Derecho: Privacy Pools with Proof-Carrying Disclosures

Josh Beal and Ben Fisch

Yale University
{josh.beal,ben.fisch}@yale.edu

Abstract. A *privacy pool* enables clients to deposit units of a cryptocurrency into a shared pool where ownership of deposited currency is tracked via a system of cryptographically hidden records. Clients may later withdraw from the pool without linkage to previous deposits. Some privacy pools also support hidden transfer of currency ownership within the pool. In August 2022, the U.S. Department of Treasury sanctioned Tornado Cash, the largest Ethereum privacy pool, on the premise that it enables illicit actors to hide the origin of funds, citing its usage by the DPRK-sponsored Lazarus Group to launder over $455 million dollars worth of stolen cryptocurrency. This ruling effectively made it illegal for U.S. persons/institutions to use or accept funds that went through Tornado Cash, sparking a global debate among privacy rights activists and lawmakers. Against this backdrop, we present *Derecho*, a system that institutions could use to request cryptographic attestations of fund origins rather than naively rejecting all funds coming from privacy pools. Derecho is a novel application of *proof-carrying data*, which allows users to propagate allowlist membership proofs through a privacy pool's transaction graph. Derecho is backwards-compatible with existing Ethereum privacy pool designs, adds no overhead in gas costs, and costs users only a few seconds to produce attestations.

1 Introduction

Bitcoin, Ethereum, and other cryptocurrencies have achieved significant market capitalization and adoption over the past decade, yet the privacy guarantees of many popular blockchains remain lacking. The traceability of transactions in blockchains such as Bitcoin and Ethereum has been well-studied [AKR+13, MPJ+13, WMW+22], and even privacy-focused blockchains are subject to deanonymization attacks [KYMM18, MSH+18, YAY+19]. Privacy solutions can be designed as add-on components to an existing blockchain or as independent blockchains.

In this work, we focus on privacy pools that use *zero-knowledge proofs* to enable anonymous transfers of assets on account-based smart contract platforms such as Ethereum. These pools are based on the design of *Zerocash* [BCG+14], which is also the basis for the cryptocurrency Zcash [HBHW22]. In a nutshell, these privacy pools enable users to deposit funds into a shared pool, anonymously transfer funds within the pool, and later withdraw funds without linkage to their previous transactions.

Tornado Cash (Nova) was the most widely used Ethereum privacy pool until U.S. regulators took action against the service in August 2022. The U.S. Department of the Treasury's Office of Foreign Assets Control (OFAC) added the Tornado Cash smart contract addresses to the Specially Designated Nationals (SDN) list, purportedly due to its usage for laundering more than $9 billion worth of cryptocurrency since 2019, including by the DPRK state-sponsored Lazarus Group that was also sanctioned in 2019. This designation forbids U.S. users, including individuals and institutions, from interacting with the service [Uni22]. It has resulted in locked funds for U.S. users of the service and has limited the options for law-abiding users that seek to improve the privacy of their transactions on Ethereum. These sanctions have brought renewed attention to the clash between privacy and regulatory oversight on smart contract platforms. In October 2022, Coin Center filed a lawsuit against the Treasury Department arguing that OFAC exceeded its statutory authority in designating Tornado Cash [Coi22]. It also sparked discussion among researchers and privacy advocates [BKB22, Fis22, Sol22], questioning both the efficacy and necessity of privacy pool sanctions in addressing illicit finance, and seeking alternative technical solutions.

A simple solution would restrict deposits into and withdrawals from the privacy pool to accounts on a specific allowlist.[1] For example, the allowlist might be the set of all public Ethereum addresses that

[1] Alternatively, the usage of the privacy pool could be limited to accounts that are not on a specific blocklist and not newly generated at the time of deposit. The second criterion is important to prevent the situation where an attacker move funds to a fresh address in order to evade the restrictions.

are not on the U.S. Treasury's SDN list. However, allowlists are expected to vary by jurisdiction and may be updated dynamically. In practice, there are widely used "compliance-as-a-service" providers, such as Chainalysis and TRM Labs, that assess the risk of public Ethereum addresses, incorporating multiple risk factors such as interactions with government-sanctioned addresses and exploited smart contracts, and continuously update their risk assessments. Today, U.S. regulated exchanges utilize these risk scores for risk management, e.g., to decide whether or not to accept deposits from a given Ethereum address. Allowlists can be viewed as binary risk scores.

An alternative solution to restricting deposits and withdrawals is for users of privacy pools to generate attestations when necessary, selectively disclosing information about the provenance of funds withdrawn from the pool. When cryptocurrency is deposited into a privacy pool like Tornado Cash, a digital receipt in the form of a cryptographic commitment is generated, and the depositor retains a secret key required to use this receipt later. A user withdraws x units of cryptocurrency from the pool by presenting a zero-knowledge proof that it knows the secret key of an unused receipt for this exact amount of cryptocurrency, and a keyed hash of the receipt called a *nullifier*. The nullifier still hides the receipt but prevents it from being used twice. While this zero-knowledge proof reveals little information by default (other than transaction validity), a user may choose to reveal more information about the origin of a withdrawal to an interested party (e.g., an exchange). In fact, zero-knowledge proofs can be used to selectively disclose information about the unique deposit receipt, such as membership of the depositing address on an allowlist or a risk score that a provider has assigned to that public address. Similar solutions were proposed more than a decade ago in the context of Tor and blocklisting of IP-addresses [JKTS07, TKCS09, BG13].

However, this system of user-generated disclosures becomes more challenging in pools that support in-pool transfers. The recipient of funds must retain the ability to prove facts about its provenance, in particular, that the funds originated via deposits from accounts on a given set of allowlists. We solve this problem using *proof-carrying data* [CT10], a generalization of incrementally verifiable computation [Val08] that offers a powerful approach to recursive proof composition. When a user makes their first transaction within the privacy pool, the user generates membership proofs for a set of allowlists. Subsequent transactions within the privacy pool generate new membership proofs that are derived from (i.e., prove knowledge of) the previous membership proofs of the transaction inputs and the details of the current transaction. These membership proofs, which we call *proof-carrying disclosures*, can be verified efficiently and may be communicated directly to the recipient. While our solution focuses on allowlists, it can easily be generalized to handle non-binary risk scores.

1.1 Our Contributions

To summarize, our main contributions are as follows:

- We formalize and present Derecho, a system for cryptographic attestation of funds originating from privacy pools. Our system addresses the key legal challenges of privacy pools through the usage of *proof-carrying disclosures*, a novel application of proof-carrying data.
- We show that our disclosure system achieves practical proving and verification times for a range of system parameters. For a typical configuration, the proving time was 3.4 seconds, and the verification time was 1.9 seconds. Since membership proofs are verified off-chain by the recipient, our system adds no overhead in gas costs.

2 Technical Overview

2.1 Design Goals

A key goal in the system design was to develop a solution that can be introduced as an add-on component to existing privacy pools on Ethereum and other smart contract platforms. To facilitate adoption of the system, the design should not require changes to the functionality of the privacy pool or introduce any gas costs to users. Furthermore, it should maintain the existing security properties of the privacy pool while optionally allowing for attestations of allowlist membership.

We rule out solutions that involve changes to the privacy pool or limitations on the number of allowlists. For instance, a naive solution to this problem would involve augmenting the coin commitment openings with a field storing the allowlist identifier and enforcing membership consistency in the

transfer proofs of the privacy pool. However, this solution would not be backwards-compatible with existing privacy pools. Furthermore, if an arbitrary number of allowlists is supported, the increase in gas costs would be substantial. To support an unbounded number of allowlists in the naive solution, the privacy pool would need to let the allowlist field have unbounded size or restrict the size and make use of an on-chain set accumulator. While such a solution is appealing in its simplicity, it would face barriers to adoption due to the costs to users and the required changes to existing systems.

2.2 Initial Approach

Derecho assumes the existence of a set of allowlists that are maintained external to the system, where each allowlist contains a list of Ethereum public-key addresses, along with a dynamic accumulator A (e.g., a Merkle tree) which aggregates the allowlists. That is, for each public key pk on allowlist with identifier al the element $H(\mathsf{al}||\mathsf{pk})$ is inserted into A, where H is a collision-resistant hash function. For simplicity, we restrict to privacy pools that manage only one cryptocurrency asset at a time, but the system easily generalizes to pools that manage multiple types of assets. The pool contract maintains an accumulator R of records, where each record is a hash digest (i.e., cryptographic commitment). When a user first deposits x units of cryptocurrency into the privacy pool from a public Ethereum address pk_s, a record of the form $H(x||\mathsf{pk}_1||r_1)$ is added to the accumulator R, where pk_1 is a *shielded* public-key address and r_1 is a nonce that will later be used to nullify the record upon a transfer or withdrawal. A transfer transaction may create a new record $H(x||\mathsf{pk}_2||r_2)$, a nullifier $n = H(r_1)$, and a zero-knowledge proof that a record $c = H(x||\mathsf{pk}_1||r_1)$ exists in R such that $n = H(r_1)$. The transfer may also create multiple output records of the form $H(x_i||\mathsf{pk}_i||r_i)$ for $i \in [2,k]$, and the zero-knowledge proof would additionally attest that $\sum_i x_i = x$. A withdrawal contains a similar zero-knowledge proof, but publicly reveals the output amount y and a destination Ethereum address pk_d, at which point y units are withdrawn from the pool and delivered to pk_d. While this example references a concrete implementation of a privacy pool for illustrative purposes, the disclosure system does not assume a specific record format or nullifier creation algorithm.

We define membership of records on allowlists recursively as follows. The initial record created upon deposit is a member of allowlist al if and only if its source Ethereum address pk_s is a member of al. A record created as the output of a transfer transaction is a member of al if and only if all the input records to the transfer are members of al. Finally, we say that a withdrawal transaction is a member of al if and only if all the input records to this withdrawal are members of al. Note that this final attestation refers to the withdrawal transaction itself rather than the Ethereum destination address pk_d, which may or may not be on the allowlist for other reasons.

Since the initial record created upon deposit is publicly linked to the Ethereum source address pk_s via the on-chain deposit transaction, it is straightforward for a user to produce a membership proof of the deposit record on a list al by providing a membership proof for pk_s using the accumulator A, which could be verified given the Ethereum transaction log. Producing membership disclosure proofs for the output records of in-pool transactions is more subtle. If a user already has membership proofs for all the input records to a transfer transaction with respect to a list al, then it can create a membership proof for an output record of this transaction by proving its knowledge of valid al membership proofs for all the input records to the transaction. The same could be done for a withdrawal transaction. In more detail, since neither the output record nor the transaction log contains explicit references linking it to transaction inputs, but only nullifiers n_i for each input record, the zero-knowledge disclosure proof repeats the logic of the transfer proof: for each n_i, it proves knowledge of an input record $c_i = H(x_i||\mathsf{pk}_i||r_i)$ such that $n_i = H(r_i)$ and *additionally* proves knowledge of a valid membership proof π_i for c_i. This recursive proof of knowledge is possible via a proof-carrying data (PCD) scheme.

2.3 Key Challenges

With this initial approach, a problem immediately arises: the validity of the membership proof π_i is not actually verifiable against the record commitment c_i alone. For example, verifying the initial membership proof of a deposit record c required checking against the blockchain transaction log to obtain the link between the record c and a source Ethereum address pk_s. Naively, if the public input required to verify allowlist membership includes the entire transaction log of the privacy pool then the recursive zero-knowledge proof statement would become impractically large.

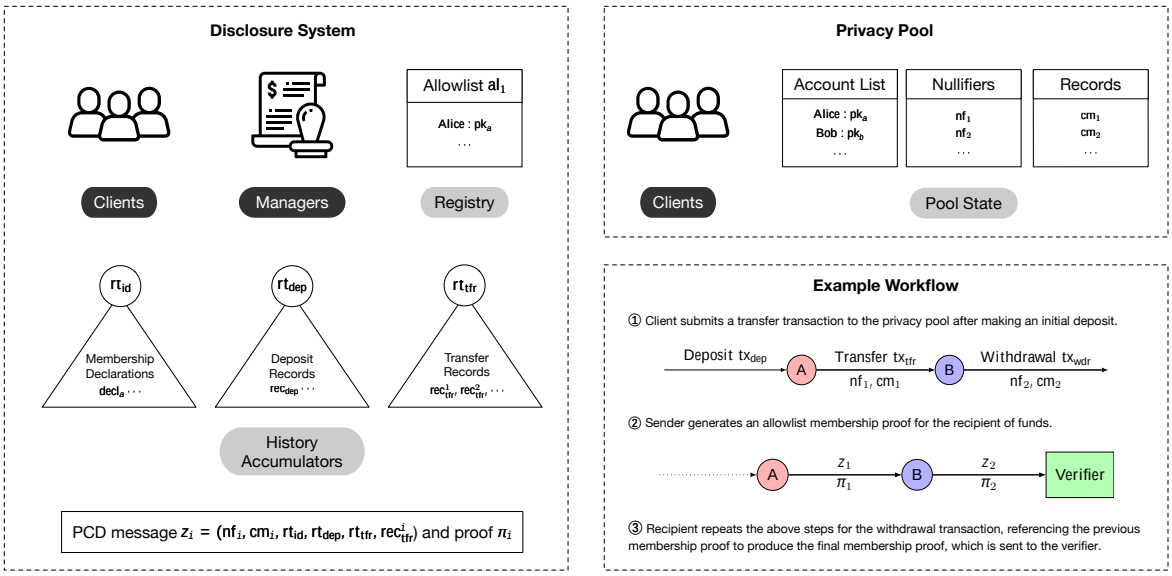

Fig. 1: Illustration of the proposed disclosure system and an example workflow.

The standard trick around this problem is to replace the transaction log public input with an accumulator digest T: the membership disclosure proof for a deposit record c now includes both an accumulator membership proof for T of a transaction linking c to pk_s and an accumulator membership proof for A showing that pk_s is on the list al at the time of the deposit.

However, yet another subtle complication arises when attempting to produce recursive membership proofs for the output records of transfers. Suppose the user has a membership proof for an input record c to a transfer creating an output record c^0. Suppose further that the accumulator digest T commits to the transaction log state at the time t that was created, and that the accumulator state T^0 commits to the transaction log state at the time t^0 that the new transfer is occurring. The value T is required as input to verify , but is unknown to the recipient of the transfer at the time t^0. Thus, T is not available as a public input to verify the recursive disclosure created for c^0, rather, the disclosure must prove knowledge of both , c and T against which is valid. Moreover, without additional restrictions, the prover would be free to invent a malicious proof valid against a T unrelated to the true blockchain state at any point in history.

To resolve this problem, we use history accumulators, which commit not only to the current state of a set but also to all historical states. History accumulators provide an efficient mechanism to prove that a digest T represents a valid historical state , which can be verified against the current digest T^0 of the history accumulator. In Section 4.2, we provide precise definitions of the disclosure system's three key data structures: membership declarations, deposit records, and transfer records. Each of these data structures has a corresponding history accumulator as illustrated above.

Altogether, these techniques result in a system that only requires a few seconds to produce attestations on a consumer-grade laptop. These attestations can be efficiently verified by the recipient of funds with respect to the current blockchain state. Due to the fact that these proofs do not need to be posted on-chain or verified by the smart contract, we were able to leverage recent developments in PCD that trade a larger proof size for faster proving times [BCL+21].

