_{x_2} one can in compute $\mathsf{acc}' = W_{x_1,x_2}$, in $O(1)$-time by using Shamir's trick [38], which is essentially an accumulator for $S \boxminus \{x_1, x_2\}$. Generalizing this, if all witnesses are precomputed $W_{x_1}, \ldots, W_{x_n}$ then one can compute acc' for any $S \boxminus \{x_{i_1}, \ldots x_{i_m}\}$, in $O(m)$ time. This would require the prover to store additional $O(n)$ group elements.

To avoid storing linear-number of elements one can use another preprocessing method, introduced by Campanelli et al. [39], that offers storage-online time tradeoffs. The storage cost is $O(n/B)$ and the online time (worst-case) $O(mB)$, for any chosen parameter B. Essentially, the more elements one stores the less resources it uses online and vice-versa.

C. Comparison with [21]

Technically speaking our approach in this section carries similarities with the one of Ozdemir et al. There are two distinguishing differences. The first is in the succinct protocol for the exponentiation $\mathsf{acc}^{u^*} = \mathsf{acc}'$. Ozdemir et al. make use of a Wesolowski proof (PoE protocol), while we propose the use of the Boneh et al. proof (PoKE protocol). The second is that we do not encode the verification of this proof inside the circuit of the SNARK.

The PoE protocol is a succinct proof of correct RSA exponentiation, introduced in [30]. It is defined for verifiers that know the exponent, i.e. the proof's input is $(\mathsf{acc}, \mathsf{acc}', u^*)$. For the non-interactive version, in order for the Fiat-Shamir transform to be sound the challenge should be generated as $\ell \leftarrow H_{\text{prime}}(\mathsf{acc}, \mathsf{acc}', u^*)$, meaning that it should take the large exponent as input. Since the verifier shall not receive the set U, it cannot generate the challenge ℓ itself. Subsequently, the prover should, in addition to the rest of the computations, prove inside the SNARK the computation $\ell = H_{\text{prime}}(\mathsf{acc}, \mathsf{acc}', u^*)$. This computation gives a significant overhead to the prover's workload.

We replace the PoE protocol with a PoKE protocol. The PoKE proof, introduced in [15], is a proof *of knowledge* of exponent. That is, here the exponent u^* is a witness instead of an input. Meaning that the Fiat-Shamir challenge is now generated as $\ell \leftarrow H_{\text{prime}}(\mathsf{acc}, \mathsf{acc}')$. All inputs are public and known to the verifier. This translates to a save on the expensive SNARK computation $H_{\text{prime}}(\mathsf{acc}, \mathsf{acc}', u^*)$. This saves m hash computations for the SNARK and a hash-to-prime computation (applied on the output of the m hash-chain). The former has a cost of ≈ 300–$45,000$ constraints per input (depending on the choice of the hash), while the latter has a fixed cost of $217,703$ constraints [21]. So overall it has a significant impact that depends on batch size. For example, for SHA256 and a moderate-sized batch size $m = 1000$, our approach saves more than 45 million constraints.

Notably this replacement does not affect the security assumptions: although the PoKE itself is secure in the generic group model [31], [32], a careful security analysis shows that when combined with $\mathsf{cp}\Pi^{\text{modarithm}}$ it can be proven secure in the standard model under the adaptive root assumption [30] (see proof of theorem 3), which is the same assumption as in [21]. Furthermore, in our favor, for this step we do not need to take the heuristic assumption of encoding a random oracle's representation inside a SNARK.

Instead of encoding the PoKE verification $Q^{\ell}\mathsf{acc}^{res} = \mathsf{acc}'$ inside the SNARK, we let the verifier perform it itself. According to [21] (figure 3) the RSA operations needed for this verification—two RSA $|\ell|$-bit exponentiations and one RSA multiplication—overall cost about 5 million constraints. Our approach has the downside of having to additionally include the PoKE proof, (Q, res), in the overall proof of set-insertion, which has an overhead of 1 RSA group elements and a 256-bit prime in the proof size. Therefore our approach can be viewed as a tradeoff: 288 bytes in the proof vs 5 million constraints less for the prover (and vice versa).

VI. Evaluation

A. Instantiations and Implementation

We instantiate the CP-SNARKs building blocks in our construction in fig. 2, $\mathsf{cp}\Pi^{\text{modarithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$, with $\mathsf{LegoGroth16}$ from [24], an efficient commit-and-prove version of Groth16 [41]. Like Groth16, it requires an elliptic curve endowed with a bilinear map. We use the curve BLS12-381 [42] for our instantiations. The proof size of $\mathsf{LegoGroth16}$ is constant (five group elements), amounting to 288 bytes in BLS12-381. For the accumulator scheme we use a 2048-bit RSA group. To be compatible with the assumptions of DDH-II in such a group, we must take at most $2\lambda = 232$ primes to hide the RSA witness in our construction. This does not affect the security provided by a 2048-bit RSA group. We enforce the relation in $\mathsf{cp}\Pi^{\text{bound}}$ through the Poseidon hash function [20].

Implementation: We provide an implementation of our construction (fig. 2) comprising $\mathsf{LegoGroth16}$, $\mathsf{cp}\Pi^{\text{arithm}}$ and PoKE. Part of our code is an extension of the C++ SNARK library $\mathsf{libsnark}$ [43] with $\mathsf{LegoGroth16}$. We use the Java library JSnark [44] to produce the circuit representation for the arithmetic relation in $\mathsf{cp}\Pi^{\text{arithm}}$. We use a chunk size $\mathsf{ChkSz} = 32$ for commitments to integer (fig. 1).

Our code consists of approximately 2000 lines of C++ code and 100 lines of Java code. We plan to release it under an open-source license. We ran all our benchmarks single-threaded on CPU i7-10510U with 16GB of RAM (we ran DID-related benchmarks on a different but comparable machine).

