Toxic Decoys: A Path to Scaling Privacy-Preserving Cryptocurrencies

Christian Cachin
University of Bern
Bern, Switzerland
christian.cachin@unibe.ch

François-Xavier Wicht
University of Bern
Bern, Switzerland
francois-xavier.wicht@unibe.ch

Abstract

Anonymous cryptocurrencies attracted much attention over the past decade, yet ensuring both integrity and privacy in an open system remains challenging. Their transactions preserve privacy because they do not reveal on which earlier transaction they depend, specifically which outputs of previous transactions are spent. However, achieving privacy imposes a significant storage overhead due to two current limitations. First, the set of potentially unspent outputs of transactions grows indefinitely because the design hides cryptographically which one have been consumed; and, second, additional data must be stored for each spent output to ensure integrity, that is, to prevent that it can be spent again. We introduce a privacy-preserving payment scheme that mitigates these issues by randomly partitioning unspent outputs into fixed-size bins. Once a bin has been referenced in as many transactions as its size, it is pruned from the ledger. This approach reduces storage overhead while preserving privacy. We first highlight the scalability benefits of using smaller untraceability sets instead of considering the entire set of outputs, as done in several privacy-preserving cryptocurrencies. We then formalize the security and privacy notions required for a scalable, privacy-preserving payment system and analyze how randomized partitioning plays a key role in both untraceability and scalability. To instantiate our approach, we provide a construction based on Merkle trees, which ensures efficient argument systems and easy pruning of the state. We finally show the storage benefits of our scheme and analyze its resilience against large-scale flooding attacks using empirical transaction data.

1 Introduction

In the cryptocurrency transaction layer, *untraceability* means that a transaction hides which funds it transfers or "spends," in the sense that the origin of the funds remains hidden among earlier transactions [21, 24, 28, 42]. In other words, untraceability consists in spending a "coin" anonymously among a group called the *untraceability set*. Currently there exist two common ways of providing this feature. One is to actively select a subset of outputs from past transactions as the untraceability set and prove ownership of a coin within this subset, usually referred to as a ring [34]. In this case, the selected outputs *explicitly* form the untraceability set, one output is actually spent, and the others are referred to as *decoys*. The other way is to prove existence and ownership of the coin in zero-knowledge over the *whole* set of outputs created so far. This means that all outputs that exist in the ledger *implicitly* form the untraceability set. The second technique offers strictly greater privacy guarantees [18, 37, 39]. The first approach has been adopted by Monero and the second by Zcash, the two most prominent privacy-preserving cryptocurrencies.

Both systems nonetheless share at least two current limitations that impact their scalability and long-term viability.

(1) The set of (potentially) unspent outputs grows indefinitely, since transactions do not reveal which of earlier transaction outputs are spent. As a result, the number of outputs included in the system keeps increasing over time.
(2) Additional cryptographic data must be produced as part of each transaction to prevent double-spending. This data that includes the so-called "nullifiers" also grow indefinitely. Deleting any of this would open the door to double-spending.

However, these limitations hold only for an unstructured set of past outputs. We now show using an example how structuring the data mitigates the above limitations. Consider two independent transactions tx_1 and tx_2 in Figure 1. The first transaction spends output o_1, and the second one spends o_2. In this example, o_1 and o_2 mutually serve as decoys for each other in tx_2 and tx_1 respectively (highlighted with dotted edges in the figure).

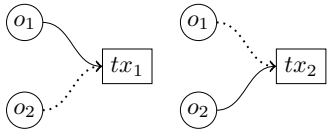

Figure 1. Illustration of two independent transactions with o_1 and o_2 as the sources of transactions tx_1 and tx_2 (linked with hard edges). Outputs o_1 and o_2 are decoys (linked with dotted edges) for tx_2 and tx_1.

While tx_1 and tx_2 cannot be traced to o_1 and o_2 directly, they jointly reveal that both o_1 and o_2 have been spent, as each output can only be spent once. Consequently, these outputs *must not* be used as decoys for subsequent transactions. Otherwise, future transactions that reference these decoys would be partially deanonymized. Such degradation of privacy through decoy pruning has been previously studied by Egger *et al.* [18] and by Vijayakumaran [37]. We refer to outputs that have exhausted their untraceability potential like this as *toxic*, since using them as decoys would poison transactions by reducing their privacy guarantees.

Proceedings on Privacy Enhancing Technologies 2025(4), 926–943

https://doi.org/10.56553/popets-2025-0165

In this paper, we do not see this issue as a challenge but rather as the key to scalability. By structuring the set of outputs into fixed-size groups that serve as independent untraceability sets, which we call *bins*, we ensure two properties:

(1) bins can be pruned once they have been referenced by m transactions, m being the size of bins; and
(2) pruning does not reveal which of the past transactions has spent which output, preserving untraceability.

This approach has the potential to limit the growth of the output set and eliminate the impact of toxic decoys. However, even though this solution looks benign, several points must be addressed. Those aspects are best summarized by outlining the weak spots of decoy sampling. To better understand the risks associated with decoy selection, we summarize four potential attacks from the literature, which we classify as *passive* and *active* attacks.

(1) Sampling discrepancy (passive): If the sampling algorithm picks decoys uniformly at random over the whole ledger, there is a divergence between the actual spending distribution and the decoy-sampling distribution. An adversary can use this divergence to infer *probabilistically* that ring members outside the spending distribution are likely decoys. Although the spending distribution is unknown, an adversary can approximate it (e.g., using the age of outputs) to realistically mount attacks [28]. If the partitioning algorithm simply groups outputs uniformly over the whole set, we face the same problem.
(2) Graph decomposition (passive): In some cases, an adversary can exploit the structure of the transaction graph to *deterministically* prune ring members of past transactions [18, 37]. This attack identifies toxic decoys and removes them from the untraceability set. While partitioning is a potential solution to this, it is crucial that groups of outputs remain immutable over their lifespan. If an output is assigned to a group of toxic decoys, any transaction using that group would be completely traceable.
(3) Flooding attack (active): An adversary can artificially create transaction outputs to flood the ledger. Any ring including adversarial outputs has thus a smaller untraceability set (for the adversary) [12, 30]. Partitioning the set of outputs greatly increases this risk since an adversary can simply target a specific bin and flood it as shown in Figure 2a.
(4) Pattern generation (active): Using the above vulnerabilities, an adversary can judiciously create transaction outputs to generate favorable deanonymization patterns, according to passive attacks.

To address these scalability and privacy challenges, we propose a privacy-preserving payment scheme based on output partitioning. Our approach ensures that outputs are grouped in a way that minimizes adversarial interference. Specifically, we introduce the following key ideas:

- Outputs are periodically partitioned into fixed-size groups that remain unchanged throughout their lifespan. This approach minimizes probabilistic inferences since outputs produced in the same period usually share the same spending distribution.

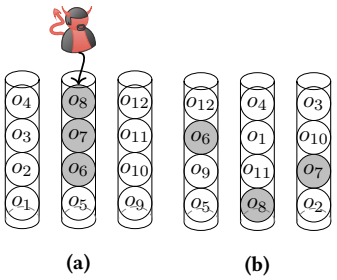

Figure 2. On the left, an adversary targets a bin containing output o_5 by submitting transactions to the ledger. The transactions produce o_6, o_7, o_8, which belong to the adversary. Thus, o_5 loses untraceability in the adversary's view. On the right, after shuffling outputs across bins, the adversary has a uniform number of outputs in each bin, reducing its ability to flood a single bin.

- The partitioning algorithm randomly assigns outputs to bins as shown in Figure 2b, which prevents an adversary from predicting the bin assignment of each output. This minimizes the risk of targeted flooding and deanonymization attacks.
- Once a group has been referenced in as many transactions as its size, it is considered toxic and may be pruned, reducing storage overhead without compromising untraceability of subsequent transactions.

In summary, our contributions are as follows.

- We highlight significant scalability benefits when using smaller untraceability sets, as opposed to privacy-preserving cryptocurrencies that consider the entire set of outputs for untraceability.
- We formalize the security and privacy properties required for a scalable, privacy-preserving cryptocurrencies.
- We analyze randomized partitioning schemes and show how crucial this property is for both untraceability and scalability.
- We provide a construction based on Merkle trees, which eases scalability over time and allows for efficient argument systems.
- We quantify the storage savings achieved by our scheme based on real transaction data.
- We analyze the presumed flooding attack of March 2024 in Monero using our scheme and show the dynamic resilience it can offer.
- We discuss open problems in privacy-preserving cryptocurrencies and suggest potential solutions using our scheme.

The remainder of this paper is structured as follows. In Section 2, we review related work. Section 3 introduces the necessary cryptographic primitives and formalizes the system model. In Section 4, we define the security and privacy properties required for a scalable privacy-preserving payment scheme. Section 5 presents our main construction, leveraging partitioning and Merkle-trees to ensure storage efficiency while maintaining untraceability. In Section 6, we assess how partitioning and pruning techniques perform on a live system using Monero transaction data. In Section 7, we analyze the resilience of our scheme against the presumed flooding

attack of Monero in March 2024. Section 8 explores practical considerations and potential optimizations. Finally, Section 9 outlines deployment paths in the two most prominent privacy-preserving cryptocurrencies – Monero and Zcash.

2 Related work

Previous works have sought to address the scalability challenges of privacy-preserving cryptocurrencies through various means. Some approaches focus on reducing the size of individual components [4], while others leverage compact data structures [5] or even propose entirely different system designs [7, 17, 19].

One notable example is QuisQuis [19], which introduces a private account-based ledger where the ledger size scales linearly with the number of accounts but remains constant with respect to transactions. However, a significant drawback of this approach is that clients must scan the entire ledger to determine their account status. This requirement not only imposes a computational burden but also exposes the system to denial-of-service attacks. Specifically, malicious users can repeatedly include honest users' accounts as decoys in transactions, effectively preventing them from accessing their own funds.

Another line of work by Chow *et al.* [14] presents a sustainability framework for ring-based anonymous systems, addressing the problem of ever-growing unspent outputs. Their solution integrates a garbage collection mechanism based on transaction graph structure alongside a sampling strategy that prioritizes outputs of the same age. Notably, their pruning condition aligns with the observations made in our introduction, yet they rigorously establish and prove it through a graph-theoretic argument. Despite these strengths, their approach suffers from structural weaknesses due to deterministic partitioning. In particular, an adversary can degrade anonymity by flooding a partition with their own outputs, effectively isolating honest users and making their transactions easily distinguishable. Additionally, their scheme lacks a mechanism to ensure balanced partitions, which can result in skewed anonymity sets and diminished privacy guarantees.

Möser *et al.* [28] informally suggest partitioning the set of outputs as another way to circumvent attacks that exploit the sampling discrepancy outlined in the introduction. They propose randomizing output assignment using the current block header. While this technique introduces an element of randomness, it remains susceptible to adversarial manipulation, as block headers are predictable and prone to adversarial bias. Moreover, their approach does not acknowledge the key benefit of partitioning, namely the possibility of pruning outputs once all elements in a bin have been spent. In contrast, our scheme circumvents several issues – sampling discrepancy, storage overhead and flooding attacks – by randomly partitioning outputs using unbiased randomness.

In a similar vein, Manevich *et al.* [27] propose a privacy-preserving scheme based on private ledger partitions. This partitioning mechanism leverages verifiable randomness derived from protocol participants, enhancing robustness against adversarial influence. This method enables an operator to distribute operations out-of-band among participants while recording only an encrypted aggregate signature and a pseudo-random identifier on the global ledger. By aggregating participant signatures, encrypting them with a symmetric key, and deriving partition identifiers through a deterministic function, this approach prevents non-participants from deducing transaction details while still allowing authorized parties to verify integrity.

Finally, another related suggestion of Boneh *et al.* [6] employs binning to improve the efficiency of leader election. This work highlights the crucial role of randomization in partitioning and underscores its significance in maintaining fairness and security.

3 Preliminaries

Notation. The concatenation of two elements a and b is denoted by $a\|b$. Sampling a uniformly random element x from a set S is denoted by $x \leftarrow\!\!\$\ S$. Let λ denote a security parameter. A function $\mathsf{negl}(\lambda) : \mathbb{N} \to \mathbb{R}^+$ is negligible if for every positive polynomial $\mathsf{poly}(\lambda)$, there exists a $\lambda_0 \in \mathbb{N}$, such that for all $\lambda > \lambda_0$: $\varepsilon(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. The notation $x \leftarrow \mathcal{A}(\cdot)$ refers to the output of a (potentially probabilistic) algorithm $\mathcal{A}$. We denote by $\mathsf{shuffle}(l, \mathsf{rnd})$ the random shuffling of l based on randomness rnd.