2.4 Example Workflow

To highlight the main ideas of our construction, we describe a typical system workflow, which is illustrated in Figure 1. We consider a user named Alice, who has generated key pairs and registered these key pairs with the privacy pool. The manager for allowlist al_1 authorizes membership for Alice's public key pk_a. As a result, Alice has an account with membership on allowlist al_1, and a membership declaration will be created and accumulated. The manager can revoke the membership of Alice's account at any point. However, since disclosure creation is optional, Alice cannot be prevented from using the privacy pool. Funds may always be spent without producing an accompanying membership proof. Next, we will see an example of how these membership proofs are generated at each step.

First, Alice deposits 1 ETH into the privacy pool. Based on the details of the deposit transaction, a deposit record $\mathsf{rec}_{\mathsf{dep}}$ will be created and accumulated. Second, Alice transfers 1 ETH to Bob using the anonymous transfer functionality of the pool. Asynchronously, Alice can create a membership proof for this transfer. This initial membership proof $_1$ will attest to the existence of a deposit record $\mathsf{rec}_{\mathsf{dep}}$ for the transaction input and the existence of a membership declaration decl_a for Alice's public key pk_a with respect to allowlist al_1. The proof ensures correspondence to a true blockchain state by attesting to the existence of an appropriate transfer record $\mathsf{rec}^1_{\mathsf{tfr}}$. Alice sends the proof $_1$ to Bob shortly after the transfer occurs. Subsequently, Bob withdraws 1 ETH from the privacy pool. Asynchronously, Bob can create a membership proof for this withdrawal. This proof $_2$ will attest to the existence of a proof $_1$ for the transaction input. As before, the proof also ensures correspondence to a true blockchain state by attesting to the existence of an appropriate transfer record $\mathsf{rec}^2_{\mathsf{tfr}}$. If Bob wishes to deposit the withdrawn funds with origin attestation at a regulated institution, he can simply present the final membership proof $_2$ directly to the recipient for verification.

3 Background

3.1 Building Blocks

Notation We let $2 \mathbb{N}$ denote the security parameter with unary representation $1^$. We let $\mathsf{negl}(\ )$ denote the class of negligible functions. We let PPT denote probabilistic polynomial time. We let $\mathcal{A}$ denote a computationally-bounded adversary modeled as a PPT algorithm. We let $[l]$ denote the set of integers $\{0,\ldots,l\ 1\}$. We let $x \ \$\ S$ denote that x is sampled uniformly at random from a set S. We let $\mathbb{F}_q$ denote a finite field of order q.

Hash functions We use hash functions satisfying the standard collision resistance property. Our system samples hash functions of the form $H_q : \{0,1\}^\ \ \{0,1\}^{\mathbb{F}_q}$. In this work, we use the arithmetic hash function $\mathsf{Poseidon}$ [GKR+21], which is commonly used in blockchain applications. The design of arithmetic hash functions is an active area of research [BBC+22, GHR+22, GKS23], and our system can be instantiated with any efficient construction of these hash functions.

Commitment schemes A commitment scheme $\mathcal{C} = (\mathsf{Com}, \mathsf{Vfy})$ is a pair of efficient algorithms defined over a message space $\mathcal{M}$ and a randomness space $\mathcal{R}$ where:

- $\mathsf{cm} \ \ \mathsf{Com}(m; r)$ is a commit algorithm that produces a commitment cm given the message $m \ 2 \ \mathcal{M}$ to be committed and the randomness $r \ \$\ \mathcal{R}$.
- $b \ \ \mathsf{Vfy}(\mathsf{cm}, m, r)$ is a verification algorithm that checks whether (m, r) is the correct opening of the commitment cm and outputs a bit $b \ 2 \ \{0,1\}$ representing accept if $b = 1$ and reject otherwise.

Informally, a commitment scheme is called binding if it is infeasible to open a commitment to a different message. It is called hiding if the commitments of any two messages are indistinguishable. Commitment schemes can be built from collision-resistant hash functions.

Accumulator schemes An accumulator scheme consists of a tuple of efficient algorithms $\mathsf{Acc} = (\mathsf{Init}, \mathsf{Update}, \mathsf{PrvMem}, \mathsf{VfyMem})$ where:

- $(\mathsf{rt}, \ ) \ \ \mathsf{Init}(1^\ )$ sets up the initial state and digest rt of the accumulator.
- $(\mathsf{rt}', \ ') \ \ \mathsf{Update}(\mathsf{rt}, \ , \mathsf{elem})$ inserts element elem into the set and outputs an updated state $'$ and digest rt'.
- $\ \ \mathsf{PrvMem}(\ , \mathsf{elem})$ outputs a set membership proof for the element elem in the set.
- $b \ \ \mathsf{VfyMem}(\mathsf{rt}, \ , \mathsf{elem})$ outputs a bit $b \ 2 \ \{0,1\}$ verifying whether is a valid proof for the accumulation of elem in rt. The output is $b = 1$ if the proof is accepted as valid and $b = 0$ otherwise.

Informally, an accumulator scheme should be correct and sound. Correctness ensures that for every element in the set, it should be easy to prove membership. Soundness ensures that for every element not in the set, it should not be feasible to prove membership.

History accumulator schemes A history accumulator is an authenticated data structure that commits to a current set state $_n$ and also to all previous set states $_1; \ldots; _{n-1}$. When the accumulator digest rt_n for $_n$ is incrementally updated for a new state $_{n+1}$, the new digest rt_{n+1} is a commitment to $_{n+1}$ and all prior states accumulated by rt_n. Some history accumulators support history proofs of additional invariants, e.g., that the current state ^{0} of a history accumulator with digest rt^0 is a superset of all historical states. Specifically, a history accumulator scheme consists of a tuple of efficient algorithms $\mathsf{HA} = (\mathsf{Init}; \mathsf{Update}; \mathsf{Remove}; \mathsf{PrvMem}; \mathsf{VfyMem}; \mathsf{PrvHist}; \mathsf{VfyHist})$ where the algorithms Init, Update, PrvMem and VfyMem work as above, and the other algorithms work as follows:

- $(\mathsf{rt}^0; {}^0) \quad \mathsf{Remove}(\mathsf{rt}; ; \mathsf{elem})$ removes element elem from the set and outputs updated state ^{0} and digest rt^0.
- $\quad \mathsf{PrvHist}(\mathsf{rt}; {}^0)$ given the current state ^{0} (which has a current digest rt^0) outputs a proof that rt is a historical state of the history accumulator.
- $b \quad \mathsf{VfyHist}(\mathsf{rt}; \mathsf{rt}^0;)$ outputs a bit $b \in \{0,1\}$ verifying whether $$ is a valid proof that rt is a digest of a historical state with respect to rt^0. The output is $b = 1$ if the proof is accepted as valid and $b = 0$ otherwise.

Informally, a history accumulator scheme should also be correct and sound. Correctness ensures that for every element in the set, it should be easy to generate a membership proof, and for every historical state, it should be easy to generate a history proof. Soundness ensures that for every element not in the set, it should be infeasible to generate a membership proof, and for every state that is not a historical state, it should be infeasible to generate a history proof.

This scheme can be instantiated by Merkle history trees [CW09, BKLZ20, TFZ+22], which are known to support efficient membership proofs and history proofs [Cro10, LLK13].

Proof-carrying data Proof-carrying data (PCD) [CT10] enables a set of parties to carry out an arbitrarily long distributed computation where every step is accompanied by a proof of correctness.

Let $V(\mathsf{G})$ and $E(\mathsf{G})$ denote the vertices and edges of a graph G. A transcript T is a directed acyclic graph where each vertex $u \in V(\mathsf{T})$ is labeled by local data $z^{(u)}_{\mathsf{loc}}$ and each edge $e \in E(\mathsf{T})$ is labeled by a message $z^{(e)}$ or $?$. The output of a transcript T, denoted $\mathsf{o}(\mathsf{T})$, is $z^{(e^0)}$ where $e^0 = (u; v)$ is the lexicographically-first edge such that v is a sink.

A vertex $u \in V(\mathsf{T})$ is $'$-compliant for a predicate $' \in \mathsf{F}$ if for all outgoing edges $e = (u; v) \in E(\mathsf{T})$ either: (1) if u has no incoming edges, $'(z^{(e)}; z^{(u)}_{\mathsf{loc}}; ?; \ldots; ?)$ evaluates to true or (2) if u has m incoming edges $e_1; \ldots; e_m$, $'(z^{(e)}; z^{(u)}_{\mathsf{loc}}; z^{(e_1)}; \ldots; z^{(e_m)})$ evaluates to true. A transcript T is $'$-compliant if all of its vertices are $'$-compliant.

A proof-carrying data system PCD for a class of compliance predicates F consists of a tuple of efficient algorithms $(\mathsf{G}; \mathsf{I}; \mathsf{P}; \mathsf{V})$, known as the generator, indexer, prover, and verifier algorithms, for which the properties of completeness, knowledge soundness, and zero knowledge hold.

Completeness. PCD has perfect completeness if for every adversary A the following holds:

$$\Pr\left[\begin{array}{c} ' \in \mathsf{F} \\ \wedge\ '(z; z_{\mathsf{loc}}; z_1; \ldots; z_m) = 1 \\ \wedge\ (\forall i; z_i = ? \vee \forall i; \mathsf{V}(\mathsf{ivk}; z_i; _i) = 1) \\ + \\ \mathsf{V}(\mathsf{ivk}; z;) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{pcd}} \quad \mathsf{G}(1^) \\ ('; z; z_{\mathsf{loc}}; [z_i; _i]_{i=1}^m) \quad A(\mathsf{pp}_{\mathsf{pcd}}) \\ (\mathsf{ipk}; \mathsf{ivk}) \quad \mathsf{I}(\mathsf{pp}_{\mathsf{pcd}}; ') \\ \mathsf{P}(\mathsf{ipk}; z; z_{\mathsf{loc}}; [z_i; _i]_{i=1}^m) \end{array}\right] = 1.$$

Knowledge soundness. PCD has knowledge soundness with respect to an auxiliary input distribution D if for every expected polynomial-time adversary P there exists an expected polynomial-time extractor $\mathsf{E}_{\tilde{\mathsf{P}}}$ such that for every set Z:

$$\Pr\left[\begin{array}{c} ' \in \mathsf{F} \\ \wedge\ (\mathsf{pp}_{\mathsf{pcd}}; \mathsf{ai}; '; \mathsf{o}(\mathsf{T}); \mathsf{ao}) \in Z \\ \wedge\ \mathsf{T} \text{ is } '\text{-compliant} \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{pcd}} \quad \mathsf{G}(1^) \\ \mathsf{ai} \quad D(\mathsf{pp}_{\mathsf{pcd}}) \\ ('; \mathsf{T}; \mathsf{ao}) \quad \mathsf{E}_{\tilde{\mathsf{P}}}(\mathsf{pp}_{\mathsf{pcd}}; \mathsf{ai}) \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c} ' \in \mathsf{F} \\ \wedge\ (\mathsf{pp}_{\mathsf{pcd}}; \mathsf{ai}; '; \mathsf{o}; \mathsf{ao}) \in Z \\ \wedge\ \mathsf{V}(\mathsf{ivk}; \mathsf{o};) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{pcd}} \quad \mathsf{G}(1^) \\ \mathsf{ai} \quad D(\mathsf{pp}_{\mathsf{pcd}}) \\ ('; \mathsf{o}; ; \mathsf{ao}) \quad \mathsf{P}(\mathsf{pp}_{\mathsf{pcd}}; \mathsf{ai}) \\ (\mathsf{ipk}; \mathsf{ivk}) \quad \mathsf{I}(\mathsf{pp}_{\mathsf{pcd}}; ') \end{array}\right] - \mathsf{negl}( ).$$

Zero knowledge. PCD has (statistical) zero knowledge if there exists a PPT simulator S such that for every honest adversary A the distributions below are statistically indistinguishable:

$$\left\{ (\mathsf{pp}_{\mathsf{pcd}}, z_{\mathsf{loc}}, z, \pi) \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{pcd}} \leftarrow \mathsf{G}(1^\lambda) \\ (z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow A(\mathsf{pp}_{\mathsf{pcd}}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathsf{I}(\mathsf{pp}_{\mathsf{pcd}}, R) \\ \pi \leftarrow \mathsf{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right\}$$

and

$$\left\{ (\mathsf{pp}_{\mathsf{pcd}}, z_{\mathsf{loc}}, z, \pi) \;\middle|\; \begin{array}{c} (\mathsf{pp}_{\mathsf{pcd}}, \tau) \leftarrow \mathsf{S}(1^\lambda) \\ (z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow A(\mathsf{pp}_{\mathsf{pcd}}) \\ \pi \leftarrow \mathsf{S}(\tau, R, z) \end{array} \right\},$$

An adversary is honest if their output results in the implicant of the completeness condition being satisfied with probability 1, i.e., $R \in \mathsf{F}$, $R(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and either $z_i = \bot$ or $\mathsf{V}(\mathsf{ivk}, z_i, \pi_i) = 1$ for each incoming edge z_i. A proof π has size $\mathsf{poly}(\lambda, |R|)$; that is, the proof size is not allowed to grow with each application of the prover algorithm P.

Our system uses the PCD construction of [BCL+21], which is based on split accumulation schemes. Other constructions are described in [BCCT13], [BCTV14], Halo [BGH19], [BCMS20], Fractal [COS20], and Halo Infinite [BDFG21]. Nova [KST22] is a recent IVC construction based on folding schemes, however we require the more general notion of PCD to support multiple transaction inputs.

3.2 Privacy Pools

The privacy pool needs to keep track of the shielded addresses, the created records, and the nullifiers that correspond to spent records. SNARK-friendly accumulators are used to support the privacy-preserving transaction functionality. Specifically, the following data structures are used:

- **Public Parameters.** In the setup of the privacy pool, a trusted party generates public parameters $\mathsf{pp}_{\mathsf{pool}}$ that are available to all participants in the system.
- **User Key Pairs.** A user generates a key pair $(\mathsf{sk}, \mathsf{pk})$ when joining the privacy pool. The public key pk is used for receiving coins and the secret key sk is used for creating transactions. The public key is typically derived from the user's Ethereum address addr and the generated secret key. The user generates a key pair $(\mathsf{sk}^0, \mathsf{pk}^0)$ for encryption and decryption of owner memos.
- **Account List.** The user's public keys are stored in the account list of the privacy pool contract upon registration. This list supports the anonymous transfer functionality of the privacy pool.
- **Coin Commitments.** A coin commitment cm is a commitment to details of a transaction output, including the value amount amt and the recipient's public key pk. The opening of the coin commitment open is used in transaction creation and membership proof generation.
- **Nullifier Sets.** The privacy pool typically maintains a nullifier set to prevent double-spending attacks. A nullifier nf can be constructed from an opening of a coin commitment cm and auxiliary information aux.
- **Owner Memos.** An owner memo memo is used by the coin owner to create the coin commitment from the encryption of the opening of the commitment. The memo can be shared with the recipient directly. Alternatively, the memo can be posted on the public ledger as part of the transaction.
- **Accumulators.** The privacy pool typically uses sparse Merkle trees to efficiently prove set membership. For example, an on-chain accumulator rt_{c} can maintain the set of coin commitments.