B. Benchmarks for Batch Membership

We evaluate our approach comparing it to Merkle Tree for benchmarks. Specifically we compare it to the following (the asterisk is a placeholder for the depth of the tree):

- $\mathsf{MT\text{-}Pos\text{-}*}$: Merkle trees based on the Poseidon hash [20].
- $\mathsf{MT\text{-}SHA\text{-}*}$: Merkle trees based on the SHA-256 hash.

These hash functions have different tradeoffs: while Poseidon has a much smaller encoding for SNARKs, it is hundreds of times slower when executed natively. For the case of SHA we (very conservatively) estimate timings for larger batches. Each of the Merkle-tree instantiations above is benchmarked by proving their (batch) opening using $\mathsf{LegoGroth16}$ as a CP-SNARK. We compare these solutions on two benchmarks: a generic computation that consists only of batch membership statements, and a DID application in which one proves membership of a batch of elements as well as additional properties of these elements.

1) General purpose Batch Membership of n Elements: We describe our evaluations in fig. 4. Notice that the performance of Merkle-tree solutions vary with the size of the accumulated set (ours does not). We benchmark both the minimal set size 2^{16} and the more realistic set size 2^{32}.

Our scheme shows an order of magnitude savings in proving time. Our verification time is slower but still highly practical: approximately 60 ms vs 30 ms for CP-SNARKs on Merkle trees for common set sizes. Our proof size is also competitive although 4x larger at 1.17 KB[15]
CRS size. Our constructions also show a better size of public parameters (not in figure). For batch sizes respectively 1, 16 and 64, we estimate the CRS size of our scheme to be lower than 1, 2.5, 8.5 MB respectively. In contrast, the smallest CRS for the Merkle-tree solutions ($\mathsf{MT\text{-}Pos\text{-}16}$ for batch size 1) is already of approximately 5 MB, 5x larger than ours. We incur even better relative or absolute savings for more expensive hash functions—$\mathsf{MT\text{-}SHA\text{-}16}$ has a CRS of more than 250MB for batch size 1—or larger batch sizes in larger sets—$\mathsf{MT\text{-}Pos\text{-}32}$ has a CRS of more than 650MB for batch size 64. Notably, these savings on CRS size immediately translate in higher scalability due to less RAM consumption.

[15]This proof size uses the fact we can optimize the two $\mathsf{LegoGroth16}$ proofs in fig. 2 as just one.

2) Decentralized Identity (DID): We now experimentally validate our membership scheme in a more realistic scenario: a Decentralized ID (DID) application on the blockchain. In this setting, issuers can broadcast signed transactions consisting of a hiding commitment to a set of valid attributes of some identity holder (e.g., a bank account balance, a month's paycheck, identity information such as age). The appropriate holder receives the commitment with its opening, while the blockchain is supposed to validate and collect these transactions in an accumulator[16]. Next, whenever a verifier requests a holder to make some claim about some of her attributes, the holder sends a batch membership proof, proving that 1) a batch of commitments are members of the accumulator and 2) the attributes in these commitments meet a given property. Finally, the verifier validates the batch membership proof against the accumulator in the blockchain.

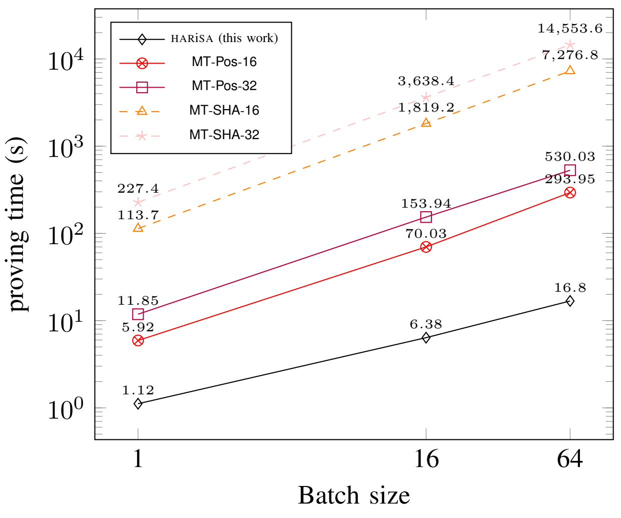
We compare our zero-knowledge batch membership proof ($\text{HARiSA}_{\text{DID}}$) against the MT-SHA-16 solution ($\text{MT-16}_{\text{DID}}$) in the above setting. Both schemes use the SHA-256 hash: ours for creating hash-to-prime commitments, MT-SHA-16 for the Merkle tree. We assume sets of 2^{16} numerical elements and that we test each attribute for a *range property* Figure 5 reports the proving time in $\text{HARiSA}_{\text{DID}}$ and $\text{MT-16}_{\text{DID}}$ as the number of proved attributes increase. The proving time for DID increases due to the two hash and attribute range check relations for each attribute. Note that the size of two hash relations dominate in $\text{HARiSA}_{\text{DID}}$ circuit (99.7%). Therefore the performance improvement in $\text{HARiSA}_{\text{DID}}$ becomes less dramatic compared to the pure batch membership proof example. The proving time in $\text{HARiSA}_{\text{DID}}$ is still 6–15x faster than in $\text{MT-16}_{\text{DID}}$. For larger sets, we can expect even larger improvements. The verification time and the proof size are equivalent to the batch membership example as shown in fig. 4.

C. *MultiSwap Benchmark*

We evaluate our MultiSwap solution built on top of B-INS-ARiSA (sec. V-B) and compare it with that of [21] and with a Merkle-tree based approach (Merkle-Swap). In all solutions we instantiate the hash functions with Poseidon. Our benchmark considers a computation consisting only of swap operations; we vary the number of swaps in 1–$10,000$ and fix the set size to 2^{20}.