Standard primitives. We use standard cryptographic tools, including commitment schemes, non-interactive argument systems, and Merkle trees. Definitions of these primitives appear later in Section 5.

Randomness beacon. Each invocation of the partitioning algorithm uses a fresh public randomness rnd. Various methods exist to generate such randomness, and extensive research explores different approaches [10, 26, 33]. Surveys on the topic include works by Choi *et al.* [13] and Kavousi *et al.* [22].

Public ledger. We assume a public ledger Λ, also known as a blockchain, hosting a privacy-preserving cryptocurrency. Users freely join the network as:

(1) clients initiating and receiving transactions, and
(2) validators verifying transactions and securing the blockchain.

Transactions transfer cryptocurrency units between users by consuming (spending) inputs and producing (creating) new outputs. Each output is linked to a public key pk, and spending it requires knowledge of the corresponding secret key sk. We use o to denote inputs and outputs, irrespective of the underlying ledger model (e.g., UTXOs or account states). Validators order transactions and maintain the state of the ledger using a Byzantine fault-tolerant [9] and sybil-resistant [1, 23, 29] atomic broadcast protocol, satisfying:

- **Validity:** If a correct process p broadcasts a transaction tx, then p eventually delivers tx.
- **No duplication:** No correct process delivers the same transaction more than once.
- **Integrity:** If a correct process delivers a transaction tx with sender p, and p is correct, then tx was previously broadcast by p.
- **Agreement:** If some correct process delivers a transaction tx, then every correct process eventually delivers tx.
- **Total order:** If two correct processes deliver transactions tx_1 and tx_2, they do so in the same order.

Moreover, temporarily unordered transactions remain in a buffer and we assume that an incentive mechanism is embedded in the protocol to reward honest validators.

Overview. Our scheme proceeds in multiple epochs. Each epoch consists of the following steps.

(1) At the beginning, several transactions generate new outputs, which lie in the transaction buffer until the end of the current epoch.
(2) At that time, outputs are randomly *partitioned* into k distinct bins.
(3) Users can then *spend* their outputs by referencing the corresponding bin and proving they know the secret for one of the outputs in said bin.
(4) Once a bin has been referenced as many times as its size, the bin can be *pruned* from the state. A partition expires once its k bins have been pruned.

4 Characterization

A scalable privacy-preserving payment scheme SPS allows secure and private transactions over the ledger Λ while enabling the pruning of outdated bins, thus improving scalability. It is composed of algorithms executed by two different types of parties: clients and validators. The scheme begins with a setup phase, during which public parameters pp are generated based on a security parameter λ. Users and validators generate key-pairs $(\mathsf{sk}, \mathsf{pk})$ for them to securely identify on the network. The initial state of the system is also composed of a few outputs, previously created and partitioned into a single bin, i.e., $k = 1$. A typical transaction is a transfer of ownership of spent outputs to newly created ones, while revealing a nullifier that prevents double-spending. For a transaction to be valid, spenders must provide a proof of ownership π of their outputs, the value of the spent outputs must match the value of the created ones, and the nullifiers must not collide with existing ones. The newly created outputs remain unordered until the next round of atomic broadcast. At this occasion, validators partition the new outputs using fresh randomness.

We moreover differentiate between probabilistic and deterministic algorithms. Probabilistic algorithms rely on internally generated randomness, producing varying outputs for the same input, whereas deterministic algorithms always yield identical outputs for identical inputs. Note that the partitioning algorithm is a deterministic algorithm but it takes randomness as input.

While our design incorporates elements that could be integrated into existing cryptocurrencies (as we discuss in later sections), we present it as a complete, standalone scheme for several important reasons. The pruning mechanism we introduce fundamentally interacts with core security properties including double-spending prevention and privacy guarantees (untraceability). These interactions require careful formal analysis that would be difficult to capture if presented merely as an extension or add-on. We now formally define the syntax and security properties of SPS through several notions – double-spending, unpredictability, untraceability, scalability, confidentiality and non-slanderability.

Definition 4.1 (Scalable privacy-preserving payment scheme). A scalable privacy-preserving payment SPS scheme consists in eight algorithms:

- $(pp, \mathsf{st}_0) \leftarrow \mathrm{Setup}(1^\lambda)$ is a probabilistic algorithm that takes the security parameter λ, returns the public parameters pp and the initial state st_0. The public parameters pp are implicit input to the following algorithms.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{KeyGen}(\cdot)$ is a probabilistic algorithm that returns a private and public key-pair.
- $(o, \mathsf{nul}) \leftarrow \mathrm{CreateOutput}(\mathsf{pk}, \mathsf{amt})$ is a probabilistic algorithm that takes as input a public key pk, an amount amt, returns a nullifier nul, and an output o.
- $\mathcal{P} \leftarrow \mathrm{Partition}(\mathsf{rnd}, O, k)$ is a deterministic algorithm that takes as input a random value rnd, a set of outputs O, an integer k, and returns a partition of k bins $\mathcal{P} = \{\mathsf{bn}_1, \ldots, \mathsf{bn}_k\}$ over O.
- $(\pi, \mathsf{nul}) \leftarrow \mathrm{Spend}(\mathsf{sk}, o, \mathsf{bn})$ is a probabilistic algorithm that takes as input a secret key sk, an output o, a bin bn, returns a proof π, and a nullifier nul.
- $\{0,1\} \leftarrow \mathrm{Link}(\mathsf{bn}, \mathsf{nul}, \mathsf{nul}')$ is a deterministic algorithm that takes as input a bin bn, nullifiers nul and nul′, and returns either 0 or 1.
- $\{0,1\} \leftarrow \mathrm{Verify}(\mathsf{st}, \mathsf{bn}, \pi, \mathsf{nul})$ is a deterministic algorithm that takes as input the state st, a bin bn, a proof π, a nullifier nul, and returns either 0 or 1.
- $\{0,1\} \leftarrow \mathrm{Prune}(\mathsf{bn}, \{\pi_i\}_{i=1}^m, \{\mathsf{nul}_i\}_{i=1}^m) \rightarrow \{0,1\}$ is a deterministic algorithm that takes as input a bin bn, a set of m proofs $\{\pi_i\}_{i=1}^m$, a set of m nullifiers $\{\mathsf{nul}_i\}_{i=1}^m$, and returns either 0 or 1.

The first notion we define is probably the most important in any cryptocurrency: double-spending. Double-spending occurs when a user attempts to spend the same output more than once. In a privacy-preserving payment scheme, preventing double-spending is challenging since transactions do not explicitly reveal which output has been spent. Instead, the system must ensure that each output can only be spent once without compromising privacy. This is typically achieved using nullifiers, which act as unique, one-way markers for spent outputs. A secure scheme must guarantee that no adversary can produce two valid spending proofs for the same output with two nullifiers that do not link. We formalize this requirement in the following experiment.

Definition 4.2 (Double-Spending). An SPS scheme prevents double-spending by ensuring that no output is spent more than once. Formally, an adversary $\mathcal{A}$ wins the double-spending experiment $\mathrm{DSExp}(\mathcal{A}, \lambda, n, k)$ if it succeeds in producing two valid proofs by referencing the same output.

The experiment proceeds as follows:

Setup. The challenger C runs the setup algorithm $\mathsf{pp} \leftarrow \mathrm{Setup}(1^\lambda)$ and creates n outputs O, partitioned into k bins using randomness rnd, i.e., $\mathcal{P} := \{\mathsf{bn}_1, \ldots, \mathsf{bn}_k\} \leftarrow \mathrm{Partition}(\mathsf{rnd}, O, k)$.

Double-spending. The adversary $\mathcal{A}$ receives pp, O, $\mathcal{P}$ and attempts to produce two valid spending proofs (π_1, nul_1) and (π_2, nul_2) for the same output o in bin bn_i, i.e.,

$$(\pi_1, \mathsf{nul}_1), (\pi_2, \mathsf{nul}_2) \leftarrow \mathcal{A}(O, \mathcal{P}).$$

Verification. The adversary $\mathcal{A}$ wins if

$$\mathsf{Verify}(\mathsf{bn}_i, \pi_1) = \mathsf{Verify}(\mathsf{bn}_i, \pi_2) = 1 \wedge \mathsf{Link}(\mathsf{nul}_1, \mathsf{nul}_2) = 0.$$

An SPS is *double-spending resistant* if no probabilistic polynomial-time adversary $\mathcal{A}$ can win this experiment with greater than negligible probability:

$$\Pr\left[\mathsf{DSExp}(\mathcal{A}, \lambda, n, k)\right] \leq \mathsf{negl}(\lambda).$$

Furthermore, as mentioned in the preamble of this work, a partitioning algorithm must be *unpredictable*. As we will see more formally, an SPS is unpredictable, if an adversary cannot predict to which bin an output is going to get assigned. In more detail, we say a partitioning algorithm is unpredictable if the adversary has only a negligible advantage of winning the following experiment. The game starts with the setup algorithm. The adversary takes as input the public parameters pp, a set of outputs O, the partitioning parameter rnd k, and outputs a partition index i and an output $o \in O$. The partitioning algorithm is ran with the random value rnd, the set of outputs O, and the value k. The algorithm outputs a partition $\{\mathsf{bn}_1, \ldots, \mathsf{bn}_k\}$ and the adversary wins the experiment if the output, o, is in the bin bn_i of the given index i.

Definition 4.3 (Unpredictability). We denote the unpredictability experiment with adversary $\mathcal{A}$, security parameter λ and for all $n, c \in \mathbb{N}$ by $\mathsf{Unpred}(\mathcal{A}, \lambda, n, k)$. The experiment is played between an adversary $\mathcal{A}$ and a challenger C, proceeding as follows:

Setup. The challenger C runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$, and creates n outputs of arbitrary denominations

$$O := \{o \mid o \leftarrow \mathsf{CreateOutput}(\mathsf{pk}, \cdot)\}, \text{ such that } |O| = n.$$

It sends the public parameters pp and the set of outputs O to the adversary $\mathcal{A}$.

Prediction. The adversary $\mathcal{A}$ takes the security parameter λ, the public parameters pp, the of outputs O, and the partitioning parameter k and outputs an index i and an output o, i.e., $(i, o) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, O, k)$.

Challenge. The challenger C generates the randomness $\mathsf{rnd} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and runs the partitioning algorithm on O using rnd and the parameter k, i.e.,

$$\mathcal{P} := \{\mathsf{bn}_1, \ldots, \mathsf{bn}_k\} \leftarrow \mathsf{Partition}(\mathsf{rnd}, O, k).$$

The adversary $\mathcal{A}$ wins if the output o is placed in the predicted bin bn_i, i.e., if $o \in \mathsf{bn}_i$.

An SPS is *k-unpredictable* if no probabilistic polynomial-time adversary $\mathcal{A}$ wins this experiment with greater than negligible advantage. That is, for all $\mathcal{A}$, for any $n \geq k \geq 2$, we have

$$\Pr\left[\mathsf{Unpred}(\mathcal{A}, \lambda, n, k)\right] \leq \frac{1}{k} + \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

The main privacy guarantee provided by our scheme is *untraceability* – the property that an adversary cannot determine which output was spent from within a bin. We formalize this notion via the following experiment.

The adversary begins by corrupting c out of the n outputs. The challenger C then runs the partitioning algorithm and provides the resulting partition $\mathcal{P}$ to the adversary. The adversary then selects a bin with the largest number of corrupted outputs – as to maximize its chance of guessing which outputs will be spent in the next step – and thus the smallest number β of honest outputs.

If β is 1, the adversary immediately wins the experiment. If β is 0, it immediately fails. Otherwise, the challenger uniformly selects one of the β honest outputs in bn, denoted o, and spends it, producing a corresponding proof π and nullifier nul.

The adversary receives π and nul, and outputs a guess o' as to which honest output was spent. It wins the experiment if $o' = o$.