Some of these data structures may also be referenced by the disclosure system. For instance, it is necessary to reference the nullifiers and the coin commitments in creating the membership proof.

The privacy pool interface is described below. The privacy pool supports an initial setup, account registration, and financial transactions. The initial setup is run by a trusted party, and the transaction processing is typically executed by a smart contract. Each of the transaction objects are created by the client. We provide a specific implementation of the privacy pool interface in Appendix D.

- $\mathsf{PrivacyPoolSetup}(1^\lambda) \to \mathsf{pp}_{\mathsf{pool}}$. This algorithm sets up the initial state of the privacy pool, including the accumulator and configurable parameters. Returns the public parameters $\mathsf{pp}_{\mathsf{pool}}$.
- $\mathsf{ProcessDepositTx}(\mathsf{pp}_{\mathsf{pool}}, \mathsf{tx}_{\mathsf{dep}}) \to b$. This algorithm validates the deposit amount and transfers funds from the sender address to the privacy pool contract address. Returns accept/reject bit b.
- $\mathsf{ProcessTransferTx}_{n,m}(\mathsf{pp}_{\mathsf{pool}}, \mathsf{tx}_{\mathsf{tfr}}) \to b$. This algorithm checks the value invariant and verifies the transfer proof. If the transaction is valid, input nullifiers are added to the nullifier set and output coin commitments are accumulated. Returns accept/reject bit b.

- $\mathsf{ProcessWithdrawalTx}_n(\mathsf{pp}_{\mathsf{pool}}; \mathsf{tx}_{\mathsf{wdr}}) \to b$. This algorithm validates the input coin commitments and verifies the withdrawal proof. If the transaction is valid, input nullifiers are added to the nullifier set and funds are sent to the recipient. Returns accept/reject bit b.
- $\mathsf{ProcessRegistrationTx}(\mathsf{pp}_{\mathsf{pool}}; \mathsf{tx}_{\mathsf{reg}}) \to b$. The pool will store the public keys $(\mathsf{pk}; \mathsf{pk}^0)$ for the user in the account list. Returns accept/reject bit b.

The client supports the following operations for interacting with the privacy pool:

- $\mathsf{GenerateKeyPair}(\mathsf{pp}_{\mathsf{pool}}; \mathsf{addr}) \to (\mathsf{sk}; \mathsf{pk}; \mathsf{sk}^0; \mathsf{pk}^0)$. Given public parameters $\mathsf{pp}_{\mathsf{pool}}$ and an address addr, output a user key pair $(\mathsf{sk}; \mathsf{pk})$ and an encryption key pair $(\mathsf{sk}^0; \mathsf{pk}^0)$.
- $\mathsf{CreateRegistrationTx}(\mathsf{pp}_{\mathsf{pool}}; \mathsf{pk}) \to \mathsf{tx}_{\mathsf{reg}}$. Given public parameters $\mathsf{pp}_{\mathsf{pool}}$ and the user's public keys $(\mathsf{pk}, \mathsf{pk}^0)$, output a registration transaction $\mathsf{tx}_{\mathsf{reg}} = (\mathsf{pk}, \mathsf{pk}^0)$. This transaction registers the public keys, which enables other users to send funds to the account using the key pk and encrypt the owner memos using the encryption key pk^0.
- $\mathsf{CreateDepositTx}(\mathsf{pp}_{\mathsf{pool}}; \mathsf{amt}; \mathsf{pk}) \to \mathsf{tx}_{\mathsf{dep}}$. Given public parameters pp, a value amount amt, and a user public key pk, output a deposit transaction $\mathsf{tx}_{\mathsf{dep}} = (\mathsf{cm}; \mathsf{amt})$. The deposit transaction will transfer amt units of value from the sender to the privacy pool contract address.
- $\mathsf{CreateTransferTx}_{n;m}(\dots) \to \mathsf{tx}_{\mathsf{tfr}}$. Given public parameters $\mathsf{pp}_{\mathsf{pool}}$, a list of input user secret keys $\mathbf{sk}_{\mathbf{in}}$, a list of openings of input coin commitments $\mathbf{open}_{\mathbf{in}}$, a list of input addresses $\mathbf{addr}_{\mathbf{in}}$, a list of openings of output coin commitments $\mathbf{open}_{\mathbf{out}}$, and a list of encryption public keys $\mathbf{pk}'$, output a transfer transaction $\mathsf{tx}_{\mathsf{tfr}} = (\mathbf{nf}; \mathbf{cm}; \mathbf{memo}; \mathsf{rt}_{\mathsf{c}}; \pi_t)$ where π_t is the transfer proof. This transaction will transfer value from the input coin owners to the output coin owners. This algorithm is parametrized by the number of transaction inputs n and the number of transaction outputs m.
- $\mathsf{CreateWithdrawalTx}_n(\dots) \to \mathsf{tx}_{\mathsf{wdr}}$. Given public parameters $\mathsf{pp}_{\mathsf{pool}}$, a list of sender secret keys $\mathbf{sk}_{\mathbf{in}}$, a list of openings of input coin commitments $\mathbf{open}_{\mathbf{in}}$, a list of input addresses $\mathbf{addr}_{\mathbf{in}}$, an opening of a placeholder output coin commitment $\mathsf{open}_{\mathsf{out}}$, and an output address $\mathsf{addr}_{\mathsf{out}}$, output a withdrawal transaction $\mathsf{tx}_{\mathsf{wdr}} = (\mathsf{amt}; \mathsf{addr}_{\mathsf{out}}; \mathbf{nf}; \mathsf{cm}; \mathsf{rt}_{\mathsf{c}}; \pi_w)$ where π_w is the withdrawal proof. The withdrawal transaction will transfer amt units of value from the input coins to the output address $\mathsf{addr}_{\mathsf{out}}$. This algorithm is parametrized by the number of transaction inputs n.

The building blocks of the privacy pool must support the following interface:

- Coin Commitment Creation. A coin commitment cm is computed from its opening open. This operation is denoted by $\mathsf{cm} := \mathsf{Com}(\mathsf{open})$.
- Nullifier Creation. A nullifier is computed from the opening of the coin commitment open and the auxiliary input aux. This operation is denoted by $\mathsf{nf} := \mathsf{Nullify}(\mathsf{open}; \mathsf{aux})$. Nullifiers must be binding and hiding. This is typically achieved by using commitment schemes.
- Amount Extraction. The opening of the coin commitment must support the field extraction operation $\mathsf{amt} := \mathsf{Value}(\mathsf{open})$, which yields the value amount.

4 Definitions

4.1 System Entities

The disclosure system consists of accounts, allowlists, clients, managers, and the registry. The system references the state of the privacy pool and user transactions, however it does not change the functionality of the privacy pool. Hence it is backwards-compatible with existing privacy pool designs.

- Account. An externally-owned account controls units of a cryptocurrency and has a corresponding public/private key pair. For simplicity, we refer to externally-owned accounts as accounts.
- Allowlist. A list of accounts that are not prohibited from financial interactions in certain settings, such as a geographical jurisdiction. Each list is managed by a trusted party and updated regularly.
- Client. A client operates one or more accounts on the Ethereum blockchain and interacts with the privacy pool through cryptocurrency deposits, transfers, and withdrawals. The client generates membership proofs on a set of allowlists when transferring or withdrawing funds.
- Manager. An allowlist manager is responsible for specifying the members of the allowlist. The manager may update the list over time by adding or removing members.
- Registry. The registry stores the current members of each of the allowlists in order to support generation of membership proofs by the clients.
- Privacy Pool Contract. The privacy pool contract supports deposits and withdrawals of coins from the pool and anonymous transfers of coins within the pool.

4.2 Data Structures

The following data structures are introduced to support the disclosure system functionality. In Section 2.4, we described an example of how membership declarations, deposit records, and transfer records are produced in a typical workflow. These objects are managed by history accumulators.

- Public Parameters. In the setup of the disclosure system, a trusted party generates public parameters $\mathsf{pp}_{\mathsf{disc}}$ that are available to all participants in the system.
- Allowlists. An allowlist consists of a unique identifier al and a set of authorized addresses.
- Membership Proof Lists. A membership proof list is a set of membership proofs for a coin commitment. Each membership proof asserts membership on a specific allowlist in the system.
- Membership Declarations. A membership declaration $\mathsf{decl} := H_q(\mathsf{al} \| \mathsf{pk}_s)$ is a public reference to an allowlist identifier al and a user's public key pk_s.
- Deposit Records. A deposit record $\mathsf{rec}_{\mathsf{dep}} := H_q(\mathsf{amt} \| \mathsf{pk}_s \| \mathsf{cm} \| \mathsf{uid} \| \mathsf{rt}_{\mathsf{id}})$ is a public record for a deposit into the privacy pool that is derived from the value amount amt, the user's public key pk_s, the coin commitment cm generated upon deposit, the unique identifier uid of the deposit, and the current digest of the membership declaration history accumulator $\mathsf{rt}_{\mathsf{id}}$.
- Transfer Records. A transfer record $\mathsf{rec}_{\mathsf{tfr}} := H_q(\mathsf{nf} \| \mathsf{cm})$ is a public record for a pool transfer that is derived from the nullifier nf for a transaction input and the coin commitment cm for a transaction output. A transfer record is generated for each input-output pair.
- History Accumulators. The disclosure system uses sparse Merkle history trees to efficiently prove set membership and ensure historical consistency. There are three history accumulators: the membership declaration history accumulator with digest $\mathsf{rt}_{\mathsf{id}}$, the deposit record history accumulator with digest $\mathsf{rt}_{\mathsf{dep}}$, and the transfer record history accumulator with digest $\mathsf{rt}_{\mathsf{tfr}}$. These history accumulators are maintained off-chain using public information derived from the blockchain state.

The following definitions will be helpful for specifying the system's interface:

- Asset. An asset $A = (\mathsf{cm}; I)$ consists of a coin commitment cm and auxiliary input $I \in \mathcal{I}$ containing details of the system's public state, where $\mathcal{I} = \{0,1\}^*$ is the auxiliary input distribution. We let $\mathbf{A} = (A_i)_{i=1}^n$ denote a list of n assets where n is the number of transaction inputs.
- Transaction Parameter. A transaction parameter $\mathsf{tx} \in \{0,1\}^*$ consists of details of a deposit, transfer, or withdrawal transaction that are defined by the privacy pool construction and auxiliary information that is needed to prove asset membership. In our construction, this string contains information related to the membership declarations, deposit records, and history accumulators.

4.3 System Operations

The disclosure system supports the operations below, including an initial setup and transaction processing methods to update the history accumulators. The update operations reference transactions that are produced by the privacy pool, whose interface is described in Section 3.2.

- $\mathsf{DisclosureSystemSetup}(1^\lambda) \to \mathsf{pp}_{\mathsf{disc}}$. This algorithm sets up the initial state of the disclosure system, including history accumulators and account lists. Returns the public parameters $\mathsf{pp}_{\mathsf{disc}}$.
- $\mathsf{ProcessDepositTx}(\mathsf{pp}_{\mathsf{disc}}; \mathsf{tx}_{\mathsf{dep}}) \to (b; \mathsf{rec}_{\mathsf{dep}}; \mathsf{uid})$. This algorithm creates and accumulates the deposit record and generates a unique identifier for the deposit transaction. Returns accept/reject bit b, deposit record $\mathsf{rec}_{\mathsf{dep}}$, and unique identifier uid for the deposit transaction.
- $\mathsf{ProcessTransferTx}_{n;m}(\mathsf{pp}_{\mathsf{disc}}; \mathsf{tx}_{\mathsf{tfr}}) \to (b; \mathbf{rec}_{\mathbf{tfr}})$. This algorithm creates and accumulates transfer records for in-pool transfers. Returns accept/reject bit b and a list of transfer records $\mathbf{rec}_{\mathbf{tfr}}$.
- $\mathsf{ProcessWithdrawalTx}_n(\mathsf{pp}_{\mathsf{disc}}; \mathsf{tx}_{\mathsf{wdr}}) \to (b; \mathbf{rec}_{\mathbf{tfr}})$. This algorithm creates and accumulates transfer records for pool withdrawals. Returns accept/reject bit b and a list of transfer records $\mathbf{rec}_{\mathbf{tfr}}$.

The client supports the following operations for interacting with the disclosure system. When a client transfers funds within the privacy pool, the client will separately create a membership proof list for each of the output coin commitments. When a client withdraws from the privacy pool, the client generates a final membership proof list. The privacy pool contract checks for transaction validity, but the contract does not have access to the membership proofs. These proofs can be communicated through a direct channel to the recipient rather than being posted on the blockchain.

- $\mathsf{CreateMembershipProof}_{n;m}(\mathsf{pp}_{\mathsf{disc}}; A; \ ; A_{\mathbf{in}}; \ _{\mathbf{in}}; \mathbf{al}) \ ! \ (A_{\mathbf{out}}; \ _{\mathbf{out}})$. Given public parameters $\mathsf{pp}_{\mathsf{disc}}$, a list of assets A, a list of transaction parameters , a list of input assets $A_{\mathbf{in}}$, a list of input membership proof lists $_{\mathbf{in}}$, and a list of allowlists $\mathbf{al}$, output a list of assets $A_{\mathbf{out}}$ and a list of membership proof lists $_{\mathbf{out}}$. A membership proof list is generated for each transaction output with respect to the allowlists $\mathbf{al}$. This algorithm is run by the sender and parametrized by the number of transaction inputs n and the number of transaction outputs m.
- $\mathsf{VerifyMembershipProof}(\mathsf{pp}_{\mathsf{disc}}; A; \ ; \mathsf{al}) \ ! \ b$. Given public parameters $\mathsf{pp}_{\mathsf{disc}}$, an asset A, a membership proof , and an allowlist al, return $b \ 2 \ f0; 1g$. This algorithm is run by the recipient.

The manager supports the following operations for interacting with the disclosure system. These operations result in the addition or removal of an address from the specified allowlists.

- $\mathsf{AuthorizeAccount}(\mathsf{pp}_{\mathsf{disc}}; \mathsf{pk}_S; \mathbf{al}) \ ! \ (b; \mathbf{decl})$. Given public parameters pp, a user public key pk_S, and a set of allowlists $\mathbf{al}$, the registry updates the user's membership on allowlists $\mathbf{al}$ if the user is not already a member of each allowlist. Membership declarations are created and added to the history accumulator. Returns accept/reject bit b and membership declaration list $\mathbf{decl}$.
- $\mathsf{RevokeMembership}(\mathsf{pp}_{\mathsf{disc}}; \mathsf{pk}_S; \mathbf{al}) \ ! \ b$. Given public parameters pp, a user public key pk_S, and a set of allowlists $\mathbf{al}$, the registry revokes the user's membership on allowlists $\mathbf{al}$ if the user is currently a member of each allowlist. Membership declarations are removed from the history accumulator. Returns accept/reject bit b.

4.4 Security Goals

The security goals of privacy pools consist of correctness, availability, confidentiality, and unlinkability.

Correctness ensures that a pool does not allow clients to spend coins that have already been spent or that they do not own. Availability ensures that clients cannot be prevented from using the privacy pool. Once coin commitments have been added to the contract state, clients cannot be prevented from spending coins that they own and have not previously spent. Confidentiality and unlinkability are the key privacy considerations. A pool ensures confidentiality of transactions if only the sender and recipient learn the value amount associated with each transaction. A pool ensures unlinkability of transfers and withdrawals if an adversary has a negligible advantage in guessing an input coin commitment associated with a given transfer or withdrawal transaction.