Proving costs. For this evaluation, we use a cost-model analysis using, as metric, the number of constraints, that we compute based on the model of fig. 6. For Merkle-Swap the number of constraints is the only metric that conditions proving time. For our (resp. [21]) MultiSwap, the proving cost is made of the zkSNARK prover cost (which again depends on the number of constraints reported in fig. 6) plus the cost of RSA group operations to compute the accumulator acc' after deleting elements and the PoKE (resp. PoE) proof. To estimate

[16]We can assume that a smart contract checks the signature of the transaction and updates the accumulator by inserting the new commitment, which guarantees that every commitment in the accumulator is authenticated by an issuer.



Scheme	**V time** (ms)	**Proof size** (KB)
MT-$*$	31	0.29
HARiSA	63	1.17

Fig. 4: Comparison of batched set-membership in zero-knowledge: our scheme (HARiSA) vs LegoGroth16 on Merkle tree circuits. Plot is in log-scale. Verification time and proof size are $O(1)$ and independent of set/batch size. Due to constraints on RAM in our machine, timings for MT-SHA-* for batch size larger than 1 are an extrapolated lower bound.

the latter costs, we extract an equivalent measure in number of constraints based on [21, Sec. 4.4].

We report our results in fig. 7. Our MultiSwap solution has proving cost larger than Merkle-Swap for small batches, but it breaks even at ≈ 140 swaps. Also, it strictly improves over [21] MultiSwap, which has a larger fixed cost, and a break-even point w.r.t. Merkle-Swap at ≈ 1400 swaps.

Verification time and proof size. For this evaluation we consider an instantiation of all solutions with LegoGroth16 as a CP-SNARK. Similarly to the batch-membership case, our solution has slower verification and larger proofs, which are still practical. Our MultiSwap proof is 1.4 KB whereas proofs for Merkle-Swap and [21] are 288 bytes. Our verification time is $\approx$ 120ms and is about 4 times slower than that of Merkle-Swap and [21].

VII. Related Work

Succinct proofs for RSA accumulators. The works that are closest to ours are those of Benarroch et al. [22] and Ozdemir et al. [21], both concerning the efficient use of RSA accumulators with zkSNARKs. Comparing to [22], we achieve constant-size proofs of membership for batches of elements whereas [22] can only prove membership of a single (committed) element. In particular, the technique of [22] does not seem extendable to support batching with constant size proofs: they mainly rely on a new sigma-protocol for proving that two commitments, one in a prime order group and one

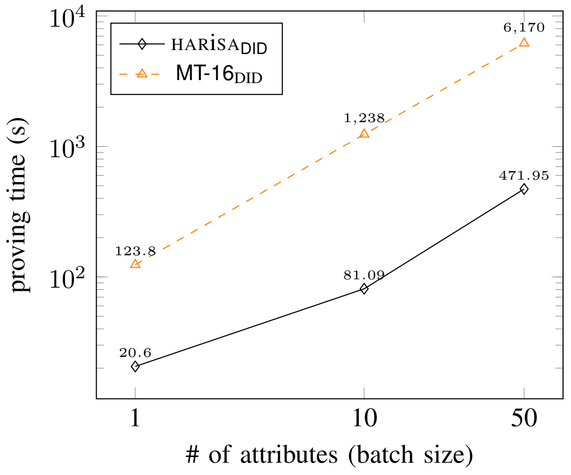


Fig. 5: Comparison of batched set-membership for the DID application: our scheme (HARiSA$_{DID}$) vs MT-16$_{DID}$. Plot is in log-scale. Due to constraints on RAM in our machine, timings for Merkle Tree solution for batch size larger than 1 are an extrapolated lower bound.

in a hidden-order group, open to the same integer value (the element in the accumulated set), but the size of its proof is linear in the integer's size. This means that extending this protocol to batch RSA witnesses would lead to linear-size proofs. Comparing to [21], our protocol for batch insertions is similar but has the following key differences. We employ a PoKE protocol instead of a proof-of-exponentiation (PoE); this allows us to generate the PoKE random oracle challenge based on the short and public verifier's input as opposed to the long and unknown exponent as in [21]. Thanks to this we can avoid encoding in the SNARK an expensive and long hashing along with a prime certification of its outputs. The second major difference is that Ozdemir et al. technique is not ZK-friendly and could not be used to do batch membership; to the best of our knowledge, hiding the RSA accumulator witness would require encoding an RSA group operation in the constraints.

Another work related to batch proofs is that of Boneh et al. [15] who construct such proofs for RSA accumulators with an efficient verification procedure. In their constructions, however, the verifier knows the elements for which it is verifying (non)membership. In contrast, our goal is to build proofs that can be verified by having only has a succinct commitment to the batch of elements, over which one can also verify additional properties.

Verifiable computation with state. Verifiable computation and zkSNARKs have a vast literature; a complete coverage goes beyond the scope of this paper, yet some seminal papers include [45]–[47]. More relevant to our work are some works that address the problem of verifiable computation (or zkSNARKs) with respect to succinct digests. Pantry [25] use Merkle trees to model RAM computations. Fiore et al. [48] propose hash&prove and accumulate&prove protocols that avoid expensive hash encodings in the circuit, but their solutions require the SNARK prover to do work *linear* in the hashed/accumulated set, which limits their scalability to large sets. The same limitation applies to the efficient commit-and-prove SNARKs [24], [49] as well as to the vSQL scheme of Zhang et al. [50] and TRUESET by Kosba et al. [51]. Also, all these schemes [24], [48], [50] require public parameters linear in the largest set. ADSNARK [52] can generate proofs on authenticated data; this setting is similar to accumulated sets except that inserting data in the set requires a secret authentication key; also [52] achieves succinct proofs only when the verifier knows the secret authentication key.