Definition 4.4 (Untraceability). We denote the untraceability experiment with adversary $\mathcal{A}$, security parameter λ and for all $n, c, k \in \mathbb{N}$ by $\mathsf{Untrace}(\mathcal{A}, \lambda, n, c, k)$. The experiment is played between an adversary $\mathcal{A}$ and a challenger C, proceeding as follows:

Setup. The challenger C runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$, creates n outputs of arbitrary denominations O, such that $|O| = n$.
It sends the public parameters pp and the set of outputs O to the adversary $\mathcal{A}$.

Corruption. The adversary $\mathcal{A}$ corrupts a set of c outputs, i.e.,

$$C \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, O, R, k), \text{ such that } |C| = c.$$

Partitioning. The challenger C generates $\mathsf{rnd} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ runs the partitioning algorithm on O, gets the following partition $\mathcal{P}$

$$\mathcal{P} := (\mathsf{bn}_1, \ldots, \mathsf{bn}_k) \leftarrow \mathsf{Partition}(\mathsf{rnd}, O, k),$$

and sends the partition $\mathcal{P}$ to the adversary $\mathcal{A}$.

Selection. The adversary $\mathcal{A}$ selects a bin bn_i from the partition $\mathcal{P}$ with β honest outputs $\mathsf{bn}_i \leftarrow \mathcal{A}(\mathcal{P})$ where $\beta = |\mathsf{bn}_i \cap O|$. The adversary immediately wins if $\beta = 1$ and fails if $\beta = 0$. The adversary sends bn_i to the challenger.

Transaction. The challenger selects an honest output uniformly at random, i.e., $o \leftarrow\!\!\$\ \mathsf{bn}_i \cap O$. It then conducts a transaction using the bin bn_i and obtains

$$(\pi, t) \leftarrow \mathsf{Spend}(\mathsf{sk}_o, o, \mathsf{bn}_i).$$

The challenger sends the partition the bin bn_i, the proof π, and the nullifier nul to the adversary.

Challenge. The adversary $\mathcal{A}$ attempts to guess which output was spent by computing $o' \leftarrow \mathcal{A}(\pi, \mathsf{bn}_i, \mathsf{nul})$. The adversary wins if $o' = o$.

An SPS is *β-untraceable* if no stateful probabilistic polynomial-time adversary $\mathcal{A}$ can win this experiment with greater than negligible advantage. That is, for all $\mathcal{A}$, for any n, c, and k, we have

$$\Pr\left[\mathsf{Untrace}\left(\mathcal{A}, \lambda, n, c, k\right)\right] \leq \frac{1}{\beta} + \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

In the introduction, we have highlighted the importance of unpredictability of the partitioning algorithm, especially with regards to untraceability. Unpredictability guarantees that even an adversary corrupting a fraction of the outputs cannot systematically place corrupted outputs into unfavorable configurations for honest users. This property prevents the adversary from gaining an advantage in the untraceability experiment.

We show here formally that, if an adversary cannot predict the placement of any output with better than negligible advantage, then the distribution of corrupted outputs across the partitioning

remains sufficiently uniform. This ensures that, except with negligible probability, each bin contains a substantial number of honest outputs, thus maintaining a lower bound on the untraceability parameter β. The following lemma, proved in Appendix B, quantifies this relationship by establishing a bound on β under the assumption that the scheme is unpredictable.

Lemma 4.5. *If an SPS scheme is k-unpredictable, then it is at most β-untraceable for any*

$$\beta < \frac{n - ec}{k},$$

with high probability, where e is Euler's number, given security parameter λ, n outputs, k bins, and at least $c \geq 2k \log_2(k)$ corrupted outputs, against any stateful probabilistic polynomial-time adversary $\mathcal{A}$.

This first lemma highlights the crucial role of unpredictability in our scheme. In a predictable partitioning scheme, an adversary only needs to corrupt outputs based on the size of bins. More specifically, it only needs to corrupt $n/k - 1$ outputs to compromise the untraceability of a specific bin. However, when the partitioning scheme is unpredictable, the number of corrupted outputs required to compromise untraceability is $n - (k + 1)$.

The adversary therefore does not gain a significant advantage as k increases. This fundamental shift greatly enhances the resilience of the system against adversarial corruption. Figure 3 illustrates this effect, demonstrating how increasing k leads to a more pronounced difference between predictable and unpredictable partitioning schemes. Due to this, unpredictability grants the scheme the

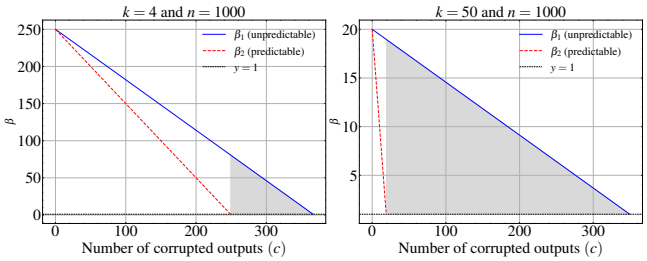

Figure 3. Illustration of the impact of corruption over untraceability (β) for an unpredictable scheme (β_1 depicted with a solid blue line) and a predictable one (β_2 depicted with a dashed red line) with varying values of $k \in \{4, 50\}$ and $n = 1000$ outputs. An unpredictable scheme is more robust to adversarial corruptions for higher values of k.

flexibility to increase the number of bins without a drastic decrease in untraceability, thus enabling better scalability as we will see with the next notion.

Typically, scalability is not framed as a security notion, as it is commonly assessed through performance measurements in non-adversarial settings. However, in our case, scalability is directly influenced by adversarial behavior. Specifically, an adversary can inject transactions that degrade system performance, leading to denial-of-service or flooding attacks [7, 12]. Given that such attacks pose a realistic threat in decentralized systems, we argue that scalability must be treated as a security property.

We define scalability as the ability of the scheme to prune bins with non-negligible probability while preserving its core security guarantees. Specifically, scalability ensures that, as transactions are conducted, bins can be removed efficiently without increasing the adversary's ability to trace spent outputs. This property is crucial for maintaining a practical and lightweight system without compromising untraceability. We now formalize this notion.

Definition 4.6 (Scalability). We denote the scalability experiment with security parameter λ by $\mathsf{Scale}(\mathcal{A}, \lambda, n, t, k)$. The experiment is played between an adversary $\mathcal{A}$ and a challenger C, proceeding as follows:

Setup. The challenger C runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ creates n outputs O of arbitrary denominations, and generates the partitioning randomness rnd. It then runs the partitioning algorithm on O into k bins

$$\mathcal{P} := \{\mathsf{bn}_1, \ldots, \mathsf{bn}_k\} \leftarrow \mathsf{Partition}(\mathsf{rnd}, O, k)$$

and sends the adversary $\mathcal{A}$ the parameters pp, the set of outputs O, the randomness rnd, and the partition $\mathcal{P}$.

Spending. The adversary $\mathcal{A}$ conducts t transactions, using oracle access that selects bins uniformly at random from the set of partitions. Each transaction produces a proof π and nullifier nul

$$(\pi_i, \mathsf{nul}_i) \leftarrow \mathsf{Spend}(\mathsf{sk}_{o_i}, o_i, \mathsf{bn}_i), \quad \text{for } i \in [t].$$

The adversary then sends the t proofs $\pi_1, \ldots, \pi_t$ and nullifiers $\mathsf{nul}_1, \ldots, \mathsf{nul}_t$ to the challenger.

Pruning. The experiment returns 1 if there exists a bin bn_i in the partition $\mathcal{P}$ where the pruning algorithm outputs 1 for a subset of the proofs and nullifiers, i.e.,

$$\exists \mathsf{bn}_i : \quad \mathsf{Prune}(\mathsf{bn}_i, \{\pi_j, t_j\}_{j \in S}) = 1, \quad \text{for } S \subseteq [t].$$

An SPS is then *scalable* if the following conditions hold

(1) the probability of success in the scalability experiment is *non-negligible* in the number t of transactions. That is, for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a constant $c > 0$ such that

$$\Pr\left[\mathsf{Scale}(\mathcal{A}, \lambda, n, t, k) = 1\right] \geq c.$$

(2) the probability of success in the double-spending experiment before the scalability experiment, $\mathsf{DSExp}(\mathcal{A}, \lambda, n, k)$ and after, $\mathsf{DSExp}^*(\mathcal{A}, \lambda, n, k)$, remains negligible for all probabilistic polynomial-time adversaries $\mathcal{A}$, such that

$$\Pr[\mathsf{DSExp}(\mathcal{A}, \lambda, n, k)] - \Pr[\mathsf{DSExp}^*(\mathcal{A}, \lambda, n, k)] \leq \mathsf{negl}(\lambda).$$

(3) the probability of success in the untraceability experiment before the scalability experiment, $\mathsf{Untrace}(\mathcal{A}, \lambda, n, k, c)$ and after, $\mathsf{Untrace}^*(\mathcal{A}, \lambda, n, k, c)$ remains negligible for all probabilistic polynomial-time adversaries $\mathcal{A}$, such that

$$\Pr[\mathsf{Untrace}(\mathcal{A}, \lambda, n, k, c)] - \Pr[\mathsf{Untrace}^*(\mathcal{A}, \lambda, n, k, c)] \leq \mathsf{negl}(\lambda).$$

The scalability experiment captures the fundamental requirement that bins must be pruned with non-negligible probability in the number of transactions without compromising double-spending and untraceability. To analyze this probability formally, we examine the probability that a given bin reaches the pruning threshold, i.e.,

is referenced as many times as its size. Since we assume outputs to be evenly distributed, the size of each bin is simply n/k.

In what follows, we establish a probabilistic bound on the success of the pruning condition. This bound is derived under the assumption that transactions reference bins uniformly at random, which is encapsulated in the above definition. Consequently, we model the number of times each bin is referenced as a binomially distributed random variable and derive the probability that at least one bin reaches the pruning threshold. The following lemma provides the formal derivation of this result and we give more details on how to derive this bound in the corresponding proof in Appendix B.

Lemma 4.7 (Probability of successful pruning). *Let an SPS operate with n outputs partitioned into k bins. Assume that transactions reference bins uniformly at random, with probability $p = 1/k$ for any bin being referenced in a transaction. We denote by t the number of transactions. Then, the probability that at least one bin becomes prunable after t transactions is given by*

$$\Pr\left[\exists i \in \{1,\ldots,k\} : X_i = \frac{n}{k}\right] = 1 - p^k,$$

where

$$p = \sum_{m=0}^{\frac{n}{k}-1} \Pr[X = m], \text{ with } X \sim B\left(t, \frac{1}{k}\right).$$

To further illustrate this result, we analyze how the probability of successful pruning evolves as the size of the partition, k, varies. Figure 4 presents this relationship for different values of k, which showcases the direct impact of partitioning on the scalability of the scheme. As expected, increasing k accelerates the pruning probability, allowing bins to reach their threshold for removal more rapidly. Conversely, smaller values of k lead to longer retention times for each bin, requiring a greater number of transactions before a bin can be pruned. This result shows the critical trade-off between untraceability and scalability: increasing k enhances efficiency by accelerating pruning, but at the cost of reducing the untraceability set size per partition. However, as shown in Lemma 4.5, randomization mitigates this drawback by preventing structured attacks and ensuring that each output's placement remains resistant to adversarial inference. Randomization is thus a crucial requirement for output partitioning and the optimal choice of k must strike a balance between maintaining strong privacy guarantees and achieving a scalable ledger.

Another aspect that has an impact on untraceability is *confidentiality*, i.e., hiding the transacted amounts. The SPS scheme works with outputs of different denominations. However, without confidentiality, that means that the partitioning should group outputs with the same denomination in the same bin. This can become challenging for outputs of rare or unique denomination. Knowing this, an adversary could attempt to conduct transactions to produce unique amounts to isolate a target. A workaround is to obfuscate the amount of each output.

We call this notion confidentiality and define it under an experiment, in which the adversary outputs two distinct denominations d_0 and d_1. The challenger picks a random bit b and creates an output with denomination d_b. The adversary wins this experiment if it is able to guess which of the two denominations has been chosen.

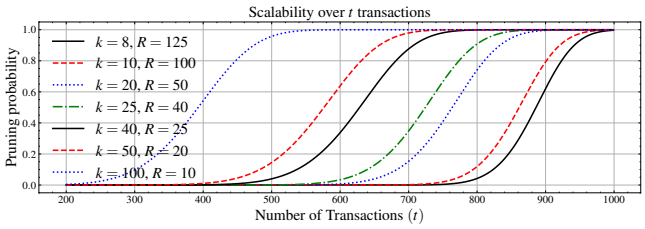

Figure 4. Scalability impact of varying the number of bins k (and thus the pruning threshold R) for a fixed number of outputs $n = 1000$. Each curve corresponds to a different value of $k \in \{8, 10, 20, 25, 40, 50, 100\}$, and shows how the probability of a bin becoming prunable evolve as transactions occur. Reducing k leads to greater retention time with curves shifting to the right.

