DIV: Resolving the Dynamic Issues of Zero-knowledge Set Membership Proof in the Blockchain

Zihuan Xu
The Hong Kong University of Science and Technology
Hong Kong SAR, China
zxuav@cse.ust.hk

Lei Chen
The Hong Kong University of Science and Technology
Hong Kong SAR, China
leichen@cse.ust.hk

ABSTRACT

Zero-knowledge set membership (ZKSM) proof is widely used in blockchain to enable private membership attestation. However, existing mechanisms do not fully consider dynamic issues in the blockchain scenario. Particularly, frequent addition/removal of set elements, not only brings the significant cost to keep public parameters up to date to provers and verifiers but also affects mechanism efficiency (*e.g.*, generation time of the proof and verification, etc.).

In this paper, we propose DIV to shard elements on the blockchain into independent subsets with the same cardinality to reduce the effect of dynamic issues. However, due to the diverse proof frequency, an improper element-set assignment can result in frequently used elements being easily inferred and corrupted. Thus, we formalize the assignment problem under both element addition and removal cases as two optimization problems and prove their NP-hardness. For each problem, we consider two cases if each element proof frequency is known in advance by the set maintainer or not, and propose solutions with theoretical guarantees. We implement DIV on both Merkle tree and RSA-based ZKSM mechanisms to evaluate its efficiency and effectiveness and apply DIV on a ZKSM-based application named zkSync to demonstrate its applicability. Results show that DIV can achieve $O(1)$ time/space cost on ZKSM under dynamic situations while protecting the information about frequently used elements. It also notably reduces the system latency of zkSync.

CCS CONCEPTS

• **Security and privacy** → *Pseudonymity, anonymity and untraceability*; • **Computer systems organization** → Maintainability and maintenance.

KEYWORDS

blockchain; zero-knowledge set membership proof

ACM Reference Format:
Zihuan Xu and Lei Chen. 2021. DIV: Resolving the Dynamic Issues of Zero-knowledge Set Membership Proof in the Blockchain. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3457248

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '21, June 20–25, 2021, Virtual Event, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457248

1 INTRODUCTION

Blockchain privacy issues are highlighted in many researches [27, 57]. Among the privacy-preserving techniques, zero-knowledge set membership (ZKSM) proof [43] is used to prove that an element belongs to a public set without disclosing any information of the element itself. It guarantees the privacy for membership proof which is the heart of many blockchain applications (*e.g.*, anonymous cryptocurrency [51], credential [19] and identity [23] systems).

Basically, ZKSM works as follows [43]: firstly, a set maintainer collects elements and commits them in a public set. Then, a prover generates element membership proofs based on public set parameters. Finally, a verifier verifies the proofs. Besides, blockchain is perfectly compatible with ZKSM due to its immutability and Byzantine fault-tolerant (BFT) features. In particular, blockchain helps to better maintain the public parameters of ZKSM for both provers and verifiers to work on an untrusted environment. However, when adopting ZKSM in blockchain scenarios, existing mechanisms do not fully consider the dynamic issues summarized as follows:

Firstly, it takes time to maintain the set when elements are dynamically changed (*i.e.*, update, addition, and removal). For example, to update an element in a *Merkle tree* [39], a new branch is recomputed to update the root which is also the public parameter for ZKSM. Besides, the maintenance cost usually depends on the number n of set elements. Specifically, the update time complexity of Merkle tree is $O(\log n)$ and RSA-accumulator takes $O(1)$ for addition but quasilinear of n for others operations [12, 45]. Moreover, in the blockchain, the delay to make new public parameters on-chain, called *on-chain delay*, becomes a bottleneck. In particular, it is costly for nondeterministic consensus protocols (*e.g.*, PoW [44]) to finalize the changes (*e.g.*, tens or hundreds of seconds in Ethereum and Bitcoin [53]). Meanwhile, to update the set, some applications (*e.g.*, stateless blockchain [2, 15, 38]) even require a computational heavy *zero-knowledge* correctness proof for privacy and security. Thus, due to the on-chain delay, sometimes, ZKSM can be invalid since the public parameters may already be out of date.

Secondly, the dynamics also affect the ZKSM efficiency such as the proof size, public parameter size, and the time to generate and verify a proof. For instance, Merkle tree-based ZKSM mechanism is widely used in blockchain applications (*e.g.*, ZCash [51]). With n elements, both the space complexity of its *proof size* and the time complexity of the *proof generation* are $O(\log n)$. When continuously adding elements to a set, ZKSM will be more costly in terms of time and space which is not desirable for applications (*e.g.*, credential and cryptocurrency system) where constant cost is required.

An easy fix to enable current mechanisms to handle the dynamic issue is that instead of maintaining a unique set for all elements, we can assign them into subsets of constant size to make the ZKSM efficiency of each set independent of each other. However, it brings the third challenge that imbalanced element-set assignment may expose which elements are frequently used, thus providing additional information to the adversary. Moreover, by combining this knowledge with other side information (*e.g.*, an entity's real-world trading rate), it increases the possibility for attackers to discover the link between proofs and elements [30] which obeys the principle of "zero-knowledge".

In this paper, we propose a mechanism named DIV to address the above three challenges. Specifically, we define two optimization problems for set element addition and removal, respectively to minimize the possible information leak brought by the set division aiming to improve the efficiency of ZKSM under the dynamic updates. For each problem, we consider whether the element proof frequency is the algorithm input or not, to meet different requirements (*e.g.*, provers may want their estimated element proof frequency to be even hidden from the set maintainer). We propose an approximation algorithm with the theoretical bound to each case separately. Finally, we conduct extensive experiments on both real and synthetic datasets to show the efficiency and effectiveness of our proposed techniques, DIV.

In summary, we have made the following contributions:

1) We propose DIV to scale the zero-knowledge set membership proof in the blockchain scenario.

2) We define two optimization problems to obtain a balanced element-set assignment exposing the minimum information of the elements' proof frequency in two dynamic cases and prove their NP-hardness.

3) For each problem, we design algorithms with a constant approximation bound for both cases that element proof frequency is known to or hidden from the set maintainer.

4) We implement DIV on both Merkle tree and RSA based ZKSM mechanisms as well as a ZKSM-based application named zkSync and conduct extensive experiments. Results verify that DIV can resolve the dynamic issues and be applicable on a real application to reduce the latency brought by ZKSM. Meanwhile, the makespan of each set usage frequency is bounded.

The rest of this paper is organized as follows: Sec. 2 describes the background and challenges of DIV. Sec. 3 formally defines our optimization problems. We propose approximation algorithms in Sec. 4 and Sec. 5, respectively. Experimental results are shown in Sec. 6. We discuss related works in Sec. 7 and conclude in Sec. 8.

2 BACKGROUND AND OVERVIEW

2.1 Zero-knowledge Set Membership Proof

Proving membership (given an element e and a set S to prove $e \in S$) plays an important role in blockchain applications (*e.g.*, currency transfer, identity system, etc.). While, cryptographic accumulators [12] provide the efficient solution. Typically, they are classified into: *Merkle Tree-based* [39], *pairing-based* [21, 26, 45, 58] and *RSA-based* [8, 14, 22] mechanisms. For privacy requirements, one may even want to prove $e \in S$ without disclosing e. Such property is called *zero-knowledge proof* (ZKP) [32]. Nowadays, to enable succinct ZKP for not only membership, but also other general properties (*e.g.*, correctness of digital signatures [51]), the technique called SNARK [13] is proposed. A well-known example is zkSNARKs [10] derived from GGPR [31] via circuit satisfiability building blocks [11, 31, 33, 47][1] where the proving problem is converted to a satisfiable problem of a *rank-1 constraint system* (R1CS) and a pair of proving and verification keys are used to generate and verify the proof.

Specifically, generalized ZKSM process in a blockchain consists of three roles [43]: the set maintainer, prover, and verifier. As shown in Fig. 1, a set maintainer first setups a set S^0 and records the public parameters (*e.g.*, Merkle root) on the blockchain through transactions. For different applications, the underlying set elements can be public (*e.g.*, public compliance) or hidden (*e.g.*, the asset) by using the cryptography commitment technique (*e.g.*, Pedersen commitment [48]). Then, a prover uses the proving key to generate the ZKP of $e_1 \in S^0$. Finally, a verifier uses the verification key to verify the proof and provide services based on the result. Besides, the maintainer can update the set through transactions.

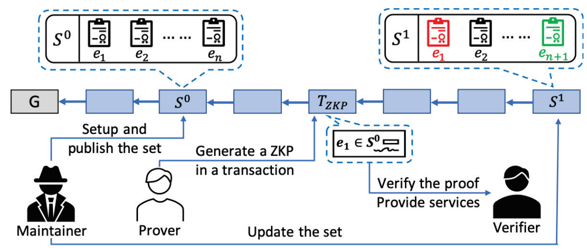


Figure 1: General process of ZKSM with blockchain

2.2 Applications and Dynamic Issues

Motivation Example. zkSync [37] is a well-known layer-2 (L2) application aiming to scale the Ethereum. Fig. 2 shows its basic architecture where a L1 on-chain smart contract handles the state update and asset migration between L1 and L2. The idea is to reduce L1 on-chain transactions by local processing on the L2. Specifically, each L2 asset state is recorded in a Merkle-tree with the upper-layer (24 layers by default) account tree and the lower-layer (8 layers by default) balance tree. The state digest (Merkle root) is recorded in the L1 smart contract. In running time, L2 block proposers roll up a batch of L2 transactions in one block and execute them to update states and their digest. Then block committers generate validity ZKPs for the new block and state digest including ZKSM (checking if sender/receiver accounts and the updated asset states are valid members of the digest) and other ZKPs (*e.g.*, signature check of each sender). Thus, a L1 smart contract can periodically update the state digest and verify its correctness by ZKPs which is faster than processing on-chain transactions.

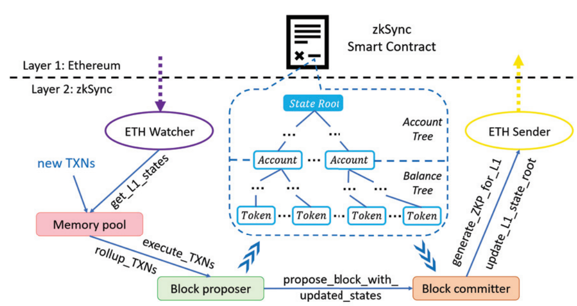


Figure 2: Basic architecture of zkSync

[1] This work is also compatible with other SNARK frameworks, *e.g.*, CS Proofs [40], Zaatar [52], Aurora [9], Ligero [6], Bulletproofs [16], etc.

Although, batch-based off-chain processing improves the system throughput, the latency (the duration from a transaction being proposed to its block ZKP is committed) becomes a new concern since the ZKP is time-costly (*e.g.*, a few minutes to generate a single ZKSM for a 64-layer Merkle tree [51]). In terms of ZKSM, zkSync states are compressed in the Merkle tree whose maintainers are block proposers and provers are block committers proving the membership correctness for L1 smart contract executors (verifiers). Beside the high generation time cost, the ZKSM also suffers from the following dynamic issues.