Derecho does not alter the functionality of the privacy pool and thus preserves these correctness and privacy properties. However, we also need to define additional correctness and privacy goals for *proof-carrying disclosures*. Correctness ensures that a client cannot attest membership of a transaction output on a given allowlist unless each of the transaction inputs has an attestation of membership on this allowlist or is a deposit from an address that is registered on this allowlist. Privacy ensures that the allowlist membership proof does not reveal anything besides the allowlist membership of the transaction output (e.g., it does not reveal the private transaction details).

In our construction, correctness will follow from the definition of the compliance predicate and privacy from the zero-knowledge property of the underlying PCD scheme. These properties are formalized below and are adapted from the general definition of PCD that is presented in Section 3.1.

4.5 Private Asset Membership

Definitions We require the following definitions to discuss the security of our construction.

- Historical State. The state of the system is represented with history accumulators S_T for transactions in the privacy pool and a history accumulator S_M for membership declarations. These history accumulators are managed offline and computed from the blockchain state.
- Transaction Validation. Let D_T be a decider for transaction correctness and acceptance. Let $D_T(S_T; A; \ ; (A_i)_{i=1}^n) = 1$ if there is a corresponding correctly-formed $\mathsf{tx}_{\mathsf{tfr}}$ or $\mathsf{tx}_{\mathsf{wdr}}$ object that has been accepted by the privacy pool operation ProcessTransferTx or ProcessWithdrawalTx, respectively, and has $(A_i)_{i=1}^n$ corresponding to its inputs and has A corresponding to one of its outputs. Let $D_T(S_T; A; \ ; ?) = 1$ if there is a corresponding well-formed $\mathsf{tx}_{\mathsf{dep}}$ object that has been accepted by the privacy pool operation ProcessDepositTx and has A as its output.
- Membership Validation. We let $D_M(S_M; A; \ ; \mathsf{al})$ be a decider that checks if the sender is a member of allowlist al at the time of deposit.

– Membership-Preserving Transaction. A membership-preserving transaction is one that satisfies the following rule. For a non-empty set of assets A and asset B such that A are inputs to the transaction and B is an output of the transaction, the asset B has membership on allowlist al if and only if for each $A \in A$, A has membership on allowlist al. We also consider a deposit transaction to be membership-preserving.
– Transaction Graph. A transaction graph G is a directed acyclic graph that consists of a set of vertices $V(G)$ and a set of edges $E(G)$ where each vertex $v \in V(G)$ corresponds to a deposit, transfer, or withdrawal transaction and each edge $e \in E(G)$ corresponds to a transaction output.
– Provenance Transcript. A provenance transcript PT is a transaction graph where each vertex $u \in V(\mathsf{PT})$ is labeled by a transaction parameter $^{(u)}$ and each edge $e \in E(\mathsf{PT})$ is labeled by an asset $A^{(e)}$ $\mathfrak{G}$?. The output of the provenance transcript, denoted $o(\mathsf{PT})$, is $A^{(e^0)}$ where $e^0 = (u, v)$ is the first edge such that v is a sink in the lexicographic ordering of the edges. We say that a provenance transcript is *compliant* if all of the transactions corresponding to the labels are correctly-formed, accepted by the privacy pool, and membership-preserving. We let $D_C(S_T, S_M, \mathsf{PT})$ be a decider for whether a provenance transcript is compliant according to the labels and the historical state.

Algorithms Given a privacy pool POOL, a disclosure system DISC, and a security parameter , a *Private Asset Membership* scheme is a tuple $\mathsf{PAM} = (\mathsf{G}, \mathsf{P}, \mathsf{V})$, where G is called the generator, P is called the prover, and V is called the verifier. These algorithms work as follows:

– The generator $\mathsf{G}(1^\lambda) \to \mathsf{pp}$, given a security parameter , generates the public parameters pp. The public parameters contain global information such as the history accumulators S_T and S_M.
– The prover $\mathsf{P}(\mathsf{pp}, A, , A_{\mathsf{in}}, _{\mathsf{in}}, \mathsf{al}) \to$, given asset A, a transaction parameter , input assets $A_{\mathsf{in}} = (A_i)_{i \in [n]}$, input proofs $_{\mathsf{in}} = (_i)_{i \in [n]}$, and allowlist al, returns membership proof .
– The verifier $\mathsf{V}(\mathsf{pp}, A, , \mathsf{al}) \to b$, given asset A, membership proof , and allowlist al, returns a decision $b \in \{0,1\}$ for whether asset A has membership on allowlist al.

Properties The completeness, knowledge soundness, and zero knowledge properties must hold for G, P, and V. These properties are defined below.

Completeness. It must always be possible to prove the membership of an asset with membership on allowlist al. For every adversary A, the following holds:

$$\Pr\left[\begin{array}{l}(D_T(S_T, A, , ?) = 1 \wedge D_M(S_M, A, , \mathsf{al}) = 1) \\ \mathfrak{G}\ (D_T(S_T, A, , (A_i)_{i=1}^n) = 1 \wedge \forall i, \mathsf{V}(\mathsf{pp}, A_i, _i, \mathsf{al})) \\ + \\ \mathsf{V}(\mathsf{pp}, A, , \mathsf{al}) = 1\end{array} \;\middle|\; \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda) \\ (A, , [A_i, _i]_{i=1}^n, \mathsf{al}) \leftarrow A(\mathsf{pp}) \\ \mathsf{P}(\mathsf{pp}, A, , [A_i, _i]_{i=1}^n, \mathsf{al})\end{array}\right] = 1.$$

Knowledge soundness. If the verifier accepts a proof for an asset generated by some adversary, then the asset has membership on allowlist al, and moreover, the adversary "knows" a compliant provenance transcript PT with output A.

Formally, a PAM scheme PAM has the knowledge soundness property if, for every expected polynomial-time adversary A, there exists an expected polynomial-time knowledge extractor E that can output a provenance transcript PT such that, for every sufficiently large security parameter ,

$$\Pr\left[\begin{array}{c}\mathsf{V}(\mathsf{pp}, A, , \mathsf{al}) = 1 \\ \wedge\ (o(\mathsf{PT}) \mathfrak{G} A \_ D_C(S_T, S_M, \mathsf{PT}) = 0)\end{array} \;\middle|\; \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda) \\ (A, , \mathsf{al}) \leftarrow A(\mathsf{pp}) \\ \mathsf{PT} \leftarrow E(\mathsf{pp})\end{array}\right] \leq \mathsf{negl}( ).$$

Zero knowledge. The PAM proofs reveal nothing besides the allowlist membership of the assets. Formally, the proofs are (statistical) zero knowledge if there exists a PPT simulator S such that for every honest adversary A the distributions below are statistically indistinguishable.

$$\left\{(\mathsf{pp}, A, , \mathsf{al}) \;\middle|\; \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda) \\ (A, , [A_i, _i]_{i=1}^n, \mathsf{al}) \leftarrow A(\mathsf{pp}) \\ \mathsf{P}(\mathsf{pp}, A, , [A_i, _i]_{i=1}^n, \mathsf{al})\end{array}\right\}, \text{ and } \left\{(\mathsf{pp}, A, , \mathsf{al}) \;\middle|\; \begin{array}{c}(\mathsf{pp},) \leftarrow S(1^\lambda) \\ (A, , [A_i, _i]_{i=1}^m, \mathsf{al}) \leftarrow A(\mathsf{pp}) \\ S(, A, \mathsf{al})\end{array}\right\},$$

5 Construction

We now present the details of our construction. After outlining the building blocks of the system in Section 5.1, we precisely define the membership proof statement in Section 5.2. We present a detailed specification of the disclosure system algorithms in Section 5.3. We present our formal construction of the Derecho Private Asset Membership scheme in Section 5.4 and sketch the security proof in Section 5.5. Since our system does not involve any changes to the functionality of the privacy pool, we defer a detailed discussion of the privacy pool algorithms to Appendix D.

5.1 Building Block Algorithms

Below we describe how to compute each of the key objects of the disclosure system. These algorithms are used in updating the history accumulators and producing the membership proofs.

- Membership Declaration Creation. For an allowlist al and public key pk_s, the membership declaration is computed by $\mathsf{decl} := H_q(\mathsf{al} k \mathsf{pk}_s)$.
- Deposit Record Creation. A deposit record is computed using a hash function that is applied to the value amount amt, the user's public key pk_s, the coin commitment cm generated upon deposit, the unique identifier uid of the deposit transaction, and the current digest of the membership declaration history accumulator $\mathsf{rt}_{\mathsf{id}}$. The deposit record is computed by $\mathsf{rec}_{\mathsf{dep}} := \mathsf{DepositRecord}(\mathsf{open}; \mathsf{pk}_s; \mathsf{uid}; \mathsf{rt}_{\mathsf{id}}) = H_q(\mathsf{Amt}(\mathsf{open}) k \mathsf{pk}_s k \mathsf{Com}(\mathsf{open}) k \mathsf{uid} k \mathsf{rt}_{\mathsf{id}})$, where open is the opening of the coin commitment cm generated upon deposit.
- Transfer Record Creation. A transfer record is computed using a hash function that is applied to an input nullifier nf and an output coin commitment cm. The transfer record is computed by $\mathsf{rec}_{\mathsf{tfr}} := H_q(\mathsf{nf} k \mathsf{cm})$ without reference to any private values.

5.2 Recursive Membership Proofs

This section defines the PCD system for attestations of allowlist membership for a transaction output. Let n be the number of transaction inputs, m be the number of transaction outputs, and l be the number of allowlists. For simplicity, we fix the set of allowlists $(\mathsf{al}_j)_{j \in [l]}$ to yield a set of compliance predicates $(\Phi_j)_{j \in [l]}$. We refer to Section 3.1 for a complete description of proof-carrying data.

The compliance predicate Φ_j is a function of the message z, the local data z_{loc}, and the incoming messages $(z_i)_{i \in [n]}$. Each compliance predicate Φ_j is defined with respect to a specific allowlist al_j. A message z consists of public data associated with the transaction output: the input nullifiers, the output coin commitment, and additional data related to the transfer records and history accumulators. The local data z_{loc} consists of private data associated with the transaction output: input and output coin commitment openings, auxiliary inputs for the nullifier computation, membership declarations, deposit information, membership witnesses for the accumulated elements (i.e., deposit records, transfer records, and membership declarations), and history proofs for the history accumulator digests.

Each transfer transaction corresponds to a vertex in the proof-carrying data graph G. This vertex typically has n incoming edges and m outgoing edges. However, this vertex has no incoming edges when all transaction inputs consist of fresh deposits to the privacy pool. For a node with incoming edges, each message z_i corresponds to a message that was generated as the output of a previous transaction. If the node has no incoming edges, $z_i = ?$. A vertex u is Φ_j-compliant if for all outgoing edges with message z either: (1) if u has no incoming edges, $\Phi_j(z; z_{\mathsf{loc}}; ?; \dots; ?)$ evaluates to true or (2) if u has n incoming edges, $\Phi_j(z; z_{\mathsf{loc}}; z_1, \dots; z_n)$ evaluates to true. Note that $z_i = ?$ or $\mathsf{V}(\mathsf{ivk}_j; z_i; \quad_i) = 1$ for each incoming edge z_i. The prover generates an output proof $\quad = \mathsf{P}(\mathsf{ipk}_j; z; z_{\mathsf{loc}}; [z_i; \quad_i]_{i=1}^m)$.

If a vertex has no incoming edges, this indicates that each transaction input is a coin that was generated upon deposit to the pool. This is the base case of the compliance predicate. In this case, the predicate performs a series of checks for each transaction input. The predicate checks that the deposit record is correctly computed from the public data (i.e, the value amount, the user's public key, the deposit coin commitment, the unique identifier of the deposit transaction, and the current digest of the membership declaration history accumulator) and verifies that the deposit record is accumulated. The predicate checks that the membership declaration is correctly computed from the allowlist identifier and the user's public key and verifies that the membership declaration is accumulated. In this case, the prover is computing an attestation from public information in such a way that the initial attestation can be reused in subsequent attestations.

If a vertex has n incoming edges, this indicates that each transaction input is a coin that was the output of a previous transfer transaction. In this case, the membership proof for this transaction will attest to the validity of previous membership proofs with respect to previous messages. However, a problem arises where the history accumulator digests of previous messages may be stale with respect to the current state of the history accumulator for the current message. We thus additionally need to prove that the prior history accumulator digests represent correct historical states with respect to the current history accumulator digest. Otherwise, there is no guarantee that the prior history accumulator digests correspond to valid prior contract states. The predicate will ensure consistency by verifying that the prior history accumulator digest of message z_i is a valid historical digest according to the current history accumulator digest of message z. The predicate will verify history proofs for three history accumulators: the membership declaration history accumulator, the deposit record history accumulator, and the transfer record history accumulator. The history accumulators store public information derived from the blockchain state that is useful for ensuring that the membership proofs are consistent with the current state of the blockchain.

In both cases, the predicate computes nullifiers for the transaction inputs based on the input coin commitment openings and auxiliary data. The predicate computes the output coin commitment from its opening. The details of this logic are determined by the privacy pool. The predicate ensures the consistency of the output coin commitments of the previous messages with the input coin commitment openings of the current local data. Finally, the predicate computes the transfer record for each pair of input nullifier and output coin commitment and verifies that the transfer record is accumulated.

While these computations reference the (private) local data, the resulting proofs can be verified with respect to the corresponding (public) message. Each transaction output corresponds to an outgoing edge in the PCD graph, so each transaction output has a corresponding membership proof.

From the recipient's perspective, it is important to check the validity of the public information in the message z with respect to the privacy pool contract state, in addition to verifying the proof with respect to the message z. Otherwise, it is not guaranteed that the membership proof is meaningful. The recipient should be able to perform this check at any time with access to the current state.

Our design offers flexibility in combining coins with membership proofs on distinct sets of allowlists that have a non-empty intersection. For instance, a coin with membership proofs on allowlists al_1 and al_2 may be spent along with a coin with a membership proof on allowlist al_1 to produce a new coin with a membership proof on allowlist al_1 only.

Recall that membership proof generation does not require changes to the functionality of the privacy pool. Membership proofs are generated alongside the privacy pool transactions and provided directly to recipients. As a result, there is no increase in the gas costs of these transactions.