Definition 4.8 (Confidentiality). We denote the confidentiality experiment with security parameter λ by $\mathsf{Confid}(\mathcal{A}, \lambda)$. The experiment is played between an adversary $\mathcal{A}$ and a challenger C, proceeding as follows:

Setup. The challenger runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and sends the public parameters pp to the adversary $\mathcal{A}$.

Choice. The adversary $\mathcal{A}$ selects two distinct denominations

$$\mathsf{amt}_0, \mathsf{amt}_1 \leftarrow \mathcal{A}(\mathsf{pp})$$

and submits them to the challenger.

Selection. The challenger flips a random bit $b \leftarrow\!\!\$\ \{0,1\}$ and creates an output with denomination amt_b

$$o \leftarrow \mathsf{CreateOutput}(\mathsf{pk}, \mathsf{amt}_b).$$

The challenger provides o to the adversary $\mathcal{A}$.

Challenge. The adversary $\mathcal{A}$ outputs a guess $b' \leftarrow \mathcal{A}(o, \mathsf{pp})$. It wins this experiment if $b' = b$.

An SPS is *confidential* if no probabilistic polynomial-time adversary $\mathcal{A}$ wins this experiment with greater than negligible advantage. That is, for all $\mathcal{A}$, we have:

$$\Pr\left[\mathsf{Confid}(\mathcal{A}, \lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

Finally, a widespread notion in the literature for linkable ring signatures is non-slanderability. Informally, this means that an adversary cannot produce a nullifier that links to an output it does not know the secret of. In this experiment, the challenger sends an output o to the adversary. $\mathcal{A}$ then attempts to produce a nullifer for o. In the end, the challenger spends the output with nullifier nul' and the adversary wins if $\mathsf{Link}(\mathsf{nul}', \mathsf{nul}) = 1$.

Definition 4.9 (Non-Slanderability). We denote the non-slanderability experiment with security parameter λ by $\mathsf{NonSland}(\mathcal{A}, \lambda)$. The experiment proceeds as follows between an adversary $\mathcal{A}$ and a challenger C:

Setup. The challenger C generates the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and creates an output of arbitrary denomination amt: $o \leftarrow \mathsf{CreateOutput}(\mathsf{pk}, \mathsf{amt})$. C sends the public parameters pp and the output o to the adversary $\mathcal{A}$.

Challenge. The adversary $\mathcal{A}$ produces a nullifier nul for the output o: $\mathsf{nul} \leftarrow \mathcal{A}(o, \mathsf{pp})$.

Spend. The challenger $\mathcal{C}$ spends the output o and produces a proof π and a nullifier nul': $(\pi, \mathsf{nul}') \leftarrow \mathsf{Spend}(\mathsf{sk}, o)$. The adversary wins this experiment if both nullifiers nul' and nul are linked: $1 \leftarrow \mathsf{Link}(\mathsf{nul}', \mathsf{nul})$.

An SPS is *non-slanderable* if no probabilistic polynomial-time adversary $\mathcal{A}$ wins this experiment with greater than negligible advantage. That is, for all $\mathcal{A}$, we have:

$$\Pr\left[\mathsf{NonSland}(\mathcal{A}, \lambda)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

5 Constructions

In this section, we first present cryptographic building blocks and second our tree-based construction. We prove its security in Appendix D. Finally, we discuss other potential instantiations and their weaknesses compared to our main construction.

5.1 Building blocks

Our construction in Section 5.2 leverages three fundamental cryptographic primitives: commitment schemes, argument systems, and Merkle trees. We present each of these building blocks briefly below and defer their formal security properties to Appendix C.

A commitment scheme allows users to commit to a message while keeping it hidden to others. They can later reveal the committed value, which cannot be changed once it is committed to.

Definition 5.1 (Commitment scheme). A commitment scheme is a tuple of two algorithms (Setup, Commit):

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Com.KeyGen}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter 1^λ, outputs a public and private key $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Com} \leftarrow \mathsf{Com.Commit}(\mathsf{pk}, m, r)$ is a deterministic algorithm that takes as input a public key pk, a message m, a blinding factor r, and outputs a commitment c.
- $\{0, 1\} \leftarrow \mathsf{Com.Open}(\mathsf{sk}, m, r)$ is a deterministic algorithm that takes as input a secret key sk, a message m, a blinding factor r, and returns either 0 or 1.

We furthermore require commitments to be binding and hiding. Binding means that it must be hard to find two distinct messages m_0 and m_1 that open to the same commitment Com. Hiding means that the commitment does not reveal any information about the message m. We also use non-interactive argument system for a language $\mathcal{L}_\mathcal{R}$ with witness relation $\mathcal{R}$.

Definition 5.2 (Non-Interactive Argument System). A non-interactive argument system ARG for $(x, w) \in \mathcal{R}_\mathcal{L}$, where x is a public statement about an NP language $\mathcal{L}_\mathcal{R}$ defined by the relation $\mathcal{R}$ and w is the witness, consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{ARG.Setup}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter 1^λ and outputs the public parameters pp.
- $\pi \leftarrow \mathsf{ARG.Prove}(x, w)$ is a probabilistic algorithm that takes as input the statement x and the witness w and outputs a proof π.
- $\{0, 1\} \leftarrow \mathsf{ARG.Verify}(x, \pi)$ is a deterministic algorithm that takes as input the statement x and the proof π and returns either 0 or 1.

We require ARG to satisfy three standard properties. Completeness ensures that any valid statement x with a corresponding witness w always yields a proof π such that $\mathsf{ARG.Verify}(x, \pi) = 1$. Knowledge soundness guarantees that no probabilistic polynomial-time (PPT) adversary can produce a valid proof π for a statement without knowing a corresponding witness. Zero-knowledge requires that the proof π does not reveal any information about the witness w, beyond the validity of the statement x. Finally, we use Merkle trees to authenticate data and enable efficient membership proofs.

Definition 5.3 (Merkle tree). A Merkle tree MT is an authenticated data structure that consists in three algorithms:

- $\mathsf{rt} \leftarrow \mathsf{MT.Init}(1^\lambda, X)$ takes the security parameter λ and a list of elements $X = (x_1, \ldots, x_n)$ as inputs, constructs a tree that stores $x_1, \ldots, x_n$ in the leaves, and finally outputs a root rt.
- $\mathsf{path} \leftarrow \mathsf{MT.Prove}(x, X)$ takes as input an element x, a list X, and outputs the proof path, which can prove that x is in X.
- $\{0, 1\} \leftarrow \mathsf{MT.Verify}(x, \mathsf{rt}, \mathsf{path})$ takes as input an element x, the root rt, a proof path, and outputs either 0 or 1.

5.2 Tree-based construction

In our construction, we use the above standard cryptographic primitives and demonstrate how they effectively accommodate the binning and pruning logic. First, the partitioning algorithm shuffles the list of outputs by randomly choosing a permutation using unbiased randomness rnd. Second, it arranges the shuffled list of outputs as the leaves of a Merkle tree T. Each output is assigned a Merkle subtree $T'_{\log(k)} \subseteq T$ which lies at height $\log(k)$ of T. Spending an output involves showing in zero-knowledge that one knows the opening of a commitment in the path of a Merkle subtree $T'_{\log(k)}$ and disclosing the nullifier nul. Preventing double-spending requires checking the past nullifiers of the same bin for potential conflicts. Once m proofs for a bin of size m with m distinct nullifiers have been recorded, pruning of the m outputs, proofs and nullifiers is allowed. We illustrate the partitioning of four outputs with $k = 2$, using an example. At epoch e_1, four outputs o_1, o_2, o_3 and o_4 are partitioned and arranged as the leaves of a Merkle tree with root rt_1. As shown in Figure 5, by epoch e_2, both outputs o_1 and o_2 have been spent. Since this bin has been referenced twice, these two outputs can be pruned from the ledger (depicted as dashed and grayed out). Simultaneously, new outputs o_5, o_6, o_7 and o_8 form a new Merkle tree with root rt_2. In this construction, we present a formal description of our scalable privacy-preserving payment scheme. Below, we describe the eight core algorithms that constitute our instantiation of the SPS scheme. We assume that the global state of the system st is initially empty and is subsequently updated through the consensus mechanism to incorporate newly minted outputs and nullifiers of spent outputs.

Setup and key generation. The setup algorithm initializes the system parameters. The key generation algorithm produces public and secret keys for users, as shown in Figure 6.

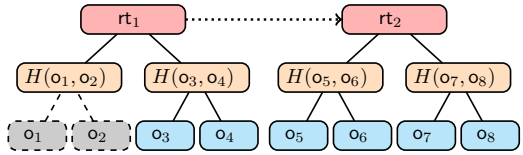

Figure 5. Illustration of our pruning mechanism across epochs. Initially at epoch e_1, a Merkle tree with root rt_1 contained four commitments o_1, o_2, o_3 and o_4. By epoch e_2, outputs o_1 and o_2 have been pruned (depicted as dashed and grayed out), leaving only commitments o_3 and o_4 in the original tree, while a new Merkle tree with root rt_2 contains fresh commitments o_5, o_6, o_7 and o_8.

Setup (1^λ)

1 : $\mathsf{pp}_1 \leftarrow \mathsf{ARG.Setup}(1^\lambda)$
2 : $\mathsf{pp} \leftarrow (1^\lambda, \mathsf{pp}_1)$
3 : $\mathsf{st}_0 \leftarrow \emptyset$
4 : **return** $(\mathsf{pp}, \mathsf{st}_0)$

KeyGen(pp)

1 : **parse** 1^λ from pp
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Com.KeyGen}(1^\lambda)$
3 : **return** $(\mathsf{pk}, \mathsf{sk})$

Figure 6. Algorithms for system initialization and user key generation.

Creating and spending outputs. Users generate outputs by committing to a secret key (by passing the public key pk to the commitment scheme), an amount and a randomly chosen nullifier. Spending an output involves proving in zero-knowledge that one knows the secret key sk corresponding to the public key pk, the amount amt and the nullifier nul of a commitment, the amount is in the valid range, and that the output is contained in the subtree (the bin). The nullifier is revealed and not kept secret to prevent double-spending. These algorithms are detailed in Figure 7.

CreateOutput(pk, amt)

1 : $\mathsf{nul} \leftarrow\!\!\$\ \mathbb{Z}_q^*$
2 : $o \leftarrow \mathsf{Com.Commit}(\mathsf{pk}, \mathsf{amt}, \mathsf{nul})$
3 : **return** (o, nul)

Spend(sk, o, nul, bn)

1 : **parse** bn **as** (T, rt)
2 : $\mathsf{arg} \leftarrow \{\mathsf{Com.Open}(\mathsf{sk}, \mathsf{amt}, \mathsf{nul}) = 1$
3 : $\quad \wedge\ \mathsf{amt} \in [\mathsf{range}]$
4 : $\quad \wedge\ \mathsf{path} = \mathsf{MT.Prove}(o, T)$
5 : $\quad \wedge\ \mathsf{MT.Verify}(o, \mathsf{rt}, \mathsf{path})\}$
6 : $\pi \leftarrow \mathsf{ARG.Prove}((\mathsf{sk}, \mathsf{amt}) : \mathsf{arg})$
7 : **return** (π, nul)

Figure 7. Algorithms for creating and spending outputs in the system.

Verification and linking. Verification guarantees the validity of a proof and ensures that no previously spent nullifier is reused. Nullifiers are extracted from the global state st. Both verification and linking, which identifies double-spending by comparing nullifiers, are depicted in Figure 8.

Verify(st, bn, π, nul) $\rightarrow \{0, 1\}$

1 : **parse** $\{\mathsf{nul}_i\}_{i=1}^n$ from st
2 : **if** $\exists \mathsf{nul}_i \in \{\mathsf{nul}_i\}_{i=1}^n : 1 \leftarrow \mathsf{Link}(\mathsf{nul}, \mathsf{nul}_i)$
3 : $\quad$ **return** 0
4 : **return** $\mathsf{ARG.Verify}(o, \mathsf{bn}, \pi)$

Link(bn, nul, nul′) $\rightarrow \{0, 1\}$

1 : **return** nul = nul′

Figure 8. Verification and linking algorithms for ensuring transaction validity.