Accumulators and vector commitments for stateless blockchains. In addition to the already mentioned accumulators from hidden-order groups and Merkle trees,[17] other popular schemes rely on bilinear pairings [54], [55]. Merkle trees actually generalize to vector commitments [56], of which we also know realizations from hidden-order groups and bilinear pairings. Recent works [15], [57]–[60] have extended these two primitives with additional functionalities, including batch proofs, and shown applications to stateless blockchains. In the latter approach, transactions need to be sent and known for verification. Also, their use within zkSNARKs presents the same efficiency challenges—their verification (including elliptic curve operations and pairings) is considerably expensive when compiled into constraints [44], [49]—in addition to the fact that they need public parameters linear in the largest set to be accumulated.

Acknowledgements

This work has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00) and SECURITAS (ref. RED2018-102321-T), by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339), and by a grant from Nomadic Labs and the Tezos foundation. This work was also partly supported by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2021-0-00518, High performance blockchain privacy preserving techniques based on commitment, encryption, and zero-knowledge proofs, 50%, No.2021-0-00532, Blockchain scalability solutions supporting high performance/capacity transactions, 30%, No.2021-0-00727, A Study on Cryptographic Primitives for SNARK, 10%, No.2021-0-00528, Development of Hardware-centric Trusted Computing Base and Standard Protocol for Distributed Secure Data Box, 5% and No.2021-0-00590, Decentralized High Performance Consensus for Large-Scale Blockchains, 5%). Other support was provided by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

[17]We can include under this category existing lattice-based schemes [53].

System	**Number of Constraints** Per-operation cost	Per-proof cost
Merkle-Swap	$2(c_{H_e} + \log\|S\| \cdot c_H)$	—
MultiSwap from [21]	$2(c_{H_e} + c_{H_{in}} + c_{\text{split}} + c_{+\ell}(f) + c_{\times\ell})$	$4c_{e_{\mathbb{G}_?}}(\|\ell\|) + 2c_{\times\mathbb{G}_?} + c_{H_p} + c_{\text{mod}_\ell}(b_{H_{\text{DI}}})$
B-INS-ARiSA-MultiSwap	$2(c_{H_e} + c_{\text{split}} + c_{+\ell}(f) + c_{\times\ell})$	$c_{\text{mod}_\ell}(b_{H_{\text{DI}}})$

λ	security parameter (128)	f	field elements size $\log_2\|\mathbb{F}\|$ (255)
$\|\ell\|$	prime challenge bits (256 in ours, 352 in [21])	c_{split}	strict bit split in $\mathbb{F}$ (388)
$b_{H_{\text{DI}}}$	output size of division-intractable hash H_{DI} (2048)	$c_{+\ell}(b)$	addition mod ℓ of two inputs of b-bits ($16+b$)
c_H	hash $\mathbb{F}^2 \to \mathbb{F}$ (varies)	$c_{\times\ell}$	multiplication mod ℓ (479)
c_{H_e}	set items hashing to $\mathbb{F}$, used in H_{DI} (varies)	$c_{mod_\ell}(b)$	reduction mod ℓ of b-bit input ($16+b$)
$c_{H_{in}}$	per-operation cost of full-input hash in [21] (varies)	$c_{\times\mathbb{G}_?}$	multiplication in $\mathbb{G}_?$
c_{H_p}	prime generation (217703)	$c_{e_{\mathbb{G}_?}}(b)$	exponentiation in $\mathbb{G}_?$ with b-bit exponent

Fig. 6: Constraints count model for Merkle swaps, the MultiSwap of [21], and our MultiSwap of section V-B. The values of the cost parameters are from [21]. The hash functions costs '(varies)' depend on the hash instantiation.

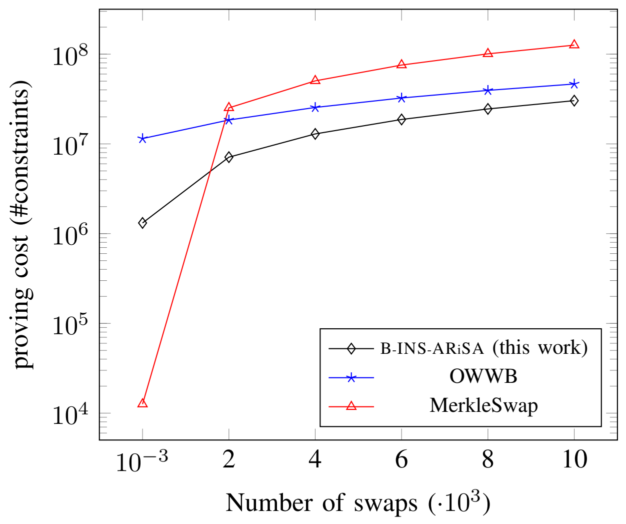


Fig. 7: Proving cost vs. # swaps in MultiSwap. Set size $|S|$ is 2^{20} and y-axis is in log-scale.

References

[1] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 459–474.

[2] "Zcash," https://z.cash.

[3] "Iden3," https://iden3.io.

[4] "Sovrin," https://sovrin.org.

[5] "Hyperledger indy," https://www.hyperledger.org/use/hyperledger-indy.

[6] R. Tamassia, "Authenticated data structures," in *ESA*, 2003.

[7] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO'87*, ser. LNCS, C. Pomerance, Ed., vol. 293. Springer, Heidelberg, Aug. 1988, pp. 369–378.

[8] J. C. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)," in *EUROCRYPT'93*, ser. LNCS, T. Helleseth, Ed., vol. 765. Springer, Heidelberg, May 1994, pp. 274–285.

[9] N. Bari and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *EUROCRYPT'97*, ser. LNCS, W. Fumy, Ed., vol. 1233. Springer, Heidelberg, May 1997, pp. 480–494.