- Message $z := (\mathbf{nf}, \mathsf{cm}, \mathsf{rt}_{\mathsf{id}}, \mathsf{rt}_{\mathsf{dep}}, \mathsf{rt}_{\mathsf{tfr}}, \mathbf{rec}_{\mathbf{tfr}})$:
  - $\mathbf{nf} := (\mathsf{nf}_i)_{i \in [n]}$: Nullifiers for transaction inputs.
  - cm: Output coin commitment.
  - $\mathsf{rt}_{\mathsf{id}}$: Digest of membership declaration history accumulator.
  - $\mathsf{rt}_{\mathsf{dep}}$: Digest of deposit record history accumulator.
  - $\mathsf{rt}_{\mathsf{tfr}}$: Digest of transfer record history accumulator.
  - $\mathbf{rec}_{\mathbf{tfr}} := (\mathsf{rec}_i^{\mathsf{tfr}})_{i \in [n]}$: Transfer records derived from public information.
- Local data $z_{\mathsf{loc}} := (\mathbf{open}_{\mathbf{in}}, \mathsf{open}_{\mathsf{out}}, \mathbf{aux}_{\mathbf{in}}, \mathbf{decl}, \mathbf{rec}_{\mathbf{dep}}, \mathbf{pk}, \mathbf{uid}, \mathbf{w}_{\mathbf{id}}, \mathbf{c}_{\mathbf{id}}, \mathbf{w}_{\mathbf{dep}}, \mathbf{c}_{\mathbf{dep}}, \mathbf{w}_{\mathbf{tfr}}, \mathbf{c}_{\mathbf{tfr}})$:
  - $\mathbf{open}_{\mathbf{in}} := (\mathsf{open}_i^{\mathsf{in}})_{i \in [n]}$: Input coin commitment openings.
  - $\mathsf{open}_{\mathsf{out}}$: Output coin commitment opening.
  - $\mathbf{aux}_{\mathbf{in}} := (\mathsf{aux}_i^{\mathsf{in}})_{i \in [n]}$: Auxiliary inputs for nullifier computation.
  - $\mathbf{decl} := (\mathsf{decl}_i)_{i \in [n]}$: Membership declarations for the allowlist al_j.
  - $\mathbf{rec}_{\mathbf{dep}} := (\mathsf{rec}_i^{\mathsf{dep}})_{i \in [n]}$: Deposit records for the deposit transactions.
  - $\mathbf{pk} := (\mathsf{pk}_i)_{i \in [n]}$: Public keys for the deposit transactions.
  - $\mathbf{uid} := (\mathsf{uid}_i)_{i \in [n]}$: Unique identifiers for the deposit transactions.
  - $\mathbf{w}_{\mathbf{id}} := (\mathsf{w}_i^{\mathsf{id}})_{i \in [n]}$: Membership witnesses for membership declarations $\mathbf{decl}$ and digest $\mathsf{rt}_{\mathsf{id}}$.
  - $\mathbf{c}_{\mathbf{id}} := (\mathsf{c}_i^{\mathsf{id}})_{i \in [n]}$: History proofs for digest $\mathsf{rt}_{\mathsf{id}}$ with respect to previous digests $\hat{\mathsf{rt}}_i^{\mathsf{id}}$.
  - $\mathbf{w}_{\mathbf{dep}} := (\mathsf{w}_i^{\mathsf{dep}})_{i \in [n]}$: Membership witnesses for deposit records $\mathbf{rec}_{\mathbf{dep}}$ and digest $\mathsf{rt}_{\mathsf{dep}}$.
  - $\mathbf{c}_{\mathbf{dep}} := (\mathsf{c}_i^{\mathsf{dep}})_{i \in [n]}$: History proofs for digest $\mathsf{rt}_{\mathsf{dep}}$ with respect to previous digests $\hat{\mathsf{rt}}_i^{\mathsf{dep}}$.
  - $\mathbf{w}_{\mathbf{tfr}} := (\mathsf{w}_i^{\mathsf{tfr}})_{i \in [n]}$: Membership witnesses for transfer records $\mathbf{rec}_{\mathbf{tfr}}$ and digest $\mathsf{rt}_{\mathsf{tfr}}$.
  - $\mathbf{c}_{\mathbf{tfr}} := (\mathsf{c}_i^{\mathsf{tfr}})_{i \in [n]}$: History proofs for digest $\mathsf{rt}_{\mathsf{tfr}}$ with respect to previous digests $\hat{\mathsf{rt}}_i^{\mathsf{tfr}}$.

- Previous messages $(z_i)_{i \in [n]}$:
  - $z_i := (\hat{\mathbf{nf}}_i, \hat{\mathsf{cm}}_i, \hat{\mathsf{rt}}_i^{\mathsf{id}}, \hat{\mathsf{rt}}_i^{\mathsf{dep}}, \hat{\mathsf{rt}}_i^{\mathsf{tfr}}, \hat{\mathbf{rec}}_{\mathbf{tfr}})$
  - z_i is a message for the i-th transaction input.
  - Each message z_i has the same format as z.
- Previous proofs $(\ _i)_{i \in [n]}$:
  - $_i$ is a proof for the i-th transaction input.
  - Each proof $_i$ can be verified with respect to z_i.
- Compliance predicate $'_j(z, z_{\mathsf{loc}}, z_1, \ldots, z_n)$ for allowlist al_j:
  - For $i \in [n]$:
    - For base case ($z_i = ?$), check the following:
      - $\mathsf{decl}_i = H_q(\mathsf{al}_j \| \mathsf{pk}_i)$
      - $\mathsf{HA.VfyMem}(\mathsf{rt}_{\mathsf{id}}, \mathsf{w}_i^{\mathsf{id}}, \mathsf{decl}_i)$
      - $\mathsf{rec}_i^{\mathsf{dep}} = \mathsf{DepositRecord}(\mathsf{open}_i^{\mathsf{in}}, \mathsf{pk}_i, \mathsf{uid}_i, \mathsf{rt}_{\mathsf{id}})$
      - $\mathsf{HA.VfyMem}(\mathsf{rt}_{\mathsf{dep}}, \mathsf{w}_i^{\mathsf{dep}}, \mathsf{rec}_i^{\mathsf{dep}})$
    - Otherwise, check the consistency of messages:
      - $\hat{\mathsf{cm}}_i = \mathsf{Com}(\mathsf{open}_i^{\mathsf{in}})$
      - $\mathsf{HA.VfyHist}(\hat{\mathsf{rt}}_i^{\mathsf{id}}, \mathsf{rt}_{\mathsf{id}}, \mathsf{c}_i^{\mathsf{id}})$
      - $\mathsf{HA.VfyHist}(\hat{\mathsf{rt}}_i^{\mathsf{dep}}, \mathsf{rt}_{\mathsf{dep}}, \mathsf{c}_i^{\mathsf{dep}})$
      - $\mathsf{HA.VfyHist}(\hat{\mathsf{rt}}_i^{\mathsf{tfr}}, \mathsf{rt}_{\mathsf{tfr}}, \mathsf{c}_i^{\mathsf{tfr}})$
  - For $i \in [n]$:
    - $\mathsf{nf}_i = \mathsf{Nullify}(\mathsf{open}_i^{\mathsf{in}}, \mathsf{aux}_i^{\mathsf{in}})$
  - $\mathsf{cm} = \mathsf{Com}(\mathsf{open}_{\mathsf{out}})$
  - For $i \in [n]$:
    - $\mathsf{rec}_i^{\mathsf{tfr}} = H_q(\mathsf{nf}_i \| \mathsf{cm})$
    - $\mathsf{HA.VfyMem}(\mathsf{rt}_{\mathsf{tfr}}, \mathsf{w}_i^{\mathsf{tfr}}, \mathsf{rec}_i^{\mathsf{tfr}})$

5.3 Disclosure System Algorithms

We present the disclosure system algorithms in Figure 2. An example implementation of the privacy pool interface is contained in Appendix D. Recall that the client may create and submit three types of financial transactions to the pool: deposit transactions, transfer transactions, and withdrawal transactions. Membership proofs are produced alongside transfer and withdrawal transactions.

5.4 Private Asset Membership

We present our formal construction of the Derecho Private Asset Membership (PAM) scheme in Algorithms 1, 2 and 3. Recall that a PAM scheme $\mathsf{PAM} = (\mathsf{G}, \mathsf{P}, \mathsf{V})$ consists of generator, prover, and verifier algorithms. We present a security proof for the properties of this construction in Section 5.5.

5.5 Disclosure System Security

We present a security proof for the properties of proof-carrying disclosures that were discussed in Section 4.4. The details of the compliance predicate were presented in Section 5.2.

Correctness of proof-carrying disclosures follows from the completeness and knowledge soundness of the PAM scheme. For privacy, we must show that the asset A and membership proof do not reveal anything about the private transaction details. First, note that the corresponding message z for the asset A consists of public information that is already known by the recipient of the message. The nullifiers $\mathbf{nf}$ and output coin commitment cm are contained in the public state and linked by the corresponding transaction. The history accumulator digests $(\mathsf{rt}_{\mathsf{id}}, \mathsf{rt}_{\mathsf{dep}}, \mathsf{rt}_{\mathsf{tfr}})$ are part of the public state. The transfer records $\mathbf{rec}_{\mathbf{tfr}}$ are determined by the nullifiers and output coin commitment. Hence, the asset A does not reveal private transaction details. The membership proof does not reveal anything about these details according to the zero knowledge property of the PAM scheme.

We now sketch the proof that the Derecho PAM scheme satisfies the desired properties. Recall that a PAM scheme $\mathsf{PAM} = (\mathsf{G}, \mathsf{P}, \mathsf{V})$ consists of generator, prover, and verifier algorithms with properties of completeness, knowledge soundness, and zero knowledge as defined in Section 4.5.

DisclosureSystemSetup(1^λ)

Sample hash function $H_q : \{0,1\}^* \to \mathbb{F}_q$
Create $(\mathsf{rt}_{\mathsf{id}}, \mathsf{st}_{\mathsf{id}}) = \mathsf{HA.Init}(1^\lambda)$
Create $(\mathsf{rt}_{\mathsf{dep}}, \mathsf{st}_{\mathsf{dep}}) = \mathsf{HA.Init}(1^\lambda)$
Create $(\mathsf{rt}_{\mathsf{tfr}}, \mathsf{st}_{\mathsf{tfr}}) = \mathsf{HA.Init}(1^\lambda)$
Construct PCD compliance predicates $(\Phi_j)_{j\in[l]}$
Perform PCD setup: $\mathsf{pp}_{\mathsf{pcd}} = \mathsf{PCD.G}(1^\lambda)$
for $j \in [l]$ **do**
  Generate keys: $(\mathsf{ipk}_j, \mathsf{vpk}_j) \leftarrow \mathsf{PCD.I}(\mathsf{pp}_{\mathsf{pcd}}, \Phi_j)$
endfor
Set $\mathsf{AuthAccountList} = \{\}$
Populate $\mathsf{AuthAccountList}$ for each allowlist al_j
Set $\mathsf{pp}_{\mathsf{disc}} = \{H_q, \mathsf{rt}_{\mathsf{id}}, \mathsf{st}_{\mathsf{id}}, \mathsf{rt}_{\mathsf{dep}}, \mathsf{st}_{\mathsf{dep}}, \mathsf{rt}_{\mathsf{tfr}}, \mathsf{st}_{\mathsf{tfr}}, (\mathsf{ipk}_j, \mathsf{vpk}_j)_{j\in[l]}, \mathsf{AuthAccountList}\}$
return $\mathsf{pp}_{\mathsf{disc}}$

ProcessDepositTx$(\mathsf{pp}_{\mathsf{disc}}, \mathsf{tx}_{\mathsf{dep}})$

Parse $(\mathsf{cm}, \mathsf{amt}) = \mathsf{tx}_{\mathsf{dep}}$
Set $\mathsf{pk}_s = \mathsf{tx}_{\mathsf{dep}}.\mathsf{sender}$
Compute unique identifier uid for deposit
Compute $\mathsf{rec}_{\mathsf{dep}} = H_q(\mathsf{amt} \| \mathsf{pk}_s \| \mathsf{cm} \| \mathsf{uid} \| \mathsf{rt}_{\mathsf{id}})$
Update $\mathsf{rt}_{\mathsf{dep}} = \mathsf{HA.Update}(\mathsf{rt}_{\mathsf{dep}}, \mathsf{st}_{\mathsf{dep}}, \mathsf{rec}_{\mathsf{dep}})$
return $(1, \mathsf{rec}_{\mathsf{dep}}, \mathsf{uid})$

ProcessTransferTx$_{n,m}(\mathsf{pp}_{\mathsf{disc}}, \mathsf{tx}_{\mathsf{tfr}})$

Parse $(\mathbf{nf}, \mathbf{cm}, \mathbf{memo}, \mathsf{rt}_c, \pi_t) = \mathsf{tx}_{\mathsf{tfr}}$
Set $\mathbf{rec_{tfr}} = []$
for $i \in [m]$ **do**
  for $j \in [n]$ **do**
    Create transfer record $\mathsf{rec}_{\mathsf{tfr}} = H_q(\mathbf{nf}[j] \| \mathbf{cm}[i])$
    Update $\mathsf{rt}_{\mathsf{tfr}} = \mathsf{HA.Update}(\mathsf{rt}_{\mathsf{tfr}}, \mathsf{st}_{\mathsf{tfr}}, \mathsf{rec}_{\mathsf{tfr}})$
    Add $\mathsf{rec}_{\mathsf{tfr}}$ to $\mathbf{rec_{tfr}}$
  endfor
endfor
return $(1, \mathbf{rec_{tfr}})$

ProcessWithdrawalTx$_n(\mathsf{pp}_{\mathsf{disc}}, \mathsf{tx}_{\mathsf{wdr}})$

Parse $(\mathsf{amt}, \mathsf{addr}_{\mathsf{out}}, \mathbf{nf}, \mathsf{cm}, \mathsf{rt}_c, \pi_w) = \mathsf{tx}_{\mathsf{wdr}}$
Set $\mathbf{rec_{tfr}} = []$
for $i \in [n]$ **do**
  Set $\mathsf{nf} = \mathbf{nf}[i]$
  Create transfer record $\mathsf{rec}_{\mathsf{tfr}} = H_q(\mathsf{nf} \| \mathsf{cm})$
  Update $\mathsf{rt}_{\mathsf{tfr}} = \mathsf{HA.Update}(\mathsf{rt}_{\mathsf{tfr}}, \mathsf{st}_{\mathsf{tfr}}, \mathsf{rec}_{\mathsf{tfr}})$
  Add $\mathsf{rec}_{\mathsf{tfr}}$ to $\mathbf{rec_{tfr}}$
endfor
return $(1, \mathbf{rec_{tfr}})$

CreateMembershipProof$_{n,m}(\mathsf{pp}_{\mathsf{disc}}, \boldsymbol{A}, \ldots, \boldsymbol{A}_{\mathsf{in}}, \pi_{\mathsf{in}}, \mathbf{al})$

Set $\boldsymbol{A}_{\mathbf{out}} = []$ and $\pi_{\mathbf{out}} = []$
Parse $\boldsymbol{z}$ from $\boldsymbol{A}$, $\boldsymbol{z}_{\mathbf{loc}}$ from $\ldots$, and $\boldsymbol{z}_{\mathbf{in}}$ from $\boldsymbol{A}_{\mathbf{in}}$
for $\mathsf{al} \in \mathbf{al}$ **do**
  Compute index j for allowlist al
  for $i \in [m]$ **do**
    Compute $\pi^{\mathsf{out}}_{j\,i} = \mathsf{PCD.P}(\mathsf{ipk}_j, z_{j\,i}, z^{\mathsf{loc}}_{j\,i}, [z^{\mathsf{in}}_{j\,k}, \pi^{\mathsf{in}}_{j\,k}]^n_{k=1})$
    Set $\boldsymbol{A}_{\mathbf{out}}[j][i] = A$ and $\pi_{\mathbf{out}}[j][i] = \pi^{\mathsf{out}}_{j\,i}$
  endfor
endfor
return $(\boldsymbol{A}_{\mathbf{out}}, \pi_{\mathbf{out}})$

VerifyMembershipProof$(\mathsf{pp}_{\mathsf{disc}}, A, \ldots, \mathsf{al})$

Parse message z from asset A
Compute index j for allowlist al
Compute verification result $b = \mathsf{PCD.V}(\mathsf{ivk}_j, z, \ldots)$
return b

AuthorizeAccount$(\mathsf{pp}_{\mathsf{disc}}, \mathsf{pk}_s, \mathbf{al})$

Set $\mathbf{decl} = []$
for $\mathsf{al} \in \mathbf{al}$ **do**
  Require pk_s is not in $\mathsf{AuthAccountList}[\mathsf{al}]$
  Add pk_s to $\mathsf{AuthAccountList}[\mathsf{al}]$
  Compute $\mathsf{decl} = H_q(\mathsf{al} \| \mathsf{pk}_s)$
  Add decl to $\mathbf{decl}$
  Update $\mathsf{rt}_{\mathsf{id}} = \mathsf{HA.Update}(\mathsf{rt}_{\mathsf{id}}, \mathsf{st}_{\mathsf{id}}, \mathsf{decl})$
endfor
return $(1, \mathbf{decl})$

RevokeMembership$(\mathsf{pp}_{\mathsf{disc}}, \mathsf{pk}_s, \mathbf{al})$

for $\mathsf{al} \in \mathbf{al}$ **do**
  Require pk_s is in $\mathsf{AuthAccountList}[\mathsf{al}]$
  Compute $\mathsf{decl} = H_q(\mathsf{al} \| \mathsf{pk}_s)$
  Update $\mathsf{rt}_{\mathsf{id}} = \mathsf{HA.Remove}(\mathsf{rt}_{\mathsf{id}}, \mathsf{st}_{\mathsf{id}}, \mathsf{decl})$
endfor
return 1

Fig. 2: Disclosure System Algorithms.