Partitioning and pruning. Partitioning involves randomly shuffling outputs and placing them as the leaves of a Merkle tree. In turn, each bin is a Merkle subtree. Once m valid proofs and m nullifiers corresponding to a bin bn have been recorded, this bin along with all its outputs and nullifiers can be pruned. At this point, the pruning algorithm returns 1 and the corresponding node can choose to discard it from its locally replicated state. Figure 9 presents the formal algorithms for these operations.

Partition(rnd, O, k)

1 : $l \leftarrow \mathsf{shuffle}(O, \mathsf{rnd})$
2 : $T \leftarrow \mathsf{MT.Init}(l)$
3 : $\{T_i\}_{i=1}^k \leftarrow$ **for** T_i at height $\log_2(k)$
4 : $\mathsf{bn}_i \leftarrow (T_i, \mathsf{rt}_i), T_i$ with root rt_i
5 : **return** $\{\mathsf{bn}_i\}_{i=1}^k$

Prune(st, bn, $\{\pi_i\}_{i=1}^m, \{\mathsf{nul}_i\}_{i=1}^m$)

1 : **parse** bn **as** (T, rt)
2 : **if** $|\{\pi_i\}_{i=1}^m| \neq |\mathsf{bn}|$: **return** 0
3 : **if** $|\{\mathsf{nul}\}_{i=1}^m| \neq |\mathsf{bn}|$: **return** 0
4 : **if** $\exists i \in [m], \quad \mathsf{Verify}(\mathsf{st}, \mathsf{bn}, \pi_i) = 0$: **return** 0
5 : **if** $\exists i, j \in [m], i \neq j, \quad \mathsf{Link}(\mathsf{bn}, \mathsf{nul}_i, \mathsf{nul}_j) = 1$: **return** 0
6 : **return** 1

Figure 9. Algorithms for partitioning outputs into bins and pruning completed bins.

5.3 Other constructions

One can explore alternative authenticated data structures for our scheme, for example cryptographic accumulators or vector commitments.

Cryptographic accumulators. This data structure is particularly attractive since accumulators allow each bin to be represented as a constant-size digest. Yet, when employing accumulators, we encounter several challenges. Consider the RSA-based accumulator, originally introduced by Camenisch and Lysyanskaya [11] and later refined by Boneh *et al.* [5] for trustless environments like blockchains. To ensure its security, the accumulator must be instantiated with a 2048-bit security parameter, which means that each bin and its associated witness will also be 2048 bits in size. As a result, its scalability advantage over a Merkle tree (instantiated with SHA-256) only becomes apparent when bins contain more than 256 outputs. Moreover, RSA accumulators are secure only when they accumulate primes, which requires additional proofs to verify the primality of the outputs – a requirement that increases the proof overhead. Furthermore, to this day, no efficient post-quantum accumulator constructions are currently known. Finally, despite mitigations via class-group instantiation [38], RSA accumulators typically rely on a trusted setup to initialize the RSA modulus.

Vector commitments. Vector commitments (VC) offer a structured way to authenticate the bins by committing to the randomly shuffled list of outputs. One possible approach is to commit to the entire list at once, but doing so would make the pruning of individual bins difficult and offer little advantage in terms of storage. Another approach is to instantiate each bin as a separate vector commitment, allowing outputs within a bin to be referenced more efficiently. However, this comes with the same drawback as cryptographic accumulators: the size of each bin would be dictated by the security parameter, making them quite large. Moreover, most known VC constructions require a trusted setup unless instantiated using a class-group of unknown order, which adds further complexity.

6 Performance simulation

We conduct a simulation to evaluate the partitioning and pruning mechanisms in practice. For this purpose, we analyze Monero transactions over a complete year, spanning the period from May 1, 2024, to April 30, 2025. This section presents our methodology and findings.

Preliminary storage savings. Monero currently uses ring signatures with a fixed ring size of 16, which sets $\beta = 16$ in the absence of corruption, i.e., $c = 0$. Each transaction must explicitly reference all 16 ring members. In contrast, our scheme requires only one reference to the bin. Assuming the bin is identified by a 32 B Merkle tree root as in Section 5, this results in a 480 B saving per transaction. Moreover, the transaction size depends only on the proof size, which grows logarithmically with the bin size under current zero-knowledge argument techniques [8]. In contrast, using explicit decoy references leads to a linear growth in transaction size with the number of decoys. Notably, despite the reduction in transaction size, the computational cost is about the same, as the underlying cryptographic operations remain largely similar and are not significantly affected by the switch from explicit decoy references to bin references.

Setup. We extract transaction data from the Monero blockchain using the `monerorpc` library and export it as a structured CSV file for analysis. We segment the chain into epochs of 5 blocks each, corresponding to approximately 10 minutes. This approach strikes a practical balance: it allows sufficient outputs to accumulate for effective partitioning, while minimizing the delay during which newly created outputs remain in the buffer and thus unavailable for spending. We discuss different strategies and their respective benefits in Section 8. For this initial simulation, we assume $c = 0$, deferring the analysis of a system under attack to Section 7. Figure 10 displays the number of outputs observed per epoch across the simulation period. We enforce a strict untraceability threshold of $\beta \geq 16$ for every epoch, matching Monero's current ring size. This parameter determines the number of bins k used for partitioning. To maintain this threshold as transaction volume fluctuates, we adjust k dynamically by setting $k = \lfloor n/\beta \rfloor$, in line with Lemma 4.5 for the case $c = 0$. The resulting values of k are plotted in Figure 10, on the y-axis on the right.

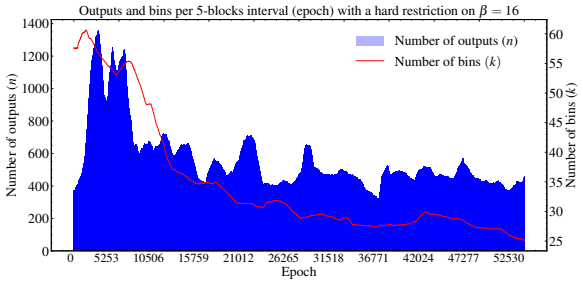

Figure 10. Distribution of outputs (n) and corresponding number of bins (k) per 5-blocks interval over the simulation period. The left y-axis shows the number of outputs, while the right y-axis shows the dynamically adjusted number of bins required to maintain $\beta \geq 16$.

Transaction simulation. To simulate realistic transaction spending patterns, we analyze the Monero dataset chronologically by epoch. Since Monero's ring signatures conceal which outputs are actually spent, we need to simulate which output from previous epochs is actually spent. We thus create a simulation model that preserves the authentic structure of Monero transactions using the real number of inputs per transaction, but that randomly select outputs from the ledger. The random selection is based on a gamma distribution that closely aligns with the spending behavior, characterized by a shape parameter of 19.28 and a rate parameter of 1.61, as detailed by Möser *et al.* [28]. Similarly, the default wallet application of Monero employs a gamma distribution with the same parameters for its decoy-sampling distribution. Our simulation randomly selects outputs from previous epochs as inputs and tracks spent outputs and their corresponding bins. Our approach focuses on tracking individual outputs rather than sampling bins directly. This distinction is crucial as it allows us to differentiate between outputs that are spent and those that become prunable when their bin reaches the reference threshold. In summary, for each transaction with m inputs from the dataset, our simulation executes the following steps:

(1) Randomly selects m outputs from previous epochs as inputs, using the true number of inputs from the dataset;

(2) Marks these outputs as spent;
(3) Updates the reference count for each corresponding bin; and
(4) Marks outputs as prunable once their bin reaches the reference threshold of 16.

Pruning over time. Figure 11 shows the evolution of the cumulative number of spent outputs and prunable outputs across the 52530 epochs. The gap between the two curves corresponds to outputs that have been spent but cannot be pruned because their bins have not been referenced in 16 transactions. This gap remains significant at the beginning since the random sampling picks outputs that are close to the current epoch due to the gamma distribution. By the

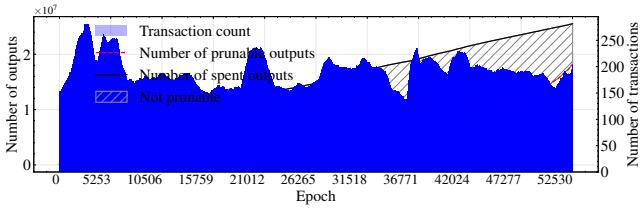

Figure 11. Cumulative number of spent and prunable outputs over time. The widening gap reflects spent outputs that cannot yet be pruned because their bin has not reached the reference threshold of 16.

end of our simulation period, approximately 85% of outputs are spent. This number aligns with Bitcoin's statistics, where only a small portion of outputs remain in the UTXO set [16]. Our results further indicate that roughly 60% of outputs are prunable. The remaining 15 percentage points correspond to spent outputs whose bins have not yet reached the pruning threshold, and thus cannot be discarded. As shown in Figure 12, without pruning the number of outputs would grow to about 779 MB, with spent outputs accounting for more than 1.5 GB with nullifiers included. By contrast, our pruning mechanism allows discarding 1.1 GB (nullifiers included), implying that non-prunable outputs with their nullifiers account for a mere 446.4 MB. This gap illustrates the substantial storage savings enabled by pruning. Overall, these results demonstrate that our pruning strategy offers significant scalability gains. This holds especially for privacy-preserving cryptocurrencies that lack pruning mechanisms and would have to store over 1.6 GB of data (896.4+779.1 MB) against 349.5 MB with pruning. More importantly, binning and pruning did not have to compromise on privacy in this case since we were able to maintain an untraceability level of 16.

Discussion. Our simulation results confirm that partitioning and pruning enable scalable deployment while maintaining the same level of untraceability as currently set in Monero. First, the system lets itself naturally partitionned over the multiple epochs while providing enough outputs to guarantee untraceability to the given threshold. The adjustment of k ensures that the required threshold β is consistently enforced without manual tuning. This is easily enforceable at the protocol level during the consensus mechanism, a validator would simply compute $k = \lfloor n/\beta \rfloor$ with β being a system parameter. Second, our simulation shows that pruning enables a 60% drop in retained outputs and demonstrates the value of integrating partitioning to enable the pruning logic. Lastly, the partitioning

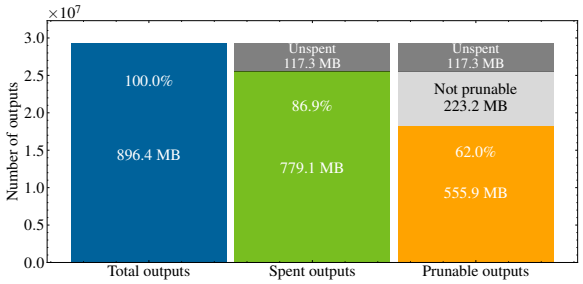

Figure 12. Storage impact of pruning at the end of the simulation period. The blue bar (left) represents the total number of outputs, the green bar (middle) the subset that has been spent, and the orange bar (right) those that are prunable. The difference between the blue and green bars corresponds to unspent outputs, while the difference between the green and orange bars reflects spent outputs that are not prunable yet.

of the output set eliminates the need for explicit decoy references in transactions, yielding substantial bandwidth savings all while maintaining similar computational costs.

7 Attack resilience

In this section, we extend the simulation to adversarial settings with $c > 0$. We show how dynamic adjustment of k, based on an estimated number of corrupted outputs, continues to enforce privacy even under worst-case corruption scenarios.

Estimating the number of corrupted outputs $\hat{c}$ allows setting k to maintain a threshold on the untraceability of the scheme, i.e., β, while ensuring scalability over time. This parameter is particularly useful as it can be adjusted to mitigate flooding attacks. For instance, to maintain a minimum untraceability level of 16 one should ensure that $k = \lfloor n - e\hat{c}/16 \rfloor$.