[10] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO 2002*, ser. LNCS, M. Yung, Ed., vol. 2442. Springer, Heidelberg, Aug. 2002, pp. 61–76.

[11] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *ACNS 07*, ser. LNCS, J. Katz and M. Yung, Eds., vol. 4521. Springer, Heidelberg, Jun. 2007, pp. 253–269.

[12] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash," in *CRYPTO'99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, Heidelberg, Aug. 1999, pp. 555–572.

[13] P. Todd, "Making utxo set growth irrelevant with low-latency delayed txo commitments," https://petertodd.org/2016/delayed-txo-commitments, 2016.

[14] J. Drake, "Accumulators, scalability of utxo blockchains, and data availability," https://ethresear.ch/t/accumulators-scalability-of-utxo-blockchains-and-data-availability/176, 2017.

[15] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 561–586.

[16] S. Micali, "CS proofs (extended abstracts)," in *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453.

[17] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 326–349.

[18] barry WhiteHat, "roll_up: Scale ethereum with snarks," https://github.com/barryWhiteHat/roll_up.

[19] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *ASIACRYPT 2016, Part I*, ser. LNCS, J. H. Cheon and T. Takagi, Eds., vol. 10031. Springer, Heidelberg, Dec. 2016, pp. 191–219.

[20] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[21] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators," in *USENIX Security 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 2020, pp. 2075–2092.

[22] D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos, "Zero-knowledge proofs for set membership: Efficient, succinct, modular," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 393–414.

[23] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *34th ACM STOC*. ACM Press, May 2002, pp. 494–503.

[24] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM Press, Nov. 2019, pp. 2075–2092.

[25] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *Proc. of the ACM SOSP*, 2013.

[26] J. Buchmann and S. Hamdy, "A survey on iq cryptography," in *Public-Key Cryptography and Computational Number Theory*. De Gruyter, 2011, pp. 1–16.

[27] R. Cramer, "Modular design of secure yet practical cryptographic protocols," *Ph. D. Thesis, CWI and University of Amsterdam*, 1996.

[28] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Springer, Heidelberg, Aug. 1987, pp. 186–194.

[29] R. Canetti, "Towards realizing random oracles: Hash functions that hide all partial information," in *CRYPTO'97*, ser. LNCS, B. S. Kaliski Jr., Ed., vol. 1294. Springer, Heidelberg, Aug. 1997, pp. 455–469.

[30] B. Wesolowski, "Efficient verifiable delay functions," in *EUROCRYPT 2019, Part III*, ser. LNCS, Y. Ishai and V. Rijmen, Eds., vol. 11478. Springer, Heidelberg, May 2019, pp. 379–407.

[31] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *EUROCRYPT'97*, ser. LNCS, W. Fumy, Ed., vol. 1233. Springer, Heidelberg, May 1997, pp. 256–266.

[32] U. M. Maurer, "Abstract models of computation in cryptography (invited paper)," in *10th IMA International Conference on Cryptography and Coding*, ser. LNCS, N. P. Smart, Ed., vol. 3796. Springer, Heidelberg, Dec. 2005, pp. 1–12.

[33] I. Damgard, C. Hazay, and A. Zottarel, "Short paper on the generic hardness of ddh-ii."

[34] J. Bartusek, F. Ma, and M. Zhandry, "The distinction between fixed and random generators in group-based assumptions," in *CRYPTO 2019, Part II*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11693. Springer, Heidelberg, Aug. 2019, pp. 801–830.

[35] I. Damgård and M. Koprowski, "Generic lower bounds for root extraction and signature schemes in general groups," in *EUROCRYPT 2002*, ser. LNCS, L. R. Knudsen, Ed., vol. 2332. Springer, Heidelberg, Apr. / May 2002, pp. 256–271.

[36] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number is the product of two safe primes," in *EUROCRYPT'99*, ser. LNCS, J. Stern, Ed., vol. 1592. Springer, Heidelberg, May 1999, pp. 107–122.

[37] J.-S. Coron and D. Naccache, "Security analysis of the Gennaro-Halevi-Rabin signature scheme," in *EUROCRYPT 2000*, ser. LNCS, B. Preneel, Ed., vol. 1807. Springer, Heidelberg, May 2000, pp. 91–101.

[38] A. Shamir, "On the generation of cryptographically strong pseudorandom sequences," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 1, pp. 38–44, 1983.

[39] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *ASIACRYPT 2020, Part II*, ser. LNCS, S. Moriai and H. Wang, Eds., vol. 12492. Springer, Heidelberg, Dec. 2020, pp. 3–35.

[40] I. Damgård, "Towards practical public key systems secure against chosen ciphertext attacks," in *CRYPTO'91*, ser. LNCS, J. Feigenbaum, Ed., vol. 576. Springer, Heidelberg, Aug. 1992, pp. 445–456.

[41] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Heidelberg, May 2016, pp. 305–326.

[42] S. Bowe, "Bls12-381: New zk-snark elliptic curve construction," *Zcash Company blog, URL: https://z. cash/blog/new-snark-curve*, 2017.

[43] https://github.com/scipr-lab/libsnark, libsnark.

[44] https://github.com/akosba/jsnark, JSnark.

[45] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *40th ACM STOC*, R. E. Ladner and C. Dwork, Eds. ACM Press, May 2008, pp. 113–122.

[46] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 238–252.

[47] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO 2013, Part II*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8043. Springer, Heidelberg, Aug. 2013, pp. 90–108.

[48] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 1304–1316.

[49] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 253–270.

[50] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 863–880.

[51] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos, "TRUESET: Faster verifiable set computations," in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 765–780.

[52] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 271–286.

[53] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, Heidelberg, May 2013, pp. 353–370.