Dynamic issues. In practice, the set element will not remain static. For example, in zkSync, each L2 transaction will lead to the update of states in the digest, while the asset transform between L1 and L2 will cause set member (account) creation and removal. Such dynamics can bring the following two issues:

One is the set maintenance cost. In particular, as discussed in Sec. 2.1, existing ZKSM solutions convert the proving problem to a R1CS by pre-computing constraints in a setup process. However, for Merkle-based ZKSM, the proving problem will be different if the set size changes where re-setup is needed. Besides, existing elements count can affect the set update cost (*e.g.*, $O(\log n)$ for Merkle tree) which means the more the elements, the more cost needed to update a set. Most importantly, the on-chain delay (finalize update on-chain) may arise in blockchain, makes it possible to generate invalid proofs when updating the set. Specifically, to complete a block updating the state digest in zkSync, a ZKP from L2 and the consensus of the change in L1 are required. Before the finalization, every proof based on the old digest is invalid, which may cause the collision problem and introduce a race condition to generate proofs in parallel [1], severely decreasing the system performance.

The other is the efficiency issue of ZKSM proof generation and verification. For Merkle tree, the constraint system is to ensure the prover knows a valid branch which can produce the public-known root. Besides, for the tree with more elements, the branch will be longer. Therefore, the ZKSM constraint count and proof generation time/space cost are proportional to the tree height [51]. ZKSM will be more costly and the transaction processing latency will be longer in zkSync. Although, the cost of RSA-accumulator can be independent from the inside elements, it outperforms Merkle tree only when the set size is large (*e.g.*, 128-bit RSA outperforms Merkle tree when the set size is larger than 2^{20}). Moreover, additional element map function is needed for RSA-accumulator which makes it less practical than Merkle tree.

To avoid frequently re-setup, existing solutions maintain a *universal set* for all possible elements (*e.g.*, zkSync uses one set digest for all account states and ZCash uses a 2^{64} Merkle tree for supported coins). However, it restricts the system to have more elements than the universal set. Besides, regardless of the actual element count, the ZKSM always inefficiently proves membership of the universal set (*e.g.*, causing high transaction processing latency in zkSync).

Other Applications. ZKSM is also widely used in other blockchain Apps. We only list part of them. More details can be found in [43].

Stateless Blockchain. This concept [54] is proposed to scale the blockchain where zkSync is a typical implementation. Other examples are Rollup [2], Coda [38], and Zexe [15] where transactions are processed locally with a succinct correctness proof and only a short states commitment is recorded on-chain. Meanwhile, ZKSM is used to check if a given state is a valid member of the latest digest.

Anonymous Credentials. Since ZKSM can ensure compliance with privacy guarantee, a typical use case is the anonymous credential (*e.g.*, Idemix [20] in Hyperledger [18]) where the service provider (verifier) requires the customer (prover) to provide a ZKSM proving the membership of an accepted or legitimate identities set (*a.k.a.* KYC check) issued by a certificate authority (set maintainer).

Reputation Validation. ZKSM can also check any auditable qualification privately. For example, [43] describes a reputation validation application using ZKSM to check if a company is a member of the high reputation set. Such applications often utilize blockchain to record the qualification change based on participants' behavior .

2.3 Challenges of DIV

Intuitively, we can divide the universal set with all possible elements into subsets with the same cardinality. As a consequence, for set maintenance, addition only creates new subsets and removal/update only affects subsets containing related elements. Meanwhile, the ZKSM cost can remain constant under element addition.

Notice that, an element can be assigned to multiple sets and there is no difference to declare its membership in any subset. Thus, we assume a prover will declare an element's membership in each assigned set with equal frequency (*e.g.*, in Fig. 1, if e_2 is assigned to sets S^0 and S^1, the prover will generate ZKSM of $e_2 \in S^0$ and $e_2 \in S^1$ in the same frequency). Because, without knowing the proof strategy of other provers, compared to declaring an element's membership from a single set, the equal declaration in multiple sets mixes the proof with more possible elements which decreases the proof-element linkability.

Although an adversary still cannot obtain any information from the ZKSM after set division, one can perform other attacks. For example, [51] describes a network partition attack to block a specific node from the latest public parameters to deny services. Based on the blockchain adversary's behaviors described in [50], an adversary with limited resources will perform attacks to frequently used elements to maximize their profits. Especially, an unbalanced element-set assignment can leak the importance information of elements. In particular, for each ZKSM proof, one needs to at least specify which set it is extracted from. Due to the difference in each element's *proof frequency*, it will make the *usage frequency* of each subset different as well. As a consequence, an adversary can perform attacks (*e.g.*, network partition) on those important elements. Therefore, our main challenge is to get the optimal element-set assignment to hide the importance of each element the best. Then, we provide an example to illustrate the problem.

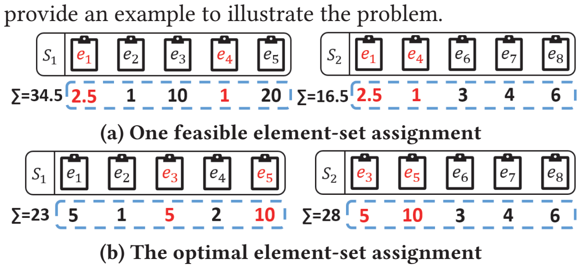


Figure 3: Two feasible element-set assignments

Example 1. *Suppose we have 8 elements (e_1 to e_8) with proof frequency* $\{5, 1, 10, 2, 20, 3, 4, 6\}$ *respectively and we assign them to* *two subsets with the cardinality of* 5*. Fig. 3a shows one feasible assignment. The number below each element is its proof frequency contributing to the subset. For instance, as* e_1 *is assigned to both sets, its proof frequency in both set is* 2.5*. The usage frequencies of two sets are* 34.5 *and* 16.5 *respectively. However, this is not the optimal assignment, since the usage frequencies indicate that elements in* S_1 *are more frequently used than those in* S_2*. While, Fig. 3b gives the optimal assignment where the usage of each set is relatively balanced which gives the adversary less information about the elements.*

In this paper, for both batch-based element addition and removal dynamic situations, we define an optimization problem respectively and provide solutions to the set maintainers to obtain the balanced element-set assignment before they set up and publish the sets.

3 PROBLEM DEFINITION

In this section, we formally define our problems named as the *Set Membership Proof Optimization (SMPO)* problems in both the cases of adding and removing a batch of elements.

3.1 Basic Concepts

In this paper, we consider a blockchain system where some functionalities depend on the element set membership proof (*i.e.*, a valid ZKSM proof of elements is required to activate some functions) and the public parameters of these sets are recorded on-chain.

Definition 3.1. **Element and Set.** We use $\{e_1, e_2 ..., e_n\}$ to denote elements and use p_i to denote the frequency for provers to do the membership proof on e_i. Meanwhile, S_i denotes a subset of elements with cardinality k where k is a system defined parameter to control the efficiency and security of ZKSM.

Notice that, we do NOT assume the element proof frequencies remain static. Instead, we suppose it is an estimated result during a period. To divide the elements with the same property into subsets, the element-set assignment is essential which is defined as:

Definition 3.2. **Element-set Assignment.** For each element e_i, we use a function $\mathcal{A}(e_i)$ to represent the collection of sets that e_i is assigned to where $\mathcal{A}(e_i) = \{S_j | e_i \in S_j\}$.

To assign an element to multiple sets can give a prover more choices to declare the membership which increases the element-proof unlinkability. Extremely, given n elements and set cardinality k, there are $\binom{n}{k} \propto n^k$ feasible sets that can decrease the element-proof linkability to the minimum. However, it is impractical to generate all feasible sets. Thus, we use a parameter δ_s to control the ratio of the set count to the element count.

Definition 3.3. **Set Number Ratio.** We denote δ_s as the ratio of the expected number of sets to the number of elements.

For instance, n elements need to be assigned to $\lceil \delta_s n \rceil$ sets. When choosing the values of k and δ_s, we need to make sure $k\delta_s \geq 1$. Because the maximum capacity of the sets is $k\lceil \delta_s n \rceil$ which should be able to accommodate at least one copy of each element.

Recall that we assume a rational prover will declare an element's membership in each assigned set with equal frequency. Meanwhile, with many valid sets, a proof should specify which set it applies on and this information can be obtained by anyone who can see the proof. Thus, we define the set usage frequency as:

Definition 3.4. **Set Usage Frequency.** The usage frequency of a set S_i is denoted by u_i, where $u_i = \sum_{e_j \in S_i} \frac{p_j}{|\mathcal{A}(e_j)|}$.

Due to the variance of elements' proof frequency, the set usage frequency varies from element-set assignments. Especially, as the set usage frequency is easily obtained by the adversary, when some sets are much more frequently used, the importance information of elements in the set is leaked. Thus, with limited resources, adversaries can perform dedicated attacks (*e.g.*, consensus delay [29], wallet theft [7], etc.) to "important" elements and their owners.

Information entropy of set usage frequency. To measure the aforementioned information leak of frequently used elements, in this paper, we use Shannon entropy [34] which is widely used in measuring the information that an adversary can obtain [17]. We define it as $H(U) = -\sum_{\forall S_i} \frac{u_i}{\sum_{\forall S_i} u_i} log \frac{u_i}{\sum_{\forall S_i} u_i}$. In addition, the higher the entropy is, the less information an adversary can get [34], and for $H(U)$, the more balance of each set's usage frequency, the higher the entropy will be. Thus, our goal is to balance the set usage frequency of all generated sets (minimize the makespan which is also equivalent to minimize the maximum set usage frequency [56]) during the maintenance of the sets and we define specific problems in two dynamic scenarios in next subsection.

3.2 Adding and Removing Elements Problems

The first dynamic case is that with some existing sets (could be empty for the first batch), a batch of new elements need to be added.

Definition 3.5. **Set Membership Proof Optimization with Adding Elements (SMPO-A) Problem.** Given a set of existing elements E which have been assigned to a collection of sets S with the set usage frequency u_i for each, a set of new elements E^* to be added, the set number ratio δ_s, and the set cardinality constraint k, the goal of SMPO-A is to assign $E \bigcup E^*$ into new sets S^* where $|S^*| = \lceil \delta_s |E \bigcup E^*| \rceil - |S| \leq \lceil \delta_s |E^*| \rceil$ to **minimize the maximum set usage frequency** $u_{max} = \max_{S_i \in S \bigcup S^*} u_i$, subject to the following constraints:

- **Cardinality Constraint.** After the assignment, the cardinality of each set is less than or equal to k.
- **Completeness Constraint.** Each element in E^* must be assigned once to make them become available.
- **Unbiased Constraint.** Each element in E can only be assigned at most once to the sets in S^* to avoid bias.