In March 2024, Monero likely suffered a flooding attack. As depicted in Figure 13, the number of transactions surged dramatically within a short period of several weeks. This anomaly suggests the presence of a malicious actor generating an excessive volume of transactions [32]. We use this event as a case study to assess the robustness of the partitioning mechanism in handling adversarial conditions. To estimate the number of corrupted outputs, we apply

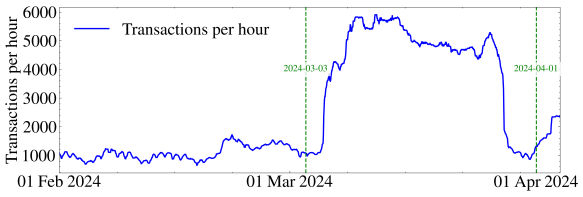

Figure 13. Plot of the number of transactions in Monero: the sharp increase in March 2024 suggests a flooding attack.

a simple and rather conservative metric. Specifically, we define the estimated corruption $\hat{c}$ as the number of outputs exceeding the arithmetic mean number of outputs over a window of 100 days observed prior to the attack. This method, while straightforward, provides an

upper-bound approximation of corruption. Figure 14 illustrates the daily evolution of $\hat{c}$ during March 2024, showing a significant spike aligning with the transaction surge. This estimation provides a crucial insight into the potential scale of corruption. Without dynamic adaptation based on $\hat{c}$, the effective untraceability level β would decrease during flooding attacks. Our estimation approach can be compared with the analysis presented by Rucknium [35]. That study applies a slightly more sophisticated metric to estimate corruption levels during the attack, but our approach yields a roughly similar number of corrupted outputs. Our estimation metric is also lightweight, which aligns with an on-chain dynamic adjustment of k. Figure 15 presents a hypothetical adaptation of k as a function of $\hat{c}$. Note that here k should be rounded down, since the number of bins is a discrete number. By adjusting k accordingly, the scheme mitigates the impact of a flooding attack while preserving its scalability and privacy guarantees to the expense of increasing the size of spending proofs. The flexibility of k therefore presents an effective countermeasure against adversarial flooding. Future research could explore alternative estimation techniques to further refine the accuracy of $\hat{c}$ and dynamic on-chain adjustment of k to enhance the resilience of the system. The next section discusses practical considerations, including alternative methods to mitigate flooding attacks.

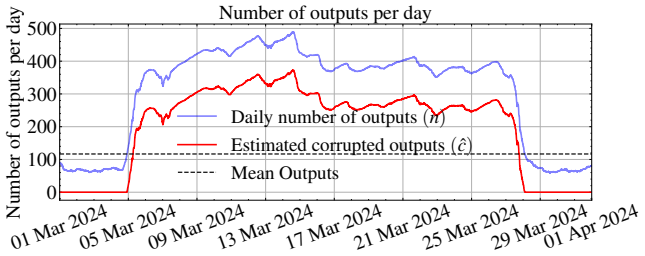

Figure 14. Plot of estimated number of corrupted outputs $\hat{c}$ during Monero flooding attack of March 2024: the estimation of the number of corrupted outputs $\hat{c}$ is the positive difference between the number of outputs and the average number of outputs.

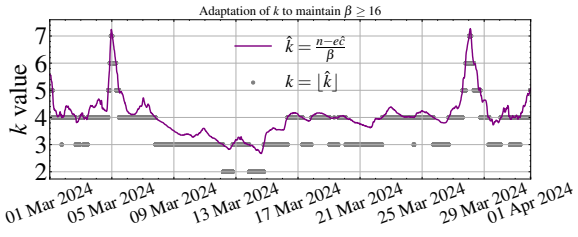

Figure 15. Plot of potential adaptation of k based on the estimated corruption $\hat{c}$: this adjustment ensures a controlled level of untraceability despite adversarial interference. The value k should be rounded down since it represents the number of bins in the partition.

8 Practical considerations

This section discusses improvement ideas, clarifies some technical details and addresses some practical concerns of our scheme.

Flooding attacks. As discussed earlier, shuffling outputs ensures an even distribution of adversarial outputs across partitions, thus limiting the impact of flooding attacks. However, a well-resourced adversary can still attempt to overflood each partition, corrupting $n/k - 1$ adversarial outputs per bin. A natural defense mechanism is to increase transaction fees, which will raise the cost of such an attack. Yet, an adversary with sufficient financial resources may still scale the attack despite elevated fees. Another approach is requiring proof of work for transaction submissions, but this could exclude low-resource clients and increase the chain's ecological footprint. As seen before, a built-in countermeasure is reducing k, which limits the adversary's control over partitions at the cost of increasing the size of spending proofs. However, this adjustment impacts long-term scalability because of the increased proof size and the lower k-value. Ultimately, no perfect solution exists against a well-funded adversary, as even membership proofs over the entire set of coins may prove insufficient if the adversary's monetary resources are substantial.

Account-based models. In Section 4, we established that our formalization of inputs and outputs remains ledger-model agnostic. We recognize, however, that our terminology and formalization bears stronger resemblance to UTXO-based models, thus we provide clarification on integration with account-based models. Account-based models supporting confidentiality typically use commitments to account balances. Transactions involve recomputing two commitments while proving that the sum of old and the sum of new commitments preserve balance equality. This approach appears in privacy-preserving cryptocurrencies (e.g., Quisquis [19] and Nopenena [2]) and numerous CBDC prototypes (e.g., PRCash [40], Platypus [41], UTT [36] and presumably others). In Quisquis and Nopenena, untraceability is achieved by selecting decoy accounts for randomization without impeding owners' spending capabilities. Upon account randomization by another user, owners must retrieve their account's updated cryptographic representation from the ledger. This technique is particularly appealing as the ledger need only maintain the most current version of these commitments, yielding extremely compact storage requirements. However, any user can designate any account as a decoy, potentially preventing *slower* owners from accessing their accounts – as highlighted by Bünz *et al.* [7]. To address this vulnerability, accounts can be implemented as commitments to balances augmented with nullifiers. Spending from an account requires releasing said nullifier, with untraceability enforced by selecting decoy accounts and proving knowledge of one commitment's opening. Mitigating the aforementioned availability issue thus necessitates compromising on ledger representation succinctness, as we cannot discard previous account representations that may remain valid. Here our partitioning and pruning techniques become relevant. Each account commitment would be assigned to a bin alongside $m - 1$ similar commitments. Once a bin has been used in m transactions, the ledger can prune these m commitments and their corresponding nullifiers.

Further optimizing pruning. Pruning can only occur once m outputs within a bin have been spent, this is known only when a bin has been referenced m-many times. This can create a bottleneck where unspent outputs indefinitely delay scalability because a bin has been referenced less than m times. A natural approach to mitigate this issue is to encourage users to spend their outputs by offering incentives such as reduced fees or minor rewards in newly minted coins. However, introducing such mechanisms could alter spending patterns in unpredictable ways, potentially shaping a new transaction economy that warrants further investigation. A more aggressive yet straightforward solution is to impose an expiration date on bins. Once a bin surpasses this expiration threshold, it would be pruned unconditionally, regardless of whether all outputs have been spent. Users willing to preserve their funds could transfer their outputs to themselves, and a notification in their wallet application could alert them of a bin soon expiring. This approach guarantees a steady pruning rate, but it also introduces concerns. In particular, if a bin is pruned while it still contains some unspent outputs, these outputs would be permanently burnt from the ledger. If the ledger is confidential – output values are hidden – this could result in a loss of transparency about the total money in circulation, which may affect estimations such as the market capitalization of the cryptocurrency.

Generating randomness. A key component of our scheme is the generation of unbiased randomness to ensure unpredictability in partitioning. While we have assumed access to such randomness, producing it in an open setting requires care. The source must be publicly verifiable and resistant to manipulation. Blockchain-derived variables, such as block hashes or timestamps, are often insufficient, as adversaries can influence them. Several existing chains embed protocol-level randomness mechanisms that meet these requirements. For instance, Ethereum 2.0 uses RanDAO [15], the Internet Computer includes onchain randomness in every block [20], and SUI provides native randomness [25]. These examples demonstrate that secure randomness generation is feasible and already deployed in practice. External solutions like drand [31] can also be used if the protocol does not embed randomness directly. However, randomness must be available at each epoch and synchronized with the protocol to avoid safety or liveness issues. We discuss concrete deployment options in the next section.

Scheduling epochs and choosing k. In the simulation outlined in Section 6, we implemented a fixed epoch schedule with a 5-block interval. This approach effectively provides sufficient outputs for partitioning the set while adhering to a strict constraint on the untraceability level, denoted as β. However, this fixed scheduling may not be optimal for systems experiencing low transaction volumes. In such scenarios, a dynamic scheduling approach could be more beneficial, initiating a new epoch as soon as the number of available outputs exceeds β.

Furthermore, it is possible to maintain a single bin per epoch by setting $k = 1$ until the system reaches sufficient inertia. In this scenario, there is no need to generate randomness. The choice of k should otherwise primarily depend on the desired level of untraceability, β, which is a crucial system parameter. The value of k will naturally vary based on the number of outputs in the current epoch, denoted as n, and the estimated number of corrupted outputs, $\hat{c}$. Adjustments to β and the estimation metric for $\hat{c}$ are managed by a mechanism within the consensus protocol. Increasing β will result in a smaller k and a longer bin retention time, which can affect scalability. Similarly, adopting a more conservative estimation metric for $\hat{c}$ will also reduce k. Therefore, system designers should carefully consider the selection of β and the estimation metric to effectively balance these trade-offs.

9 Deployment

Deploying the scheme in existing privacy-preserving cryptocurrencies would probably require significant modifications to their protocol. We sketch a deployment path using Monero and Zcash as examples. In both systems, miners operate in pools, which are naturally suited to run a distributed random beacon (DRB). Participants should contribute entropy to the DRB, which remains secure as long as one participant is honest. The DRB runs concurrently with transaction collection and finalizes at the end of the epoch. Once the DRB is released, no new transactions should be accepted in the current epoch. This prevents adversaries from reacting to the revealed randomness and flooding bins accordingly.

The block producer then uses the generated randomness to partition outputs and broadcasts the sealed block with the necessary information. This includes the epoch number, the randomness beacon, and potentially any auxiliary data required for verification. In Monero, this step can follow the process described in Section 6. Untraceability is preserved as long as there are enough outputs to at least fill a bin. In Zcash, however, partitioning weakens privacy. The current model requires proving ownership over the full set of shielded outputs, and restricting this set to a partition – even a large one – reduces the size of the untraceability set.

Pruning requires no additional signaling from the block producer or the validators. Once all outputs in a bin are spent and the corresponding nullifiers appear on-chain, the bin becomes obsolete and can be discarded locally. If an adversary attempts to double-spend from a pruned bin, two outcomes are possible. Either the validator has pruned the bin and rejects the transaction because the bin is missing, or has retained it and rejects it upon detecting a duplicated nullifier. The pruning logic is thus implicit in the ledger state and does not require coordination.

Despite requiring substantial protocol-level modifications, the deployment reuses core components of Monero and Zcash and remains compatible with their consensus and transaction logic.

10 Conclusion

In this work, we presented a scalable privacy-preserving payment scheme that directly addresses the challenge of ledger growth and graph-based attacks in privacy-focused cryptocurrencies. By partitioning the set of outputs into immutable, fixed-size groups using verifiable randomness, our approach mitigates both passive and active adversarial attacks while enabling efficient pruning of spent outputs. Our formalization of key security notions provides a robust framework for analyzing privacy guarantees in such systems and shows how randomized partitioning plays a key role in both privacy and scalability. Finally, our evaluation and analysis using a dataset of real transactions shows the storage benefits and resilience of our scheme against large-scale flooding attacks.

Acknowledgments

We would like to express our sincere gratitude to Jayamine Alupotha for the insightful discussions that greatly contributed to the development of this work. We also extend our appreciation to Christian Sillaber and Mirjam Eggen for their engaging and thought-provoking discussions, which enriched our perspective on the subject. Additionally, we thank the anonymous reviewers for their valuable suggestions and constructive comments, which have significantly enhanced the quality and clarity of this paper. This work has been supported by the Initiative for Cryptocurrencies and Contracts (IC3).