[54] L. Nguyen, "Accumulators from bilinear pairings and applications," in *CT-RSA 2005*, ser. LNCS, A. Menezes, Ed., vol. 3376. Springer, Heidelberg, Feb. 2005, pp. 275–292.

[55] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC 2009*, ser. LNCS, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Heidelberg, Mar. 2009, pp. 481–500.

[56] D. Catalano and D. Fiore, "Vector commitments and their applications," in *PKC 2013*, ser. LNCS, K. Kurosawa and G. Hanaoka, Eds., vol. 7778. Springer, Heidelberg, Feb. / Mar. 2013, pp. 55–72.

[57] R. W. F. Lai and G. Malavolta, "Subvector commitments with application to succinct arguments," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 530–560.

[58] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, 2018, https://ia.cr/2018/968.

[59] S. Gorbunov, L. Reyzin, H. Wee, and Z. Zhang, "Pointproofs: Aggregating proofs for multiple vector commitments," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 2007–2023.

[60] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich, "Aggregatable subvector commitments for stateless cryptocurrencies," in *SCN 20*, ser. LNCS, C. Galdi and V. Kolesnikov, Eds., vol. 12238. Springer, Heidelberg, Sep. 2020, pp. 45–64.

[61] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[62] T. Attema, R. Cramer, and L. Kohl, "A compressed Σ-protocol theory for lattices," in *CRYPTO 2021, Part II*, ser. LNCS, T. Malkin and C. Peikert, Eds., vol. 12826. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 549–579.

[63] A. Buldas, P. Laud, and H. Lipmaa, "Accountable certificate management using undeniable attestations," in *ACM CCS 2000*, D. Gritzalis, S. Jajodia, and P. Samarati, Eds. ACM Press, Nov. 2000, pp. 9–17.

Appendix

The PoKE protocol [15]

For the ease of presentation we recall the PoKE protocol in fig. 8. Although the interactive version of the protocol is secure with λ-sized challenges its non-interactive version is only secure with 2λ-sized challenges, due to a subexponential attack [61].

Remark 4. *We note that the proof of fig. 8 is not originally secure for arbitrary bases A, but rather for random ones. For arbitrary bases extra care should be taken, that give a proof of additional* 2 *group elements. We wil show that the protocol still suffices for our needs, since we combine it with a SNARK for the relation* $x = res \mod \ell$. *In a nutshell, a* PoKE *for random bases with a SNARK for* $x = res \mod \ell$

give a succinct proof of knowledge of exponent for arbitrary bases.

```
$\underline{\mathsf{Setup}(1^\lambda):}$
    $(\mathbb{G}_?, g_?) \leftarrow \mathcal{G}_?(1^\lambda)$
    $\mathbf{return}\ \mathsf{crs} := (\mathbb{G}_?, g_?)$

$\underline{\mathsf{Prove}\,(\mathsf{crs}, A, B; x):}$
    $\ell \leftarrow H_{\text{prime}}(\mathsf{crs}, A, B)$
    $Q \leftarrow A^{\lfloor \frac{x}{\ell} \rfloor}, res \leftarrow x \mod \ell$
    $\mathbf{return}\ \pi = (Q, res)$

$\underline{\mathsf{Verify}\,(\mathsf{crs}, A, B, \pi):}$
    Parse $\pi$ as $(Q, res)$
    $\ell \leftarrow H_{\text{prime}}(\mathsf{crs}, A, B)$
    Reject if $A, B, Q \notin \mathbb{G}_?$ or $res \notin [0, \ell - 1]$
    Reject if $Q^\ell A^{res} \neq B$
```

Fig. 8: The succinct argument of knowledge PoKE [15]. H_{prime} denotes a cryptographic hash function that outputs a prime of size 2λ, modeled as a random oracle.

Deferred Security proofs

A. Security of the construction of section IV

In fig. 9 we describe an interactive version of our construction.

Theorem 3. *The construction in fig. 9 for the relation $\tilde{R}^{mem}_{\mathsf{ck}}$ is a secure CP-NIZK: succinct, knowledge-sound under the adaptive root assumption and zero-knowledge under the DDH-II assumption.*

Proof. Succinctness: Comes from inspection and from the assumption that $\mathsf{cp}\Pi^{\text{modarithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$ are succinct.

$(2, M)$*-Special Soundness*: assume that we have a tree of $(2, M)$ successful transcripts, for $M = \mathsf{poly}(\lambda) > \left\lceil \frac{\|p^*\| + \|u^*\| + \lambda}{2\lambda} \right\rceil$, i.e.

$$\left\{ \left(\hat{W}_{\vec{u}}, c_{s,r}, R \right), h, \ell^{(j)}, \left(Q^{(i)}, res^{(j)} \right), \pi_2^{(j)}, \pi_3^{(j)} \right\}_{j=1}^{M}$$

and

$$\left\{ \left(\hat{W}_{\vec{u}}, c_{s,r}, R \right), \tilde{h}, \tilde{\ell}'^{(j)}, \left(\tilde{Q}^{(i)}, \tilde{res}^{(j)} \right), \tilde{\pi}_2^{(j)}, \tilde{\pi}_3^{(j)} \right\}_{j=1}^{M}$$

We construct an extractor Ext that works as follows.