Hardness Proof. We prove the NP-hardness of SMPO-A problem by reducing it from the *load-balancing problem* [56] which is a well-known NP-hard problem. Due to space limit, we refer the details of proof in the appendix [5].

THEOREM 3.6. *The SMPO-A problem is NP-hard.*

Notice that, we do not require the cardinality of each set to be exactly k. Because when the set usage frequency of a none-full set is already the maximum, filling it with additional elements makes the result deviate from the optimal. However, the different number of elements makes the proof-element unlinkability of each set different. Thus, when a set is not full, the maintainer can generate some dummy elements with 0 proof frequency to fill the set.

Another dynamic case is that when a batch of elements are removed from the currently valid element sets which is defined as:

Definition 3.7. **Set Membership Proof Optimization with Removing Elements (SMPO-R) Problem.** Given a set of elements

E which have been assigned to a collection of sets S with the set usage frequency u_i for each, a set of elements E^d ($E^d \subset E$) to be removed and we denote S^d as the sets containing elements in E^d ($\forall S_i \in S^d, \exists e$ *s.t.* $e \in S_i, e \in E^d$), the set number ratio δ_s, and the set cardinality constraint k, the goal of SMPO-R is to reassign elements in $E^r = \{e | e \in S^d, e \notin E^d\}$ into new sets S^r where $|S^r| = |S^d| - (|S| - \lceil \delta_s |E - E^d| \rceil)$ to **minimize the maximum set usage frequency** $u_{max} = \max_{S_i \in (S - S^d) \bigcup S^r} u_i$, subject to the following constraints:

- **Cardinality Constraint.** After the assignment, the cardinality of each set is less than or equal to k.
- **Completeness Constraint.** $\forall e \in E^r$ we denote $\lambda(e) = |\{S' | S' \in S^d, S' \in \mathcal{A}(e)\}|$, and e must be assigned $\mathcal{I}(e) = \lceil \frac{\lambda(e) * |S^r|}{|S^d|} \rceil$ times to preserve their original distribution proportion in the sets.
- **Unbiased Constraint.** Each element in $E - E^d - E^r$ can only be assigned at most once to the sets in S^r to avoid bias.

Since $|S^d| \leq \lceil \delta_s(|E^d \bigcup E^r|) \rceil$, we have $|S^r| \leq \lceil \delta_s(|E^d \bigcup E^r|) \rceil - \delta_s|E| + \lceil \delta_s |E - E^d| \rceil \leq \lceil \delta_s |E^r| \rceil$. Meanwhile, for the completeness constraint, it is different from the one in the SMPO-A problem. Because, during multiple batches of assignments, different existing sets may contain duplication of the same element. When E^d are removed from S^d, the instances of the remaining elements may also have duplication. Therefore, this constraint is to preserve the original proportion of each element. Otherwise, if we only keep one instance of these elements, their provers will have a less set choice when doing the ZKSM proof. It will cause the usage of other sets containing these elements to increase dramatically.

Hardness Proof. By using similar reduction to the SMPO-A problem, SMPO-R problem can also be proved as NP-hard. For more details, please refer to the appendix [5].

Theorem 3.8. *The SMPO-R problem is NP-hard.*

Table 1 shows the main notations used in this paper.

3.3 Two Situations of Each Problem

For both SMPO-A and SMPO-R problems, we consider two situations whether the parameter proof frequency of each element is treated as the input of the designed algorithm or not.

Frequency is the Input. In some applications, especially in the *permissioned chain*, one may choose to count on the set maintainer and provide the estimated element proof frequency to get better services. For example, in a digital credential system, the set maintainer is also the authority who can issue or revoke credentials (elements). With our mechanism, the maintainer needs to take responsibility for not only credential issuing and revoking, but also the element-set assignment to maintain the credentials. Therefore, in this setting, a set maintainer is considered to have two properties: 1) execute the assignment algorithms properly and 2) keep the element proof frequency private. Therefore, under these requirements, for both SMPO-A and SMPO-R problems, we treat the proof frequency of each element as the algorithm input.

Frequency is NOT the Input. On the other hand, especially in the *permissionless chain*, sometimes the set maintainer is not the authority who can create or destroy the elements. For instance, the miners in ZCash need to maintain the latest coin commitment list but cannot create or destroy any commitment arbitrarily and they can be potential adversaries. Thus, in this setting, we do not assume the maintainer to behave honestly. In particular, provers will not expose their proof frequency which will not be the input of the assignment algorithm either. Instead, we only assume the proof frequency of each element follows *independent and identically distribution* (*i.i.d.*). Meanwhile, an additional mechanism is required to audit if the set maintainer executes the algorithm correctly.

Table 1: Summary of major notations

Symbol	Description
E	A set of existing elements that have been assigned to on-chain sets
p_i	The proof frequency of an element e_i
S	A set of on-chain sets containing existing elements
u_i	The usage frequency of the set S_i
δ_s	The ratio of the expected number of sets to the number of elements
k	The cardinality constraint of each set
E^*	A set of new elements to be added and published on-chain
S^*	A set of new sets to contain E^*
E^r	Remaining elements in affected sets after the removal of elements
S^r	A set of new sets to contain E^r
$\mathcal{I}(e_i)$	Number of instances that need reassignment of the element e_i
$\mathcal{A}(e_i)$	A set of sets that e_i has been assigned to

4 ELEMENT ADDITION OPTIMIZATION

In this section, we propose approximation solutions to the SMPO-A problem for both cases that the element proof frequency is known to or hidden by the set maintainer.

There are two challenges to solve the SMPO-A problem. The first is the cardinality constraint which makes each set can only contain up to k elements. The second is that we need to consider two types of elements to achieve our goal. One is the new element in E^* which has not been assigned to any set yet. To assign such an element e_i to a new set $s_j \in S^*$, it makes the usage u_j of s_j increase by p_i. The other is the element in E that has been assigned to some existing sets in S. These elements can be duplicated and assigned to the set in S^* as well. Recall that, once we duplicate an existing element, according to Def. 3.4, the usage frequency of each set containing this element will change. Specifically, to duplicate and assign such an element e_i to a new set s_j, it makes the usage u_j increase by $\frac{p_i}{|\mathcal{A}(e_i)|+1}$, meanwhile, $\forall S_k \in \mathcal{A}(e_i)$, the usage of each of them will decrease by $\frac{p_i}{|\mathcal{A}(e_i)|} - \frac{p_i}{|\mathcal{A}(e_i)|+1}$ as well.

4.1 Proof Frequency is the Input

We propose an algorithm named *highest frequency for adding (HF-A)* algorithm where the element proof frequency is the input.

Basic Idea. The basic idea is to divide the algorithm into two phases to deal with the two types of elements separately. In the first phase, we assign E^* to S^*. Each time, we pick the element with the highest proof frequency to the new set with the minimum current usage and containing less than k elements. In the second phase, we keep on adjusting the sets in S which currently has the maximum usage and duplicating elements inside it to assign to the new sets.

Algorithm Details. As shown in Algo. 1, each time we find the element in E^* with the highest proof frequency and assign it to the set in S^* with the minimum usage (lines 3-6). After phase 1, only when the set S_{max} with the maximum current usage is in S

and there is extra space in S^* and assignable elements in S_{max}, we enter the second phase. Because it means we can further reduce the makespan by duplicating and assigning an element in S_{max} to the set in S^*. We pick an assignable $e_i \in S_{max}$ which can produce the maximum usage decrement computed by $\frac{p_i}{|A(e_i)|} - \frac{p_i}{|A(e_i)|+1}$ and $S_{min} \in S^*$ with the minimum usage and less than k elements. Next we check if to duplicate and assign e_i can reduce the maximum usage (line 13). If not, since S_{min} already has the minimum usage, it means there is no place to assign e_i such that the maximum usage can be reduced. Thus, we mark e_i as unassignable. Otherwise, we duplicate and assign e_i to S_{min} (lines 14-16).

Algorithm 1: HF-A Algorithm

```
   Input  : Elements E and E* with each proof frequency p_i, Sets S with each
            usage u_j, current assignment 𝒜, cardinality k and ratio δ_s.
   Output : A feasible assignment 𝒜.
1  create empty sets S* s.t. |S*| = ⌈δ_s |E ⋃ E*|⌉ − |S| ;
2  /* Phase 1: assign elements in E* to S*                          */
3  while |E*| > 0 do
4      find e_i ∈ E* with the maximum p_i;
5      find s ∈ S* with the minimum usage s.t. |s| < k and let 𝒜(e_i) = {s};
6      remove e_i from E* and update the usage of s;
7  /* Phase 2: utilize the left space in S*                         */
8  space = k * |S*| − |E*|;
9  find S_max in S ⋃ S* with the maximum usage;
10 while space > 0, S_max ∈ S and S_max has assignable elements do
11     find assignable e_i ∈ S_max with the maximum p_i/|𝒜(e_i)| − p_i/(|𝒜(e_i)|+1);
12     find S_j ∈ S* with the minimum usage s.t. |S_j| < k;
13     if u_j + p_i/(|𝒜(e_i)|+1) < u_max then
14         𝒜(e_i) ⋃ {S_j}, update u_k : ∀S_k ∈ 𝒜(e_i) and space − = 1;
15     mark e_i as unassignable;
16     find S_max in S ⋃ S* with the maximum usage;
```

Time Complexity Analysis. In Algo. 1, phase 1 has $O(|E^*|)$ loops. Each loop takes $O(log|E^*| + log|S^*|)$ to assign each element. Since $|S^*| = \lceil \delta_s |E \bigcup E^*| \rceil - |S|$, $|S| = \lceil \delta_s |E| \rceil$ and $\delta_s \leq 1$, the first phase takes $O(|E^*| log(|E^*|))$. In the second phase, it has at most $O(k\delta_s|E^*|)$ loops. It takes $O(log(\delta_s(|E| + |E^*|)))$ to find the S_{max} and $O(logk + log(\delta_s|E^*|))$ to assign e_i to S_{min}. Thus, the second step takes no more than $O(k\delta_s|E^*|(logk + log(\delta_s(|E| + |E^*|)))$. Overall, the time complexity of Algo. 1 is $O(k\delta_s|E^*|(logk + log(\delta_s(|E| + |E^*|)))$.

Approximation Analysis. We first analyze the lower bound of the SMPO-A problem.

Lemma 4.1. *The lower bound of SMPO-A problem is* $\max\{L_0 = \frac{\sum_{e_i \in E^*} p_i}{|S^*|}, L_1 = \max_{\forall e_i \in E^*} p_i, L_2 = \frac{\max_{\forall S_j \in S}\{u_j\}}{2}, L_3 = \frac{\sum_{e_i \in E \bigcup E^*} p_i}{|S \bigcup S^*|}\}$.