Algorithm 1 Derecho generator $G(1^\lambda)$

Input: security parameter
Output: public parameters pp

1: $pp_{pool} \leftarrow POOL.PrivacyPoolSetup(1^\lambda)$
2: $pp_{disc} \leftarrow DISC.DisclosureSystemSetup(1^\lambda)$
3: **return** (pp_{pool}, pp_{disc})

Algorithm 2 Derecho prover $P(pp, A, \ldots, A_{in}, \ldots_{in}, al)$

Input: asset A, transaction parameter $\ldots$, input assets $\boldsymbol{A}_{in}$, membership proofs $\ldots_{in}$, allowlist al
Output: membership proof

1: $(\boldsymbol{A}_{out}, \ldots_{out}) \leftarrow DISC.CreateMembershipProof(pp_{disc}, [[A]], [[\ldots]], [\boldsymbol{A}_{in}], [\ldots_{in}], [al])$
2: Check $A \stackrel{?}{=} \boldsymbol{A}_{out}[0][0]$
3: **return** $\ldots_{out}[0][0]$

Algorithm 3 Derecho verifier $V(pp, A, \ldots, al)$

Input: asset A, membership proof $\ldots$, allowlist al
Output: verification result $b \in \{0,1\}$

1: $b \leftarrow DISC.VerifyMembershipProof(pp_{disc}, A, \ldots, [al])$
2: **return** b

Completeness We prove correctness of the compliance predicate in two parts. These parts correspond to the base case (deposit) and the regular case (transfer/withdrawal) of the compliance predicate. To simplify the analysis, we let $n = 1$, where n is the number of transaction inputs. It is easy to see that the case of $n > 1$ follows directly from the proofs in these two parts.

First, let z be a message with local data z_{loc} that corresponds to an accepted transaction that spends a deposit originating from an address pk_1^{in} on allowlist al. The correctness of the computation of membership declaration $decl_1$ and deposit record rec_1^{dep} follows from the logic of the ProcessDepositTx and AuthorizeAccount algorithms. Furthermore, the correctness of set membership verification follows from the correctness of the history accumulator scheme. The correctness of nullifier computation and output coin commitment computation follows from the acceptance of the transaction by the privacy pool. In particular, these computations are verified in the CreateTransferTx and ProcessTransferTx algorithms of the privacy pool, and the compliance predicate repeats this logic. The correctness of the computation of transfer record rec_{tfr} follows from the logic of the ProcessTransferTx algorithm.

Second, let z be a message with local data z_{loc} that corresponds to an accepted transaction that spends the output of a previous transfer transaction with membership on allowlist al. The correctness of historical state verification follows from the correctness of the history accumulator scheme. The correctness of the computation of the previous output coin commitment follows from the completeness of the PCD scheme. The correctness of the computation of the nullifier, the current output coin commitment, and the transfer record follows as above. Likewise, the correctness of set membership verification follows from the correctness of the history accumulator scheme.

The successful verification of the previous membership proof follows from the completeness of the PCD scheme. Hence the final proof will be convincing with probability 1.

Knowledge Soundness The compliant provenance transcript can be viewed as a transcript in the PCD scheme, where the assets are the messages on the edges, and the transaction parameters are the local data at the nodes. The knowledge soundness of the PAM scheme then follows from the knowledge soundness of the PCD scheme.

Let A be an adversary that is attacking the Derecho PAM scheme. To show knowledge soundness, we must find a polynomial-time extractor E such that whenever a convincing proof $\ldots$ is found that A is an asset with membership on the allowlist al, E produces the evidence $\ldots$.

Using A, we construct A_{PCD}, an adversary attacking the PCD scheme. Given the public parameters of the PAM scheme (including system state) as auxiliary input, the adversary A_{PCD} will run the adversary A to produce a PCD message o and a proof $\ldots$ corresponding to the output of A. From an equivalent formulation of the PCD knowledge soundness property, there is an extractor P such that (for sufficiently large $\ldots$) and every (polynomial-length) auxiliary input ai, the following holds:

$$\Pr\left[\begin{array}{c} ' \in \mathsf{F} \\ \wedge\ (o(\mathsf{T}) \notin o\_\mathsf{T} \text{ is not } '\text{-compliant}) \\ \wedge\ \mathsf{V}(\mathsf{ivk}; o; \ ) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{pcd}} \quad G(1\ ) \\ \mathsf{ai} \quad D(\mathsf{pp}_{\mathsf{pcd}}) \\ (\ '; o; \ ; \mathsf{ao}) \quad \mathsf{P}(\mathsf{pp}_{\mathsf{pcd}}; \mathsf{ai}) \\ (\mathsf{ipk}; \mathsf{ivk}) \quad \mathsf{I}(\mathsf{pp}_{\mathsf{pcd}}; \ ') \end{array}\right] \quad \mathsf{negl}(\ ):$$

The extractor E works as follows. When A outputs an asset A and a proof $\ $, invoke the extractor P and read off the compliant provenance transcript from the output transcript's labels. Since A outputs a PCD proof, it follows that $\mathsf{V}(A; \ ; \mathsf{al}) = 1$ if the difference in the probability that T is $'$-compliant and the probability that PT is compliant is negligible. Recall that a transcript T is $'$-compliant if all of its vertices are $'$-compliant. A vertex is $'$-compliant for a predicate $' \in \mathsf{F}$ if for all outgoing edges $e = (u; v) \in E(\mathsf{T})$ in the transcript, either: (1) if u has no incoming edges, $'(z^{(e)}; z_{\mathsf{loc}}^{(u)}; ?; \ldots; ?) = 1$ or (2) if u has m incoming edges $e_1; \ldots; e_m$, $'(z^{(e)}; z_{\mathsf{loc}}^{(u)}; z^{(e_1)}; \ldots; z^{(e_m)}) = 1$. Similarly, a provenance transcript PT is compliant if all of its vertices are compliant, i.e., all of the transactions that correspond to to the labels of the provenance transcript are correctly-formed, accepted by the pool, and membership-preserving. If T is $'$-compliant but PT is not compliant, then the compliance predicate $'$ has not captured the desired security properties, i.e., an asset has an invalid attestation of allowlist membership.

To show these probabilities are negligibly close, we need to consider two cases for the transaction input logic in the compliance predicate. In the base case ($z_i = ?$), the difference in these probabilities is negligible according to the soundness of the history accumulator scheme, the collision-resistance of the hash function, and the binding property of the commitment scheme. Specifically, it is hard to find an invalid membership witness w that causes the VfyMem algorithm of the history accumulator to accept, and it is hard to find an invalid history proof c that causes the $\mathsf{VfyHist}$ algorithm of the history accumulator to accept. Likewise, it is hard to find a collision for the hash function or open the commitment to an invalid value. In the regular case ($z_i \notin ?$), the difference in these probabilities is negligible according to the soundness of the history accumulator scheme and the binding property of the commitment scheme. The transaction logic is sound as it simply repeats the transaction logic that is defined by the privacy pool. The transfer record logic is sound according to the collision-resistance of the hash function and the soundness of the history accumulator scheme.

Zero Knowledge The (statistical) zero knowledge property of the PAM scheme follows directly from the zero knowledge property of the PCD scheme. In particular, the PCD simulator S can be used to construct the PAM simulator S needed to show that the PAM scheme is zero knowledge.

6 Evaluation

We implement our construction of proof-carrying disclosures using Rust and the Arkworks ecosystem [ac22]. Our implementation consists of 5500 lines of code and is available open source.[2] The implementation consists of the compliance predicate circuit and a system for generating proof-carrying disclosures for example transactions. The constraints and proof-carrying data primitives are implemented using Arkworks. We instantiate the PCD scheme of [BCL+21] with the Pasta cycle of elliptic curves [Hop20].[3] This scheme optimizes for prover efficiency and uses a transparent setup.

The compliance predicate circuit consists of 51,062 constraints. A component-wise breakdown of the constraints is provided in Table 3. The verification of the membership proofs and history proofs accounts for the majority of the constraints in the compliance predicate. The constraint count has been optimized by using the $\mathsf{Poseidon}$ [GKR+21] hash function.

We evaluated the performance on a laptop with an Apple M1 Max processor. Figure 4 contains proving/verification times for a range of Merkle tree depth values. For a Merkle tree depth of 20 and single-threaded execution, the setup time was 5.5 seconds, the proving time was 19.5 seconds, and the verification time was 13.0 seconds. With multi-threaded execution, the setup time was 2.0 seconds, the proving time was 3.4 seconds, and the verification time was 1.9 seconds. The proof size was 6.3 MB for a tree depth of 20. Other PCD schemes such as [BCTV14] could be used, resulting in different

[2] `https://github.com/joshbeal/derecho`

[3] If the privacy pool uses a different curve for hash computation (e.g., the BN-254 curve), there will be overhead from non-native field arithmetic in verifying the nullifier and coin commitment computations. In this evaluation, we assume usage of the Poseidon hash function with the Pallas curve as in Zcash [HBHW22].

Component	Sub-component	Constraints
Membership	Value Computation	1,258
	Membership Proof	6,161
	History Proof	6,161
Deposit Record	Value Computation	7,142
	Membership Proof	6,161
	History Proof	6,161
Transfer Record	Value Computation	1,990
	Membership Proof	6,161
	History Proof	6,161
Transaction	Value Computation	3,705
	Value Consistency	1
Total	–	**51,062**

Fig. 3: Component-wise breakdown of R1CS constraints for the compliance predicate in the recursive membership proof.

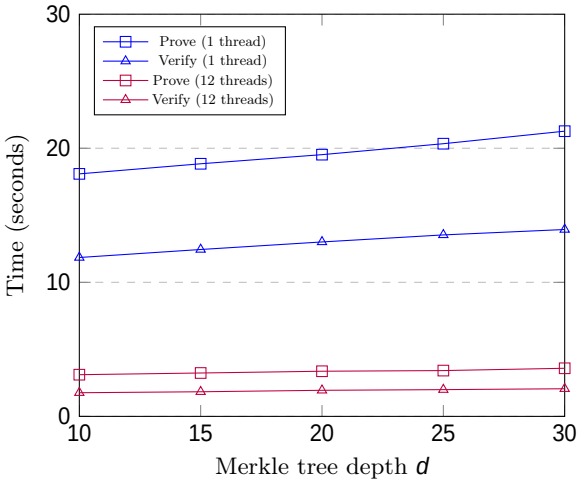

Fig. 4: Recursive membership proof generation time and verification time for a range of system parameters and parallel processing configurations.

tradeoffs in the setup type, proving/verification time, and proof size. The scheme of [BCTV14] results in short proofs (200 bytes) but yields a significantly higher proving time.

Our design does not introduce additional gas costs and does not change the functionality of the privacy pool contract. The registry containing the allowlists and the history accumulators can be maintained offline based on the blockchain state.

7 Discussion

Extensions Derecho supports attestations of membership on allowlists. Allowlists can also be used to implement blocklists by simply including every *unblocked* address on the allowlist. The challenge with supporting blocklists more directly is that a user can easily transfer funds from a blocklisted address pk_s to a fresh Ethereum address pk_s^0 before depositing into the privacy pool. The funds might then be transfered several hops within the pool before the blocklist could be updated to include pk_s^0. At this stage, the output records of these hops would include valid proofs of non-membership. It would be infeasible to require these proofs to be updated relative to the newer states of the blocklists because the holders of those records do not have knowledge of the records' origin, only the non-membership attestations that were valid against the previous blocklist states. In the case of allowlists, users can be required to register new addresses on the allowlists so that freshly created addresses are not automatically included, preventing users from undermining the disclosure proof system by creating a fresh Ethereum address before depositing into the pool. Given that using allowlists to implement blocklists results in enormously large lists, it would be interesting to develop alternative constructions that more directly support blocklist non-membership proofs, while preventing attacks of the nature described above. One solution is to require a proof of account age to avoid the issue with freshly created addresses. There are known solutions to efficiently proving account age using historical blockchain data [Int23]. Furthermore, there are PCD-friendly accumulators such as indexed Merkle trees [TKPS21] that support relatively efficient non-membership proofs.

Derecho addresses the common scenario where a sanctioned organization exploits a smart contract to steal cryptocurrency and launders the stolen funds through a privacy pool. However, a potential concern is that an entity who had funds in the pool before becoming sanctioned or receives funds through an in-pool transfer might subsequently transfer those funds to another person/entity. One solution to this problem, requiring minimal changes to the compliance predicate, is as follows. Users can additionally register their shielded addresses on allowlists. The shielded address pk can be computed by rerandomizing the public Ethereum address pk_s (e.g., $\mathsf{pk} = H_q(\mathsf{sk}k\mathsf{pk}_s)$ where sk is the secret key). When creating a disclosure, the sender can additionally prove that its current shielded address is allowed. In this extension, the transfer record should reference the current state of the membership declaration accumulator $\mathsf{rt}_{\mathsf{id}}$ to enforce that the current state of the allowlist is used at each step.

While it may be difficult to sanction an entity based on its actions inside a privacy pool, this is an independent concern. The allowlist manager may rely on other information for authorization.

Limitations Under stricter circumstances, an exchange may wish to know that certain funds did not recently pass through newly sanctioned entities in the pool (even if they were allowed at the time of transfer). While Derecho supports removal of addresses from allowlists to facilitate sanctions on previously approved entities, it may be desirable to additionally support *revocation* of allowlist membership proofs for existing transaction outputs. However, this would conflict with the privacy goals. Suppose there exists a construction where a single address may be removed from an allowlist al to create a new allowlist al' such that old membership proofs for al can be updated to support verification against al'. Then a membership proof that was previously valid against the old allowlist al but not the new allowlist al' can be used to break the privacy guarantee of unlinkability. More precisely, this implies that the party who is able to compute al' and the party who is able to update a membership proof for funds stored at a given record within the privacy pool would be able to collude at any time to discover all addresses on al from which the funds originated. We leave exploration of relaxed privacy models that might be compatible with membership proof revocation for future work.