References

[1] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. 2017. Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10625)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 357–379. https://doi.org/10.1007/978-3-319-70697-9_13

[2] Jayamine Alupotha, Mathieu Gestin, and Christian Cachin. 2024. Nopenena Untraceable Payments: Defeating Graph Analysis with Small Decoy Sets. *IACR Cryptol. ePrint Arch.* (2024), 903. https://eprint.iacr.org/2024/903

[3] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. 1999. Balanced Allocations. *SIAM J. Comput.* 29, 1 (1999), 180–200. https://doi.org/10.1137/S0097539795288490

[4] 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, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, Berkeley, CA, USA, 459–474. https://doi.org/10.1109/SP.2014.36

[5] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Santa Barbara, CA, USA, 561–586. https://doi.org/10.1007/978-3-030-26948-7_20

[6] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. 2020. Single Secret Leader Election. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*. ACM, New York, NY, USA, 12–24. https://doi.org/10.1145/3419614.3423258

[7] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, Kota Kinabalu, Malaysia, 423–443. https://doi.org/10.1007/978-3-030-51280-4_23

[8] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 315–334. https://doi.org/10.1109/SP.2018.00020

[9] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer. https://doi.org/10.1007/978-3-642-15260-3

[10] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *J. Cryptol.* 18, 3 (2005), 219–246. https://doi.org/10.1007/S00145-005-0318-0

[11] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2442)*, Moti Yung (Ed.). Springer, Santa Barbara, CA, USA, 61–76. https://doi.org/10.1007/3-540-45708-9_5

[12] João Otávio Massari Chervinski, Diego Kreutz, and Jiangshan Yu. 2021. Analysis of transaction flooding attacks against Monero. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021*. IEEE, Sydney, Australia, 1–8. https://doi.org/10.1109/ICBC51069.2021.9461084

[13] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 75–92. https://doi.org/10.1109/SP46215.2023.10179419

[14] Sherman S. M. Chow, Christoph Egger, Russell W. F. Lai, Viktoria Ronge, and Ivy K. Y. Woo. 2023. On Sustainable Ring-Based Anonymous Systems. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*. IEEE, Dubrovnik, Croatia, 568–583. https://doi.org/10.1109/CSF57540.2023.00035

[15] RanDAO contributors. 2025. RanDAO: Decentralized Random Number Generator. https://github.com/randao/randao. Accessed: 2025-05-14.

[16] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. 2018. Analysis of the Bitcoin UTXO Set. In *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10958)*, Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala (Eds.). Springer, 78–91. https://doi.org/10.1007/978-3-662-58820-8_6

[17] Benjamin E. Diamond. 2021. Many-out-of-Many Proofs and Applications to Anonymous Zether. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, San Francisco, CA, USA, 1800–1817. https://doi.org/10.1109/SP40001.2021.00026

[18] Christoph Egger, Russell W. F. Lai, Viktoria Ronge, Ivy K. Y. Woo, and Hoover H. F. Yin. 2022. On Defeating Graph Analysis of Anonymous Transactions. *Proc. Priv. Enhancing Technol.* 2022, 3 (2022), 538–557. https://doi.org/10.56553/POPETS-2022-0085

[19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. 2019. Quisquis: A New Design for Anonymous Cryptocurrencies. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11921)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, Kobe, Japan, 649–678. https://doi.org/10.1007/978-3-030-34578-5_23

[20] DFINITY Foundation. 2025. Internet Computer Consensus: Onchain randomness. https://internetcomputer.org/docs/building-apps/network-features/randomness. Accessed: 2025-05-14.

[21] Daira Hopwood, Sean Bow, Taylor Hornby, and Nathan Wilcox. 2021. Zcash Protocol Specification. https://zips.z.cash/protocol/protocol.pdf.

[22] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. 2024. SoK: Public Randomness. In *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024*. IEEE, 216–234. https://doi.org/10.1109/EUROSP60621.2024.00020

[23] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Santa Barbara, CA, USA, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[24] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. 2017. A Traceability Analysis of Monero's Blockchain. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10493)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer, Oslo, Norway, 153–173. https://doi.org/10.1007/978-3-319-66399-9_9

[25] Mysten Labs. 2025. Sui: On-Chain Randomness. https://docs.sui.io/guides/developer/advanced/randomness-onchain. Accessed: 2025-05-14.

[26] Arjen K. Lenstra and Benjamin Wesolowski. 2015. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.* (2015), 366. http://eprint.iacr.org/2015/366

[27] Yacov Manevich, Jason Karl Yellick, and Angelo De Caro. 2024. Privacy-Preserving Payment Scheme. https://patents.google.com/patent/US11968307/en. https://patents.google.com/patent/US11968307/en

[28] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. 2018. An Empirical Analysis of Traceability in the Monero Blockchain. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 143–163. https://doi.org/10.1515/POPETS-2018-0025

[29] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. Whitepaper. http://bitcoin.org/bitcoin.pdf.

[30] S. Noether, S. Noether, and A. Mackenzie. 2014. A note on chain reactions in traceability in cryptonote 2.0. *Research Bulletin* (2014). https://www.getmonero.org/resources/research-lab/pubs/MRL-0001.pdf

[31] League of Entropy. 2019. drand - Distributed Randomness Beacon. https://drand.love/. Accessed: 2025-02-28.

[32] Luke "Kayaba" Parker. 2024. Full-Chain Membership Proofs Development. https://www.getmonero.org/2024/04/27/fcmps.html Accessed: 2025-05-27.

[33] Krzysztof Pietrzak. 2019. Simple Verifiable Delay Functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA (LIPIcs, Vol. 124)*, Avrim Blum (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 60:1–60:15. https://doi.org/10.4230/LIPICS.ITCS.2019.60

[34] Ronald L. Rivest, Adi Shamir, and Yael Tauman. 2001. How to Leak a Secret. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on*

the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2248), Colin Boyd (Ed.). Springer, Gold Coast, Australia, 552–565. https://doi.org/10.1007/3-540-45682-1_32

[35] Rucknium. 2024. March 2024 Suspected Black Marble Flooding Against Monero: Privacy, User Experience, and Countermeasures. https://github.com/Rucknium/misc-research/blob/main/Monero-Black-Marble-Flood/pdf/monero-black-marble-flood.pdf. https://github.com/Rucknium/misc-research/blob/main/Monero-Black-Marble-Flood/pdf/monero-black-marble-flood.pdf

[36] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. 2022. UTT: Decentralized Ecash with Accountable Privacy. *IACR Cryptol. ePrint Arch.* (2022), 452. https://eprint.iacr.org/2022/452

[37] Saravanan Vijayakumaran. 2023. Analysis of CryptoNote Transaction Graphs Using the Dulmage-Mendelsohn Decomposition. In *5th Conference on Advances in Financial Technologies, AFT 2023, October 23-25, 2023, Princeton, NJ, USA (LIPIcs, Vol. 282)*, Joseph Bonneau and S. Matthew Weinberg (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Princeton, NJ, USA, 28:1–28:22. https://doi.org/10.4230/LIPICS.AFT.2023.28

[38] Benjamin Wesolowski. 2019. Efficient Verifiable Delay Functions. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, 379–407. https://doi.org/10.1007/978-3-030-17659-4_13

[39] François-Xavier Wicht, Zhipeng Wang, Duc Viet Le, and Christian Cachin. 2024. A Transaction-Level Model for Blockchain Privacy. In *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 14745)*, Jeremy Clark and Elaine Shi (Eds.). Springer, 293–310. https://doi.org/10.1007/978-3-031-78679-2_16

[40] Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. 2019. PRCash: Fast, Private and Regulated Transactions for Digital Currencies. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, 158–178. https://doi.org/10.1007/978-3-030-32101-7_11

[41] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. 2022. Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 2947–2960. https://doi.org/10.1145/3548606.3560617

[42] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. 2019. New Empirical Traceability Analysis of CryptoNote-Style Blockchains. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, Frigate Bay, St. Kitts and Nevis, 133–149. https://doi.org/10.1007/978-3-030-32101-7_9

A Generalization

The binning structure introduced in this work can be extended to a broader class of privacy-preserving mechanisms in which a set of agents perform anonymous actions. Specifically, this generalization applies to any setting where:

(1) The action benefits from privacy by being indistinguishable among a set of other agents performing the same action.
(2) Each agent is limited in the number of times they can conduct the action.
(3) A mechanism ensures that the prescribed limit is not exceeded.

By leveraging these properties, we can construct an efficient privacy-preserving execution model based on partitions that reduces data overhead while maintaining strong anonymity guarantees.

The benefit of this approach becomes particularly evident in long-running systems with a large number of agents. Without partitioning, privacy-preserving mechanisms often require accumulating and storing data for all past actions, leading to monotonically growing overhead. Instead, the binning structure ensures that some data can be pruned once a set of conditions is fulfilled. This abstraction demonstrates that binning is a general technique applicable beyond our specific payment scheme and thus extends to a variety of privacy-preserving actions in decentralized and regulated systems.

B Proofs of theorems

Proof of Lemma 4.5. This proof takes elements of typical occupancy problems as for instance given by Azar *et al.* [3]. Based on $\mathsf{Untrace}(\mathcal{A}, \lambda, n, c, k)$, the adversary may corrupt c out of n outputs. If the scheme is unpredictable, then each corrupted output j falls in bin i with probability $\frac{1}{k} + \mathsf{negl}(\lambda)$. We analyze here the maximum number of corrupted outputs in a bin. Let X_i denote the number of corrupted outputs in bin i. Our goal is to analyze $\max X_i$. We omit the negligible term in the following and deem it implicit. Clearly, we have

$$E[X_i] = \frac{c}{k}.$$

By the union bound, for every $C > 0$, we have

$$\Pr[X_i \geq C] \leq \sum_{i=1}^{k} \Pr[X_i \geq C].$$

Therefore to estimate the probability that the maximum number of corrupted outputs is larger than some value, it is enough to focus on a single bin and show that for this bin, the probability that the number of corrupted outputs in bin i exceeding C is very small. We analyze the probability distribution of the number of corrupted outputs in a fixed bin i using a Chernoff bound. We thus obtain

$$\Pr[X_i \geq (1+\delta) \cdot E[X_i]] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{\frac{c}{k}}.$$

Assuming $\delta \geq e - 1$, this gives

$$\frac{e^{\delta}}{(1+\delta)^{1+\delta}} = \frac{1}{1+\delta} \cdot \left(\frac{e}{1+\delta} \right)^{\delta} \leq \frac{1}{1+\delta} \leq \frac{1}{2}.$$

For $c \geq 2k \log_2 k$, this simplifies to

$$\Pr[X_i \geq (1+\delta) \cdot E[X_i]] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{\frac{c}{k}} \leq 2^{-\frac{c}{k}} \leq \frac{1}{k^2}.$$

Using the union bound, we obtain

$$\Pr[C \geq \frac{e \cdot c}{k}] = \Pr[X_i \geq \frac{e \cdot c}{k}] \leq \sum_{i=1}^{c} \Pr[X_i \geq \frac{e \cdot c}{k}] \leq k \cdot \frac{1}{k^2} = \frac{1}{k}.$$

Thus, using the complement event we conclude that

$$\Pr[C < \frac{ec}{k}] \geq 1 - \frac{1}{k}.$$

Consequently, for every $c \geq 2k \log_2 k$, the minimum number of honest outputs per bin satisfies

$$\beta < \frac{n}{k} - \frac{ec}{k} = \frac{n - ec}{k}$$

with probability at least $1 - \frac{1}{k}$, which establishes the desired bound, i.e.,

$$\Pr[\beta < \frac{n - ec}{k}] \geq 1 - \frac{1}{k}.$$

□

Proof of Lemma 4.7. Assuming that transactions reference bins uniformly at random, a bin is referenced by a transaction with probability $p = \frac{1}{k}$. At most $t \leq n$ transactions may take place over k bins. Every bin contains $R = \frac{n}{k}$ outputs and can be pruned once it has been referenced R times, i.e., is prunable. We want to compute the probability that a bin is prunable after t transactions. Let X_i be the random variable denoting the number of times a bin i has been referenced. Furthermore, we can express X_i as the sum of random variables $X_{i1} + \ldots + X_{it}$, where

$$X_{ij} = \begin{cases} 1 \text{ if output } j \text{ falls in bin } i, \\ 0 \text{ otherwise.} \end{cases}$$

The variables X_i has thus binomial probability distribution $X_i \sim B\left(t, \frac{1}{k}\right)$.