Proof. L_0 and L_1 consider the assignment of E^*. Since we need to assign every element in E^* to S^* once, the optimal set usage cannot be less than either L_0, which is to evenly distribute the proof frequency of elements in E^* to sets in $|S^*|$, or L_1, which is the maximum proof frequency among the E^*. L_2 and L_3 consider the assignment of E. Since we cannot reassign them to S in any batch, the usages of sets in S will be non-increasing. According to Def. 3.4, to duplicate and assign an $e_i \in E$ to S^*, the usage decrement of each set containing e_i in S is computed by $\frac{p_i}{|A(e_i)|} - \frac{p_i}{|A(e_i)|+1} = \frac{p_i}{|A(e_i)|(|A(e_i)|+1)}$. Since $|A(e_i)| \geq 1$, the maximum usage decrement of an element e_i is $\frac{p_i}{2}$. As we restrict the elements in E can only be assigned once in each batch, $\forall S_j \in S$ its usage cannot be reduced to lower than $\frac{u_j}{2}$. Thus, L_2 holds. Due to the same reason of L_0, we apply it on $E \bigcup E^*$ and $S \bigcup S^*$ to make L_3 hold. □

Theorem 4.2. *The approximation ratio of Algo. 1 is 2.*

Proof. We denote OPT as the result of the optimal solution. For Algo. 1, S_{max} is the set producing the final maximum usage u_{max}. The algorithm has the following two possible running states:
State 1: $S_{max} \in S^*$ after phase 1 (stop after phase 1). If it never occurs, an element cannot be assigned to a set due to the cardinality constraint. After assigning the last element e_n to S_{max}, we have $u_{max} - p_n \leq u_j(\forall S_j \in S^*) \leq L_0$. Thus, $u_{max} \leq L_0 + L_1 \leq 2OPT$. Else, we denote e_1 as the first element that cannot be assigned to S_a ($|S_a| = k$) with the minimum current usage u_a (since $\forall S_j \in S^* : u_a \leq u_j \Rightarrow u_a \leq OPT$). When e_1 is considered, we denote $\hat{u}_{max}$ as the current usage of S_{max} and e_2 as the current last element that has been assigned to S_{max}. Since the elements are sorted, we have $p_1 \leq p_2$. Also, $\hat{u}_{max} - p_2 \leq u_a$, otherwise e_2 is supposed to be assigned to S_a. When e_2 is assigned, either S_a already has some elements or not. In the former case, we have $p_2 + (k-1)p_1 \leq u_a \Rightarrow u_{max} \leq \hat{u}_{max} + (k-1)p_1 = (\hat{u}_{max} - p_2) + (p_2 + (k-1)p_1) \leq 2u_a \leq 2OPT$. In the later case, e_2 will be the first element assigned to S_{max}. Otherwise, it should be assigned to S_a since S_a is empty. Thus, we have $u_{max} \leq p_2 + (k-1)p_1 \leq L_1 + u_a \leq 2OPT$. Hence, Algo. 1 can achieve 2-approximation in State 1.
State 2: $S_{max} \in S$ after phase 1. We denote $S_a \in S$ as the second last S_{max}. Then, the last step that can affect the result of this state must be assigning an element $e_i \in S_a$ to the final S_{max}. There are two conditions. One is the final $S_{max} \in S^*$. According to line 13 of Algo. 1, we have $u_{max} \leq u_a \leq \max_{\forall S_j \in S}\{u_j\} \leq 2L_2 \leq 2OPT$. The other is the final $S_{max} \in S$, since we do not assign elements into sets in S, we also have $u_{max} \leq \max_{\forall S_j \in S}\{u_j\} \leq 2L_2 \leq 2OPT$. Hence, Algo. 1 can achieve 2-approximation in State 2 as well.

Overall, the approximation ratio of Algo. 1 is 2. □

Avoid frequency detection from the assignment. In the algorithm, the higher an element's proof frequency is, the more copies of the element will be generated. It also gives the adversary the information we are trying to hide. However, when a copy of an element is demanded, the set maintainer (also the creator of the elements) can generate a different commitment to the same element by changing the random value in the commitment function. It makes an adversary cannot distinguish the elements in the assignment.

4.2 Proof Frequency is Unknown

Since the main purpose of blockchain is to reduce the trusted third-party, we also propose a randomized algorithm named *RD-A* to address the case of SMPO-A problem where the proof frequency of each element is hidden.

Basic Idea. RD-A also contains two phases. In the first phase, we randomly assign elements in E^* to S^* and make sure each set in S^* contains balanced number of elements (the largest element count difference is 1). In the second phase, we find two sets S_{max} with the maximum (in S) and S_{min} with the minimum (in S^*) current usage. Then randomly pick an assignable element from S_{max} following a specific probability and duplicate and assign it to S_{min}.

Algorithm Details. Algo. 2 shows the algorithm details. In phase 1, we first randomly assign the new elements to each set of S^* in a round-robin fashion (lines 3-5). Importantly, each time we assign an element to a set, we add the usage of that set by twice of the expectation proof frequency calculated by using the existing set usages. We enter the second phase based on the same condition as Algo. 1. After finding S_{max} we obtain the picking probability of each element inside it and randomly pick an assignable e_i (line 10). Note that, the e_j in the probability calculation can only be the assignable element. We treat p_i as its expectation value and check if to assign e_i to S_{min} can reduce the current maximum usage. If so, we assign the element and update the related set usage by treating p_i as its expectation value as well (lines 13-18).

Algorithm 2: RD-A Algorithm

Input : Elements E and E^*, Sets S with each usage u_j, current assignment $\mathcal{A}$, cardinality k and ratio δ_s.
Output : A feasible assignment $\mathcal{A}$.

1 create empty sets S^* *s.t.* $|S^*| = \lceil \delta_s |E \bigcup E^*| \rceil - |S|$;
2 /* **Phase 1: assign elements in E^* to S^*** */
3 **for** $i = 0;\ i < |E^*|;\ i{+}{+}$ **do**
4 $\quad S_j = S^*[i\ mod\ |S^*|]$ and $\mathcal{A}(E^*[i]) = \{S_j\}$;
5 $\quad u_j \mathrel{+}= 2 * \frac{\sum_{\forall S_i \in S} u_i}{|E|}$;
6 /* **Phase 2: utilize the left space in S^*** */
7 $space = k * |S^*| - |E^*|$;
8 find S_{max} in $S \bigcup S^*$ with the maximum usage;
9 **while** $space > 0$, $S_{max} \in S$ *and* S_{max} *has assignable elements* **do**
10 $\quad$ Randomly pick one assignable $e_i \in S_{max}$, with the probability of $\frac{1}{|\mathcal{A}(e_i)| \sum_{e_j \in S_{max}} \frac{1}{|\mathcal{A}(e_j)|}}$ for each;
11 $\quad$ find $S_j \in S^*$ with the minimum usage *s.t.* $|S_j| < k$;
12 $\quad$ **if** $u_j + \frac{\sum_{\forall S_i \in S} u_i}{|E| * (|\mathcal{A}(e_i)| + 1)} < u_{max}$ **then**
13 $\quad\quad$ **foreach** $S_k \in \mathcal{A}(e_i)$ **do**
14 $\quad\quad\quad u_k \mathrel{-}= \frac{\sum_{\forall S_i \in S} u_i}{|E| * |\mathcal{A}(e_i)| * (|\mathcal{A}(e_i)| + 1)}$;
15 $\quad\quad u_j \mathrel{+}= \frac{\sum_{\forall S_i \in S} u_i}{|E| * (|\mathcal{A}(e_i)| + 1)}$;
16 $\quad\quad \mathcal{A}(e_i) \bigcup \{S_j\}$ and $space -= 1$;
17 $\quad$ mark e_i as unassignable;
18 $\quad$ find S_{max} in $S \bigcup S^*$ with the maximum usage;

Time Complexity Analysis. In Algo. 2, phase 1 takes $O(|E^*|)$ and for phase 2 there is at most $O(k\delta_s|E^*|)$ loops. In each loop, it takes $O(log(\delta_s(|E| + |E^*|)))$ to find the S_{max} and $O(k + log\delta_s|E^*|)$ to assign an element. Therefore, the total complexity of Algo. 2 is $O(k\delta_s|E^*|(k + log\delta_s(|E| + |E^*|)))$.
Approximation Analysis. Since the Algo. 2 is a randomized algorithm, we analyze its expected approximation ratio. We first prove a lemma to help our analysis. Let x_i be an *i.i.d.* random variable with finite mean μ and variance σ^2. Let $S_i(m) = \sum_{\forall x_j \in S_i} x_j$ be the summation of x_j in a set S_i with m members and let $M_n(m) = \max\{S_1(m), \ldots, S_n(m)\}$ as the maximum value of n sets:

Lemma 4.3. *$\forall \alpha > 1, \epsilon > 0$, if $m \geq n^{\frac{1}{\alpha}}$, $\mathbb{E}(M_n(m)) \leq \mu m + \epsilon m^{\frac{\alpha}{2}}$.*

Proof. Since all x_i is *i.i.d.*, for all $S_i(m)$, it has mean μm and standard deviation $\sigma\sqrt{m}$. For $\mathbb{P}(M_n(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}}) = \mathbb{P}(\exists i, S_i(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}})$ which equals to compute: $\mathbb{P}(\bigcup_{i=1}^{n}\{S_i(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}}\}) \leq n\mathbb{P}(S_i(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}})$ (Boole's inequality [42]). According to the Chebyshev's inequality [42] $n\mathbb{P}(S_i(m) - \mu m \geq \sigma(\frac{\epsilon}{\sigma} m^{\frac{\alpha}{2}})) \leq \frac{\sigma^2}{\epsilon^2}\frac{n}{m^\alpha}$. When $m \geq n^{\frac{1}{\alpha}}$, $\frac{\sigma^2}{\epsilon^2}\frac{n}{m^\alpha} \leq \frac{\sigma^2}{\epsilon^2}$. With the Borel-Cantelli lemma [24], $\sum_{i=1}^{\infty}\mathbb{P}(S_i(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}}) < \infty$. It means, with probability one, the event $\{M_n(m) \geq \mu m + \epsilon m^{\frac{\alpha}{2}}\}$ only occurs for finite times which completes the proof. □

Theorem 4.4. *The expected approximation ratio of Algo. 2 is 2 when $\delta_s \leq \frac{1}{\sqrt[3]{|E^*|}}$ and $\sqrt[3]{|E^*|}$ when $\frac{1}{\sqrt[3]{|E^*|}} < \delta_s \leq 1$.*

Proof. Similar to Theorem 4.2, we also analysis two states:
State 1: $S_{max} \in S^*$ after phase 1 (stop after phase 1). In this case, after phase 1, there are $\lceil \delta_s |E^*| \rceil$ sets where each contains $\lceil \frac{1}{\delta_s} \rceil$ elements on average. According to Lemma 4.3, let $n = \lceil \delta_s |E^*| \rceil$, $m = \lceil \frac{1}{\delta_s} \rceil$, $\epsilon = \mu$, $\alpha = 2$, we have: when $\delta_s \leq \frac{1}{\sqrt[3]{|E^*|}}$, $\mathbb{E}(u_{max}) = \mathbb{E}(\max_{\forall S_i \in S^*} S_i(\frac{1}{\delta_s})) \leq \frac{2\mu}{\delta_s} = \frac{2\sum_{\forall S_i \in S^*} u_i}{\delta_s |E^*|} = 2L_0 \leq 2OPT$. When $\frac{1}{\sqrt[3]{|E^*|}} < \delta_s \leq 1$, $\mathbb{E}(u_{max}) \leq \frac{\max_{\forall e_i \in E^*} p_i}{\delta_s} \leq \frac{L_1}{\delta_s} \leq \sqrt[3]{|E^*|}OPT$.
State 2: $S_{max} \in S$ after phase 1. When entering phase 2, due to line 5 in Algo. 2, regardless of the value of δ_s, we must have $\forall S_i \in S^* : \mathbb{E}(u_i) \leq 2OPT$. Also we always have $\forall S_j \in S : \mathbb{E}(u_j) \leq 2L_2 = 2OPT$. We denote $S_a \in S$ as the second last S_{max}. The last step in phase 2 must be assigning an element $e_i \in S_a$ to the final S_{max}. According to line 12 in Algo. 2 we have $\mathbb{E}(u_{max}) \leq \mathbb{E}(u_a) \leq 2L_2 \leq 2OPT$.