8 Related Work

Sander and Ta-Shma [ST99] and Camenisch et al. [CHL06] established the foundations of accountable privacy for ecash systems. With the growing popularity of cryptocurrencies, several works have examined trade-offs between privacy and accountability/auditability in the design of decentralized payment systems. Garman et al. [GGM16] demonstrates how to add privacy-preserving policy enforcement mechanisms to the Zerocash design. UTT [TBA+22] designs a decentralized payment system that limits the the amount of currency sent per month using the notion of an anonymity budget. Platypus [WKDC22] and PEReDi [KKS22] explore the design of central bank digital currencies (CBDCs) with privacy-preserving regulatory functionality. Platypus focuses on enforcement of anonymity budgets and total balance limits. PEReDi supports compliance with regulations such as Know Your Customer (KYC), Anti Money Laundering (AML), and Combating Financing of Terrorism (CFT). Their system aims to avoid a single point of failure by distributing the policy enforcement mechanism. CAP [Esp22] introduces Configurable Asset Privacy schemes, which support private transfers of heterogeneous assets with custom viewing and freezing policies. ZEBRA [RPX+22] develops anonymous credentials that support auditability and revocation while enabling efficient on-chain verification. We refer to [CBC21] for a more detailed study of these research challenges.

ZEXE [BCG+20] provides a general framework for privacy-preserving blockchain applications in which the application state is a system of records, transactions create and nullify records, and all records have birth and death predicates defining the conditions under which they can be created or nullified. Transactions contain zero-knowledge proofs that these predicates are satisfied. As the authors note, this captures membership proofs of records on allowlists/blocklists as a special case (described in detail through a "regulation-friendly private stablecoin" example). In terms of comparison to *Derecho*, the ZEXE regulation-friendly stablecoin example restricts users of the stablecoin to a single allowlist (or blocks users on a single blocklist), represented as a credential assigned to the public key address of a user, while Derecho does not alter the functionality of privacy-preserving cryptocurrencies, enabling users to separately disclose allowlist provenance off-chain. Unlike Derecho, ZEXE does not address how users can prove statements about the origin of records within a hidden transaction graph, nor the added challenge that the users themselves cannot see the full details of transaction history aside from allowlist membership proofs of their existing records.

Proof-carrying data [CT10] (PCD) generalizes the notion of incrementally verifiable computation [Val08] (IVC) from sequential computation to distributed computation over a directed acyclic graph. The initial paper proposing PCD proposed several applications to the integrity of distributed computations, including distributed program analysis, type safety, IT supply chains, and conjectured applications to financial systems. Naveh and Tromer [NT16] proposed an application of PCD to image authentication, i.e., proving the authenticity of photos even after they have been edited according to a permissible set of transformations (e.g., cropping, rotation, scaling), which would invalidate signatures on the original image data. PCD (and IVC as a special case) has been used to construct authenticated data structures with richer invariants, such as append-only dictionaries [TFZ+22] and incrementally verifiable ledger systems [BMRS20, CCDW20].

9 Acknowledgments

This work was supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

References

ac22. arkworks contributors. `arkworks` zksnark ecosystem, 2022. `https://github.com/arkworks-rs/`.

AKR+13. E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in Bitcoin. In *FC 2013*, *LNCS* 7859, pages 34–51. Springer, Heidelberg, April 2013.

BBC+22. C. Bouvier, P. Briaud, P. Chaidos, L. Perrin, and V. Velichkov. Anemoi: Exploiting the link between arithmetization-orientation and CCZ-equivalence. Cryptology ePrint Archive, Report 2022/840, 2022. `https://eprint.iacr.org/2022/840`.

BCCT13. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

BCG+14. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

BCG+20. S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020.

BCL+21. B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. Proof-carrying data without succinct arguments. In *CRYPTO 2021, Part I*, *LNCS* 12825, pages 681–710, Virtual Event, August 2021. Springer, Heidelberg.

BCMS20. B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. Proof-carrying data from accumulation schemes. In *Theory of Cryptography*. Springer, 2020.

BCTV14. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO 2014, Part II*, *LNCS* 8617, pages 276–294. Springer, Heidelberg, August 2014.

BDFG21. D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Annual International Cryptology Conference*, pages 649–680. Springer, 2021.

BG13. S. Bayer and J. Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *EUROCRYPT 2013*, *LNCS* 7881, pages 646–663. Springer, Heidelberg, May 2013.

BGH19. S. Bowe, J. Grigg, and D. Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, 2019. `https://eprint.iacr.org/2019/1021`.

BKB22. J. Burleson, M. Korver, and D. Boneh. Privacy-protecting regulatory solutions using zero-knowledge proofs. 2022. `https://a16zcrypto.com/wp-content/uploads/2022/11/ZKPs-and-Regulatory-Compliant-Privacy.pdf`.

BKLZ20. B. Bünz, L. Kiffer, L. Luu, and M. Zamani. FlyClient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy*, pages 928–946. IEEE Computer Society Press, May 2020.

BMRS20. J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020. `https://eprint.iacr.org/2020/352`.

CBC21. P. Chatzigiannis, F. Baldimtsi, and K. Chalkias. SoK: Auditability and accountability in distributed payment systems. In *ACNS 21, Part II*, *LNCS* 12727, pages 311–337. Springer, Heidelberg, June 2021.

CCDW20. W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522, 2020. `https://eprint.iacr.org/2020/1522`.

CHL06. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *SCN 06*, *LNCS* 4116, pages 141–155. Springer, Heidelberg, September 2006.

Coi22. Coin Center. Tornado cash complaint, 2022. `https://www.coincenter.org/app/uploads/2022/10/1-Complaint-Coin-Center-10-12-22.pdf`.

COS20. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT 2020, Part I*, *LNCS* 12105, pages 769–793. Springer, Heidelberg, May 2020.

Cro10. S. A. Crosby. *Efficient tamper-evident data structures for untrusted servers*. Rice University, 2010.

CT10. A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010.

CW09. S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security 2009*, pages 317–334. USENIX Association, August 2009.

Esp22. Espresso Systems. Configurable asset privacy. 2022. https://github.com/EspressoSystems/cap/blob/main/cap-specification.pdf.

Fis22. B. Fisch. Privacy-protecting regulatory solutions using zero-knowledge proofs. 2022. https://www.espressosys.com/blog/configurable-privacy-case-study-partitioned-privacy-pools.

GGM16. C. Garman, M. Green, and I. Miers. Accountable privacy for decentralized anonymous payments. In *FC 2016*, *LNCS* 9603, pages 81–98. Springer, Heidelberg, February 2016.

GHR+22. L. Grassi, Y. Hao, C. Rechberger, M. Schofnegger, R. Walch, and Q. Wang. A new feistel approach meets fluid-SPN: Griffin for zero-knowledge applications. Cryptology ePrint Archive, Report 2022/403, 2022. https://eprint.iacr.org/2022/403.

GKK+22. C. Ganesh, H. Khoshakhlagh, M. Kohlweiss, A. Nitulescu, and M. Zając. What makes fiat–shamir zksnarks (updatable srs) simulation extractable? In *International Conference on Security and Cryptography for Networks*, pages 735–760. Springer, 2022.

GKR+21. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

GKS23. L. Grassi, D. Khovratovich, and M. Schofnegger. Poseidon2: A faster version of the poseidon hash function. Cryptology ePrint Archive, Report 2023/323, 2023. https://eprint.iacr.org/2023/323.

Gro16. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 305–326. Springer, Heidelberg, May 2016.

GWC19. A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

HBHW22. D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. 2022. https://zips.z.cash/protocol/protocol.pdf.

Hop20. D. Hopwood. The pasta curves for halo 2 and beyond, 2020. https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/.

Int23. Intrinsic Technologies. Introducing axiom. 2023. https://www.axiom.xyz/blog/intro.

JKTS07. P. C. Johnson, A. Kapadia, P. P. Tsang, and S. W. Smith. Nymble: Anonymous IP-address blocking. In *PET 2007*, *LNCS* 4776, pages 113–133. Springer, Heidelberg, June 2007.

KKS22. A. Kiayias, M. Kohlweiss, and A. Sarencheh. PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. In *ACM CCS 2022*, pages 1739–1752. ACM Press, November 2022.

KST22. A. Kothapalli, S. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.

KYMM18. G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. In *USENIX Security 2018*, pages 463–477. USENIX Association, August 2018.

LLK13. B. Laurie, A. Langley, and E. Kasper. Rfc 6962: Certificate transparency, 2013. https://www.rfc-editor.org/rfc/rfc6962.

MPJ+13. S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013.

MSH+18. M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, July 2018.

NT16. A. Naveh and E. Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy*, pages 255–271. IEEE Computer Society Press, May 2016.

RPX+22. D. Rathee, G. V. Policharla, T. Xie, R. Cottone, and D. Song. ZEBRA: Anonymous credentials with practical on-chain verification and applications to KYC in DeFi. Cryptology ePrint Archive, Report 2022/1286, 2022. https://eprint.iacr.org/2022/1286.

Sol22. A. Soleimani. Permissioned privacy pools. 2022. https://ethresear.ch/t/permissioned-privacy-pools/13572.

ST99. T. Sander and A. Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In *FC'99*, *LNCS* 1648, pages 46–61. Springer, Heidelberg, February 1999.

TBA+22. A. Tomescu, A. Bhat, B. Applebaum, I. Abraham, G. Gueta, B. Pinkas, and A. Yanai. UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452, 2022. https://eprint.iacr.org/2022/452.

TFZ+22. N. Tyagi, B. Fisch, A. Zitek, J. Bonneau, and S. Tessaro. VeRSA: Verifiable registries with efficient client audits from RSA authenticated dictionaries. In *ACM CCS 2022*, pages 2793–2807. ACM Press, November 2022.
TKCS09. P. P. Tsang, A. Kapadia, C. Cornelius, and S. W. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Transactions on Dependable and Secure Computing*, 8(2):256–269, 2009.
TKPS21. I. Tzialla, A. Kothapalli, B. Parno, and S. Setty. Transparency dictionaries with succinct proofs of correct operation. Cryptology ePrint Archive, Report 2021/1263, 2021. https://eprint.iacr.org/2021/1263.
Uni22. United States Department of the Treasury. U.s. treasury sanctions notorious virtual currency mixer tornado cash. 2022. https://home.treasury.gov/news/press-releases/jy0916.
Val08. P. Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC 2008*, *LNCS* 4948, pages 1–18. Springer, Heidelberg, March 2008.
WKDC22. K. Wüst, K. Kostiainen, N. Delius, and S. Capkun. Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In *ACM CCS 2022*, pages 2947–2960. ACM Press, November 2022.
WMW+22. M. Wu, W. McTighe, K. Wang, I. A. Seres, N. Bax, M. Puebla, M. Mendez, F. Carrone, T. De Mattey, H. O. Demaestri, et al. Tutela: An open-source tool for assessing user-privacy on ethereum and tornado cash. *arXiv preprint arXiv:2201.06811*, 2022. https://arxiv.org/abs/2201.06811.
YAY+19. Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau. New empirical traceability analysis of CryptoNote-style blockchains. In *FC 2019*, *LNCS* 11598, pages 133–149. Springer, Heidelberg, February 2019.

A Cryptographic Primitives

We provide definitions of additional cryptographic primitives that are commonly used in the implementation of privacy pools. We describe an example construction of a privacy pool in Appendix D.

Public-key encryption schemes A public-key encryption scheme is of a triple of efficient algorithms $E = (\mathsf{Gen}; \mathsf{Enc}; \mathsf{Dec})$ where:

- $(\mathsf{pk}; \mathsf{sk})$ $\mathsf{Gen}(1^\lambda)$ is a PPT key generation algorithm that outputs a key pair consisting of a public key pk and a private key sk. The public key defines a message space M_{pk}.
- ct $\mathsf{Enc}(\mathsf{pk}; \mathsf{msg})$ is a PPT encryption algorithm that outputs a ciphertext ct when given a public key pk and a message $\mathsf{msg} \in M_{\mathsf{pk}}$.
- msg $\mathsf{Dec}(\mathsf{sk}; \mathsf{ct})$ is a polynomial-time decryption algorithm that given a ciphertext ct and the secret key sk whose corresponding public key pk was used to generate the ciphertext, outputs the encrypted message in plaintext. The output msg is a special reject value if decryption failed.

We require that $\Pr[\mathsf{Dec}(\mathsf{sk}; \mathsf{Enc}(\mathsf{pk}; \mathsf{msg})) = \mathsf{msg}] = 1$ for all key pairs and messages. We require that the scheme has the IND-CPA and IK-CPA properties. The ElGamal scheme has these properties.

zk-SNARKs A preprocessing zk-SNARK (zero-knowledge succinct non-interactive argument of knowledge) with universal SRS (structured reference string) consists of a tuple of efficient algorithms $\mathsf{ARG} = (G; I; P; V)$ where:

- srs $G(1^\lambda; N)$ is a PPT generation algorithm that samples an SRS that supports indices of size up to N. This is the universal setup, which is carried out once and used across all future circuits.
- $(\mathsf{ek}; \mathsf{vk})$ $I_{\mathsf{srs}}(i)$ is a polynomial-time indexing algorithm that outputs the proving key ek and verification key vk for a circuit with description i. This algorithm has oracle access to the SRS.
- $P(\mathsf{ek}; x; w)$ is a PPT proving algorithm that outputs the proof given the instance x and the witness w.
- b $V(\mathsf{vk}; x; \ )$ is a polynomial-time verification algorithm that outputs an accepting bit $b \in \{0; 1\}$ given the verification key vk, the instance x, and a proof . The bit $b = 1$ denotes acceptance of the proof for the instance, while $b = 0$ denotes rejection of the proof.

We require the standard security properties of completeness, knowledge soundness, zero knowledge, and succinctness. We additionally require simulation extractability. Informally, this property ensures that an adversary cannot generate a proof unless the adversary has seen the witness. This is needed to guarantee non-malleability of proofs. We refer to [GKK+22] for a formal definition of this property.

B Building Block Algorithms

We provide a specific implementation of the building block algorithms of the privacy pool below. This is a basic implementation of the algorithms that illustrates the key ideas. Some minor changes, such as including a function of the secret key in the nullifier computation, may be needed for a production-ready system. These algorithms may be generalized to support assets of multiple types.

- Coin Commitment Creation. A coin commitment is computed using a hash function that is applied to the input elements and randomness. For a coin with value amt owned by public key pk, the coin commitment cm is computed by $\mathsf{Com}(\mathsf{amt}; \mathsf{pk}; r) := H_q(\mathsf{amt} \| \mathsf{pk} \| r)$. We may also write $\mathsf{cm} := \mathsf{Com}(\mathsf{open})$ for the opening $\mathsf{open} = (\mathsf{amt}; \mathsf{pk}; r)$.
- Nullifier Creation. A nullifier is computed using a hash function that is applied to the randomness in the opening of the coin commitment. The nullifier for a coin commitment with opening $\mathsf{open} = (\mathsf{amt}; \mathsf{pk}; r)$ and auxiliary input $\mathsf{aux} = ?$ is computed by $\mathsf{Nullify}(\mathsf{open}; \mathsf{aux}) := H_q(r)$.
- Memo Encryption. ct $\mathsf{Enc}_{\mathsf{pk}}(m; r)$ denotes an ElGamal encryption algorithm that computes ciphertext ct from public key pk, message m, and randomness r.