Ext uses the extractor of $\mathsf{cp}\Pi^{\text{modarithm}}$ to extract $\vec{u}^{(j)}, s^{(j)}, r^{(j)}$, openings of $c_{\vec{u}}$ and $c_{s,r}$ respectively, such that $res^{(j)} = s^{(j)} h \prod_i u_i^{(j)} + r^{(j)} \mod \ell^{(j)}$. From the binding of the commitments we get that $\vec{u}^{(j)} = \vec{u}^{(j')}, s^{(j)} = s^{(j')}, r^{(j)} = r^{(j')}$ for each transcript, since they refer to the same commitments. So we denote the extracted values as $\vec{u}, s, r$ and get:

$$sh \prod_i u_i + r = res^{(j)} \mod \ell^{(j)}, \text{ for each } j \in [M]$$

Using the Chinese Remainder Theorem we get a k such that

$$k = sh \prod_i u_i + r \mod \left(\prod_{j=1}^{M} \ell^{(j)} \right)$$

M can be set sufficiently large (but still polynomial-sized) so that $\prod_{j=1}^{M} \ell^{(j)} > sh \prod_i u_i + r$ and thus $k = sh \prod_i u_i + r$ over the integers. Furthermore, $k = res^{(j)} \mod \ell^{(j)}$ for each $j \in [M]$.

As shown in [15] the fact that for any accepting proof, (ℓ, Q, res), it holds that $Q^\ell \hat{W}_{\vec{u}}^{res} = \mathsf{acc}^h R$ and $k = res \mod \ell$ (the latter in our case is ensured by the SNARK) then under the adaptive root assumption we get:

$$\hat{W}_{\vec{u}}^{k} = \hat{\mathsf{acc}}^h R$$

(we refer to [15] appendix C.2 for the formal reduction).

Then the extractor does the same for the second set of transcripts to get $\tilde{k}, \tilde{\vec{u}}, \tilde{s}, \tilde{r}$ such that $\hat{W}_{\tilde{\vec{u}}}^{\tilde{k}} = \hat{\mathsf{acc}}^{\tilde{h}} R$ and $\tilde{k} = \tilde{s}\tilde{h} \prod_i \tilde{u}_i + \tilde{r}$ over the integers. Now since $\tilde{\vec{u}}, \tilde{s}, \tilde{r}$ refer to the same commitment as $\vec{u}, s, r$ (recall that the commitment were sent a priori) from the binding of the pedersen commitment we get that $\tilde{\vec{u}} = \vec{u}, \tilde{s} = s, \tilde{r} = r$, which gives us that $\tilde{k} = s\tilde{h} \prod_i u_i + r$.

From the above we have: $\hat{W}_{\vec{u}}^{k} = \hat{\mathsf{acc}}^h R$ and $\hat{W}_{\vec{u}}^{\tilde{k}} = \hat{\mathsf{acc}}^{\tilde{h}} R$. Combining the two we get that

$$\hat{W}_{\vec{u}}^{k - \tilde{k}} = \hat{\mathsf{acc}}^{h - \tilde{h}} \Leftrightarrow$$

$$\hat{W}_{\vec{u}}^{sh \prod_i u_i + r - s\tilde{h} \prod_i u_i - r} = \hat{\mathsf{acc}}^{h - \tilde{h}} \Leftrightarrow$$

$$\hat{W}_{\vec{u}}^{(s \prod_i u_i)(h - \tilde{h})} = \hat{\mathsf{acc}}^{h - \tilde{h}}$$

From the low order assumption (which is implied by the adaptive root assumption) we get $\hat{W}_{\vec{u}}^{s \prod_i u_i} = \hat{\mathsf{acc}}$.

Finally, the extractor runs once the extractor of $\mathsf{cp}\Pi^{\text{bound}}$ to get that $u_i > 2\lambda$.

To conclude the proof, $(2, M)$-special soundness implies knowledge-soundness [62].

Zero-Knowledge: It comes directly from the standard rewinding-simulation Σ-method and the use of the simulators of $\mathsf{cp}\Pi^{\text{modarithm}}$ and $\mathsf{cp}\Pi^{\text{bound}}$. ☐

B. Security of the construction of section V

We give a formal statement of the security of the scheme and then give an overview of the security proof. The proof can be seen as a simplification of the proof of theorem 3.

Theorem 4. *Let H_{prime} a hash-to-prime function modeled as a random oracle, H_{DI} be a division-intractable hash function and $\mathsf{cp}\Pi^{modarithm}$, $\mathsf{cp}\Pi^{H_{DI}}$ be secure CP-SNARKs. The construction in fig. 3 for the relation $\tilde{R}^{ins}_{\mathsf{ck}}$ is a succinct and knowledge-sound, under the adaptive root assumption, CP-SNARK.*

Succinctness: is inherited from the succinctness of $\mathsf{cp}\Pi^{\text{modarithm}}$, $\mathsf{cp}\Pi^{H_{\text{DI}}}$ and PoKE.

Knowledge-Soundness intuition: the extractor proceeds similarly to a PoKE extractor, it rewinds the prover until it gets $M = \mathsf{poly}(\lambda)$ proofs $\{Q^{(i)}, res^{(i)}, \pi_2^{(i)}\}_{i \in [M]}$ and $\ell^{(i)}$ such that each proof verifies, $Q^{(i)\ell^{(i)}} \mathsf{acc}^{res^{(i)}} = \mathsf{acc}'$. For each proof it runs the corresponding extractor $\mathsf{cp}\Pi^{\text{modarithm}}$ and gets a (common) $\vec{u}$ such that $u^* = res^{(i)} \mod \ell^{(i)}$ for each $i \in [M]$. As argued in the proof of theorem 3 for a sufficiently large $M > \|u^*\|/\|\ell\|$ we get that $\mathsf{acc}^{u^*} = \mathsf{acc}'$. This by using the CRT and a reduction to the adaptive root assumption from [15].

To conclude the extraction we additionally need a single π_3 and run the extractor of $\mathsf{cp}\Pi^{H_{\text{DI}}}$ to get that $u_i = H_{\text{DI}}(x_i)$.