Overall, the expected approximation ratio of Algo. 2 is 2 when $\delta_s \leq \frac{1}{\sqrt[3]{|E^*|}}$ and $\sqrt[3]{|E^*|}$ when $\frac{1}{\sqrt[3]{|E^*|}} < \delta_s \leq 1$. □

Verifiable randomness function. As discussed in Sec. 3.3, when the maintainers potentially being malicious, one should be able to verify if our algorithm is executed correctly. We use Verifiable randomness function (VRF) [28, 41] to audit the execution. Specifically, VRF maps its input to verifiable pseudorandom outputs. Each maintainer has a key pair <*sk, pk*> where *sk* is used to generate the output $VRF_{sk}(X)$ on input X and pk is for others to verify the output. Maintainer uses the output as the random seed to execute the randomized algorithm and others can audit if it is executed correctly. Meanwhile, the input of VRF in each assignment is the output of VRF in the last assignment.

5 ELEMENT REMOVAL OPTIMIZATION

In this section, we discuss the approximation solution to the SMPO-R problem. Similarly, we give two solutions to different situations when the proof frequency is given or not.

The main difference between SMPO-A and SMPO-R problems is that in the former, elements in E^* are never assigned and can only be assigned once in a batch of assignment. However, in the later, set E^r contains the remaining elements in S^d after the removal of E^d. Thus, as discussed in Sec. 3.2, elements in E^r may have multiple instances. Meanwhile, we need to make sure instances of the same element cannot be assigned to the same set twice. Thus, the hardness of SMPO-R problem is to handle the duplicated instances in E^r. We skip the process to remove $e \in E^d$ from the current assignment and directly give the algorithm to reassign instances in E^r.

5.1 Proof Frequency is the Input

We first provide the deterministic algorithm named *highest frequency for removing (HF-R)* for SMPO-R problem.
Basic Idea. Basically, we still divide the algorithm into two steps. We first assign E^r to S^r by sorting the element instances by proof frequency and assign the highest one to the set with minimum current usage. Meanwhile, we make sure any two instances of the

same element are not assigned to the same set. Then, we consider how to utilize the extra space of S^r.

Algorithm 3: HF-R Algorithm

```
   Input  : Elements E and E^r with each proof frequency p_i and instance
            count I(e_i), Sets S − S^d with each usage u_j, current assignment 𝒜,
            cardinality k and ratio δ_s.
   Output : A feasible assignment 𝒜.
1  create empty sets S^r;
2  /* Phase 1: assign elements in E^r to S^r */
3  while ∃ unassigned instance of an element in E^r do
4      find e ∈ E^r with the maximum p_i / (|𝒜(e)| + I(e));
5      find s ∈ S^r with the minimum usage s.t. |s| < k and e ∉ s;
6      𝒜(e) ⋃ {s}, remove an e instance and update the usage of s;
7  mark all instances of elements in E^r as unassignable;
8  space = k * |S^r| − Σ_{∀e_i ∈ E^r} I(e_i);
9  find S_max in (S − S^d) ⋃ S^r with the maximum usage;
10 while space > 0, S_max ∈ (S − S^d) and S_max has assignable elements do
11     /* Similar phase 2 to HF-A algorithm */ ... ...
```

Algorithm Details. Algo. 3 shows the details of the algorithm. We first assign the instances of elements in E^r (lines 3-6). In particular, the proof frequency of each instance of element e_i is computed as $\frac{p_i}{|\mathcal{A}(e_i)|+\mathcal{I}(e_i)}$ where $|\mathcal{A}(e_i)|$ denotes the assignment count of e_i in the sets of $S - S^d$ and $\mathcal{I}(e_i)$ denotes the number of instances to be assigned in S^r. After assigning one instance, we mark it as unassignable. The second phase is similar to Algo. 1. The difference is that to compute the remaining space, we need to take all instances of elements in E^r into consideration. Notice that, after phase 1, all elements in E^r are marked as unassignable (line 7). It means their instances in the sets of $S - S^d$ are not considered in phase 2 either.
Time Complexity Analysis. The analysis of Algo. 3 is similar to Algo. 1, since they have the same main body. Thus, the complexity of Algo. 3 is $O(k\delta_s|E^r|(logk + log(\delta_s|E|))$.
Approximation Analysis. We can also derive the lower bound of the SMPO-R problem by using the same idea as Lemma 4.1.

Lemma 5.1. *The lower bound of SMPO-R problem is* $\max\{L_0^r = \frac{\sum_{e_i \in E^r} \frac{p_i \mathcal{I}(e_i)}{|\mathcal{A}(e_i)|+\mathcal{I}(e_i)}}{|S^r|}, L_1^r = \max_{\forall e_i \in E^r} \frac{p_i}{|\mathcal{A}(e_i)|+\mathcal{I}(e_i)}, L_2^r = \frac{\max_{\forall S_j \in (S-S^d)} \{u_j\}}{2},$ $L_3^r = \frac{\sum_{e_i \in E-E^d} p_i}{|(S-S^d) \bigcup S^r|}\}$.

Theorem 5.2. *The approximation ratio of Algo. 3 is 2.*

Proof. We denote OPT, u_{max} as the optimal solution and final result of Algo. 3 respectively and use the same notations as Algo. 3.
State 1: $S_{max} \in S^r$ after phase 1 (stop after phase 1). If it never occurs that an instance cannot be assigned to a set due to any constraint, after assigning the last instance of element e_n to S_{max}, we have $u_{max} - \frac{p_n}{|\mathcal{A}(e_n)|+\mathcal{I}(e_n)} \leq L_0^r$. Thus, $u_{max} \leq L_0^r + L_1^r \leq 2OPT$. Otherwise, we have the following two cases:

Case 1: Each time, an e_i instance cannot be assigned to S_j only because S_j already contains an e_i instance. We denote S_a with usage u_a as the set which finally has the minimum usage. Without loss of generality, we consider the last assigned element e_n as the one that can effect the final result. We have $u_a \leq OPT$. If S_a does not contain an instance of e_n, we have $u_{max} - \frac{p_n}{|\mathcal{A}(e_n)|+\mathcal{I}(e_n)} \leq u_a \Rightarrow u_{max} \leq u_a + L_1^r \leq 2OPT$. Else, we know S_a contains some instances of elements in S_{max} that makes e_n cannot be assigned to S_a. Thus, we have $u_{max} - u_a \leq \max_{\forall e_i \in E^r} \frac{p_i}{|\mathcal{A}(e_i)|+\mathcal{I}(e_i)}$. Otherwise, there must be some elements in S_{max} that should be assigned to S_a. Therefore, $u_{max} \leq 2OPT$ as well.

Case 2: At one stage, an instance of e_j firstly cannot be assigned to S_a due to $|S_a| = k$. Also we have $u_a \leq OPT$. In this stage, we denote $\hat{u}_{max}$ as the usage of S_{max} and $e_{j'}$ as the last element assigned to S_{max}. According to Algo. 3, we have $\frac{p_j}{|\mathcal{A}(e_j)|+\mathcal{I}(e_j)} \leq \frac{p_{j'}}{|\mathcal{A}(e_{j'})|+\mathcal{I}(e_{j'})}$. When $e_{j'}$ is assigned to S_{max}, similar to Case 1, we have $\hat{u}_{max} - u_a \leq \frac{p_{j'}}{|\mathcal{A}(e_{j'})|+\mathcal{I}(e_{j'})}$ (at most S_a already contains an instance of $e_{j'}$). Otherwise, $e_{j'}$ should be assigned to S_a. Also, we have $\frac{p_{j'}}{|\mathcal{A}(e_{j'})|+\mathcal{I}(e_{j'})} + (k-1)\frac{p_j}{|\mathcal{A}(e_j)|+\mathcal{I}(e_j)} \leq u_a$. Therefore, $u_{max} \leq \hat{u}_{max} + (k-1)\frac{p_j}{|\mathcal{A}(e_j)|+\mathcal{I}(e_j)} = (\hat{u}_{max} - \frac{p_{j'}}{|\mathcal{A}(e_{j'})|+\mathcal{I}(e_{j'})}) + (\frac{p_{j'}}{|\mathcal{A}(e_{j'})|+\mathcal{I}(e_{j'})} + (k-1)\frac{p_j}{|\mathcal{A}(e_j)|+\mathcal{I}(e_j)}) \leq 2u_a \leq 2OPT$.
State 2: $S_{max} \notin S^r$ after phase 1. Similar to the proof of State 2 in Theorem 4.2, we can use L_2^r to get $u_{max} \leq 2OPT$.

Thus, the approximation ratio of Algo. 3 is 2. □

5.2 Proof Frequency is Unknown

With unknown proof frequency, we also propose a randomized algorithm named *RD-R* to address the situation.