The probability that bin i is *not* prunable, i.e., $\Pr[X_i < R]$ can thus be computed using the following observation

$$\Pr[X_i < R] = \sum_{m=0}^{R-1} \Pr[X_i = m]$$

where $\Pr[X_i = m]$ is given by the binomial probability mass function with $p = \frac{1}{k}$

$$\Pr[X_i = m] = \binom{t}{m} p(1-p)^{t-m}.$$

Since bins are independent, we obtain the probability that no bin has been referenced R times with

$$\Pr[\forall i \in \{1, \ldots, k\} : X_i < R] = \prod_{i=1}^{k} \Pr[X_i < R] = \Pr[X_i < R]^k.$$

This gives us

$$\Pr[\exists i \in \{1, \ldots, k\} : X_i \geq R] = 1 - \Pr[X_i < R]^k.$$

However, a bin cannot be referenced more than R times. Taking that into account the above probability becomes

$$\Pr[\exists i \in \{1, \ldots, k\} : X_i = R] = 1 - \Pr[X_i < R]^k.$$

□

C Standard cryptographic primitives

We recall the standard definitions of cryptographic primitives used in this work, including commitment schemes, argument systems, and Merkle trees and include their formal security properties.

A commitment scheme allows users to commit to a message while keeping it hidden to others. They can later reveal the committed value, which cannot be changed once it is committed to.

Definition C.1 (Commitment scheme). A commitment scheme is a tuple of two algorithms (Setup, Commit):

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Com.KeyGen}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter 1^λ, outputs a public and private key $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Com} \leftarrow \mathsf{Com.Commit}(\mathsf{pk}, m, r)$ is a deterministic algorithm that takes as input a public key pk, a message m, a blinding factor r, and outputs a commitment c.
- $\{0, 1\} \leftarrow \mathsf{Com.Open}(\mathsf{sk}, m, r)$ is a deterministic algorithm that takes a public key pk, a message m, a blinding factor r, and returns either 0 or 1.

A commitment is furthermore binding and hiding. Binding means that it must be hard to find two distinct messages m_0 and m_1 that open to the same commitment Com. Hiding means that the commitment does not reveal any information about the message m.

Definition C.2 (Binding). A commitment scheme is binding for all adversary if,

$$\Pr\left[\begin{array}{c|c} m_0 \neq m_1 \wedge \mathsf{Com.Commit}(\mathsf{pk}, m_0, r_0) = c & \mathsf{pp} := (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Com.KeyGen}(1^\lambda) \\ \wedge\, r_0 \neq r_1 \wedge \mathsf{Com.Commit}(\mathsf{pk}, m_1, r_1) = c & (c, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(1^\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition C.3 (Hiding). A commitment scheme is hiding for all adversary if,

$$\Pr\left[\begin{array}{c|c} b' = b & \begin{array}{c} \mathsf{pp} := (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Com.KeyGen}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \\ b \leftarrow\$ \{0, 1\} \\ c \leftarrow \mathsf{Com.Commit}(\mathsf{pk}, m_b, r) \\ b' \leftarrow \mathcal{A}(c) \end{array} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We define a non-interactive argument system for a language $\mathcal{L}_\mathcal{R}$ with witness relation $\mathcal{R}$.

Definition C.4 (Non-Interactive Argument System). A non-interactive argument system ARG for $(x, w) \in \mathcal{R}_\mathcal{L}$, where x is a public statement about an NP language $\mathcal{L}_\mathcal{R}$ defined by the relation $\mathcal{R}$ and w is the witness, consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{ARG.Setup}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter 1^λ and outputs the public parameters pp.
- $\pi \leftarrow \mathsf{ARG.Prove}(x, w)$ is a probabilistic algorithm that takes as input the statement x and the witness w and outputs a proof π.
- $\{0, 1\} \leftarrow \mathsf{ARG.Verify}(x, \pi)$ is a deterministic algorithm that takes as input the statement x and the proof π and returns either 0 or 1.

We now define the required properties for a non-interactive argument system.

Definition C.5 (Completeness). A non-interactive argument system ARG is complete if for any $\lambda \in \mathbb{N}$, any public parameters $\mathsf{pp} \leftarrow \mathsf{ARG.Setup}(1^\lambda)$, and any valid pair $(x, w) \in \mathcal{R}$, it holds that:

$$\mathsf{ARG.Verify}(x, \pi) = 1, \quad \text{for } \pi \leftarrow \mathsf{ARG.Prove}(x, w).$$

Definition C.6 (Knowledge Soundness). A non-interactive argument system ARG is knowledge-sound if for any PPT adversary A, there exists an expected polynomial-time knowledge extractor Ext such that:

$$\Pr\left[\mathsf{ARG.Verify}((x, \pi)) = 1 \wedge (x, w) \notin \mathcal{R} \mid (x, w, \pi) \leftarrow \mathsf{Ext}(\mathsf{pp})\right] \leq \mathsf{negl}(\lambda).$$

where $\mathsf{pp} \leftarrow \mathsf{ARG.Setup}(1^\lambda)$.

Definition C.7 (Zero-Knowledge). A non-interactive argument system ARG is zero-knowledge if there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for any PPT adversary A,

$$\left|\Pr[A^{O_0}(\mathsf{pp})] - \Pr[A^{O_1}(\mathsf{pp})]\right| \leq \mathsf{negl}(\lambda)(\lambda),$$

where:

- $O_0(x, w)$ returns $\mathsf{ARG.Prove}(x, w)$ if $(x, w) \in \mathcal{R}$.
- $O_1(x)$ returns $\mathsf{Sim}_1(td, x)$ where $(\mathsf{pp}, td) \leftarrow \mathsf{Sim}_0(1^\lambda)$.

Definition C.8 (Merkle tree). A Merkle tree MT is an authenticated data structure that consists in three algorithms:

- $\mathsf{rt} \leftarrow \mathsf{MT.Init}(1^\lambda, X)$ takes the security parameter λ and a list of elements $X = (x_1, \ldots, x_n)$ as inputs, constructs a tree that stores $x_1, \ldots, x_n$ in the leaves, and finally outputs a root rt.
- $\mathsf{path} \leftarrow \mathsf{MT.Prove}(x, X)$ takes as input an element x, a list X, and outputs the proof path, which can prove that x is in X.
- $\{0, 1\} \leftarrow \mathsf{MT.Verify}(x, \mathsf{rt}, \mathsf{path})$ takes as input an element x, the root rt, a proof path, and outputs either 0 or 1.

D Security

In this section, we prove the security of the construction presented in Section 5; we refer to this construction with Π.

Theorem D.1. *Our construction of the SPS scheme is k-unpredictable if the randomness used in partitioning is unbiased, meaning that no PPT adversary $\mathcal{A}$ can predict the bin of an output with probability better than $\frac{1}{k} + \mathsf{negl}(\lambda)$.*

Proof. (Sketch.) We prove this theorem using a sequence of hybrid arguments.

H_0: The real security experiment where the adversary attempts to predict the bin assignment of an output.

H_1: We replace the randomness used in partitioning with a truly random value. Since the randomness used is assumed to be unbiased, this change only alters the distribution of outputs in a negligible factor with H_0.

H_2: We replace the partitioning algorithm with a random selection from the set of permutations of outputs. This change in the experiment does not alter the distribution with H_1. Each output is thus placed in any of the k bins with probability exactly $\frac{1}{k}$.

Thus, the adversary's advantage in predicting the bin remains at most $\frac{1}{k} + \mathsf{negl}(\lambda)$. Therefore, our construction of SPS is unpredictable. □

Theorem D.2. *Our construction of the SPS scheme is β-untraceable, meaning that no probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ can win the untraceability game* $\mathsf{Untrace}(\mathcal{A}, \lambda, n, c, k)$ *with an advantage greater than $\frac{1}{\beta} + \mathsf{negl}(\lambda)$, where β is the number of honest outputs in the chosen bin.*

Proof. (Sketch.) We prove this theorem using a sequence of hybrid arguments.

H_0: The real security experiment where the adversary attempts to distinguish the spent output.

H_1: We replace the spending proof π with a simulated proof generated by the zero-knowledge simulator. By the zero-knowledge property of the proof system, the adversary's view remains computationally indistinguishable.

H_2: We replace the partitioning function with a truly random assignment. Since the partitioning function is unpredictable, this change introduces at most a negligible advantage.

Since in H_2, the adversary has no information to distinguish the spent output beyond guessing, their probability of success is at most $\frac{1}{\beta} + \mathsf{negl}(\lambda)$. Thus, our construction of SPS is β-untraceable. □

Theorem D.3. *Our construction of the SPS scheme is scalable, meaning that, with non-negligible probability, bins are pruned while preserving untraceability. The probability of pruning a bin after t transactions is $1 - p^k$, where $p = \sum_{m=0}^{R-1} \Pr[X = m]$ with $X \sim B(t, \frac{1}{k})$.*

Proof. (Sketch.) We proceed with a sequence of hybrid arguments.

H_0: The real execution of the scalability experiment, where transactions occur over time, bins accumulate spending proofs, and pruning is performed once a bin has been referenced as many times as its size.

H_1: We modify the experiment so that, instead of performing pruning directly, bins are marked as prunable once all outputs within them have been referenced. Since pruning depends only on the deterministic state of transactions, this change is indistinguishable from H_0.

H_2: We analyze the system before pruning occurs and consider double-spending resistance. In this hybrid, an adversary who succeeds in spending the same output twice without linkable nullifiers must break the linkability function (for two distinct nullifiers) or the binding property of commitments. Since the system satisfies double-spending resistance before pruning, this change does not alter the adversary's success probability, and this hybrid remains indistinguishable from H_1.

H_3: We execute the pruning process and remove spent outputs while retaining their corresponding nullifiers. Since all outputs that were referenced have unique nullifiers, and pruning only affects spent bins, double-spending resistance remains intact. The adversary cannot claim an output was not spent, nor can they introduce a new valid proof for a previously spent output. Since pruning does not modify the linkability function, this change is indistinguishable from H_2.

H_4: We consider the untraceability guarantee before pruning. An adversary's probability of linking a transaction to a specific output is at most $\frac{1}{\beta} + \mathsf{negl}(\lambda)$. Since no additional information is revealed about which outputs were spent before pruning, the adversary's advantage in the untraceability experiment remains unchanged. This change does not increase the adversary's success probability, making this hybrid indistinguishable from H_3.

H_5: We analyze the system after pruning and consider the impact on untraceability. The system removes references to spent outputs while keeping the structure necessary for future transactions. Since pruning does not introduce new information about which outputs were spent, the adversary's advantage in the untraceability experiment remains at most $\frac{1}{\beta} + \mathsf{negl}(\lambda)$. Thus, this change is indistinguishable from H_4.

H_6: We allow the system to continue executing transactions and pruning bins over multiple rounds. Since bins are pruned only once all their outputs have been referenced, the probability of at least one bin reaching the pruning condition after t transactions larger than

a constant c as shown in Lemma 4.7. Moreover, since all previous hybrids establish that security properties remain unchanged before and after pruning, the scheme maintains its guarantees over time. □

Theorem D.4. *Our construction of the SPS scheme is confidential, meaning that no PPT adversary $\mathcal{A}$ can distinguish between two different denominations d_0 and d_1 with probability better than $\frac{1}{2} + \mathsf{negl}(\lambda)$ in the confidentiality experiment* $\mathsf{Confid}(\mathcal{A}, \lambda)$.

Proof. (Sketch.) We use a sequence of hybrid games to prove confidentiality.

H_0: The real security experiment where the adversary tries to distinguish d_0 and d_1.

H_1: Replace the commitment to the amount with a commitment to a random value. The hiding property of the commitment scheme ensures that the adversary's advantage only varies with a negligible factor.

H_2: Replace the zero-knowledge proof with a simulated proof. Since the proof does not reveal any information, this transition is indistinguishable.

At the final hybrid, the adversary's advantage is bounded by $\frac{1}{2} + \mathsf{negl}(\lambda)$. Thus, our construction of SPS is confidential. □

Theorem D.5. *Our construction of the scalable privacy-preserving payment scheme SPS is non-slanderable, meaning that no PPT adversary $\mathcal{A}$ can produce a tag t for an output o it does not control such that* $\mathsf{Link}(t', t) = 1$, *where t' is the tag of the actual spender.*

Proof. (Sketch.) We prove non-slanderability using a sequence of hybrid games.

H_0: The real experiment where the adversary tries to produce a linkable tag for an unknown output.

H_1: Replace the tag-generation function with a random function. Since the tag function is collision-resistant, this change does not increase the adversary's advantage.

H_2: Replace the adversary's view of previously generated tags with simulated values. The security of the commitment scheme ensures indistinguishability.

At the final step, the adversary's success probability remains negligible. Thus, our construction of SPS is non-slanderable. □