C Privacy Pool Proofs

For completeness, we provide a recap of the zk-SNARK statements for transactions within privacy pools. Unlike the recursive membership proofs, the proofs of these statements are verified by the smart contract. Hence it is more important to optimize the verification costs of these proofs.

Transfer Proofs This is the zk-SNARK statement for the validity of an anonymous transfer in the privacy pool. The proof shows that the value amount is preserved in the transfer, the sender knows the secret keys for each of the input coin commitments, the input coin commitments are accumulated, the owner memo of each output is correctly encrypted, the nullifiers for the input coin commitments are correctly computed, and the output coin commitments are correctly computed.

Let n be the number of transaction inputs and m be the number of transaction outputs. We allow placeholder coins with a placeholder public key and a zero amount to support transaction padding.

- Statement:
  For $i \in [n]$:
  - $\mathsf{pk}_i^{\mathsf{in}} = H_q(\mathsf{sk}_i^{\mathsf{in}} \| \mathsf{addr}_i^{\mathsf{in}})$
  - $\mathsf{nf}_i = H_q(r_i^{\mathsf{in}})$
  - $\mathsf{Acc.VfyMem}(\mathsf{rt_c}; \mathsf{w}_i^{\mathsf{in}}; \mathsf{Com}(\mathsf{amt}_i^{\mathsf{in}}; \mathsf{pk}_i^{\mathsf{in}}; r_i^{\mathsf{in}}))$

  $\sum_{i \in [n]} \mathsf{amt}_i^{\mathsf{in}} = \sum_{i \in [m]} \mathsf{amt}_i^{\mathsf{out}}$

  For $i \in [m]$:
  - $\mathsf{cm}_i = \mathsf{Com}(\mathsf{amt}_i^{\mathsf{out}}; \mathsf{pk}_i^{\mathsf{out}}; r_i^{\mathsf{out}})$
  - $\mathsf{memo}_i = \mathsf{Enc}_{\mathsf{pk}_i^{\emptyset}}((\mathsf{amt}_i^{\mathsf{out}}; \mathsf{pk}_i^{\mathsf{out}}; r_i^{\mathsf{out}}); \ \ _i)$
- Public inputs:
  $(\mathsf{nf}_i)_{i \in [n]}$: List of nullifiers for the transaction inputs.
  $(\mathsf{cm}_i)_{i \in [m]}$: List of output coin commitments.
  $(\mathsf{memo}_i)_{i \in [m]}$: List of output owner memos.
  $\mathsf{rt_c}$: Digest of coin commitment accumulator.
- Private inputs:
  $(\mathsf{amt}_i^{\mathsf{in}}; \mathsf{pk}_i^{\mathsf{in}}; r_i^{\mathsf{in}})_{i \in [n]}$: List of openings of input coin commitments.
  $(\mathsf{addr}_i^{\mathsf{in}})_{i \in [n]}$: List of owner addresses for transaction inputs.
  $(\mathsf{sk}_i^{\mathsf{in}})_{i \in [n]}$: List of user secret keys for transaction inputs.
  $(\mathsf{w}_i^{\mathsf{in}})_{i \in [n]}$: List of membership witnesses for transaction inputs.
  $(\mathsf{amt}_i^{\mathsf{out}}; \mathsf{pk}_i^{\mathsf{out}}; r_i^{\mathsf{out}})_{i \in [m]}$: List of openings of output coin commitments.
  $(\mathsf{pk}_i^{\emptyset})_{i \in [m]}$: Public keys for encryption of transaction outputs.
  $(\ \ _i)_{i \in [m]}$: Encryption randomness values for transaction outputs.

Withdrawal Proofs This is the zk-SNARK statement for the validity of a withdrawal from the privacy pool. The proof shows that the value amount is preserved in the withdrawal, the sender knows the secret keys for each of the input coin commitments, the input coin commitments are accumulated, and the nullifiers are correctly computed. Let n be the number of transaction inputs for the withdrawal. As in the case of the transfer statement, we allow placeholder coins to support transaction padding. We include the recipient address in the public inputs to ensure that a front-running adversary cannot change the address in the withdrawal transaction. This defense relies on the non-malleability of the proof, which is guaranteed by the simulation extractability property of the zk-SNARK scheme.

- Statement:
  For $i \in [n]$:
  - $\mathsf{pk}_i^{\mathsf{in}} = H_q(\mathsf{sk}_i^{\mathsf{in}} \| \mathsf{addr}_i^{\mathsf{in}})$
  - $\mathsf{nf}_i = H_q(r_i^{\mathsf{in}})$
  - $\mathsf{Acc.VfyMem}(\mathsf{rt_c}, \mathsf{w}_i^{\mathsf{in}}, \mathsf{Com}(\mathsf{amt}_i^{\mathsf{in}}, \mathsf{pk}_i^{\mathsf{in}}, r_i^{\mathsf{in}}))$
  - $\sum_{i \in [n]} \mathsf{amt}_i^{\mathsf{in}} = \mathsf{amt}_{\mathsf{out}}$
- Public inputs:
  $(\mathsf{nf}_i)_{i \in [n]}$: List of nullifiers for the transaction inputs.
  $\mathsf{addr}_{\mathsf{out}}$: Output recipient address.
  $\mathsf{amt}_{\mathsf{out}}$: Output value amount.
  $\mathsf{rt_c}$: Digest of coin commitment accumulator.
- Private inputs:
  $(\mathsf{amt}_i^{\mathsf{in}}, \mathsf{pk}_i^{\mathsf{in}}, r_i^{\mathsf{in}})_{i \in [n]}$: List of openings of input coin commitments.
  $(\mathsf{addr}_i^{\mathsf{in}})_{i \in [n]}$: List of owner addresses for transaction inputs.
  $(\mathsf{sk}_i^{\mathsf{in}})_{i \in [n]}$: List of user secret keys for transaction inputs.
  $(\mathsf{w}_i^{\mathsf{in}})_{i \in [n]}$: List of membership witnesses for transaction inputs.

D Privacy Pool Algorithms

We present a specification of the privacy pool algorithms in Figures 5 and 6. Privacy pools typically use a proof system with trusted setup, such as Plonk [GWC19] or Groth16 [Gro16]. The setup may be universal or circuit-specific, and the setup commonly involves participation from multiple parties.

GenerateKeyPair(pp_{pool}, addr)

Generate encryption keys (pk^0; sk^0) E:Gen(1)
Generate secret key sk $\$ \{0,1\}$
Generate public key $pk = H_q(sk \| addr)$
return (sk; pk; sk^0; pk^0)

CreateDepositTx(pp_{pool}, amt, pk)

Sample $r \$ \{0,1\}$
Compute coin commitment $cm = H_q(amt \| pk \| k \| r)$
Set open = (amt; pk; r)
Set tx_{dep} = (cm; amt)
return tx_{dep}

CreateTransferTx$_{n,m}$(:::)

Inputs: pp_{pool}; $\mathbf{sk_{in}}$; $\mathbf{open_{in}}$; $\mathbf{addr_{in}}$; $\mathbf{open_{out}}$; $\mathbf{pk'}$
Set **nf** = [] and $\mathbf{w_{in}}$ = []
for i 2 [n] **do**
  Set sk = $\mathbf{sk_{in}}[i]$
  Parse (amt; pk; r) = $\mathbf{open_{in}}[i]$
  Compute coin commitment $cm = H_q(amt \| pk \| k \| r)$
  Compute nullifier $nf = H_q(r)$
  Compute w = Acc:PrvMem($_c$; cm)
  Add nf to **nf**; w to $\mathbf{w_{in}}$
endfor
Set **cm** = []; **memo** = []; and = []
for i 2 [m] **do**
  Parse (amt; pk; r) = $\mathbf{open_{out}}[i]$
  Compute coin commitment $cm = H_q(amt \| pk \| k \| r)$
  Sample $\$ \{0,1\}$
  Set $pk^0 = \mathbf{pk'}[i]$
  Compute owner memo memo = $Enc_{pk^0}(amt \| pk \| k \| r;$)
  Add cm to **cm**; memo to **memo**; to
endfor
Set instance x = (**nf**; **cm**; **memo**; rt_c)
Set witness w = ($\mathbf{open_{in}}$; $\mathbf{addr_{in}}$; $\mathbf{sk_{in}}$; $\mathbf{w_{in}}$; $\mathbf{open_{out}}$; $\mathbf{pk'}$;)
Generate transfer proof $_t$ = ARG:P(ek$_t$; x; w)
Set tx_{tfr} = (**nf**; **cm**; **memo**; rt_c; $_t$)
return tx_{tfr}

CreateWithdrawalTx$_n$(:::)

Inputs: pp_{pool}; $\mathbf{sk_{in}}$; $\mathbf{open_{in}}$; $\mathbf{addr_{in}}$; $open_{out}$; $addr_{out}$
Set **nf** = [] and $\mathbf{w_{in}}$ = []
for i 2 [n] **do**
  Set sk = $\mathbf{sk_{in}}[i]$
  Parse (amt; pk; r) = $\mathbf{open_{in}}[i]$
  Compute coin commitment $cm = H_q(amt \| pk \| k \| r)$
  Compute nullifier $nf = H_q(r)$
  Compute w = Acc:PrvMem($_c$; cm)
  Add nf to **nf**; w to $\mathbf{w_{in}}$
endfor
Set **cm** = []
Parse (amt; pk; r) = $open_{out}$
Compute coin commitment $cm = H_q(amt \| pk \| k \| r)$
Add cm to **cm**
Set instance x = (**nf**; amt; $addr_{out}$; rt_c)
Set witness w = ($\mathbf{open_{in}}$; $\mathbf{addr_{in}}$; $\mathbf{sk_{in}}$; $\mathbf{w_{in}}$)
Generate withdrawal proof $_w$ = ARG:P(ek$_w$; x; w)
Set tx_{wdr} = (amt; $addr_{out}$; **nf**; cm; rt_c; $_w$)
return tx_{wdr}

Fig. 5: Privacy Pool Client Algorithms.

PrivacyPoolSetup(1^λ)

Sample hash function $H_q : \{0,1\}^* \to \mathbb{F}_q$
Specify the maximum deposit amount $\mathsf{amt}_{\max} \in \mathbb{N}$ and the Merkle tree depth $d \in \mathbb{N}$
Run universal setup for zk-SNARK: $\mathsf{srs} = \mathsf{ARG}.G(1^\lambda)$
Construct circuit descriptions t for transfer proof and w for withdrawal proof
Generate keys for circuit: $(\mathsf{ek}_t, \mathsf{vk}_t) \leftarrow \mathsf{ARG}.I_{\mathsf{srs}}(t)$ and $(\mathsf{ek}_w, \mathsf{vk}_w) \leftarrow \mathsf{ARG}.I_{\mathsf{srs}}(w)$
Create $(\mathsf{rt}_c, \tau_c) = \mathsf{Acc}.\mathsf{Init}(1^\lambda)$
Set $\mathsf{NulliferList} = \{\}$, $\mathsf{DigestList} = \{\}$, and $\mathsf{AccountList} = \{\}$
Set $\mathsf{pp}_{\mathsf{pool}} = \{H_q, \mathsf{amt}_{\max}, d, \mathsf{rt}_c, \tau_c, \mathsf{ek}_t, \mathsf{vk}_t, \mathsf{ek}_w, \mathsf{vk}_w, \mathsf{NulliferList}, \mathsf{DigestList}, \mathsf{AccountList}\}$
return $\mathsf{pp}_{\mathsf{pool}}$

ProcessRegistrationTx($\mathsf{pp}_{\mathsf{pool}}$, $\mathsf{tx}_{\mathsf{reg}}$)

Parse $(\mathsf{pk}, \mathsf{pk}^0) = \mathsf{tx}_{\mathsf{reg}}$
Set $\mathsf{AccountList}[\mathsf{tx}_{\mathsf{reg}}.\mathsf{sender}] = (\mathsf{pk}, \mathsf{pk}^0)$
return 1

ProcessDepositTx($\mathsf{pp}_{\mathsf{pool}}$, $\mathsf{tx}_{\mathsf{dep}}$)

Parse $(\mathsf{cm}, \mathsf{amt}) = \mathsf{tx}_{\mathsf{dep}}$
Check amt is less than or equal to $\mathsf{amt}_{\max}$
Update $\mathsf{rt}_c = \mathsf{Acc}.\mathsf{Update}(\mathsf{rt}_c, \tau_c, \mathsf{cm})$
Add rt_c to $\mathsf{DigestList}$
Transfer amt units from $\mathsf{tx}_{\mathsf{dep}}.\mathsf{sender}$ to contract address
return 1

ProcessTransferTx$_{n,m}$($\mathsf{pp}_{\mathsf{pool}}$, $\mathsf{tx}_{\mathsf{tfr}}$)

Parse $(\mathbf{nf}, \mathbf{cm}, \mathbf{memo}, \mathsf{rt}_c^0, \pi_t) = \mathsf{tx}_{\mathsf{tfr}}$
Require $\mathsf{rt}_c^0 \in \mathsf{DigestList}$
for $i \in [n]$ **do**
  Set $\mathsf{nf} = \mathbf{nf}[i]$
  Require $\mathsf{nf} \notin \mathsf{NulliferList}$
  Add nf to $\mathsf{NulliferList}$
endfor
for $i \in [m]$ **do**
  Set $\mathsf{cm} = \mathbf{cm}[i]$
  Update $\mathsf{rt}_c = \mathsf{Acc}.\mathsf{Update}(\mathsf{rt}_c, \tau_c, \mathsf{cm})$
  for $j \in [n]$ **do**
    Set $\mathsf{nf} = \mathbf{nf}[j]$
  endfor
endfor
Add rt_c to $\mathsf{DigestList}$
Require $\mathsf{ARG}.V(\mathsf{vk}_t, [\mathbf{nf}, \mathbf{cm}, \mathbf{memo}, \mathsf{rt}_c^0], \pi_t)$
return 1

ProcessWithdrawalTx$_n$($\mathsf{pp}_{\mathsf{pool}}$, $\mathsf{tx}_{\mathsf{wdr}}$)

Parse $(\mathsf{amt}, \mathsf{addr}, \mathbf{nf}, \mathsf{cm}, \mathsf{rt}_c^0, \pi_w) = \mathsf{tx}_{\mathsf{wdr}}$
Require $\mathsf{rt}_c^0 \in \mathsf{DigestList}$
for $i \in [n]$ **do**
  Set $\mathsf{nf} = \mathbf{nf}[i]$
  Require $\mathsf{nf} \notin \mathsf{NulliferList}$
  Add nf to $\mathsf{NulliferList}$
endfor
Require $\mathsf{ARG}.V(\mathsf{vk}_w, [\mathbf{nf}, \mathsf{amt}, \mathsf{addr}, \mathsf{rt}_c^0], \pi_w)$
Transfer amt units from contract address to recipient addr
return 1

Fig. 6: Privacy Pool Contract Algorithms.