Algorithm 4: RD-R Algorithm

```
   Input  : Elements E and E^r with each instance count I(e_i), Sets S − S^d with
            each usage u_j, current assignment 𝒜, cardinality k and ratio δ_s.
   Output : A feasible assignment 𝒜.
1  create empty sets S^r and let index = 0;
2  /* Phase 1: assign elements in E^r to S^r */
3  for i = 0; i < |E^r|; i++ do
4      for j = 0; j < I(E^r[i]); j++ do
5          S_k = S^r[(index + j) mod |S^r|];
6          u_k += 2 * (Σ_{∀S_l ∈ S} u_l) / (|E| * (|𝒜(E^r[i])| + I(E^r[i])));
7          𝒜(E^r[i]) ⋃ {S_k};
8      index += I(E^r[i]);
9  mark all instances of elements in E^r as unassignable;
10 space = k * |S^r| − Σ_{∀e_i ∈ E^r} I(e_i);
11 find S_max in (S − S^d) ⋃ S^r with the maximum usage;
12 while space > 0, S_max ∈ (S − S^d) and S_max has assignable elements do
13     /* Similar phase 2 to RD-A algorithm */ ... ...
```

Basic Idea. Similar to RD-A algorithm, in the first phase of RD-R, we also evenly assign each element instance to each set in S^r. Differently, each time we randomly select one element and assign all of its instances to different sets. In this case, the randomization can make sure none of the instances of the same element is assigned to the same set. For phase 2, the same process of Algo. 2 is used.
Algorithm Details. As shown in Algo. 4, the major difference between RD-R and RD-A algorithm is in the first phase (lines 3-8). In RD-R, we first generate a random permutation on the elements in E^r. Each time, we pick an element with unassigned instances and assign each of its instance to a new set (lines 5-7). The main aim of phase 1 is also to assign every instance evenly to each set. For the phase 2, it is the same as the RD-A algorithm.
Time Complexity Analysis. According to the complexity analysis of Algo. 2 algorithm, we can directly obtain the complexity of RD-R algorithm as $O(k\delta_s|E^r|(k + log(\delta_s|E|)))$.

Approximation Analysis. The approximation analysis of Algo. 2 shares similar ideas of Theorem 4.4. Due to space limit, we only show the result. For more details please refer to the appendix [5].

THEOREM 5.3. *The expected approximation ratio of Algo. 4 is 2 when* $\delta_s \leq \frac{1}{\sqrt[3]{|E^r|}}$ *and* $\sqrt[3]{|E^r|}$ *when* $\frac{1}{\sqrt[3]{|E^r|}} < \delta_s \leq 1$.

6 EXPERIMENTAL EVALUATIONS

In this section, we report our evaluation study on DIV, specifically on answering the following questions:
1. The effectiveness of our approximation algorithms. (Sec. 6.1)
2. How DIV can help to improve the efficiency of ZKSM? (Sec. 6.2)
3. How does the DIV perform in a real application? (Sec. 6.3)
Experiment environment. All experiments are conducted on the machine with 8 Intel(R) Xeon(R) E5-2667 v3 @ 3.20GHz CPUs and 256G RAM running Ubuntu 18.04. For nondeterministic results (*e.g.*, randomized algorithms and execution time estimation), we conduct each experiment 30 times and report the average result.

6.1 DIV Algorithm Effectiveness Measurement

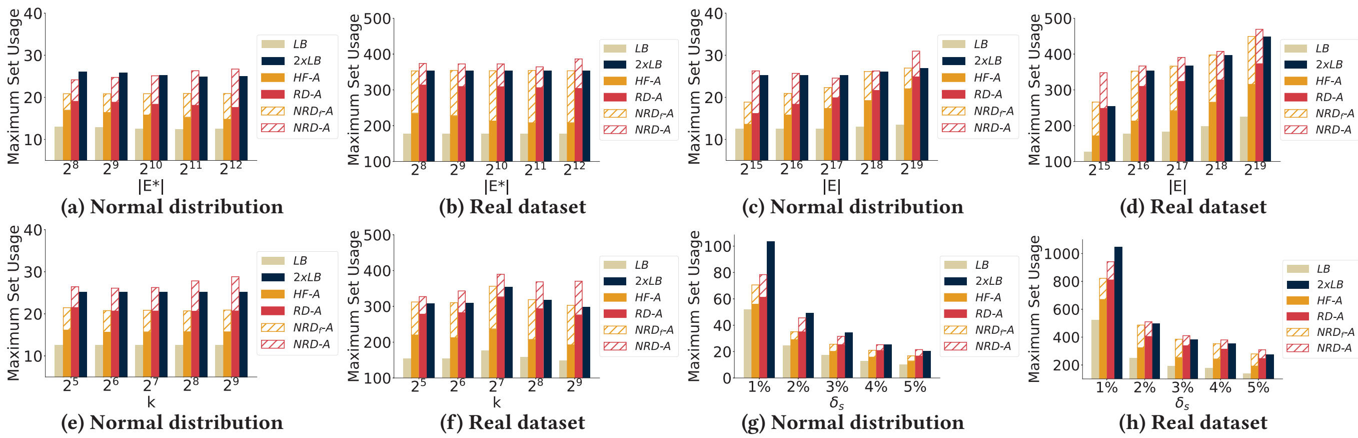
Table 2: Experiment parameters settings

Parameters	Settings
New elements count $\lvert E^*\rvert$ (for **SMPO-A only**)	$2^8, 2^9, \mathbf{2^{10}}, 2^{11}, 2^{12}$
Existing elements count $\lvert E\rvert$	$2^{15}, \mathbf{2^{16}}, 2^{17}, 2^{18}, 2^{19}$
Cardinality of each set k	$2^5, 2^6, \mathbf{2^7}, 2^8, 2^9$
Set number ratio δ_s	1%, 2%, 3%, **4%**, 5%
Affected sets percentage ρ (for **SMPO-R only**)	2%, 4%, **6%**, 8%, 10%
σ of normal distribution	0.05, 0.1, **0.15**, 0.2, 0.25
a of power law distribution	1, 2, **3**, 4, 5

Real Dataset. To obtain the element proof frequency in a real application, we use the Ethereum dataset. Specifically, we extract transactions from the Ethereum blocks from #6M to #11M (generated from Jul-20-2018 to Oct-06-2020) containing 40,900,640 succeed transactions. Since transactions update the states of involved addresses, in the ZK scenario (*e.g.*, zkSync mentioned in Sec. 2.2), a ZKSM is required for each address state update. Therefore, we can use the frequency of each Ethereum address being involved in the transactions (*e.g.*, the ERC-20 token sender or receiver) , as the element proof frequency of a ZK set (*e.g.*, state digest in zkSync). We observe that 99% proof frequencies are in the range of $[1, 100]$, and 90% are less than 50 which approximates power-law distribution.
Synthetic Dataset. To capture more cases, we generate the synthetic elements with proof frequency in different distributions. Specifically, an element is represented by a random 256-bits hash value and we generate different number of elements based on the setting in Table 2. For each element, we randomly generate its proof frequency in the range of $(0, 1)$ based on **uniform**, **normal** (with $\mu = 0.5$ and very σ) and **power law** (vary a) distributions.
Effectiveness Metrics. We measure our algorithms' effectiveness by the produced maximum set usage frequency. We also implement naive randomized algorithms (NRD) as the baselines which first randomly assign elements in E^*/E^r to S^*/S^r and then randomly pick and assign elements in $S/S - S^d$ to S^*/S^r to produce feasible solutions. Specifically, for SMPO-A problem, the NRD_f*-A* algorithm works with a known frequency. Thus, during the random process, one can check if the maximum usage appears in S^* to stop the algorithm. Meanwhile *NRD-A* works with an unknown frequency where the stop condition is when there is no space left in S^*. Similarly, for SMPO-R problem we have NRD_f*-R* and *NRD-R* for the cases of known and unknown frequency, respectively. We vary the parameters in Table 2 where the default values are in bold and compare our algorithms with the theoretical bound and NRD baselines. In each experiment, we first construct the existing element-set assignment based on E, k and δ_s and randomly assign elements in E to sets in S. The reason we use random assignment is to simulate the unpredictable dynamically change of the proof frequency of each element. Then, for the SMPO-A problem, we use *HF-A*, NRD_f*-A*, *RD-A* and *NRD-A* algorithms respectively to add elements in E^* in one batch. For SMPO-R problem, we randomly remove a batch of elements (E^d in the problem) based on a given affected percentage of existing sets (to obtain $|S^d|$ in the problem) and use *HF-R*, NRD_f*-R*, *RD-R* and *NRD-R* algorithms respectively to do the reassignment.

The algorithms are implemented with GNU C++. Due to the space limit, we omit the experimental results of uniform distribution which is similar to the normal distribution, power-law distribution which is approximated by the real dataset as well as varying σ and a in normal and power-law distributions, since they do not have distinct influence on the compared algorithms.

6.1.1 Experimental Results of SMPO-A Problem. Experiments show that *HF-A* and *RD-A* always have lower set usage (at least $0.18\times$ in synthetic data and $0.15\times$ in real data) than NRD_f*-A* and *NRD-A*. Because, if the elements that affect the final result the most cannot be correctly picked, the algorithm effectiveness is low as well. Besides, there is no obvious trend of the baseline algorithms' performance when varying the parameters in Table 2. Due to space limit, we only analyze the influence of parameters on our algorithms.
Impact of the new elements count $|E^*|$. As shown in Fig. 4a and Fig. 4b, with the $|E^*|$ increases, the results of both *HF-A* (1.3× to 1.18× in synthetic data and 1.33× to 1.18× in real data higher than *LB*) and *RD-A* (1.46× to 1.41× in synthetic data and 1.77× to 1.72× in real data higher than *LB*) are closer to the theoretical lower bound. The reason is that with fixed δ_s, larger $|E^*|$ means more new sets providing more space for duplicating the existing elements to make the overall usage balanced. Meanwhile, the performance of *HF-A* is always better than *RD-A* and their ratio ($\frac{RD-A}{HF-A}$) becomes higher (1.12× to 1.19× in synthetic data and 1.33× to 1.46× in real data). Because in phase 2 of *RD-A*, each element's proof frequency is treated as the expectation of all elements, which makes the element with lower proof frequency not considered to be assigned to the set, thus, reducing the extra space utilization rate.
Impact of the existing elements count $|E|$. In Fig. 4c and Fig. 4d, as the $|E|$ increases, there is no obvious trend of the comparison ratio between our algorithms (*HF-A* and *RD-A*) and the *LB*. It means, under the same system settings (*e.g.*, same k and δ_s), our algorithms' performance will not be affected by the number of existing elements during the long-term usage. Meanwhile, the performance of both *HF-A* (1.08× to 1.63× in synthetic data and 1.2× to 1.4× in real data higher than *LB*) and *RD-A* (1.28× to 1.84× in synthetic data and 1.65× to 1.95× in real data higher than *LB*) can always be bounded within 2*LB*, which supports our theoretical approximation ratio.
Impact of the set cardinality k. As shown in Fig. 4e and Fig. 4f, when k changes from 2^5 to 2^9 the result of *HF-A* (1.28× to 1.24×



(a) Normal distribution (b) Real dataset (c) Normal distribution (d) Real dataset

(e) Normal distribution (f) Real dataset (g) Normal distribution (h) Real dataset

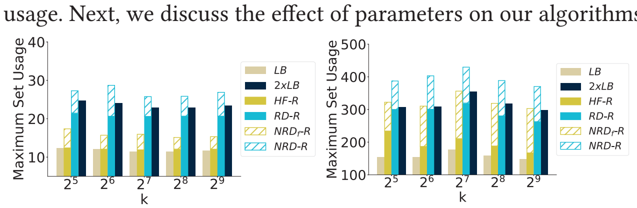
Figure 4: Results of varying $|E^*|$,$|E|$,k and δ_s in SMPO-A problem

in synthetic data and 1.43× to 1.29× in real data higher than *LB*) is closer to the lower bound. Because with larger k the possibility that an element cannot be assigned to the set with the minimum current usage decrease as well. However, the result of *RD-A* (1.63× to 1.7× in synthetic data and 1.81× to 1.85× in real data higher than *LB*) is further from the lower bound, due to the lower probability to pick the correct reassigned elements from a lager set.