Extending our CP-SNARK for batch membership

C. Dealing with sets of arbitrary elements

The scheme described in the section IV works for sets whose elements are suitably large prime numbers. Working with primes can be a limitation in practical applications. Here we describe how to get rid of this limitation and can support sets of arbitrary elements, such as binary strings. The idea is common in previous work and is to use a suitable collision-resistant hash function that maps arbitrary strings to prime numbers. What is a bit more complicated in our setting is that in order to prove membership of an arbitrary element, we need to prove the mapping to a prime.

Thanks to the commit-and-prove modularity of our protocol we can do this extension easily. This is the same idea used in [22]. Say that the prover holds a commitment $\hat{c}$ to a vector of binary strings $(\hat{u}_1, \ldots, \hat{u}_m)$. To prove the mapping the prover creates a commitment c to the primes $(u_1, \ldots, u_m)$ such that $u_i = H_{\text{prime}}(\hat{u}_i)$, runs our CP-SNARK with c and adds a proof $\pi_{H_{\text{prime}}}$ showing that $c, \hat{c}$ commit to elements such that $\forall i : u_i = H_{\text{prime}}(\hat{u}_i)$. The latter proof can be generated via a CP-SNARK for this hashing relation. In particular, although a computation of H_{prime} involves several computations of a collision resistant hash function until reaching a prime, for the sake of proving one can use nondeterminism and prove a single hash evaluation (see [22] for details).

D. Succinct batch proofs of non-membership

We observe that by using a CP-SNARK for batch membership it is also possible to prove batch non-membership, if one accumulates sets using an interval-based encoding. The idea is that the elements of the set $S = \{x_i\}_i$ are ordered, $x_1 < x_2 < \cdots < x_n$, and the accumulator actually contains hashes of consecutive pairs $u_i = H(x_{i-1}, x_i)$. This way, proving that $x \notin S$ translates into proving that there is an element $u_j = H(x_{j-1}, x_j)$ in the accumulator such that $x_{j-1} < x < x_j$. The idea of interval-based non-membership proofs was introduced by Buldas et al. in the context of Merkle trees [63]. The drawback of this method is that it needs some coordination in order to add the data to the accumulator in this structured form and has an additional cost to the prover to prove the range. As an alternative, we can also extend our technique of this section to generate succinct zero-knowledge batch membership proofs for generic RSA accumulators. The idea is to build a randomization method for the batch non-membership witnesses of [11], [15] and add a similar combination of a sigma protocol for its verification together with a PoKE proof. We leave the formalization of this extension to the full version of the paper.

```
$\underline{\mathsf{Setup}\left(1^{\lambda}, \mathsf{ck}, \mathsf{pp}_{\mathsf{Acc}}\right):}$
    $\mathsf{crs}_2 \leftarrow \mathsf{cp\Pi}^{\text{arithm}}.\mathsf{Setup}(1^{\lambda}, \mathsf{ck}, \tilde{R}^{\text{arithm}}_{\mathsf{ck}})$
    $\mathsf{crs}_3 \leftarrow \mathsf{cp\Pi}^{\text{bound}}.\mathsf{Setup}(1^{\lambda}, \mathsf{ck}, R^{\text{bound}})$
    $\mathbf{return}\ \mathsf{crs} := (\mathsf{ck}, \mathsf{pp}_{\mathsf{Acc}}, \mathsf{crs}_2, \mathsf{crs}_3)$

$\underline{P\,(\mathsf{crs}, \hat{\mathsf{acc}}, c_{\vec{u}}; W_{\vec{u}}, \vec{u}, o_{\vec{u}}, S)}$                                   $\underline{V\,(\mathsf{crs}, \hat{\mathsf{acc}}, c_{\vec{u}})}$

  $\pi_3 \leftarrow \mathsf{cp\Pi}^{\text{bound}}.\mathsf{Prv}(\mathsf{crs}_3, c_{\vec{u}}, p_{2\lambda}; \vec{u}, o_{\vec{u}})$
  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 $r \leftarrow\$ \{0,1\}^{\|p^*\|+\|u^*\|+\lambda}$
  $c_{s,r} \leftarrow \mathsf{Comm}_{\mathsf{ck}}(s, r; o_{s,r})$
  $R \leftarrow \hat{W}_{\vec{u}}^{r}$

                    $\xrightarrow{\pi_3, \hat{W}_{\vec{u}}, c_{s,r}, R}$

                                                  $h \leftarrow\$ \{0,1\}^{\lambda}$

                    $\xleftarrow{h}$

  $k \leftarrow r + (s \prod_i u_i)h$

                                                  $\ell \leftarrow\$ \mathbb{P}_{2^{2\lambda}}$

                    $\xleftarrow{\ell}$

  $Q \leftarrow A^{\lfloor \frac{k}{\ell} \rfloor}, res \leftarrow x \mod \ell$
  $\pi_2 \leftarrow \mathsf{cp\Pi}^{\text{modarithm}}.\mathsf{Prv}(\mathsf{crs}_2, \boxed{c_{\vec{u}}}, \boxed{c_{s,r}}, h, \ell, res)$

                    $\xrightarrow{(Q, res), \pi_2}$

                                                  Accept if all are true:
                                                  • $Q \in \mathbb{G}_{?}$ and $res \in [0, \ell - 1]$
                                                  • $Q^{\ell} \hat{W}_{\vec{u}}^{res} = \hat{acc}^{h} R$ and
                                                  • $\mathsf{cp\Pi}^{\text{modarithm}}.\mathsf{Vfy}(\mathsf{crs}_2, c_{\vec{u}}, c_{s,r}, h, \ell, res, \pi_2) = 1$
                                                  • $\mathsf{cp\Pi}^{\text{bound}}.\mathsf{Vfy}(\mathsf{crs}_{\text{bound}}, c_{\vec{u}}, p_{2\lambda}, \pi_3) = 1$
```

Fig. 9: Interactive version of our protocol for batch membership.