Impact of the set number ratio δ_s. As shown in Fig. 4g and Fig. 4h, with larger δ_s, in both datasets, the lower bound usage decreases. Because there are more sets to assign the elements. Meanwhile, the results of both *HF-A* (1.08× to 1.27× in synthetic data and 1.28× to 1.4× in real data higher than *LB*) and *RD-A* (1.18× to 1.64× in synthetic data and 1.55× to 1.79× in real data higher than *LB*) are further from the theoretical lower bound. Because larger δ_s makes both $|S|$ and $|S^*|$ larger as well. Meanwhile, since we choose $|E^*| = 2^{10} \ll |E| = 2^{16}$, when δ_s increases, more space is provided for $|E|$ which makes the theoretical lower bound decrease faster than the enhancement of two algorithms.

6.1.2 *Experimental Results of SMPO-R Problem.*

Similar to the analysis in the SMPO-A problem, *HF-R* and *RD-R* can always outperform NRD_f*-R* and *NRD-R*. Specifically, our algorithms can achieve at least 0.2× in synthetic data and 0.23× in real data lower maximum usage. Next, we discuss the effect of parameters on our algorithms.

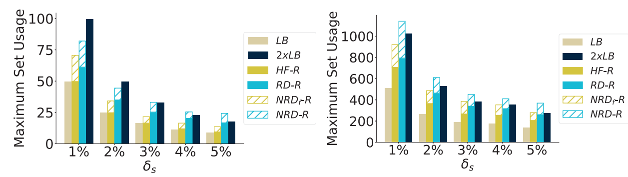


(a) Normal distribution (b) Real dataset

Figure 5: Results of varying k in SMPO-R problem

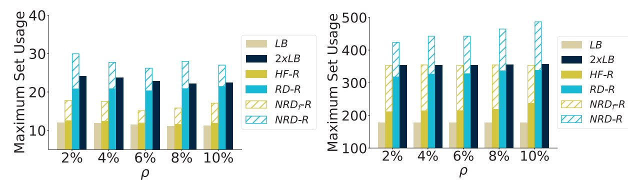
Impact of the set cardinality k. As shown in Fig. 5, when k changes from 2^5 to 2^9 the results of both *HF-R* (1.06× to 1.01× in synthetic data and 1.52× to 1.12× in real data higher than *LB*) and *RD-R* (1.81× to 1.71× in synthetic data and 1.95× to 1.77× in real data higher than *LB*) are closer to the lower bound. The reason is the same as varying k in the SMPO-A problem.

Impact of the set number ratio δ_s. As shown in Fig. 6, with δ_s increase, both *HF-R* (1.0002× to 1.05× in synthetic data and 1.38× to 1.5× in real data higher than *LB*) and *RD-R* (1.23× to 1.86× in synthetic data and 1.55× to 1.9× in real data higher than *LB*) are further from the lower bound. The reason is similar to vary δ_s in SMPO-A problem.



(a) Normal distribution (b) Real dataset

Figure 6: Results of varying δ_s in SMPO-R problem



(a) Normal distribution (b) Real dataset

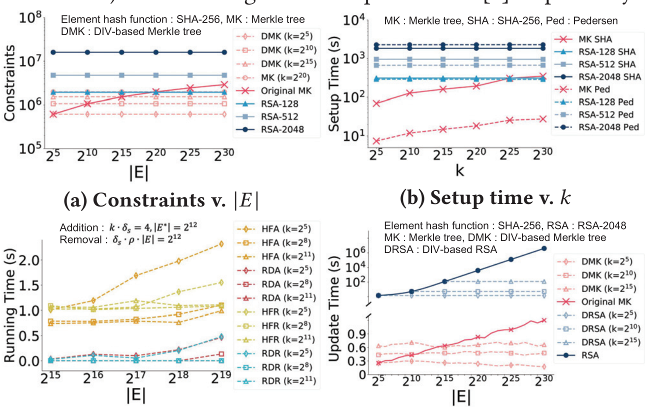
Figure 7: Results of varying ρ in SMPO-R problem

Impact of the affected sets percentage ρ. As shown in Fig. 7, with ρ increases, both *HF-R* (1.04× to 1.06× in synthetic data and 1.19× to 1.33× in real data higher than *LB*) and *RD-R* (1.73× to 1.91× in synthetic data and 1.79× to 1.9× in real data higher than *LB*) are further from the lower bound. This is because with larger ρ, the $|E^r|$ is larger. Moreover, elements in $|E^r|$ have more duplications which can easily cause an element not to be assigned to the set with the minimum current usage due to the containing of its duplication.

6.2 DIV-based ZKSM Efficiency Measurement

Efficiency Metrics. We measure ZKSM efficiency from three aspects: proof generation, verification and set maintenance. Specifically, we use the count of R1CS constraints as the metric of proof generation. Because, for the prover, both time and space cost are dominated by the constraint count [10]. For proof verification, we measure its time cost. While, for the set maintenance, we measure the time cost to prepare a set (including the circuit setup, ZKP keys generation and the element assignment algorithm running for DIV-based solution) and update elements in a set (*a.k.a.* swaps: deleting the old and adding the new elements). For the on-chain delay, we capture it in the real application experiment (Sec. 6.3). We choose widely used Merkle tree and RSA-accumulator as examples to perform the DIV and we measure the aforementioned metrics by varying the existing element count $|E|$ and the set cardinality k.

Implementation. Our experiments are based on one of the latest Rust implementations of both Merkle tree and RSA accumulators [46]. Specifically, it uses the Bellman [35] lib to build constraint systems and synthesizes constraints over the BLS12-381 curve [49]. The ZKPs are generated with pairing-based argument based on Groth [33]. For the element hashing functions, we choose SHA-256 (each operation costs 45567 constraints) and Pedersen commitment [48] based on the JubJub elliptic curve [55] (each operation costs 2753 constraints) respectively. For the RSA-accumulator, we choose the RSA modulus size in 2048-bit, 512-bit, and 128-bit (the larger the securer) with challenge numbers provided in [4] respectively.



(a) Constraints v. $|E|$

(b) Setup time v. k

(c) Algo. running time v. $|E|$

(d) Update time v. $|E|$

Figure 8: Results of the ZKSM efficiency measurement

Results of proof generation and verification. Fig. 8a shows the constraint count (the lower the better) indicating the time/space cost of one-time ZKSM for a single element. With more $|E|$, the circuit constraints of original Merkle tree increases as well. However, with fixed subset cardinality k, DIV can ensure each ZKSM with constant cost (see *DMKs* in Fig. 8a). Thus, the dynamic issue that more existing elements will lead to higher proof generation cost can be solved. Moreover, for DIV, with 50% shorter tree height ($k = 2^{10}$ v. $k = 2^5$), the constraint count is reduced by 42.57%. Although, RSA-accumulator's constraints are not affected by the set size, it only outperforms original Merkle tree for large set. For instance, the break even point for RSA with 128-bits modulus appears in $|E| = 2^{20}$ and for RSA with larger modulus, the break even point comes later. While, by applying DIV, the cardinality of each subset is bounded and with well-chosen k (*e.g.*, *DMK* with $k < 2^{20}$), DIV-based Merkle tree can always outperform RSA.

On the other hand, the proof verification time of all accumulators with SHA-256 element hash function is negligible ($\approx 4ms$) and independent from the count of existing elements.

Results of set preparation. Fig. 8b and Fig. 8c show the set setup and DIV algorithms running time respectively. In particular, to setup a set is to pre-compute the common reference string (CRS) for provers and verifiers, which only depends on the input bounds of proving problem [10]. To support more elements (with larger k), Merkle tree setup time also increases and for original Merkle tree, it depends on the total number of supported elements. While, setup time of RSA is independent from the set size. Since DIV fixes subsets with the same cardinality, the problem input bounds of each set is the same as well (proving membership of homogeneous sets). Thus, even new sets will be created for new elements, it only needs to be setup once and the CRS can be reused. Therefore, since k in DIV is always lower than the total number of supported elements, DIV-based Merkle tree has less setup cost than both original Merkle tree (*e.g.*, with lower k, MK SHA and MK Ped cost lower time) and RSA accumulator (by choosing k less than 2^{30}). Also, the same accumulator with more efficient element hash function has less setup cost (*e.g.*, MK SHA v. Ped and RSA SHA v. Ped in Fig. 8b).

Besides, Fig. 8c shows algorithms' running time of DIV-based method by varying $|E|$ according to Table 2 with fixed new elements ($|E^*| = 2^{12}$) for *HF-A* and *RD-A* algorithms and affected elements ($\delta_s \rho |E| = 2^{12}$) for *HF-R* and *RD-R* algorithms (due to space limit, we omit the results by varying $|E^*|$ and ρ). When $|E|$ increases, the running time of our four algorithms growth logarithmically which meets our complexity analysis. Meanwhile, comparing with the set update time (shown in Fig. 8d and will be discussed later), the algorithm running time is tolerable which does not bring too much additional cost to create or delete sets.

Results of element update. Fig. 8d shows the update (addition after removal) time of an element. For Merkle tree, both addition and removal are to re-compute a branch whose time is determined by the tree height. Thus, with more $|E|$, the update time of original method increases while the time of DIV-based method (*DMK*) is steady and the smaller k is, the shorter the time is. For RSA, the addition cost is $O(1)$, however, the removal cost depends on the existing element count in the set. Therefore, with more $|E|$, the update cost of original RSA increases quasilinearly with $|E|$. For DIV-based RSA, after $|E|$ exceeding the set cardinality k of *DRSA*, the update time in a full-set remains steady which has significant improvement (*e.g.*, when $|E| = 2^{20}, k = 2^{15}$, DIV reduces the time by 97%) and the dynamic issue of set maintenance cost is solved.

6.3 Real App Performance: zkSync as example

Performance Metrics. We use zkSync as the real application example to evaluate the performance of DIV. Specifically, as discussed in Sec. 2.2, we focus on the transaction latency bottleneck by applying DIV on the universal state digest/set. Particularly, we divide the account tree and vary the set cardinality as well as the block chunks size which controls the throughput. Since the under layer Bellman implementation automatically pads any circuit size up to the next power of two to apply efficient radix-2 DFT algorithm [25], we cannot obtain the exact time cost of the ZKP in arbitrary circuit size. Thus, similar to experiments in Sec. 6.2, we use the constraint count as the efficiency metric. Moreover, to illustrate the relationship between the constraint count and latency, we measure the latency under the circuit size from 2^{21} to 2^{26}. Despite zkSync supports many transaction types, due to space limit, we choose the most frequently used transactions - token transfer (occupying two chunks per transaction), to measure its transaction latency.

Implementation. Our experiments are based on the Rust implementation of zkSync v0.4.2 [36]. In particular, to apply DIV on zkSync, we mainly did the following modifications: **1.** we divide the account tree into subtrees (balance tree depth remains 8) with given cardinality such that the universal state digest/set is separated into subsets (stored in one or more L1 contracts) to serve different accounts. **2.** To simulate a set maintainer to get the account-set assignment, we use the Ethereum dataset described in Sec. 6.1 and our HFA algorithm with $|E^*| = 2^{10}$. **3.** To support the ZKP after applying DIV, we slightly modify the zkSync circuit. Specifically, the following simplified official pseudocode [3] shows the basic circuit logic of transfer operation.

```
def transfer(op, cur):  # cur: current Merkle branch
  # check sender's validity (s: sender, r: receiver)
  s_valid:=s.sig_msg==('transfer',s.account,s.token,r.pub_key,op.
  other_args) and s.pub_key==cur.owner_pub_key and other_checks()
  if s_valid: # deduct sender's balance
    cur.balance -= (op.amount + op.fee)
  # check receiver's validity (s: sender, r: receiver)
  r_valid:= not cur.receiver_is_empty and pubdata==(op.tx_type,
  s.account,s.token,op.amount,r.account,op.fee) and other_checks()
  if r_valid: # increase receiver's balance
    cur.balance += op.amount
  return s_valid or r_valid
```

Previously, the circuit first checks the validity of sender's signature and the membership of the current universal Merkle set (*cur* in the code) for sender and receiver respectively. Then, performing balance operations. For DIV, we modify the input of *cur* to Merkle branches (cur_s, cur_r) of two sets containing sender/receiver and check their membership as well as perform balance operations in each branch. Finally, we run the entire system locally to test the latency without considering the network affection.

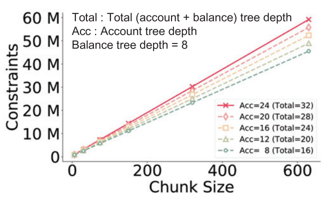


Figure 9: zkSync Constraints

Constraints	Latency (s)
$2^{21} \approx 2.10M$	193.85
$2^{22} \approx 4.19M$	310.73
$2^{23} \approx 8.39M$	544.49
$2^{24} \approx 16.78M$	1012.02
$2^{25} \approx 33.55M$	1947.09
$2^{26} \approx 67.11M$	3817.21

Table 3: zkSync Latency

Experimental Results. Fig. 9 shows the result by varying the chunk size (from *3 tps* to *315 tps* of the transfer transaction) and set cardinality (account tree depth which also affects the total tree depth). Notice that, majority of zkSync constraints are for signature and ZKSM check. With shorter account tree (Acc in Fig. 9), the ZKSM constraints decrease which follows the ZKSM efficiency experimental results in Sec. 6.2. Besides, with more chunk size (higher throughput), more percentage of constraints is ZKSM, because other overheads which are independent from the chunk size becomes negligible. In particular, by decreasing the total tree depth (account + balance) by 12.5% (Acc = 24 v. Acc = 20), the constraints decrease from 3.85% (for *3 tps*) to 5.76% (for *315 tps*). Moreover, the constraint count is proportional to the latency. As shown in Table 3, when constraints decrease 50% (*e.g.*, from 2^{26} to 2^{25}), the latency decreases 49%. Therefore, DIV can help to reduce the system latency (*e.g.*, 50% shorter tree can reduce the latency by 23%) and with higher throughput, DIV has more improvement.

7 RELATED WORKS

To make the ZKSM proof more efficient and compatible to general cases, mechanisms are proposed based on different cryptography assumptions. Specifically, existing ZKSM methods are classified into three types: *Merkle Tree-based* (*e.g.*, ZCash [51]), *pairing-based* (*e.g.*, Nguyen accumulator [45]) and *RSA-based* (*e.g.*, RSA accumulator [22]). Merkle tree-based ZKSM is widely used in the blockchain scenario which allows $O(1)$ public parameters but $O(logn)$ proof size and generation complexity, where n is the size of the set. While pairing-based solutions [21, 45] require small proof size (*e.g.*, a single element with 256 bits of a prime order bilinear group) but $O(n)$ parameters and trusted setup which is undesirable in the blockchain scenario. Although the RSA accumulator can achieve $O(1)$ size of proof and public parameter, it requires a large prime order group (*e.g.*, for 128-bits security level, the group order should be $\geq 2^{259}$) to hold the commitment space. Besides, since it requires all elements to be prime numbers, additional map schemes are required which makes it harder to be applied in blockchain.

Another attempt to scale set membership proof is to perform the ZKSM operations in batches. Specifically, [46] implements an RSA accumulator inside of a SNARK to replace the Merkle tree which enables proving the membership of a batch of elements from the same set simultaneously. Another work [14] enables batching operations on the RSA-based accumulator which can work in distributed environment without the trusted set maintainer. These methods improve the ZKSM efficiency by allowing proving the membership for a batch of elements at the cost of one-time process.

DIV differs from the above works that we not only consider accelerating the efficiency of proof and verification but also targeting on reducing the set maintenance cost when elements are frequently updated. Moreover, since DIV does not depend on the underlying ZKSM mechanism, all these works mentioned above are compatible with DIV to further scale ZKSM.

8 CONCLUSIONS

In this paper, we focus on applying zero-knowledge set membership proof in blockchain systems and identify the dynamic issues brought by set elements addition and removal. Specifically, when the set is updated, not only extra cost is needed to finalize the updated set on the blockchain, but also the ZKSM time/space cost will be affected. To address the dynamic issues, we propose DIV which is to divide the universal set into subsets. Meanwhile, to prevent the information leak of which elements are frequently used, we propose the SMPO problems in both elements addition and removal scenarios and prove their NP-hardness. For each problem, we consider whether the actual frequency is the input or not to meet different use cases and design an algorithm with an approximation guarantee to address each of them. We conduct extensive experiments on both real and synthetic datasets as well as implement DIV on Merkle tree and RSA-based ZKSM mechanisms and a ZKSM-based blockchain application named zkSync to evaluate DIV and our optimization algorithms. Results verify that DIV can achieve $O(1)$ time/space cost on ZKSM under the dynamic update situation and protect the information about frequently used elements.

ACKNOWLEDGMENTS

This work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC GRF Project 16213620, CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, Didi-HKUST joint research lab, HKUST-Webank joint research lab grants.

REFERENCES

[1] 2018. https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477/7.
[2] 2018. roll_up: Scale ethereum with SNARKs. https://github.com/barryWhiteHat/roll_up.
[3] 2020. https://github.com/matter-labs/zksync/blob/v0.4.2/docs/circuit.py#L339.
[4] 2020. RSA numbers. https://en.wikipedia.org/wiki/RSA_numbers.
[5] 2021. https://github.com/xzhflying/SIGMOD21-DIV/blob/main/paper/Appendix.pdf.
[6] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.
[7] Tobias Bamert, Christian Decker, Roger Wattenhofer, and Samuel Welten. 2014. Bluewallet: The secure bitcoin wallet. In *International Workshop on Security and Trust Management*. Springer.
[8] Niko Barić and Birgit Pfitzmann. 1997. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*. Springer.
[9] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. 2019. Aurora: Transparent succinct arguments for R1CS. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer.
[10] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct non-interactive zero knowledge for a von Neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*.
[11] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2017. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica* 79, 4 (2017).
[12] Josh Benaloh and Michael De Mare. 1993. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer.
[13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*.
[14] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Annual International Cryptology Conference*. Springer.
[15] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[16] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[17] Christian Cachin. 1997. *Entropy measures and unconditional security in cryptography*. Ph.D. Dissertation. ETH Zurich.
[18] Christian Cachin et al. 2016. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, Vol. 310.
[19] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. 2017. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.
[20] Jan Camenisch, Maria Dubovitskaya, Robert R Enderlein, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. 2014. Concepts and languages for privacy-preserving attribute-based authentication. *Journal of information security and applications* 19, 1 (2014).
[21] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *International workshop on public key cryptography*. Springer.
[22] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*. Springer.
[23] Jan Camenisch and Els Van Herreweghen. 2002. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*.
[24] Yuan Shih Chow and Henry Teicher. 2003. *Probability theory: independence, interchangeability, martingales*. Springer Science & Business Media.
[25] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965).
[26] Ivan Damgård and Nikos Triandopoulos. 2008. Supporting Non-membership Proofs with Bilinear-map Accumulators. *IACR Cryptol. ePrint Arch.* 2008 (2008).
[27] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. 2018. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering* 30, 7 (2018).
[28] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*. Springer.
[29] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*.
[30] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. 2019. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications* 126 (2019).
[31] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer.
[32] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989).
[33] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer.
[34] Edwin T Jaynes. 1957. Information theory and statistical mechanics. *Physical review* 106, 4 (1957).
[35] Matter Labs. 2020. Bellman circuit library. https://github.com/matter-labs/bellman..
[36] Matter Labs. 2020. zkSync v0.4.2. https://github.com/matter-labs/zksync/releases/tag/v0.4.2.
[37] matter labs. 2020. zkSync. https://zksync.io.
[38] Izaak Meckler and Evan Shapiro. 2018. Coda: Decentralized cryptocurrency at scale. *O (1) Labs whitepaper. May* 10 (2018).
[39] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*.
[40] Silvio Micali. 1994. CS proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE.
[41] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE.
[42] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.
[43] Eduardo Morais, Tommy Koens, Cees Van Wijk, and Aleksei Koren. 2019. A survey on zero knowledge range proofs and applications. *SN Applied Sciences* 1, 8 (2019).
[44] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[45] Lan Nguyen. 2005. Accumulators from bilinear pairings and applications. In *Cryptographers' track at the RSA conference*. Springer.
[46] Alex Ozdemir, Riad Wahby, Barry Whitehat, and Dan Boneh. 2020. Scaling verifiable computation using efficient set accumulators. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.
[47] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE.
[48] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer.
[49] Bowe S. 2017. BLS12-381: New zk-SNARK elliptic curve construction. https://z.cash/blog/new-snark-curve/..
[50] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. 2019. Exploring the attack surface of blockchain: A systematic overview. *arXiv preprint arXiv:1904.03487* (2019).
[51] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE.
[52] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J Blumberg, Bryan Parno, and Michael Walfish. 2013. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*.
[53] Melanie Swan. 2015. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.".
[54] Peter Todd. [n.d.]. Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments, May 2016.
[55] Riad S Wahby and Dan Boneh. 2019. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019).
[56] David P Williamson and David B Shmoys. 2011. *The design of approximation algorithms*. Cambridge university press.
[57] Cheng Xu, Ce Zhang, and Jianliang Xu. 2019. vchain: Enabling verifiable boolean range queries over blockchain databases. In *Proceedings of the 2019 International Conference on Management of Data*.
[58] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2017. An expressive (zero-knowledge) set accumulator. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE.