ZEBRA: Anonymous Credentials with Practical On-chain Verification and Applications to KYC in DeFi

Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song
University of California, Berkeley
{*deevashwer, guruvamsip, tianc.x, rcottone, dawnsong*}*@berkeley.edu*

***Abstract*—ZEBRA is an Anonymous Credential (AC) scheme, supporting auditability and revocation, that provides practical on-chain verification for the first time. It realizes efficient access control on permissionless blockchains while achieving both privacy and accountability. In all prior solutions, users either pay exorbitant fees or lose privacy since authorities granting access can map users to their wallets. Hence, ZEBRA is the first to enable DeFi platforms to remain compliant with imminent regulations without compromising user privacy.**

We evaluate ZEBRA and show that it reduces the gas cost incurred on the Ethereum Virtual Machine (EVM) by $11.8\times$ when compared to Coconut [NDSS 2019], the state-of-the-art AC scheme for blockchains. This translates to a reduction in transaction fees from 94 USD to 8 USD on Ethereum in August 2022. However, 8 USD is still high for most applications, and ZEBRA *further* drives down credential verification costs through batched verification. For a batch of 512 layer-1 and layer-2 wallets, the gas cost is reduced by $35\times$ and $641\times$ on EVM, and the transaction fee is reduced to just 0.23 USD and 0.0126 USD on Ethereum, respectively. For perspective, these costs are comparable to the minimum transaction costs on Ethereum.

1. Introduction

Over the last decade, blockchains have gained popularity over their centralized counterparts, owing to their strong integrity and availability, and transparent and permissionless nature. Although a majority of blockchain applications benefit from permissionless access, there are applications where access control is useful and sometimes even legally mandated. For instance, recently proposed guidelines from the Financial Action Task Force (FATF) require decentralized finance (DeFi) applications to restrict their service to KYC-verified users [59]. In prior access control solutions [120], [117], [100], [118], [30], [7], certificate authorities (CAs) issue an on-chain credential to a user nominated wallet after ensuring the user satisfies the application's access policy. These credentials can be thought of as signatures from the CA attesting to a user's access privileges. This approach, however, comes at a huge privacy cost for users: the CA can link users to their wallets, potentially even across chains, and this sensitive information could even be leaked if the CA suffers from a data breach.

The natural solution to resolve this privacy tension is to use an anonymous credential (AC), which is a cryptographic primitive that enables a user to prove that they hold a valid credential in an *unlinkable* way, i.e., even if the verifier and the CA collude, they cannot identify the user or link *multiple* showings of the same credential. Hence, the CA can issue an AC instead, and the user can prove on-chain that they hold a valid credential without revealing any additional information. To ensure accountability for misbehaving users, the AC scheme must be *auditable* to enable authorized auditors to identify the owner of a maliciously behaving wallet. Additionally, *revocation* support is also necessary as credentials are often lost or stolen, and credentials of malicious users need to be revoked.

Insufficiency of Existing Anonymous Credentials. While there is a long line of work on AC schemes [42], [41], [109], [104], [77], [35], [36], [63], [127], [47], with some also supporting revocation and auditability [42], [43], [39], [40], [26], none of them are suitable for implementing access control on permissionless blockchains. In particular, existing AC schemes are very expensive to verify on-chain (§ 8.1.3), and this high cost is due to the following reasons: (i) their verifier relies on operations that are not efficiently supported on Ethereum Virtual Machine (EVM) [124], the de-facto runtime environment for smart contracts [89], and (ii) their verifier performance degrades linearly with verification predicate complexity. Consequently, Coconut [109], the state-of-the-art AC scheme for blockchains requires 4.2M gas for on-chain verification on EVM (§ 8.1.3), which is equivalent to 94 USD on Ethereum in August 2022 [6]. Moreover, this is not a one-time cost paid by users when they join the system. Credential verification has to be performed for every transaction that interacts with an access-controlled application since it is not and in fact should not be possible to know if the previous verification of a wallet was associated with a credential that has since been revoked.

Anonymous Credentials for Blockchains. In this work, we propose an AC scheme ZEBRA[1] that provides *practical* on-chain verification for the first time and supports auditability and revocation. With the primary goal of minimizing

1. ZEBRA stands for zero-knowledge (or anonymous), batched, revocable, and auditable credentials.

verifier cost[2] (and thereby transaction fees), we build our scheme on top of zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) because they address both sources of inefficiency in existing AC schemes: SNARKs can be verified efficiently on EVM and their verifier is sublinear in the verification predicate complexity. However, the use of SNARKs by itself does not lead to a practical AC scheme:

- SNARKs typically have high proof generation costs which are often unreasonable for weak end-user devices.
- The SNARK with cheapest on-chain verification [71] still costs 8 USD on Ethereum, which is too high to pay for every transaction.

ZEBRA addresses both of these issues by designing a small credential verification circuit (§ 2.3.4) and leveraging existing access control solutions to avoid credential verification with every transaction (§ 2.3.1). Moreover, ZEBRA *further* reduces the cost of credential verification through batched verification as 8 USD is quite expensive for most applications, especially on layer-2 (L2). Batched verification relies on an *untrusted aggregator* to verify many credential verification proofs and *recursively* prove their validity to the contract with a single SNARK proof, the cost of which is amortized across multiple users. While this idea may seem straightforward, it has two efficiency challenges:

- Due to restrictions imposed by EVM for efficient on-chain verification, prior works on proof recursion either have high aggregator latency or high user overhead (§ 8.2.3).
- Each credential proof is accompanied with transaction data that needs to be processed on-chain for each user in the batch, and this imposes a high lower bound on the gas cost per user, especially for L2 wallets (§ 8.2.1).

To this end, we propose a new EVM-compatible solution with practical aggregator and user overhead (§ 2.3.2) that reduces and even *removes* per-user costs from batched verification of L1 and L2 wallets, respectively (§ 2.3.3).

Our performance evaluation demonstrates the practicality of ZEBRA: the single (core) verification protocol requires 355K gas on EVM or 8 USD on Ethereum (§ 8.1.2), and the user overhead is under 6s on a smartphone with single thread (§ 8.1.1). The aggregator can batch verification of 512 credentials within 4 minutes (§ 8.2.2) on a 64-core machine, while achieving 10.2K and 561 gas per L1 and L2 wallet (§ 8.2.1), respectively. For perspective, the average gas usage per transaction on Ethereum is around 87K [4] in August 2022, and the minimum transaction gas cost is 21K and 500 for L1 and L2 [130] wallets, respectively.

Practical and Privacy-Preserving KYC in DeFi. Recently, some DeFi platforms have moved towards implementing KYC checks [93], [116], [9], [52] in response to FATF's guidelines [59] and to attract institutional investors. While the steps taken by these platforms are important in regulating DeFi and establishing it as a mainstream financial technology, they are similar to prior access control approaches [120], [117], [100], [118], [30], [7] and come with a huge privacy loss to users. To this end, ZEBRA provides the first practical solution that protects user's privacy and enables KYC in DeFi at the same time[3]. This solution leverages the minimal gas costs of our batched verification since KYC in DeFi requires frequent credential verifications to support over 600K users and 1.8M transactions every day [5]. To enable credentials from multiple CAs, we also add *multi-CA* support to ZEBRA using which credentials from different CAs can be verified with a unified function without revealing which CA issued the credential. As a result, users holding credentials from multiple CAs can be batched together and the anonymity set of users is also increased. Finally, since ZEBRA relies on SNARKs, it is easy to extend it to richer attributes and predicates to support expressive selective disclosure without significantly changing the on-chain verification cost (see § 5.4 for discussion and more extensions). Hence, it is ideal for implementing more complex checks when further regulation is mandated, and is of independent interest for implementing on-chain privacy-preserving access control in general. In particular, some other applications of ZEBRA are decentralized voting, proof-of-personhood, private NFT drops, authentication and authorization.

In summary, we make the following contributions:

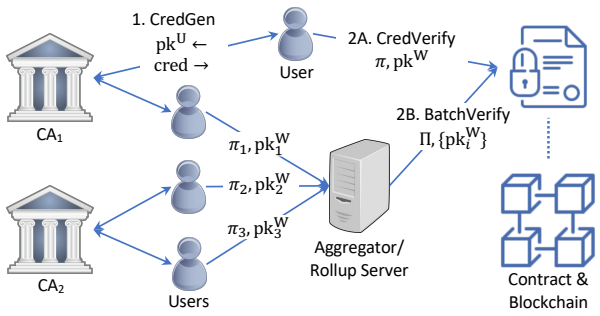
- We propose an AC scheme ZEBRA that provides practical on-chain verification for the first time and offers *auditability*, *revocation*, and *multi-CA* support with practical user overheads (§ 8.1). ZEBRA also leverages and extends the existing access control solutions to avoid credential verification overhead per transaction.
- We propose a practical and EVM-compatible solution for batching ZEBRA credentials. Compared to our solution, simply using prior proof recursion works leads to 9× higher gas cost (§ 8.2.1), and either 6.3× higher aggregator latency or 11× higher user overhead (§ 8.2.3).
- We implement (§ 7) and benchmark all components of ZEBRA and demonstrate their practicality on EVM. Our core verification protocol improves gas cost of existing AC schemes by 11.8× (§ 8.1.3), and with batched verification of just 512 credentials, ZEBRA achieves *further* gas improvements of 35× and 641× for L1 and L2 wallets (§ 8.2.1), respectively.
- We address the problem of privacy-preserving KYC in DeFi for permissionless blockchains and enable the first practical solution for it using ZEBRA. For a discussion on related work, see § 9.

2. Overview

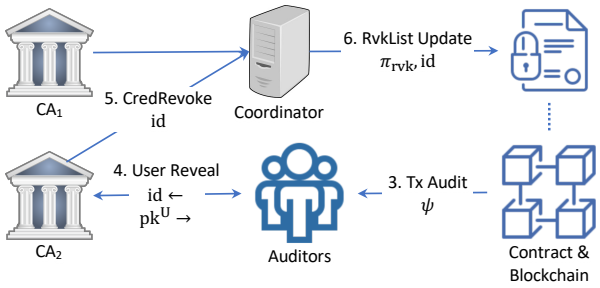
In this section, we first discuss the system model and high-level goals of our system in § 2.1 and § 2.2, respectively, and then provide a summary of our techniques in § 2.3 that realize these goals.

2. By verifier cost, we mean both the verifier runtime and the proof size.

3. There are other privacy-preserving solutions that provide practicality at the cost of introducing additional trust assumptions, and we discuss them in related work § 9.



(a) Credential Generation and Verification



(b) Transaction Audit and Credential Revocation

Figure 1: System Architecture.

2.1. System Model

Figure 1 summarizes the system architecture and the various entities and protocols in our system. Now, we describe the role of each entity while referring to the corresponding protocols:

- *Credential Verification Smart Contract*: This smart contract is responsible for verifying a credential (protocols 2A and 2B) and issuing an on-chain access token to the nominated wallet. These tokens can then be efficiently checked by the smart contract of the access-controlled application before granting access to the wallet. It also stores the list of approved CAs and their revocation lists.
- *Users*: Users can obtain a credential cred associated with their public-key pk^{U} from any CA approved by the access-controlled application if the user satisfies the issuance policy (protocol 1). They can then prove on-chain that they hold a valid credential in a privacy-preserving way by providing a proof π to get an access token for their wallet pk^{W} (protocol 2A). We assume users are malicious and can act arbitrarily to get verified without holding a valid credential.
- *Certificate Authorities (CAs)*: CAs are organizations trusted to issue credentials to users if they satisfy the application's credential issuance policy (protocol 1). A CA can revoke a credential it issued by adding its unique identifier id to the revocation list (protocol 5). They also help auditors in deanonymizing malicious users of their issued credentials (protocol 4). We assume the CAs are trusted for integrity since they are reputed organizations authorized by the application, but even if they behave arbitrarily, we prove that they cannot deanonymize users. Furthermore, in § 5.4, we discuss how ZEBRA can be extended to support other issuance models.
- *Aggregator/Rollup Server*: The aggregator/rollup server is an untrusted party that batches credential verification transactions for L1 and L2 wallets, respectively, to reduce on-chain verification costs. It does so by sending a short proof Π to the contract which ensures that the server knows a valid credential verification proof π_i for each $\mathsf{pk}_i^{\mathsf{W}}$ in the batch (protocol 2B). As in zk-rollups [34], the only thing a malicious server can do is censor user's transactions. In case of a censoring aggregator, the users can simply move to another aggregator. For a censoring rollup server, we can use standard techniques from zk-rollups [130], [85].
- *Auditors*: Auditors can audit a credential verification transaction via its associated audit token ψ if a majority of them agree to it (protocol 3). This could happen, for instance, in case law enforcement requests it. The audit reveals a unique credential identifier id that embeds the CA that issued it. The auditors then share id with the issuing CA, and the CA deanonymizes the user by sending its user's public key pk^{U} or any other identity information to the auditors (protocol 4). Our system can handle a minority of malicious auditors. We also note that it is relatively easier to find a large committee of auditors to sufficiently distribute trust since they are called upon infrequently.
- *Contract Coordinator*: The coordinator is responsible for managing the credential verification contract, and its responsibilities are updating the list of approved CAs according to the access policy, and batched updates to revocation lists of all CAs[4]. We don't trust the coordinator to post correct updates to the revocation list and require that it provide a proof π_{rvk} ensuring that it knows a valid signature from the CA associated with the credential identifier id it is adding to the revocation list (protocol 6).

2.2. System Goals

Privacy Goal

- The credential verification transaction from a wallet does not reveal any information that can be used to identify the wallet's user among the set of all verified users (across all approved CAs), unless a majority of auditors agree to audit the transaction.

Integrity Goals

- Only approved CAs can generate credentials that will be successfully verified by the credential verification contract, and the integrity of this verification is equivalent to the integrity of the underlying blockchain.
- Only users with non-revoked credentials can get access tokens for their wallets.
- Each verified wallet can be traced back to a credential and its issuing CA by the auditors, and from there to the wallet's user with the help of the CA.
- A credential can only be revoked by its issuing CA.

Performance Goals

4. The CAs can also post the update themselves.

We first look at our goals for on-chain verification costs:

- Access tokens are verified with minimal gas cost before granting access to a wallet. All access tokens expire when users are revoked, and credential verification is only performed to renew access tokens.
- The gas cost of single credential verification should be practical enough for applications where accesses or revocations are infrequent, both of which imply infrequent credential verifications.
- The gas cost of batched verification should be comparable to the cheapest on-chain transaction. Batched verification is suitable for L1 applications with a large volume of transactions and users, and for L2 applications. Such applications have many users and they require frequent credential verifications to reduce the latency in enforcing revocations.

Throughout this paper, we use EVM *gas cost* as our on-chain cost metric as EVM is the de-facto runtime environment for smart contracts [89], and stress that our discussion and ideas are applicable to any blockchain that supports smart contracts. Next, we look at off-chain overhead goals:

- The user can create a credential verification proof on a smartphone within a few seconds.
- The overheads on the coordinator and the CAs are reasonable with commodity hardware.
- The aggregator's latency is reasonable to verify a batch of wallets large enough to match the cheapest on-chain transaction cost. The monetary cost incurred by aggregator in performing this computation is negligible compared to the monetary cost of batched verification.

2.3. Technical Overview

2.3.1. Access Tokens with Universal Revocation

Although SNARKs significantly improve credential verification overhead, it still costs $4\times$ the average transaction cost on Ethereum in August 2022 [4], which is impractical to incur with every transaction. To circumvent this limitation, we leverage the existing access control solutions to issue an on-chain access token to the wallet which is verified. Subsequently, the application can do a cheaper access token verification before granting access. Revocations, however, pose a problem: each credential revocation requires all access tokens to be revoked in our setting because the mapping between credentials and wallets is not known. Since existing access control solutions only support selective revocation, it would be intractable to selectively revoke all tokens one-by-one. To this end, we extend the existing solutions with a simple mechanism that revokes all tokens at once with just 5K gas. Our idea is to simply add an epoch identifier to each token that denotes the epoch when the token was assigned. A token is only valid if its epoch identifier matches the current epoch. Every time a batch of credentials is revoked, the current epoch is incremented, invalidating all the tokens. Consequently, users perform credential verification that costs 355K gas once per epoch, and for all subsequent transactions within the epoch, the application can simply check the access token using just 4200 gas.

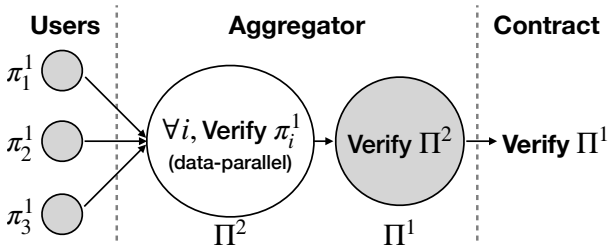


Figure 2: Batched verification workflow. Circles denote computations inside a SNARK and the circle labels denote the respective SNARK proofs. Grey circles represent pairing-based SNARK^1 over BN254 and white circle represents the (data-parallel) discrete-log-based $\mathsf{simdSNARK}^2$ over Grumpkin.

2.3.2. Batched Verification of SNARK Proofs

The idea behind batched verification is simple: the aggregator *recursively* proves to the contract that N credential verification proofs are valid using a single SNARK proof. However, to keep gas costs low for reasonable batch sizes[5], the proof sent to the contract should be a pairing-based SNARK proof over the BN254 curve [21], the only curve supported by EVM. Due to this restriction, none of the prior proof recursion approaches lead to a practical solution in our setting: they either have high aggregator latency or high user overhead (§ 8.2.3). To this end, we propose a *new approach* that relies on recursively verifying discrete-log-based SNARKs, which are not suitable for recursion in general as their verifier complexity is at least $\mathcal{O}_\lambda(\sqrt{n})$ for a circuit of size n[6]. Despite this limitation, we still manage to achieve a practical solution through careful use of the discrete-log-based SNARK and *two layers of recursion*. In particular, our solution, illustrated in Figure 2, uses the discrete-log-based SNARK over Grumpkin [64] to verify pairing-based SNARK proofs over BN254 from users, and this offers the following efficiency benefits: (i) SNARK^1 and $\mathsf{simdSNARK}^2$ can efficiently verify each other's proofs because BN254 forms a cycle with Grumpkin, (ii) $\mathsf{simdSNARK}^2$ is verifying a *data-parallel* or SIMD computation which greatly improves its concrete cost for both prover and verifer, and (iii) although $\mathsf{simdSNARK}^2$ verification circuit is large, it is sublinear in number of users. Additionally, user overheads and on-chain costs are practical due to the use of pairing-based SNARK^1. We instantiate $\mathsf{simdSNARK}^2$ with Spartan [106] in our implementation, and observe that the R1CS constraints for its verifier improve by $8\times$ due to data-parallelism (§ 7). We further improve its verifier constraints by $1.6\times$; importantly, this optimization was necessary to batch 512 users with our batching solution (§ 7). Overall, our batching solution either has at least $6.3\times$ smaller aggregator overhead or $11\times$ smaller user overhead (§ 8.2.3).

5. STARK-based verification requires $16\times$ higher gas cost [130]

6. In contrast, pairing-based and FRI-based SNARKs have $\mathcal{O}_\lambda(1)$ and $\mathcal{O}_\lambda(\log(n))$ verifiers, respectively.

2.3.3. Audit Token Caching

Proof batching mitigates the bottleneck in credential verification, but there are still costs that scale linearly with batch size and lower-bound the amortized gas cost per user. In particular, batched verification posts a 256-Byte audit token ψ_i for each user i in the batch, which is then input to the SNARK and a signature w.r.t. $\mathsf{pk}_i^{\mathsf{W}}$ is also verified on-chain in case of L1. Even ignoring the SNARK, this lower bounds the gas cost at around 20K per L1 wallet and around 5K per L2 wallet (§ 8.2.1). We circumvent these lower bounds following the observation that there is no need to post ψ_i with each credential verification of $\mathsf{pk}_i^{\mathsf{W}}$ as it can be made static and cached. We can do so because changing credentials or posting a new audit token for each verification does not provide any additional privacy. Thus, when a wallet is verified for the first time, we can simply cache ψ_i for $\mathsf{pk}_i^{\mathsf{W}}$ on-chain for L1 and within the rollup state for L2, provided the SNARK and signature verification are successful. Additionally to save storage and lookup costs, we use a smart-contract friendly hash function $\mathsf{CRH}^{\mathsf{sc}}$ like SHA256 to store a 32-Byte commitment $h_i = \mathsf{CRH}^{\mathsf{sc}}(\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$ instead of 256-Byte ψ_i. This is especially important for L1 because retrieving cached ψ_i alone would require 16.8K gas, as opposed to just 2.1K for h_i. Overall, the lower bound is reduced to 9K for L1 wallets, removed entirely for L2 wallets, and we demonstrate at least 9× improvement in batching L2 wallets (§ 8.2.1).

2.3.4. SNARK-friendly Verification Circuit

Although we obtain efficient verification due to succinctness of SNARKs, the prover overhead also has to be practical. To this end, we carefully design a SNARK-friendly circuit that satisfies the properties outlined in § 2.2 using primitives which are efficient to compute within SNARKs. The simplest instantiation of a credential is just a signature from CA on user's public key pk^{U}. To improve performance and usability of this baseline, we embed a unique 40-bit credential identifier id in our credentials that also stores the issuing CA's ID. Specifically, this offers the following benefits: (i) we can revoke id instead of 256-bit pk^{U} and this reduces the non-membership proof costs from 148K to just 23K constraints using merkle trees, improving credential verification circuit size by 3×, (ii) we can encrypt id in audit token ψ instead of pk^{U}, and this directly reveals the issuing CA to the auditors, and (iii) id ensures that CAs can only revoke credentials they have issued. Overall, our credential verification circuit has 62K constraints and takes under 6 seconds to prove on a smartphone (§ 8.1.1).

3. Preliminaries

We provide high-level description of preliminaries mainly focusing on notation in this section and defer detailed security definitions to Appendix A.

3.1. zk-SNARKs

A zk-SNARK is a tuple of three algorithms:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{crs}_{\mathcal{R}}$: On input security parameter λ and relation $\mathcal{R}$, outputs a common reference string $\mathsf{crs}_{\mathcal{R}}$.
- $\mathsf{Prove}(\mathsf{crs}_{\mathcal{R}}, x, w) \to \pi$: On input $\mathsf{crs}_{\mathcal{R}}$ and a statement-witness pair $(x, w) \in \mathcal{R}$, outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}_{\mathcal{R}}, x, \pi) \to \{0, 1\}$: On input $\mathsf{crs}_{\mathcal{R}}$, statement x and proof π, outputs a bit to indicate if the proof is valid.

We use the security definitions of [71] and write SNARK when we demand perfect completeness, perfect zero-knowledge, and computational knowledge soundness from the argument system.

We also use simdSNARK to denote a "data-parallel" SNARK which takes as input multiple statement-witness pairs and outputs a single succinct proof π. Importantly, we relax the security definition here and require all the above properties except for zero-knowledge. We note that every simdSNARK can be viewed as a SNARK by redefining the relation. However, we make the distinction for ease of presentation.

3.2. Digital Signature

We use signature schemes that are existentially unforgeable under chosen message attacks (EUF-CMA). They consists of three algorithms $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$: on input security parameter λ, outputs a secret key sk and a public verification key vk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: on input sk and message m, output signature σ.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to \{0, 1\}$: on input vk, m and σ, outputs 1 if σ is a valid signature for m w.r.t. vk.

3.3. Threshold Public-Key Encryption

We use threshold public key encryption (TPKE) satisfying the simulation based IND-CCA2 security notion defined in [45]. We also restrict the syntax of TPKE to consist of five algorithms:

- $\mathsf{Setup}(1^\lambda, n, t) \to \{\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\}$: on input threshold t for n parties, outputs the public key pk and verification key vk, along with secret keys for each party.
- $\mathsf{Enc}(\mathsf{pk}, m; \rho) \to \mathsf{ct}$: encrypts message m under public key pk using randomness ρ.
- $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_i) \to m_i$: computes a partial decryption of ct.
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, m_i) \to \{0, 1\}$: checks whether m_i was correctly computed using pk and vk.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \{m_i\}_{i \in S \subseteq [n] \text{ s.t. } |S| \geq t+1}) \to m$: recovers message m given $t + 1$ partial decryptions which verify successfully.

3.4. Merkle Trees

Sparse Merkle trees [56] are authenticated data structures MT on key-value pairs (k, v) supporting the following operations:

- $\mathsf{Add}(k, v)$: inserts a key-value pair (k, v) in MT. If the key already exists, the value is updated.
- Root: outputs the current merkle-root of MT.
- $\mathsf{MProve}(k) \to P$: outputs a membership proof for key k.
- $\mathsf{MVerify}(\mathsf{rt}, k, v, P) \to \{0, 1\}$: on inputs root rt, key k, value v and proof P, outputs 1 if (k, v) exists in rt.

- $\mathsf{NMProve}(k) \rightarrow P$: outputs a non-membership proof for key k.
- $\mathsf{NMVerify}(\mathsf{rt}, k, P) \rightarrow \{0,1\}$: on inputs root rt, key k and proof P, outputs 1 if k does not exist in rt.

4. Definitions

At the core of our system is an anonymous credential scheme coupled with a decentralized organization which verifies user credentials and maintains a list of pseudonyms corresponding to verified users. In our concrete instantiation this organization is simply a smart contract residing on the blockchain. Users register on the system using a pseudonym (such as their wallet address) and obtain an access token for their pseudonym from the organization by providing *auxiliary* information justifying the same. Concretely, this translates to the user's pseudonym being added to a list maintained by the smart contract. In any future transactions with service providers involving this wallet, a user can prove their wallet has been verified by simply pointing to the appropriate location in the smart contract's storage.

By changing the definition of what constitutes valid auxiliary information our system can be adapted to a wide range of applications. For example:

- In the simplest case, a single CA issues credentials to users in the form of signatures on their identity. This can be extended to support revocation through a public revocation list.
- It is also possible to accommodate threshold/hierarchical credential issuance with multiple CAs [109], [35], [26].

In our concrete instantiation we use the following policy:

- A CA must have issued the user a credential that has not been previously revoked.
- When requesting verification for a pseudonym, the request must be authenticated by the pseudonym owner (in the form of a signature).

4.1. Security

We consider two notions of security (Appendix A) with three different corruption scenarios as follows:

- Simulation-based security against a fully malicious adversary corrupting up to t auditors and any number of users.
- Simulation-based security against a semi-honest CA that does not collude with any other party.
- Privacy against against a fully malicious adversary corrupting up to t auditors, any number of users and the CA. In this setting, we do not provide any guarantees regarding the correctness of an honest party's output but ensure that the adversary does not learn any information about an honest party's inputs. However, we demand correctness when all parties are honest to rule out trivial protocols.

In Figure 3 we describe the ideal functionality involving users $\{U_1, \ldots, U_N\}$ and a list of auditors $(\mathsf{Aud}_1, \ldots, \mathsf{Aud}_n)$. Credentials are awarded to users when they make a request to the ideal functionality. This is done by sending a message doc "justifying" the issue of a credential to $\mathcal{F}$, which is then forwarded to the CA who decides to approve/reject a request. If the request is approved, the corresponding user is added to a list $\mathcal{L}$. Users interact with access controlled applications through pseudonyms which have been awarded an "access token". To do so, a user U first registers a pseudonym pk^{W} and sends appropriate auxiliary information aux justifying the issue of the access token to the ideal functionality. If the request satisfies the verification policy and the user $U \in \mathcal{L}$, then the pseudonym is awarded a token by adding $(\mathsf{pk}^{\mathsf{W}}, U)$ to a key-value database where the keys (pseudonyms) are made public but the values (users) are hidden. When a CA revokes a credential it has issued, the database of verified pseudonyms is archived and a fresh database is used in its place.

As a technical detail, we allow multiple users to obtain an access token on the same pseudonym or even the same user to obtain an access token on the same pseudonym multiple times. When this happens, all parties are privy to number of times access tokens were requested for a particular pseudonym. During the audit of a pseudonym pk^{W}, all users who requested verification for a particular pseudonym are revealed to the set of auditors.

Multiple Certificate Authorities. For simplicity, we focus on a single CA in the definition of $\mathcal{F}$. This can be easily extended to support the issuance of credentials from multiple CAs. Moreover, by defining the verification policies appropriately, one can increase the anonymity set of users from those that obtained a credential from a particular CA to the set of all users who obtained a credential from *any* CA. In our concrete instantiation (§ 5) we show that this can indeed be done efficiently.

5. Anonymous Credential Scheme

In this section, we describe the core protocols in our scheme, i.e., credential generation, verification, revocation and auditing in § 5.1. Then, we describe how to efficiently batch credential verification for L1 and L2 wallets to amortize costs in § 5.2 and § 5.3. Finally, we discuss some potential extensions in § 5.4.

5.1. Core Protocols

5.1.1. Credential Generation (Figure 4)

Our system supports multiple CAs, where each CA is assigned a unique group-id g. Each CA issues independent credentials but any of these credentials can be used for verification on-chain. This offers the advantage of an increased anonymity set which now comprises of all users who obtained a credential from *any* of the CAs.

To obtain a credential, user U first samples a signature key pair $(\mathsf{pk}^{\mathsf{U}}, \mathsf{sk}^{\mathsf{U}}) \leftarrow \mathsf{SIG}^2.\mathsf{Gen}(1^\lambda)$, where SIG^2 is a signature scheme instantiated as defined in § 6. U then sends a credential generation request $(\mathsf{pk}^{\mathsf{U}}, \mathsf{doc})$ to CA_g (a CA with group-id g) where doc is used to justify the issuance of a credential. CA_g reviews the supplied information and if it satisfies the issuance policy, it samples a unique user-id u and grants the user a credential. Granted credential cred contains a credential identifier $\mathsf{id} = g \| u$, an expiration

$\mathcal{F}$

Parties: Users $\{U_1, \ldots, U_N\}$, Certificate Authority CA and Auditors $(\mathsf{Aud}_1, \ldots, \mathsf{Aud}_n)$.
Parameters: A verification policy $\mathcal{C}$, pseudonym space $\mathcal{N}$, user space $\mathcal{U}$ and auxiliary information space $\mathcal{X}$.

- **Setup.** On input $(\mathsf{Setup}, \mathcal{C})$ from the organization, publish a verification issue policy $\mathcal{C}$.
- **Credential Generation.** On input $(\mathsf{ReqCred}, \mathsf{doc})$ from user U, forward (doc, U) to CA. On input $(\mathsf{AprCred}, U)$ from CA, add U to list of users who have been awarded credentials $\mathcal{L}$ and send 1 to U.
- **Credential Verification.** Maintain a database $\mathcal{D}$ of tuples from the space $\mathcal{N} \times \mathcal{U}$. The pseudonyms in each tuple are made public while the corresponding users are hidden. On input $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, \mathsf{aux})$ from user U,
  - If $\mathcal{C}(\mathsf{pk}^{\mathsf{W}}, \mathsf{aux}) = 1$ and $U \in \mathcal{L}$, add $(\mathsf{pk}^{\mathsf{W}}, U)$ to the database and publish aux.
  - Else, return $\perp$ to U.
- **Audit.** On input $(\mathsf{Audit}, \mathsf{pk}^{\mathsf{W}})$ from any $t+1$ auditors
  - Scan $\mathcal{D}$ for all occurrences of pk^{W} and gather corresponding users. Send this list of users to all auditors.
  - If pk^{W} does not appear in the database, then return $\perp$ to all auditors.
- **Revocation.** On input (Revoke, U) from party CA
  - If $U \notin \mathcal{L}$ return $\perp$ to CA.
  - Else, add U to a public ban-list $\mathcal{L}_B$.
  - Archive the database $\mathcal{D}$ and create a fresh database $\tilde{\mathcal{D}}$. Use $\tilde{\mathcal{D}}$ for any future credential issuance.

Figure 3: Ideal Functionality for an Anonymous Credential Scheme supporting Audits and Revocations.

Credential Generation

User U:

- Sample $(\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{U}}) := \mathsf{SIG}^2.\mathsf{Gen}(1^\lambda)$.

$U \rightarrow \mathsf{CA}_g$: $\mathsf{pk}^{\mathsf{U}}, \mathsf{doc}$

CA with group-id g (CA_g):

- Validate doc w.r.t. credential issuance policy.
- Sample a unique user-id u and set $\mathsf{id} := g \| u$.
- Choose an expiration epoch e and create a credential: $\sigma := \mathsf{SIG}^1.\mathsf{Sign}(\mathsf{sk}_g^{\mathsf{CA}}, \mathsf{CRH}(\mathsf{pk}^{\mathsf{U}}, \mathsf{id}, e))$.

$\mathsf{CA}_g \rightarrow U$: $\mathsf{cred} = (\sigma, \mathsf{id}, e)$

Figure 4: Credential Generation Protocol

epoch e after which the credential is no longer valid, and a signature σ on $\mathsf{pk}^{\mathsf{U}} \| \mathsf{id} \| e$ using signature scheme SIG^1.

Although it may seem like the credential identifier is redundant given that a user's public key is already unique, using a separate identifier lends the protocol to further optimization. Note that in a real system there are far fewer users than the number of public keys. We leverage this by using *compact* 40-bit identifiers as opposed to 256-bit public keys, which in turn reduces the credential verification circuit size by $3\times$ and also leads to smaller on-chain storage costs during revocation. Credential identifier also embeds the group-id of the CA that issued the credential, which helps during audits and prevents a malicious CA from revoking credentials it did not issue.

5.1.2. Credential Revocation (Figure 11)

We allow CAs to revoke any credential they have previously issued and accumulate all revoked credentials through two merkle trees[7]:

- $\mathsf{MT}_g^{\mathsf{rl}}$: stores credential identifiers revoked by CA_g.
- $\mathsf{MT}^{\mathsf{rr}}$: stores merkle-roots of each CA's revocation list.

To revoke credentials, CA_g first updates $\mathsf{MT}_g^{\mathsf{rl}}$ with newly revoked id's and computes the updated root $\mathsf{rt}_g^{\mathsf{rl}}$. Then, the CAs send their updated roots to an *untrusted* coordinator who posts a batch update for the revocation lists of all CAs.To ensure that the coordinator cannot change the revocation roots or replay old roots, CA_g signs the tuple $(\mathsf{rt}_g^{\mathsf{rl}}, E)$, where E is the current epoch number, with its secret key $\mathsf{sk}_g^{\mathsf{CA}}$ and sends it along with $\mathsf{rt}_g^{\mathsf{rl}}$ to the coordinator. The coordinator then locally updates $\mathsf{MT}^{\mathsf{rr}}$ and creates a proof π attesting to the correctness of new revocation root $\mathsf{rt}^{\mathsf{rr}}$:

1) $\mathsf{rt}_{\mathsf{old}}^{\mathsf{rr}}$, known by the contract, is the old root of revocation merkle tree with $(g, \mathsf{rt}_{\mathsf{old},\mathsf{g}}^{\mathsf{rl}})$ as leaves.
2) $\mathsf{rt}^{\mathsf{rr}}$, provided by the coordinator, is the new root of revocation merkle tree with $(g, \mathsf{rt}_g^{\mathsf{rl}})$ as leaves.
3) $\mathsf{rt}^{\mathsf{vk}}$, known by the contract, is the root of CA's verification key merkle tree with $(g, \mathsf{vk}_g^{\mathsf{CA}})$ as leaves.
4) $\forall g \in G$ s.t. $\mathsf{rt}_{\mathsf{old},\mathsf{g}}^{\mathsf{rl}} \neq \mathsf{rt}_g^{\mathsf{rl}}$, I know a signature $\sigma_{\mathsf{rv},g}$ on $(\mathsf{rt}_g^{\mathsf{rl}}, E)$ w.r.t. $\mathsf{vk}_g^{\mathsf{CA}}$, where E is the current epoch number.

The coordinator sends this proof and associated revocation information to the smart contract which verifies the proof, updates the root and increments the epoch number. All the revoked ids are also signed by the CAs and published by the coordinator to allow users to compute proofs of non-membership.

5.1.3. Credential Verification (Figure 5)

A user owning a credential cred from a CA, say CA_g, can use it to issue an *ephemeral access token* for any wallet pk^{W} it owns. These tokens are checked by the access-controlled application before granting access and they are only valid for one epoch.

As discussed in § 5.1.2, the epoch number is incremented when the revocation list is updated, and all tokens expire as it is not possible to determine which wallets were verified with a revoked credential. We realize the ephemeral nature of tokens through a simple yet effective mechanism: we embed an epoch identifier in each token that refers to the epoch in which it was issued. Access token verification is done by checking if the epoch identifier matches the current epoch. For the sake of exposition, we consider the simplest implementation of a badge: the contract maintains a hash-map LV that maps wallet addresses to the epoch when they

7. RSA accumulators [22], [82], [43] were shown to be more efficient than merkle-trees in applications with many accesses [96], but for our application, we require one access on a tree of size at most 2^{40}, for which (sparse) merkle trees are much more efficient and cost just 23K constraints.

Credential Verification

User (owner of wallet pk^{W}):

- Parse $\mathsf{cred} = (\sigma, \mathsf{id}, e)$ and $\mathsf{id} = g \| u$.
- Set $P^{\mathsf{vk}} := \mathsf{MT}^{\mathsf{vk}}.\mathsf{MProve}(g)$.
- Retrieve $\{\mathsf{rt}_i^{\mathsf{rl}}\}$, update local $\mathsf{MT}^{\mathsf{rr}}$: $\forall i, \mathsf{MT}^{\mathsf{rr}}.\mathsf{Add}(i, \mathsf{rt}_i^{\mathsf{rl}})$, and set $P^{\mathsf{rr}} := \mathsf{MT}^{\mathsf{rr}}.\mathsf{MProve}(g)$.
- Retrieve ids revoked $\{\mathsf{id}_{g,j}\}_j$ by CA_g, update local $\mathsf{MT}_g^{\mathsf{rl}}$: $\forall j, \mathsf{MT}_g^{\mathsf{rl}}.\mathsf{Add}(\mathsf{id}_{g,j}, 1)$, and set $P^{\mathsf{rl}} := \mathsf{MT}_g^{\mathsf{rl}}.\mathsf{NMProve}(u)$.
- Set $\psi := \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{id}; \rho)$, ρ is randomness.
- Set $\sigma^{\mathsf{W}} := \mathsf{SIG}^2.\mathsf{Sign}(\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}})$.
- Let $x = (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{pk}^{\mathsf{W}}, \mathsf{pk}^{\mathsf{A}}, \psi, E)$.
- Let $w = (\mathsf{vk}_g^{\mathsf{CA}}, \mathsf{cred}, \mathsf{pk}^{\mathsf{U}}, \mathsf{sk}^{\mathsf{U}}, \mathsf{rt}_g^{\mathsf{rl}}, \sigma^{\mathsf{W}}, \rho, P^{\mathsf{vk}}, P^{\mathsf{rr}}, P^{\mathsf{rl}})$.
- Create a proof $\pi := \mathsf{SNARK.Prove}(\mathsf{crs}_{\mathsf{ver}}, x, w)$, where ver is defined as follows:

$$\mathsf{ver} = \Big\{ (x, w) \mid \mathsf{MT}^{\mathsf{vk}}.\mathsf{MVerify}(\mathsf{rt}^{\mathsf{vk}}, g, \mathsf{vk}_g^{\mathsf{CA}}, P^{\mathsf{vk}}) = 1 \\ \wedge \mathsf{SIG}^1.\mathsf{Verify}(\mathsf{vk}_g^{\mathsf{CA}}, \mathsf{CRH}(\mathsf{pk}^{\mathsf{U}}, \mathsf{id}, e), \sigma) = 1 \\ \wedge e \geq E \\ \wedge \mathsf{MT}^{\mathsf{rr}}.\mathsf{MVerify}(\mathsf{rt}^{\mathsf{rr}}, g, \mathsf{rt}_g^{\mathsf{rl}}, P^{\mathsf{rr}}) = 1 \\ \wedge \mathsf{MT}^{\mathsf{rl}}.\mathsf{NMVerify}(\mathsf{rt}_g^{\mathsf{rl}}, u, P^{\mathsf{rl}}) = 1 \\ \wedge \mathsf{SIG}^2.\mathsf{Verify}(\mathsf{pk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}}, \sigma^{\mathsf{W}}) = 1 \\ \wedge \psi = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{id}; \rho) \Big\}$$

$\mathsf{pk}^{\mathsf{W}} \rightarrow$ **Smart Contract**:

- π and $\mathsf{TX} = (\mathsf{pk}^{\mathsf{W}}, \psi)$ signed by pk^{W}.

Smart Contract:

- Verify the signature on TX, w.r.t. pk^{W}.
- Check if $\mathsf{SNARK.Verify}(\mathsf{crs}_{\mathsf{ver}}, x, \pi) = 1$.
- If both checks pass, set $\mathsf{LV}(\mathsf{pk}^{\mathsf{W}}) := E$.

Figure 5: Credential Verification Protocol

were last verified. Thus, a wallet pk^{W} holds an access token iff $\mathsf{LV}(\mathsf{pk}^{\mathsf{W}}) = E$ where E is the current epoch.

Importantly, credential verification hides all information except for the fact that a wallet wishes to obtain an access token and whether it is backed by a valid credential. Hence, this ensures that users cannot be linked to their wallets, and no information is revealed which could be used to link two wallets belonging to the same user. This is made possible by a zkSNARK π which proves the conjunction of following statements:

1) I know credential $\mathsf{cred} = (\sigma, \mathsf{id}, e)$ such that:
   - σ is a valid signature under $\mathsf{vk}_g^{\mathsf{CA}}$ on three values: owner's public key pk^{U}, cred's unique identifier id, and cred's expiration epoch number e.
   - $\mathsf{vk}_g^{\mathsf{CA}}$ is a *member* of $\mathsf{MT}^{\mathsf{vk}}$ at index g.
   - cred is not expired, i.e., cred's expiration epoch number $e \geq E$, the current epoch number.
   - cred has not been revoked:
     - u is *not a member* of $\mathsf{MT}_g^{\mathsf{rl}}$ with root $\mathsf{rt}_g^{\mathsf{rl}}$.
     - $\mathsf{rt}_g^{\mathsf{rl}}$ is a *member* of $\mathsf{MT}^{\mathsf{rr}}$ at index g.
   - $\mathsf{id} = g \| u$.
2) ψ is a well-formed audit token for cred, i.e, it encrypts id under the public key pk^{A} of the auditors.
3) pk^{W} is the wallet intended for verification i.e., I know a signature σ^{W} on pk^{W} w.r.t. pk^{U}. This prevents an adversary from reusing the proof to verify another wallet.

The credential verification transaction $\mathsf{TX} = (\mathsf{pk}^{\mathsf{W}}, \psi)$ is also signed by pk^{W} to ensure that it was indeed approved by the owner of pk^{W}.

Almost all of the information required by a user to prepare the above proof does not change over time and can hence be stored locally. The only information that needs to be updated every epoch is the revocation list. In particular, the user has to retrieve the new ids revoked by CA_g since the last proof was generated, the revocation roots $\{\mathsf{rt}_i^{\mathsf{rl}}\}_i$ from all CAs, and the root $\mathsf{rt}^{\mathsf{rr}}$ of the merkle tree $\mathsf{MT}^{\mathsf{rr}}$ which stores CA's revocation roots. It is easy to retrieve this information as it is posted as part of the revocation procedure (see § 5.1.2). In practice, for users with computationally-weak devices, merkle tree proofs can be queried from an *untrusted* server. This query will only reveal id to the server, which is not revealed as part of the credential verification, and thus, can't be used to link the user to the wallet. The user doesn't have to trust the server for integrity as the user can very efficiently verify the proofs locally given $\mathsf{rt}^{\mathsf{rr}}$ and $\mathsf{rt}_g^{\mathsf{rl}}$.

5.1.4. Transaction Audit (Figure 6)

Transaction Audit

Input: TX verifying a malicious wallet
Auditor a, $a \in A \subseteq \{1, \ldots, n\}$ s.t. $|A| \geq t + 1$:

- Extract the audit token ψ from TX and compute $\mathsf{id}_a = \mathsf{TPKE.Dec}(\mathsf{sk}_a^{\mathsf{A}}, \psi)$.
- $\forall i \in A \backslash \{a\}$, collect decryption result id_i from auditor A_i and verify that $\mathsf{TPKE.Verify}(\mathsf{pk}^{\mathsf{A}}, \mathsf{vk}^{\mathsf{A}}, \mathsf{id}_i) = 1$.
- Using any $t + 1$ shares that pass verification each auditor computes $\mathsf{id} = \mathsf{TPKE.Combine}(\{\mathsf{id}_i\}_{i \in A})$.

Figure 6: Transaction Audit Protocol

A threshold number ($t+1$-out-of-n) of auditors can deanonymize the user associated with a wallet, if they deem the wallet has displayed malicious behaviour. In particular, before a wallet is issued an access token, a user must undergo credential verification, and an audit token ψ is produced as part of each such credential verification transaction TX. The audit token is simply an encryption of the unique credential identifier id corresponding to the credential used for verification. Since id is encrypted using the public key pk^{A} of the auditors, $t+1$ of them can collaboratively decrypt ψ to recover id, which is of the form $\mathsf{id} = g \| u$. This points the auditors to the issuing CA, i.e., CA_g, who can then reveal user's identity and revoke the corresponding credential if required.

We provide formal statements of our security guarantees below, and the definitions and corresponding proofs of security are deferred to Appendix A and Appendix B, respectively.

Theorem 1. *The protocol in § 5 securely emulates the ideal functionality $\mathcal{F}$ (Figure 3) for any PPT malicious $\mathcal{A}$ corrupting a threshold number of auditors and an arbitrary number of users provided NIAoK and Public-Key Encryption schemes exist.*

Theorem 2. *The protocol in § 5 securely emulates the ideal functionality $\mathcal{F}$ (Figure 3) for any semi-honest PPT $\mathcal{A}$ corrupting the CA provided NIAoK and Public-Key Encryption schemes exist.*

Theorem 3. *The protocol in § 5 privately emulates the ideal functionality $\mathcal{F}$ (Figure 3) for any PPT malicious $\mathcal{A}$ corrupting the CA, a threshold number of auditors and an arbitrary number of users provided NIAoK and Public-Key Encryption schemes exist.*

5.2. Batched Verification (Figure 7)

In § 5.1.3, we described a credential verification protocol that allows a user to obtain an on-chain access token for its wallet. Even though this protocol has much smaller gas cost than the existing AC schemes (§ 8.1), it is still unreasonable for applications like KYC despite using the cheapest on-chain SNARK verifier [71]. In this section, we describe how multiple users can batch credential verification of their L1 wallets with the help of an *untrusted* aggregator, and pay significantly lower gas costs.

5.2.1. Batching SNARK Verification.

We first look at how credential verification SNARK proofs, which are the *bottleneck* for on-chain cost, can be batched across users through proof recursion [119], [48], [23], [21]. The high-level idea is simple: the aggregator collects credential verification proofs from N users, verifies them and *recursively* proves to the contract that it performed the verification correctly using another SNARK proof. Since the SNARK has sublinear verification, the contract has to spend sublinear effort in verifying the whole batch, and in turn, the amortized gas cost per user is much smaller.

In practice, to keep gas costs low for reasonable batch sizes, it is desirable to have the contract verify a pairing-based SNARK proof over the BN254 [21] curve – the only curve supported by EVM. The problem, however, is that none of the prior approaches to proof recursion lead to a practical solution with this restriction (§ 8.2.3). In more detail, prior works use the following approaches: (i) pairing-based [21], [76], (ii) FRI-based[8] [101], and (iii) accumulation-based [33], [32], [29], [81]. Approach (i) either requires a 2-chain (or cycle) of pairing-friendly elliptic curves which is not known[9] for BN254, or the use of expensive non-native arithmetic which leads to intractable aggregator overhead. Approach (ii) has tractable aggregator overhead, but the prover cost is high which also leads to high user overhead. Finally, approach (iii) has a *linear* verification step independent of number of users, which alone imposes a huge aggregator overhead in our setting. Hence, none of the prior approaches are suitable for our setting and we justify it concretely in § 8.2.

In this work, we adopt a *new approach* that relies on recursively verifying a discrete-log-based SNARK. In general, discrete-log-based SNARKs are not suitable for

Batched Verification

User $i, i \in [N]$:

- Create a credential verification $\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$ with proof π_i^1 using SNARK^1 as in Figure 5.

User $i \rightarrow$ **Aggregator**:

- π_i^1 and $\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$ signed by $\mathsf{pk}_i^{\mathsf{W}}$

Aggregator:

- Set $H_0 = \mathbf{0}$, $\forall i$, set $H_i = \mathsf{CRH}^{\mathsf{sc}}(H_{i-1}, h_i)$, where $h_i := \mathsf{CRH}^{\mathsf{sc}}(\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$.
- Let $\forall i$, $\mathsf{x}_i = (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{pk}^{\mathsf{A}}, E, H_{i-1}, H_i)$, $\mathsf{w}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i, \pi_i^1)$.
- Create $\Pi^2 := \mathsf{simdSNARK}^2.\mathsf{Prove}(\mathsf{crs}_{\mathsf{bat}^2}, \{\mathsf{x}_i\}_i, \{\mathsf{w}_i\}_i)$, where bat^2 is defined as follows:

$$\mathsf{bat}^2 = \Big\{(\mathsf{x}_i, \mathsf{w}_i) \mid x := (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{pk}_i^{\mathsf{W}}, \mathsf{pk}^{\mathsf{A}}, \psi_i, E) \; h_i = \mathsf{CRH}^{\mathsf{sc}}(\mathsf{pk}_i^{\mathsf{W}}, \psi_i) \wedge H_i = \mathsf{CRH}^{\mathsf{sc}}(H_{i-1}, h) \wedge \mathsf{SNARK}^1.\mathsf{Verify}(\mathsf{crs}_{\mathsf{ver}}^1, x, \pi_i^1) = 1\Big\}$$

- Let $\mathsf{x} = (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{pk}^{\mathsf{A}}, E, H_N)$, $\mathsf{w} = (\Pi^2, \{h_i, H_i\}_i)$.
- Create proof $\Pi^1 := \mathsf{SNARK}^1.\mathsf{Prove}(\mathsf{crs}_{\mathsf{bat}^1}, \mathsf{x}, \mathsf{w})$, where bat^1 is defined as follows:

$$\mathsf{bat}^1 = \Big\{(\mathsf{x}, \mathsf{w}) \mid \forall i \in [N], x_i := (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{pk}^{\mathsf{A}}, E, H_{i-1}, H_i) \; \mathsf{simdSNARK}^2.\mathsf{Verify}(\mathsf{crs}_{\mathsf{bat}^2}, \{x_i\}_i, \Pi^2) = 1\Big\}$$

Aggregator $\rightarrow$ **Smart Contract**: INIT

- $\{\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$ signed by $\mathsf{pk}_i^{\mathsf{W}}\}_{i \in [N]}, \Pi^1$

Smart Contract:

- $\forall i \in [N]$, verify signature on TX_i w.r.t. $\mathsf{pk}_i^{\mathsf{W}}$, set $h_i := \mathsf{CRH}^{\mathsf{sc}}(\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$, and set $h := \mathsf{CRH}^{\mathsf{sc}}(h_1, \ldots, h_N)$.
- If $\mathsf{SNARK}^1.\mathsf{Verify}(\mathsf{crs}_{\mathsf{bat}^1}, \mathsf{x}, \Pi^1) = 1$, $\forall i \in [N]$, set $\mathsf{LV}(\mathsf{pk}_i^{\mathsf{W}}) := E$ and $\mathsf{HC}(\mathsf{pk}_i^{\mathsf{W}}) := h_i$.

Aggregator $\rightarrow$ **Smart Contract**: CACHED

- $\{\mathsf{TX}_i = \mathsf{pk}_i^{\mathsf{W}}\}_{i \in [N]}, \Pi^1$

Smart Contract:

- $\forall i \in [N]$, set $h_i := \mathsf{HC}(\mathsf{pk}_i^{\mathsf{W}})$.
- Set $h := \mathsf{CRH}^{\mathsf{sc}}(h_1, \ldots, h_N)$.
- If $\mathsf{SNARK}^1.\mathsf{Verify}(\mathsf{crs}_{\mathsf{bat}^1}, \mathsf{x}, \Pi^1) = 1$, $\forall i \in [N]$, set $\mathsf{LV}(\mathsf{pk}_i^{\mathsf{W}}) := E$.

Figure 7: Batched Verification Protocol

8. Concurrent work on ZkEVM [115], [62] improves upon the prior work on FRI-based approach and builds a system that can batch EVM contracts. At the moment, however, it doesn't support cryptographic operations required for credential verification of ZEBRA and prior AC schemes.

9. Even outside the context of EVM, the most efficient known 2-chain with 128-bit security is BLS12-377/BW6-761 [76], where BW6-761 is 6× slower than BN254 [75], albeit at a higher security level.

recursion as their verifier complexity is at least $\mathcal{O}_\lambda(\sqrt{n})$ for a circuit of size n. Despite this limitation, we manage to get a practical solution in our setting because of the way we use the discrete-log-based SNARK. Before we discuss why our solution is practical, we first describe it in detail.

We use two SNARKs, namely, pairing-based SNARK^1 (over BN254) and discrete-log-based $\mathsf{simdSNARK}^2$ (over Grumpkin [64]), and our batching solution (see Figure 2 for illustration) works as follows with *two layers of recursion*:

- A batch of N users independently create credential verification proofs $\{\pi_i^1\}_{i\in[N]}$ using SNARK^1, and send them along with transaction data $\{\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i)\}_{i\in[N]}$ to the aggregator.
- First, the aggregator verifies the proofs $\{\pi_i^1\}_{i\in[N]}$ using $\mathsf{simdSNARK}^2$ to output Π^2.
- Then the aggregator verifies Π^2 using SNARK^1 to output Π^1, which is then sent to the contract for batched verification along with $\{\mathsf{TX}_i\}_{i\in[N]}$.
- Finally, the contract processes $\{\mathsf{TX}_i\}_{i\in[N]}$ and verifies Π^1 to ensure the validity of user proofs $\{\pi_i^1\}_{i\in[N]}$ before updating access tokens for $\{\mathsf{pk}_i^{\mathsf{W}}\}_{i\in[N]}$.

Now, we discuss the efficiency benefits of our approach: (i) there are no compatibility issues as BN254 forms a cycle with (non-pairing-friendly) Grumpkin, and thus, $\mathsf{simdSNARK}^2$ over Grumpkin can efficiently verify SNARK^1 proofs over BN254 and vice-versa, (ii) since $\mathsf{simdSNARK}^2$ is verifying N independent SNARK^1 proofs, i.e., a *data-parallel* (or SIMD) computation, the concrete cost for both prover and verifier can be improved by $8\times$ (§ 7), (iii) $\mathsf{simdSNARK}^2$ verifier complexity is $\mathcal{O}_\lambda(\sqrt{N \cdot n})$[10], where n is cost of verifying a single user proof, which is sublinear in N, and (iv) the users generate SNARK^1 proofs which impose reasonable overheads. As a result, we get a solution with SNARK^1 on-chain verification that has either $6.3\times$ less aggregator overhead or $11\times$ less user overhead compared to prior solutions (§ 8.2.3).

5.2.2. Audit Token Caching

Proof batching mitigates the bottleneck in credential verification, but there are still costs that scale with batch size and lower-bound the amortized gas cost per user. Specifically, operations on user-specific inputs ($\{\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \psi_i)\}_{i\in[N]}$), such as verifying the signature on TX_i w.r.t. $\mathsf{pk}_i^{\mathsf{W}}$ and SNARK input processing on TX_i, still scale linearly with batch size and these inputs along with signatures also have to be stored on-chain. Even *ignoring* the SNARK verification completely, this places a lower bound of 20K gas per L1 wallet, and also of 5K gas per L2 wallet (§ 8.2.1).

We observe that this issue can be circumvented by leveraging the fact that user-specific information posted on-chain does not change across epochs if the same wallet is re-verified using the same credential. We expect this to be the typical use case as using different credentials does not provide any additional privacy. Therefore, one can hope that user-specific inputs can be processed once and then cached on the smart contract for subsequent verifications. While it is easy to avoid repeated signature verification, it is not straightforward to cache the SNARK input processing which is quite expensive (83.7K gas for each TX_i) for the SNARK^1 we use in our concrete instantiation. This is so because the indices of user-specific inputs can potentially change with every new batch and the index determines the processing applied to an input.

We resolve the issue of expensive input processing by making public inputs to the batch verification circuit independent of the batch size. This is achieved by moving user-specific public inputs to private witnesses through the use of a smart-contract-friendly collision-resistant hash function $\mathsf{CRH}^{\mathsf{sc}}$. In particular, for each user $i \in [N]$, the smart contract first computes a hash of user-specific inputs $h_i := \mathsf{CRH}^{\mathsf{sc}}(\mathsf{TX}_i)$, then verifies the signature on h_i w.r.t. $\mathsf{pk}_i^{\mathsf{W}}$, and finally computes a hash $h = \mathsf{CRH}^{\mathsf{sc}}(h_1, \ldots, h_N)$. h is then used as input to the SNARK and the aggregator proves within Π^2 that $\forall i \in [N], h_i = \mathsf{CRH}^{\mathsf{sc}}(\mathsf{pk}_i^{\mathsf{W}}, \psi_i) \wedge H_i = \mathsf{CRH}^{\mathsf{sc}}(H_{i-1}, h_i)$, where $(\mathsf{pk}_i^{\mathsf{W}}, \psi_i)$ were inputs used to verify π_i^1, $H_0 = \mathbf{0}$, and $H_N = h$. If the verification of Π^1 is successful, the smart contract caches h_i for wallet $\mathsf{pk}_i^{\mathsf{W}}$ in a hashmap HC. Note that in the next epoch when the wallet has to be verified again with the same credential, the contract can simply retrieve h_i's from cache HC using $\mathsf{pk}_i^{\mathsf{W}}$'s, compute $h = \mathsf{CRH}^{\mathsf{sc}}(h_1, \ldots, h_N)$, and check if the proof Π^2 verifies successfully with h as input. This technique not only reduces user-specific costs in batched verification, it also avoids expensive SNARK input processing per user, both of which make our L1 batched verification $10\times$ cheaper overall (Figure 8).

5.3. L2 Verification

In this section, we explain how we take advantage of the zk-rollup [34] design to deliver very low gas costs per credential verification for an L2 wallet. Our L2 verification protocol is very similar to the batched verification of L1 wallets, except the last-verified map (LV) and audit-token cache (HC) are stored within the state maintained by the rollup server, as opposed to being stored on the contract.

Access token verification is very simple with this design: L2 transactions can check if wallets possess an access token by simply looking up their LV value and proving that it matches the current epoch to the contract as part of the transaction rollup proof.

To cache the audit token ψ_i for user i in HC, similar to batched verification, ψ_i will be first posted on-chain. Signature verification on the transaction, on the other hand, will be performed inside the rollup as usual. ψ_i needs to be posted for data availability to ensure that the audit token whose validity is verified by the contract is always available to the auditors. If all verifications are successful, the contract allows a state update that stores ψ_i as part of the account state of wallet $\mathsf{pk}_i^{\mathsf{w}}$. Consequently, in the next epoch, when any of these $\mathsf{pk}_i^{\mathsf{w}}$'s need to renew their access tokens, the rollup can prove that the wallet is verified with respect to the

10. This is better than accumulation approach which has verifier complexity $\mathcal{O}_\lambda(n)$, as $N \ll n$ in our setting.

cached audit token and doesn't have to send any inputs that scale with batch size to the contract. Thus, the contract just performs SNARK verification, the cost of which remains constant irrespective of batch size.

5.4. Potential Extensions

In this section, we discuss some potential extensions to ZEBRA which are orthogonal to our work:

- We've considered a simple issuance model where some CAs are trusted to issue credentials. It is easy to extend our scheme to support other issuance models that are decentralized [67], [103], legacy-compatible [86], [129], [128], and threshold/hierarchical [109], [35], [26].
- We embed our credentials with only three attributes (see § 5.1.1), but it is straightforward to extend our credentials to include arbitrary many attributes and prove arbitrarily complex statements on them. Hence, our credentials can also embed the credential/claim schemas defined by W3C [51] and iden3 [2], [102] for interoperability.
- Our current scheme allows verifying any wallet using a credential, but one can consider credentials restricted to a few user selected wallets through the use of blind signatures. Here a user obtains a signature on $\mathsf{pk}^{\mathsf{U}}||\{\mathsf{pk}_i^{\mathsf{W}}\}$ instead of pk^{U} as done in Figure 4, thereby preventing the credential from being used to verify arbitrary wallet addresses. This prevents abuse when credentials are stolen and could be essential in some applications.
- We can support single-use credentials which are useful for decentralized voting by introducing a nullifier attribute (unknown to the CA) to the credentials. The nullifier is posted on-chain during verification and the credential is verified w.r.t. the nullifier. The contract can ensure that the same nullifier is not used twice, and this ensures that a credential is only used once.
- Like any threshold system, a major concern in our scheme is that if a threshold number of auditors are corrupted then privacy of all users is lost. This can be mitigated by refreshing shares regularly [74], [87], [70] to protect against *mobile* adversaries that can eventually corrupt all parties over time.

6. Concrete Instantiation

In this section, we discuss how we concretely instantiate the primitives used in § 5 and our rationale behind these choices. In our implementation, we use 40-bit long credential identifiers, where 8-bits are reserved for the group-id of the CA, and 32-bits are reserved per CA to assign unique identifiers for the its issued credentials. This means that in the current instantiation, we can support up to 256 CAs and around 4 billion issued credentials per CA. Note that the expired credentials can be reissued with the same identifier.

6.1. Core Protocols

Collision-Resistant Hash. We use MiMC [8] to instantiate the correlation-resistant hash (CRH) in our scheme. A MiMC call on two field elements costs 550 constraints.

zk-SNARK. We use Groth16 [71] instantiated over the BN254 curve [20], [105] as the zk-SNARK in our evaluation. The circuit-specific trusted setup for Groth16 can be performed by the CAs and the auditors in our setting.

Alternatively, one could use PLONK [66], a pairing-based SNARK with universal trusted setup, for more flexibility in performing the setup. The gas costs for PLONK are comparable to Groth16, and the prover can be made just as fast with custom gates [121], [126]. We use Groth16 in our evaluation because it has much better development support.

Digital Signature We instantiate SIG^1 with EdDSA [84] on BabyJubJub curve [122], which is efficient to verify within Groth16 instantiated over the BN254 curve [21]. Verifying an EdDSA signature costs around 6500 constraints. SIG^2 is the simulation-extractable NIZKPoK+OWF signature scheme by Bellare [18], where instead of a simulation extractable NIZK we use Groth16 which is only weakly simulation extractable. However, it can be shown that the scheme satisfies EUF-CMA which is sufficient to prove security of our overall protocol. See Appendix C for a full proof. The one way function is instantiated with MiMC [8]. Generating this signature costs just 1K constraints.

Threshold Public Key Encryption We instantiate TPKE with the threshold variant of CCA-2 secure Cramer-Shoup encryption described in [45]. Verifying well-formedness of this encryption scheme costs around 24K constraints.

Sparse Merkle Tree We use the iden3's implementation of Sparse Merkle Tree [78] instantiated with MiMC. With 40-bit credential identifiers, it costs around 6K, 5K, and 18K constraints for verifying membership of $\mathsf{vk}_g^{\mathsf{CA}}$ in $\mathsf{rt}^{\mathsf{vk}}$, membership of $\mathsf{rt}_g^{\mathsf{rl}}$ in $\mathsf{rt}^{\mathsf{rr}}$, and non-membership of u in $\mathsf{rt}_g^{\mathsf{rl}}$ (see Figure 5).

6.2. Batched and L2 Verification

We instantiate SNARK^1 with pairing-based Groth16 [71] (over BN254) and $\mathsf{simdSNARK}^2$ with discrete-log-based Spartan [106] (over Grumpkin) optimized for data-parallelism. Both of these SNARKs use R1CS arithmetization. In this work, we've focused on R1CS because it has the best development support currently, and the prior works in proof recursion literature are also based on R1CS. Although our batching costs with R1CS are already practical (§ 8.2), they can be further improved significantly using plonk arithmetization [66], [65] and its custom gates, specifically for non-native arithmetic and MSMs [126].

7. Implementation

The implementation details of ZEBRA are as follows:

- Credential verification smart contract was implemented with 0.5K lines of Solidity (excluding benchmarks and autogenerated code).
- Data-parallel Spartan implementation and R1CS constraints for its verifier were implemented in

`arkworks` [14] with 3K and 1.8K lines of Rust, respectively. Data-parallelism improves the Spartan prover performance by 8× [106], and also reduces its verifier constraints by close to 8× in our setting. We also optimize constraints for multi-scalar multiplications (MSMs) by 2.05× (Appendix D), which *further* reduces the verifier constraints by 1.6×. It is worth noting that without this optimization, the constraints for Spartan verifier would be $> 2^{28}$ for a batch of 512 users, which is impossible to verify within Groth16 over BN254 due to limitations on 2-adicity of BN254's scalar field.
- Grumpkin operations and its constraints, and BN254 constraints in `arkworks` with 1K lines of Rust.
- Credential verification and credential revocation circuit in `gnark` [50] with 0.5K lines of Go.

All the Groth16 circuits are proven in `gnark` as it is more efficient than `arkworks`, and Spartan prover and the constraint generation for its verifier are run in `arkworks`.

8. Evaluation

In this section, we evaluate our implementation of ZEBRA and answer the following questions:

1) For single credential verification:
   - What is the computational cost imposed on users, especially on weak devices like smartphones (§ 8.1.1)?
   - What is the gas cost incurred for verifying a single ZEBRA credential on EVM (§ 8.1.2)?
   - How does ZEBRA compare with existing anonymous credentials (§ 8.1.3)?
2) For batched credential verification:
   - What is the improvement in gas cost incurred per user with batched verification (§ 8.2.1)?
   - For a large enough batch, what is the computational and monetary overhead on ZEBRA's aggregator (§ 8.2.2)?
   - How does our proof batching solution compare with prior approaches to proof recursion (§ 8.2.3)?

For evaluation of credential revocation, we defer the reader to Appendix E and note that credential generation and transaction audit are very lightweight in terms of cryptographic tools and not time-sensitive.

Experimental Setup. We evaluate the client on two setups: (i) a 2019 MacBook Pro with 2.4 GHz 8-Core Intel Core i9 processor and 16 GB RAM, and (ii) an Android mobile device with Qualcomm Snapdragon 855 processor and 6 GB RAM. The aggregator is benchmarked on an `m6i.32xlarge` AWS instance which has a 3.5 GHz Intel Xeon processor with 128 vCPUs and 512 GB of RAM. To benchmark EVM gas costs, we've used ganache v7.0.3 [111] and truffle suite v5.4.22 [110].

8.1. Credential Verification

8.1.1. User Overhead

Before SNARK proof generation, the user creates an audit token, a non-revocation proof, and a signed transaction. Out of these, the cost for creating the non-revocation proof, which depends on the number of revoked users and is dominated by the cost of updating the tree, is discussed in Appendix E; the runtime for the rest of the operations is less than 100 ms with a single thread.

TABLE 1: Comparison of ZEBRA's single verification with prior AC works in context of permissionless blockchains for three private attributes.

Technique	Verifier Complexity	Gas Cost
ZEBRA ⋆	9 mul-$\mathbb{G}_1$, 9 add-$\mathbb{G}_1$, 4 Pairings	360K
Coconut† [109]	5 mul-$\mathbb{G}_1$, 4 add-$\mathbb{G}_1$, 2 mul-$\mathbb{G}_2$ 2 add-$\mathbb{G}_2$, 2 Pairings	$>$ 4.2M
BASS‡ [127]	7 mul-$\mathbb{G}_1$, 8 add-$\mathbb{G}_1$, 2 mul-$\mathbb{G}_2$ 2 add-$\mathbb{G}_2$, 8 Pairings, 4 mul-$\mathbb{G}_T$, 5 add-$\mathbb{G}_T$	$>$ 12.57M*

⋆: Reduces to 1 mul-$\mathbb{G}_1$, 1 add-$\mathbb{G}_1$, 4 Pairings after caching
†: does not support revocation or auditability
‡: does not support auditability
*: assuming $\mathbb{G}_T$ operation cost $\geq$ corresponding $\mathbb{G}_2$ operation cost

The total constraints in our credential verification circuit are 62K and the proof generation takes just 250 ms on Macbook Pro using 16 threads. We also compiled the SNARK prover into WASM through `gnark` playground [28]. When run on an Android mobile device, it just took under 6s for proof generation with a single thread.

8.1.2. Gas Costs

A credential verification transaction costs 355K gas, out of which, 290K gas is for SNARK verification (including input processing), making it the bottleneck. Other than that, base transaction fee is 21K gas, signature verification on audit token and wallet address takes 9K gas, updating the access token takes 20K gas, and finally, posting the proof, the audit token, and the signature on-chain requires another 11K gas; the remaining gas cost is due to miscellaneous factors. If we have the audit token already cached, the gas cost comes down to 245K. With the current average gas price of 12 Gwei and the price of Ethereum (1866.32 USD) on August 15, 2022[11], our credential verification would require 7.95 USD for 355K gas, which is reasonable for applications with low transaction volume or rare revocations, both of which imply infrequent credential verification.

8.1.3. Comparison with Existing AC Schemes

We now compare the gas cost of ZEBRA's single verification with existing AC schemes[12]. Table 1 shows the verifier complexity and gas cost for ZEBRA, Coconut [109], [72] which is the state-of-the-art AC scheme in context of blockchains, and a subsequent work BASS [127] that adds revocation to Coconut. Neither Coconut nor BASS provide auditability. The verifier complexity in Table 1 for Coconut and BASS is for three private attributes (user public key, credential identifier, and expiration epoch) as is required in our system. We swapped the $\mathbb{G}_1$ and $\mathbb{G}_2$ operations for

11. This price was recommended by https://ethereumprice.org/gas/ for transaction confirmation within 5 minutes.

12. We focus this comparison on pairing-based AC schemes as an RSA-based AC scheme [91] requires at least 32M gas which is over the block limit on Ethereum.

both Coconut and BASS as suggested by Coconut to reduce their gas costs. We used the costs from EIP-1108 [123] to estimate the cost of $\mathbb{G}_1$ and pairing operations, and the benchmarks from Coconut's code [90] for $\mathbb{G}_2$ operations. For $\mathbb{G}_T$ operations, we assume the cost to be equal to that of $\mathbb{G}_2$ operations as there's no implementation available and $\mathbb{G}_T$ operations are always more expensive than $\mathbb{G}_2$ operations. We stress that while the verifier complexity and gas costs for ZEBRA are accurate and also include the gas cost for posting credential verification data on-chain, the costs for Coconut and BASS are only lower bounds since they would require additional verifier operations to prove predicates on the attributes.

Table 1 highlights both issues with existing AC schemes that we pointed out in § 1. First, verifier of existing AC schemes rely on $\mathbb{G}_2$ and $\mathbb{G}_T$ operations that are not natively supported by EVM and cost more than $300\times$ compared to $\mathbb{G}_1$ operations. Consequently, Coconut requires at least $11.8\times$ more gas than ZEBRA. Second, the verifier complexity of existing AC schemes grows linearly with predicate complexity. As a result, introducing revocation to Coconut in BASS makes the verifier significantly more complex and increases its gas cost by $3\times$. In contrast, adding revocation to ZEBRA had a marginal impact on gas cost as verifier complexity is independent of the predicate and only depends on number of inputs.

8.2. Batched Verification

8.2.1. Gas Costs

Figure 8 compares the gas cost per user for five kinds of credential verification: (i) ZEBRA's single verification as evaluated in § 8.1 (Single), (ii) L1 batched verification without (audit token) caching (L1-Naïve), (iii) ZEBRA's L1 batched verification with caching (L1-Cached), (iv) L2 batched verification without caching (L2-Naïve), and finally, (v) ZEBRA's L2 batched verification with caching (L2-Cached). We include (ii) and (iv) as baselines in this graph to highlight the significance of audit token caching, and the gas costs reported for batched verification with caching assume that the token is already cached.

We first analyze the gas cost for L1 batching. The figure demonstrates that ZEBRA's L1 batching reduces the gas cost by up to $35\times$ and requires just 13.7K and 10.2K gas per user for a batch of 64 and 512 users, respectively. For a batch of 512 users, this translates to just 0.23 USD on Ethereum. Without caching, the gas cost improvement is just $3.9\times$, which is largely due to expensive SNARK input processing with Groth16 which costs around 9.3K gas per 32-Byte input. Even ignoring the cost from Groth16 entirely, the gas cost without caching would still be around 20K. This shows that our caching technique is essential for batching with Groth16 on-chain verification, and it leads to at least $2\times$ improvement in gas cost for L1 batching.

Now, we focus on the gas cost for ZEBRA's L2 batching. The gas for L2 is reduced linearly with batch size and the reduction is up to $641\times$ for a batch of 512 users. For the same, the gas cost is just 561, which costs 0.0126 USD

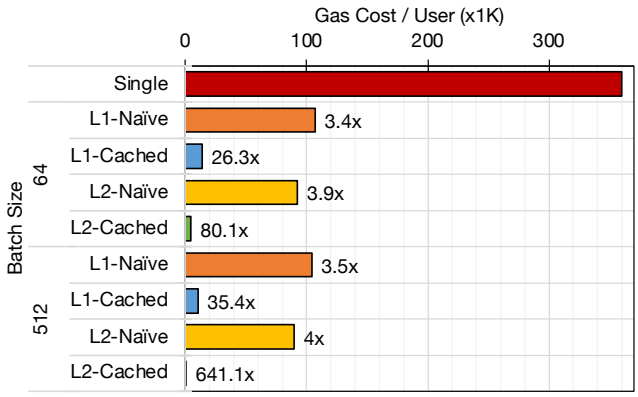


Figure 8: Gas cost per user comparison of single and batched verification of L1 and L2 wallets for batches of 64 and 512 users. The numbers next to the bars represent the improvement w.r.t. single verification.

and is comparable with the L2 transaction gas costs for ZkSync [130] and Loopring [85], the most cost-efficient L2 solutions [94]. Again, the benefit from batching is minimal without caching due to Groth16. However even ignoring Groth16 in this case, the gas cost is still as high as 5K from just posting the audit token on-chain. This is at least $9\times$ worse than ZEBRA, demonstrating that caching is crucial for batching L2 wallets.

Previously we assumed that audit tokens were already cached, and now we report the gas costs for (init) caching audit tokens. The gas cost for caching is just 62K and 13K for L1 and L2 wallets, respectively, given a batch of 512 users. This costs just 1.4 USD and 0.3 USD on Ethereum, respectively, which we believe is reasonable for any application as this is a one-time cost per user.

8.2.2. Aggregator Overhead

We first focus on aggregator overhead for L1 batching: Table 2 summarizes the aggregator runtimes for generating both proofs as well as the number of R1CS constraints they prove for a batch of 64 and 512 users. The aggregator runtime is just 98 seconds for 64 users, which already leads to a good amortization of gas costs. For a better amortization with 512 users, the total runtime is around 4 minutes, which is reasonable given credential verification is done once per epoch. Even though Spartan has a linear prover, the runtime doesn't scalar linearly with N due to imperfect parallelization in our implementation. The runtimes can be further improved with better parallelization of our Spartan implementation, and by replacing R1CS with plonk arithmetization as summarized in § 6.2.

The additional overhead for L2 batching on top of L1 is just a lookup into the rollup state and a signature verification. This is cheaper than a transfer, which costs around 30K constraints in Loopring [85] for a rollup state with 2^{32} accounts [60]. In contrast, the L1 batching requires around 334 K constraints per user. Clearly, the rollup computation doesn't affect the total runtime significantly, and the rollup server overhead for L2 batching is the same as

TABLE 2: Aggregator overhead for N users. Times are in seconds (s) and R1CS constraints are in millions (M).

SNARK	Metric	$N = 64$	$N = 512$
SPARTAN (data-parallel)	Time	31.67 s	116.42 s
	#Constraints	21.4 M	171 M
GROTH16	Time	66.79 s	117.18 s
	#Constraints	96 M	169 M
Total Aggregator Time		98.46 s	233.6 s

the aggregator overhead.

Now, we analyze the monetary cost of batched verification. The `m6i.32xlarge` we rented from AWS costs 1.3409 USD/hour (US East - Ohio). We can batch around 8K users an hour with this instance, 512 at a time. Thus, the compute cost incurred by the aggregator per user is just 0.00017 USD. This cost is negligible compared to the minimum price of gas cost per user we achieve which is 0.0126 USD, underscoring the significance of primarily minimizing gas costs.

8.2.3. Cost of Prior Recursion Approaches

We discussed why prior approaches to recursion are not suitable in our setting in § 5.2, and now we concretely justify our claim. For prior approaches, we consider number of constraints proven within Groth16 as representative of the aggregator cost, and compare that against the total constraints proven by ZEBRA's aggregator for $N = 512$. Note that this is a fair comparison because (i) we're comparing part of the baselines with our entire solution, and (ii) the prover time per constraint is almost the same for Spartan and Groth16 in our solution for $N = 512$ (Table 2).

First, we have the pairing-based approach that requires use of non-native arithmetic which introduces $\approx 1000\times$ overhead in R1CS [12]. As a result, this approach requires $\approx 2^{26} \cdot N$ constraints for N users, or $\approx 2^{35}$ constraints for $N = 512$. Second, the FRI-based approach has a recursion threshold of 2^{20} constraints for our credential verification circuit [101], resulting in 2^{29} constraints for $N = 512$. Finally, the cost of just the decider in accumulation-based recursion is $> 2^{13} \cdot n$ [13], where n is the number of constraints in each accumulated instance. In our setting, $n > 2^{18}$ (see Spartan constraints in Table 2), and thus, the decider cost is $> 2^{31}$ constraints.

In contrast, the total overhead in our solution is just $2^{28.34}$ constraints ($2^{27.33}$ within Groth16), which is at least $101\times$ and $6.3\times$ better than pairing and accumulation-based approach, respectively. Although FRI-based approach is comparable to ZEBRA in terms of aggregator overhead, it imposes an $11\times$ higher prover overhead on users (Figure 7 in [106]). Thus, prior approaches to recursion either increase aggregator overhead by $6.3\times$ or the user overhead by $11\times$.

9. Related Work

Anonymous Credentials. Following the initial work of Chaum [47], there has been a long line of work [36], [42], [31], [27], [104], [41], [17], [77], [73], [36], [63], [114], [49] with successively more efficient and expressive anonymous credentials that have been widely deployed in a number of real-world applications [97], [37], [10], [58]. Today, we have credentials that can be used a limited number of times [31], [38], [15], revoked [43], [39], [40], audited [42], delegated [16], [46], [24], [55], [35], updated [53], [25], and issued by a decentralized organization [109], [72], [67], [103]. Recent years have also witnessed a synergy between ACs and blockchains. ACs are being used in permissioned blockchains for identity management [107], [11], [26], and blockchains are being leveraged as a verifiable data registry to improve off-chain certificates [51], [95], [2], [102].

Private On-chain Access Control. Like ZEBRA, several concurrent works, namely, iden3 [1], Polygon ID [102] and Semaphore [3], propose a privacy-preserving on-chain access control solution using zk-SNARKs. However, unlike ZEBRA, they don't support auditability and batching, which are necessary for accountability and achieving practical verification costs on Ethereum, respectively.

Regulation on Blockchains. Polkadex [99] and concurrent work [98] address the problem of privacy-preserving KYC in DeFi using additional trust assumptions. In particular, they rely on a decentralized oracle to verify credentials rather than performing the verification on-chain. This approach reduces the integrity of their solution to the integrity of the oracle, which is a weaker guarantee compared to our solution. For instance, only recently, 600 million USD were lost from an exploit on the decentralized oracle of the Ethereum sidechain Ronin [88].

Espresso Systems's CAPE [113], [112] is a concurrent work that enables anonymous asset transfer on Ethereum with configurable policies per asset; one such policy is KYC credential verification. Unlike our solution, CAPE uses the UTxO model, and does not support batching and revocation.

Concordium [57] proposed another solution for incorporating regulation while preserving privacy through transaction anonymity and an identity management layer. Thus, their design is not compatible with existing widely-used permissionless blockchains. Moreover, unlike permissionless blockchains, all Concordium users have to perform identity verification before accessing the chain which limits adoption and decentralization, and unlike ZEBRA, the credentials are tied to their chain and can't be used across chains.

Other works include Azeroth [79] that provides privacy-preserving transactions with auditing. In a similar vein, there are works adding auditability and regulation to existing privacy-preserving cryptocurrencies like ZCash [68] and Monero [83], and central bank digital currencies (CBDCs) [125]. Finally, zkLedger [92] proposed privacy-preserving ledgers that can be used by banks to settle cross-organization transactions while also allowing third-party auditing.

10. Conclusion

We presented ZEBRA, an anonymous credential (AC) scheme that provides practical on-chain verification for

the first time with support for auditability and revocation. ZEBRA crucially relies on zk-SNARKs to reduce on-chain verification costs and to batch credential verifications. Use of SNARKs alone, however, leads to an inefficient AC scheme and we proposed several techniques to address these inefficiencies and achieve practicality in all aspects. The on-chain verification of ZEBRA credentials costs $11.8\times$ less compared to prior AC schemes, and with batched verification, we get even *further* improvements of up to $35\times$ and $641\times$ for verification of L1 and L2 wallets, respectively. Consequently, ZEBRA enables the first practical and privacy-preserving access control solution for permissionless blockchains, which implies the first practical solution for privacy-preserving KYC regulation in DeFi. In contrast, users of prior solutions either had to pay exorbitant fees or lose privacy.

Acknowledgements. We thank Weikeng Chen, Pratyush Mishra and Jens Ernstberger for their valuable advice, and Gonzalo Munilla Garrido, Vivek Nair, Julien Piet and SkyLab security students for their helpful feedback in improving the presentation of this paper. This work is supported by the Center for Responsible, Decentralized Intelligence at Berkeley (Berkeley RDI).

References

[1] "iden3 On-chain Verification Contracts," https://github.com/iden3/contracts/tree/master/contracts/validators.

[2] "iden3.io," https://iden3.io/.

[3] "Semaphore V2," https://semaphore.appliedzkp.org/.

[4] "Coin Metrics Network Chart," https://charts.coinmetrics.io/, 2022.

[5] "DappRadar Industry Overview," https://dappradar.com/, 2022.

[6] "Ethereum Gas Charts," https://ethereumprice.org/gas/, 2022.

[7] "Lukso," https://www.lukso.network/, 2022.

[8] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *ASIACRYPT*. Springer, 2016.

[9] I. Allison, "Aave's push for institutional defi gets second kyc provider proposal," *CoinDesk*, 2021.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*, 2018.

[11] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," ser. AFT '20. Association for Computing Machinery, 2020.

[12] arkworks rs, "nonnative," https://github.com/arkworks-rs/nonnative, 2021.

[13] ——, "r1cs-std," https://github.com/arkworks-rs/r1cs-std, 2021.

[14] ——, "arkworks-rs," https://github.com/arkworks-rs, 2022.

[15] F. Baldimtsi and A. Lysyanskaya, "Anonymous credentials light," in *CCS*. ACM, 2013.

[16] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *CRYPTO*. Springer, 2009.

[17] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "P-signatures and noninteractive anonymous credentials," in *TCC*. Springer, 2008.

[18] M. Bellare, "Lectures on nizks: A concrete security treatment," August 2021. [Online]. Available: https://cseweb.ucsd.edu/~mihir/cse208-Wi20/main.pdf

[19] M. Bellare, S. Meiklejohn, and S. Thomson, "Key-versatile signatures and applications: Rka, kdm and joint enc/sig," in *EUROCRYPT*. Springer, 2014.

[20] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*. IEEE, 2014.

[21] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *CRYPTO*. Springer, 2014.

[22] J. Benaloh and M. d. Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *EUROCRYPT*. Springer, 1993.

[23] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for SNARKS and proof-carrying data," in *STOC*. ACM, 2013.

[24] J. Blömer and J. Bobolz, "Delegatable attribute-based anonymous credentials from dynamically malleable signatures," in *ACNS*, ser. Lecture Notes in Computer Science. Springer, 2018.

[25] J. Blömer, J. Bobolz, D. Diemert, and F. Eidens, "Updatable anonymous credentials and applications to incentive systems," in *CCS*. ACM, 2019.

[26] D. Bogatov, A. D. Caro, K. Elkhiyaoui, and B. Tackmann, "Anonymous transactions with revocation and auditing in hyperledger fabric," *CANS*, 2021.

[27] D. Boneh and X. Boyen, "Short signatures without random oracles," in *EUROCRYPT*. Springer, 2004.

[28] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "gnark playground."

[29] S. Bowe, J. Grigg, and D. Hopwood, "Recursive proof composition without a trusted setup," Cryptology ePrint Archive, Report 2019/1021, 2019.

[30] P. Braendgaard, "Eip-1812: Ethereum verifiable claims," March 2019. [Online]. Available: https://github.com/ethereum/EIPs/pull/1812

[31] S. Brands and F. Légaré, "Digital identity management based on digital credentials," in *GI Jahrestagung*, ser. LNI. GI, 2002.

[32] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner, "Proof-carrying data without succinct arguments," in *CRYPTO*. Springer, 2021.

[33] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Recursive proof composition from accumulation schemes," in *TCC*. Springer, 2020.

[34] V. Buterin, "On-chain scaling to potentially 500 tx/sec through mass tx validation," 2018. [Online]. Available: https://ethresear.ch/t/on-chain-scaling-topotentially-500-tx-sec-through-mass-tx-validation

[35] J. Camenisch, M. Drijvers, and M. Dubovitskaya, "Practical uc-secure delegatable credentials with attributes and their application to blockchain," in *CCS*, 2017.

[36] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss, "Composable and modular anonymous credentials: Definitions and practical constructions," in *ASIACRYPT*. Springer, 2015.

[37] J. Camenisch and E. V. Herreweghen, "Design and implementation of the *idemix* anonymous credential system," in *CCS*. ACM, 2002.

[38] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich, "How to win the clonewars: efficient periodic n-times anonymous authentication," in *CCS*. ACM, 2006.

[39] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC*. Springer, 2009.

[40] ——, "Solving revocation with efficient update of anonymous credentials," in *SCN*. Springer, 2010.

[41] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *CRYPTO*. Springer.

[42] ——, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT*. Springer, 2001.

[43] ——, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO*. Springer, 2002.

[44] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*. IEEE, 2001.

[45] R. Canetti and S. Goldwasser, "An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack," in *EUROCRYPT*. Springer, 1999.

[46] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable signatures: New definitions and delegatable anonymous credentials," in *CSF*. IEEE, 2014.

[47] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Commun. ACM*, 1985.

[48] A. Chiesa and E. Tromer, "Proof-carrying data and hearsay arguments from signature cards," in *ICS*. Tsinghua University Press, 2010.

[49] A. Connolly, P. Lafourcade, and O. Perez-Kempner, "Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes," in *PKC*. Springer, 2022.

[50] ConsenSys, "gnark," https://github.com/ConsenSys/gnark, 2021.

[51] W. W. W. Consortium *et al.*, "Verifiable credentials data model v1.1," *W3C First Public Working Draft, https://www.w3.org/TR/vc-data-model/*, 2019.

[52] B. Cooper, "Announcing alkemi network & kyc-chain partnership," *Medium*, 2021.

[53] S. E. Coull, M. Green, and S. Hohenberger, "Controlling access to an oblivious database using stateful anonymous credentials," in *PKC*. Springer, 2009.

[54] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *CRYPTO*. Springer, 1998.

[55] E. C. Crites and A. Lysyanskaya, "Delegatable anonymous credentials from mercurial signatures," in *CT-RSA*. Springer, 2019.

[56] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees," in *NordSec*. Springer, 2016.

[57] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi, "Balancing privacy and accountability in blockchain identity management," in *CT-RSA*. Springer, 2021.

[58] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *PETS*, 2018.

[59] N. De, "State of crypto: Fatf's new guidance takes aim at defi," *CoinDesk*, 2021.

[60] B. Devos, "Loopring's zksnark prover optimizations," *Medium*, 2020.

[61] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs, "Efficient public-key cryptography in the presence of key leakage," in *ASIACRYPT*. Springer, 2010.

[62] B. Farmer, "Introducing plonky2," *Polygon Blog*, 2022.

[63] G. Fuchsbauer, C. Hanser, and D. Slamanig, "Structure-preserving signatures on equivalence classes and constant-size anonymous credentials," *J. Cryptol.*, 2019.

[64] A. Gabizon, Z. Williamson, and T. Walton-Pocock, "Aztec yellow paper," *hackmd*, 2021.

[65] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," Cryptology ePrint Archive, Paper 2020/315, 2020.

[66] 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.

[67] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," 2013.

[68] ——, "Accountable privacy for decentralized anonymous payments," in *Financial Cryptography*. Springer, 2016.

[69] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[70] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Storing and retrieving secrets on a blockchain," Cryptology ePrint Archive, Report 2020/504, 2020.

[71] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*. Springer, 2016.

[72] H. Halpin, "Nym credentials: Privacy-preserving decentralized identity with blockchains," in *CVCBT*. IEEE, 2020.

[73] L. Hanzlik and D. Slamanig, "With a little help from my friends: Constructing practical anonymous credentials," in *CCS*, 2021.

[74] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *annual international cryptology conference*. Springer, 1995, pp. 339–352.

[75] Y. E. Housni, "Benchmarking pairing-friendly elliptic curves libraries," *hackmd*, 2021.

[76] Y. E. Housni and A. Guillevic, "Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition," Cryptology ePrint Archive, Paper 2020/351, 2020.

[77] C. Hébant and D. Pointcheval, "Traceable constant-size multi-authority credentials," Cryptology ePrint Archive, Report 2020/657, 2020.

[78] iden3, "go-merkletree," https://github.com/iden3/go-merkletree, 2021.

[79] G. Jeong, N. Lee, J. Kim, and H. Oh, "Azeroth: Auditable zero-knowledge transactions in smart contracts," Cryptology ePrint Archive, Report 2022/211, 2022.

[80] A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, E. Shi *et al.*, "C ∅ c ∅: A framework for building composable zero-knowledge proofs," *Cryptology ePrint Archive, Report 2015/1093*, 2015.

[81] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," Cryptology ePrint Archive, Report 2021/370, 2021.

[82] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *International Conference on Applied Cryptography and Network Security*. Springer, 2007.

[83] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au, and D. Liu, "Traceable monero: Anonymous cryptocurrency with enhanced accountability," *IEEE Trans. Dependable Secur. Comput.*, 2021.

[84] I. Liusvaara and S. Josefsson, "Edwards-curve digital signature algorithm (eddsa)," *IETF*, 2017.

[85] Loopring, "Loopring," *URL: https://loopring.org/*, 2022.

[86] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *IEEE S&P*. IEEE, 2021.

[87] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "Churp: dynamic-committee proactive secret sharing," in *CCS*, 2019, pp. 2369–2386.

[88] M. McSweeney, "Axie infinity's ethereum sidechain ronin hit by $600 million exploit," *The Block Crypto*, 2022.

[89] MetisDAO, "Evm equivalence vs. evm compatibility," *Medium*, 2021.

[90] musalbas, "coconut-ethereum," https://github.com/musalbas/coconut-ethereum, 2018.

[91] R. Muth, T. Galal, J. Heiss, and F. Tschorsch, "Towards smart contract-based verification of anonymous credentials," Cryptology ePrint Archive, Report 2022/492, 2022.

[92] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," in *NSDI*. USENIX Association, 2018.

[93] B. Newar, "Aave launches its permissioned pool aave arc, with 30 institutions set to join," *Cointelegraph*, 2022.

[94] B. News, "Which layer 2 rollup for ethereum is the best?" *Medium*, 2022.

[95] N. Otto, S. Lee, B. Sletten, D. Burnett, M. Sporny, and K. Ebert, "Verifiable credentials use cases," *W3C First Public Working Draft, https://www.w3.org/TR/vc-use-cases/*, 2019.

[96] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators," in *USENIX*. USENIX Association, 2020.

[97] C. Paquin, "U-prove technology overview v1. 1," *Microsoft Corporation Draft Revision*, vol. 1, 2011.

[98] P. Pauwels, J. Pirovich, P. Braunz, and J. Deeb, "zkkyc in defi: An approach for implementing the zkkyc solution concept in decentralized finance," Cryptology ePrint Archive, Report 2022/321, 2022.

[99] Polkadex, "What is decentralized kyc and why polkadex is implementing it," *Medium*, 2022.

[100] C. Potti and P. Bhattacharya, "Eip-1261: Membership verification token," July 2018. [Online]. Available: https://github.com/ethereum/eips/issues/1261

[101] F. Protocol, "Fractal protocol," 2021. [Online]. Available: https://protocol.fractal.id/

[102] A. Radmilac, "A look at polygon id, a new zk-proof based web3 identity solution," *CryptoSlate*, 2022.

[103] M. Rosenberg, J. White, C. Garman, and I. Miers, "`zk-creds`: Flexible anonymous credentials from zksnarks and existing identity infrastructure," Cryptology ePrint Archive, Paper 2022/878, 2022.

[104] O. Sanders, "Efficient redactable signature and application to anonymous credentials," in *PKC*. Springer, 2020.

[105] scipr lab, "libsnark," https://github.com/scipr-lab/libsnark, 2020.

[106] S. T. V. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *CRYPTO*. Springer, 2020.

[107] W. Shao, C. Jia, Y. Xu, K. Qiu, Y. Gao, and Y. He, "Attrichain: Decentralized traceable anonymous identities in privacy-preserving permissioned blockchain," *Computers & Security*, 2020.

[108] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *EUROCRYPT*. Springer, 1998.

[109] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," in *NDSS*. The Internet Society, 2019.

[110] T. Suite, "Truffle suite - your ethereum swiss army knife." *URL: http://truffleframework.com/*, 2018.

[111] ——, "Ganache," https://github.com/trufflesuite/ganache, 2022.

[112] E. Systems, "Cape overview," https://docs.cape.tech/espresso-systems/cape/overview, 2022.

[113] ——, "Specification: Configurable asset privacy," *Github*, 2022.

[114] S. Tan and T. Groß, "Monipoly - an expressive q-sdh-based anonymous attribute-based credential system," in *ASIACRYPT*. Springer, 2020.

[115] P. Team, "The future is now for ethereum scaling: Introducing polygon zkevm," *Polygon Blog*, 2022.

[116] A. Thurman, "Aave proposal enlists fireblocks to aid defi protocol's mainstream finance push," *CoinDesk*, 2021.

[117] J. Torstensson, "Eip-780: Ethereum claims registry," November 2017. [Online]. Available: https://github.com/ethereum/EIPs/issues/780

[118] M. Tsuberi, B. Kaufman, A. Levi, and O. Sokolowsky, "Eip-1480: Access control standard," October 2018. [Online]. Available: https://github.com/ethereum/EIPs/issues/1481

[119] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *TCC*, ser. Lecture Notes in Computer Science. Springer, 2008.

[120] F. Vogelsteller, "Eip-735: Claim holder," October 2017. [Online]. Available: https://github.com/ethereum/eips/issues/735

[121] T. Walton-Pocock, "Plonk benchmarks i — 2.5x faster than groth16 on mimc," *Medium*, 2019.

[122] B. WhiteHat, M. Belles, and J. Baylina, "Eip-2494: Baby jubjub elliptic curve," https://github.com/ethereum/EIPs/pull/2494, Jan 2020.

[123] Z. Williamson and A. S. Cardozo, "Eip-1108: Reduce alt-bn128 precompile gas costs," https://github.com/ethereum/EIPs/pull/1108, May 2018.

[124] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[125] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun, "Prcash: Fast, private and regulated transactions for digital currencies," in *Financial Cryptography*. Springer, 2019.

[126] A. L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, and P. Camacho, "Veri-zexe: Decentralized private computation with universal setup," Cryptology ePrint Archive, Paper 2022/802, 2022.

[127] Y. Yu, Y. Zhao, Y. Li, X. Du, L. Wang, and M. Guizani, "Blockchain-based anonymous authentication with selective revocation for smart industrial applications," *IEEE Trans. Industr. Inform.*, 2019.

[128] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *CCS*, 2016.

[129] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls," in *CCS*. ACM, 2020.

[130] ZKSync, "Zksync," *URL: https://zksync.io/*, 2022.

Appendix

1. Extended Preliminaries

The security parameter is denoted by $\lambda \in \mathbb{N}$. A function $f : \mathbb{N} \to [0,1]$ is said to be negligible if for every $c \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $f(n) < n^{-c}$, and we write $\mathsf{negl}(\cdot)$ to denote such a function. A probability is *overwhelming* if it is equal to $1 - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\lambda)$. An algorithm $\mathcal{A}$ is PPT (probabilistic polynomial-time) if its running time is bounded by some polynomial in the size of its input. Given a distribution $\mathcal{D}$, we write $d \leftarrow \mathcal{D}$ to indicate that d is sampled according to $\mathcal{D}$. For two ensembles of random variables $\{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$, $\{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$, we write $\mathcal{D}_0 \approx_c \mathcal{D}_1$ to indicate that for all PPT $\mathcal{A}$, it holds that $\left|\Pr_{d\leftarrow\mathcal{D}_{0,\lambda}}[\mathcal{A}(d) = 1] - \Pr_{d\leftarrow\mathcal{D}_{1,\lambda}}[\mathcal{A}(d) = 1]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

The random oracle model. In the random oracle model (ROM), parties are given *oracle access* to some function H that is sampled uniformly at random from the space

of all functions $H : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are finite non-empty sets. Parties can query the oracle on an input $x \in \mathcal{X}$ and receive in return $H(x) \in \mathcal{Y}$. When proving security of a protocol Π in the random oracle model, the simulator Sim is able to "control" the oracle, observing queries made by the adversary and simulating responses.

Simulation-based security. We prove security via simulation, following the standard real/ideal world paradigm [69]. A cryptographic scheme specifies an interactive protocol Π that takes place between some parties $P_1, \ldots, P_n$ initialized with inputs $x_1, \ldots, x_n$. The protocol Π is meant to emulate some ideal functionality $\mathcal{F}$ that takes an input $x_1, \ldots, x_n$ from each party and delivers an output $y_1, \ldots, y_n$ to each party.

In the real execution, the protocol Π is executed in the presence of an adversary $\mathcal{A}$ that corrupts some subset $M \subset [n]$ of n parties. The honest parties $[n] \setminus M$ follow the instructions of Π, while $\mathcal{A}$ sends messages on behalf of parties in M. If $\mathcal{A}$ is *malicious*, these messages may be computed following an arbitrary polynomial-time strategy, while if $\mathcal{A}$ is *semi-honest*, these messages must be computed following the instructions of Π. The real execution of the protocol $\mathsf{REAL}_{\Pi,\mathcal{A}}[1^\lambda, \vec{x}, z, M]$, is defined as the output pair of honest parties and the adversary $\mathcal{A}$ from the real execution of Π. We also denote the view of party i during the execution of Π by $\mathsf{View}_i^\Pi(\vec{x})$, which consists of its input x_i, internal random coins r_i and messages received by party i during the execution.

In the ideal execution, a simulator Sim controlling some subset $M \subset [n]$ of n parties interacts with a trusted party $\mathcal{I}_\mathcal{F}$ implementing the functionality $\mathcal{F}$. Sim takes as input the security parameter 1^λ, a set of inputs $\{x_i\}_{i \in M}$, and an auxiliary input z. Each honest party $P_i \in [n] \setminus M$ sends their input x_i to $\mathcal{I}$, while Sim sends an input x_i' on behalf of each party $P_i \in M$. Let $x_1', \ldots, x_n'$ be the entire set of inputs received by $\mathcal{I}$. Next, $\mathcal{I}$ computes $(y_1, \ldots, y_n) = \mathcal{F}(x_1', \ldots, x_n')$ and delivers $\{y_i\}_{i \in M}$ to Sim. The ideal execution $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}[1^\lambda, \vec{x}, z, M]$, is defined as the output pair of the honest party and Sim from the above ideal execution.

We now present two notions of security. The first is the standard notion of simulation-based security against malicious parties. The second is a weaker notion that only guarantees privacy of honest parties' inputs. In particular, this does not guarantee correctness of an honest party's output against malicious parties who may tamper arbitrarily with the output. To rule out trivial protocols we demand that when all parties are honest, they obtain the correct output.

Definition 4. *An n-party protocol Π securely emulates an ideal functionality $\mathcal{F}$ in the presence of malicious (resp. semi-honest) adversaries corrupting a subset of parties $M \subset [n]$ if for any PPT malicious (resp. semi-honest) $\mathcal{A}$ corrupting parties M, there exists a PPT Sim such that for any set of inputs $\vec{x}$, and auxiliary input z, $\mathsf{REAL}_{\Pi,\mathcal{A}}[1^\lambda, \vec{x}, z, M] \approx_c \mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}[1^\lambda, \vec{x}, z, M]$.*

Definition 5. *An n-party protocol Π* privately *emulates an ideal functionality $\mathcal{F}$ if it is* correct *and if in the presence of malicious adversaries corrupting a subset of parties $M \subset [n]$ if for any PPT malicious (resp. semi-honest) $\mathcal{A}$ corrupting parties M, and every two series of inputs $X_1 = \{\vec{x}_\kappa^1\}$ and $X_2 = \{\vec{x}_\kappa^2\}$, $\{\mathsf{View}_\mathcal{A}^\Pi(\vec{x}_\kappa^1)\} \approx_c \{\mathsf{View}_\mathcal{A}^\Pi(\vec{x}_\kappa^2)\}$, where for every $\kappa \in \mathbb{N}$, all elements of $\vec{x}_\kappa^1$ and $\vec{x}_\kappa^1$ are equal in length.*

zk-SNARKs. Given a field $\mathbb{F}$, and an $\mathbb{F}$-arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ we denote a corresponding language by associated binary relation by $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l : C(a, x) = 0^l\}$ and a language $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists\, a \in \mathbb{F}^h, C(x, a) = 0^l\}$. A zk-SNARK for $\mathbb{F}$ is a triple of PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda, C) \to \mathsf{crs}$: Takes as input a security parameter and circuit description C, and outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: Takes as input crs and any pair $(x, w) \in \mathcal{R}_C$ and outputs a proof π for the statement $x \in \mathcal{L}_C$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: Takes as input crs, statement x and proof π and outputs a bit b with 1 indicating that verification passes.

A zk-SNARK satisfies *Completeness* which states that a proof computed from any $(x, w) \in \mathcal{R}_C$ will verify correctly with overwhelming probability. In addition it satisfies the following properties:

- *Proof of knowledge (and soundness).* For every PPT adversary $\mathcal{A}$, there is a PPT extractor ε such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$ and $(x, w) \notin \mathcal{R}_C$ with probability $\mathsf{negl}(\lambda)$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C)$, $(x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and $w \leftarrow \varepsilon(\mathsf{crs})$.
- *Perfect Zero-knowledge.* There exists a simulator Sim such that for all stateful distinguishers $\mathcal{A}$ the following probabilities are equal:

$$\Pr\left[\begin{array}{l|r} (x,w) \in \mathcal{R}_C & (\mathsf{crs}) \leftarrow \mathsf{Setup}(1^\lambda, C) \\ \mathcal{A}(\pi) = 1 \quad : & (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right],$$

$$\Pr\left[\begin{array}{l|r} (x,w) \in \mathcal{R}_C & (\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}(1^\lambda, C) \\ \mathcal{A}(\pi) = 1 \quad : & (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \pi \leftarrow \mathsf{Sim}(\mathsf{crs}, x, \tau) \end{array}\right]$$

Finally, a zk-SNARK also satisfies succinctness where an honestly-generated proof π has $O(1)$ bits and $\mathsf{Verify}(\mathsf{vk}, x, \pi)$ runs in time $O(|x|)$ up to a fixed polynomial factor in λ. If a proof system satisfies all the above properties except succinctness we refer to it as a Non-Interactive Argument of Knowledge (NIAoK).

Digital Signature. We use signature schemes that are existentially unforgeable under chosen message attacks. They consists of three algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, where $\mathsf{Gen}(1^\lambda)$ outputs a secret key sk and a public verification key vk, $\mathsf{Sign}(\mathsf{sk}, m)$ outputs a signature σ on the message m, and $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ outputs either 1 to indicate that σ is a valid signature on m, or 0 otherwise.

Definition 6. *A signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgeable under chosen message attacks if for any PPT adversary* $\mathcal{A}$*, the following probability is negligible*

$$\Pr\left[\begin{array}{l} m \notin Q \wedge \\ \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \end{array} : \begin{array}{r} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk}) \end{array}\right]$$

where Q *is the set of message queries that* $\mathcal{A}$ *makes to* $\mathsf{Sign}(\mathsf{sk}, \cdot)$.

Threshold Public-Key Encryption. We use a simulation based definition of adaptive CCA secure Threshold Public-Key Encryption (TPKE) as defined by Canetti and Goldwasser [45]. However, we restrict protocols Π_{TPKE} for TPKE to consist five PPT algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Verify}, \mathsf{Combine})$ as defined below:

- $\mathsf{Setup}(1^\kappa, n, t) \rightarrow \{\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\}$: Takes as input a security parameter and positive integers n, t and outputs a public, verification key and secret keys with threshold $t+1$.
- $\mathsf{Enc}(\mathsf{pk}, m; \rho) \rightarrow \mathsf{ct}$: Takes as input the public key pk, a message m and randomness ρ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{ct}, sk_i) \rightarrow m_i$: Takes as input a secret key and a ciphertext $\mathsf{Dec}(sk_i, \mathsf{ct})$ and outputs a partial decryption of the message m_i.
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, m_i) \rightarrow \{0, 1\}$: Takes as input the public key, verification key and a partial decryption of message and outputs $0/1$. If it outputs 1, we say the share is a valid decryption.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \{m_i\}_{i \in S \subseteq [n]}) \rightarrow m$ takes as input $t+1$ partial decryptions of the message and reconstructs m.

We require the above restriction as we create proofs of knowledge about ciphertexts in conjunction with proving predicates on the message which requires an algorithmic description of the protocol. We also demand perfect correctness of the above protocol where for all $0 < t \leq n$, $\{\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\} \leftarrow \mathsf{Setup}(1^\kappa, n, t)$,

- For any ciphertext c, if $m_i = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}_i, c)$, then

$$\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, c, m_i) = 1.$$

- If $c = \mathsf{Enc}(\mathsf{pk}, m)$, and $\{m_j\}_{j \in S}$ where $S \subset [n]$ is a $t+1$ sized subset such that $m_j = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}_j, c)$, then

$$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, \{m_j\}_{j \in S}) = m.$$

For a protocol Π_{TPKE} to be t-secure, it must emulate the following ideal functionality as described in Definition 4[13].

Canetti and Goldwasser show that the Cramer-Shoup cryptosystem [54] can be modified by having servers prove correctness of partial decryptions using zero knowledge proofs to achieve the above definition.

13. In the original paper the authors actually define and construct protocols satisfying the stronger Universally Composable notion of security [44]

$\mathcal{F}_{\mathsf{TPKE}}$

Parties: Encrypting user E and Servers $(S_1, \ldots, S_n)$.
Parameters: Space of receipts $\mathcal{C}$, number of servers n and threshold $0 < t \leq n$.

- **Setup.** Adversary specifies a distribution Γ over $\mathcal{C}$.
- **Encryption.** When E sends (Enc, m), sample a receipt $c \leftarrow \Gamma$ and store (c, m). Send c to E.
- **Decryption.** When $t+1$ servers send (Dec, c), if a tuple (c, m) has been stored, send with m to the servers. Else, send $\perp$ to the servers.

Figure 9: Ideal Functionality for a Threshold Public-Key Encryption scheme.

2. Proof of Security

We first focus on a static adversary $\mathcal{A}$ that corrupts fewer than a threshold number of auditors and arbitrary many clients. The decentralized organization is assumed to be semi-honest and does not collude with anyone.

We provide a proof sketch for the security of our scheme by describing a simulator S that interacts with the ideal functionality Figure 3 such that the transcript of $\mathcal{A}$ interacting with honest parties in the real world is computationally indistinguishable from the transcript produced by $\mathcal{A}$ when interacting with the simulator. We use the following policy:

- User must have been issued a credential by a CA that has not been revoked.
- When requesting verification for a pseudonym, the request must be authenticated by the pseudonym owner (possibly in the form of a signature).

2.1. Security against colluding Auditors and Users

Theorem 1. *The protocol in § 5 securely emulates the ideal functionality* $\mathcal{F}$ *(Figure 3) for any PPT malicious* $\mathcal{A}$ *corrupting a threshold number of auditors and an arbitrary number of users provided NIAoK and Public-Key Encryption schemes exist.*

The adversary begins by corrupting t auditors $\{\mathsf{Aud}_j\}_{j \in S}$ for $S \subset [n]$, $|S| = t$ along with an arbitrary number of users.

Setup. Sim runs the TPKE simulator $\mathsf{Sim}_{\mathsf{TPKE}}$ which outputs the public key and secret keys of malicious parties $\{\mathsf{pk}^{\mathsf{A}}, \{\mathsf{sk}_i^{\mathsf{A}}\}_{i \in S}\}$ for the TPKE scheme which are sent to $\mathcal{A}$. The simulator also runs and publishes $\mathsf{crs} \leftarrow \mathsf{NIAoK.Setup}(1^\lambda, .)$. Next, it simulates the CA by sampling $(\mathsf{vk}^{\mathsf{CA}}, \mathsf{sk}^{\mathsf{CA}}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ and publishing $\mathsf{vk}^{\mathsf{CA}}$.

Credential Generation. When Sim receives a request for a credential from a corrupt user U containing $(\mathsf{pk}^{\mathsf{U}}, \pi, \mathsf{doc})$ (Figure 4), it first sends $(\mathsf{ReqCred}, \mathsf{doc})$ to $\mathcal{F}$ on behalf of U. If Sim receives 1 as response from $\mathcal{F}$ indicating the credential was approved, and the proof of knowledge π verifies successfully, then Sim creates a credential using the secret key of simulated CA and sends it to U. Otherwise Sim sends $\perp$ to U. During this time Sim also creates a mapping from each corrupted user U to its

user-id id whenever a credential is issued.

Credential Verification. When a corrupt user U sends a proof π along with an audit token ψ and wallet address pk^{W}, Sim simulates SmrtCont by faithfully following the protocol. That is Sim issues a verification badge by adding pk^{W} to the public list of verified pseudonyms, if the proof provided by a user controlled by $\mathcal{A}$ passes verification.

Sim extracts id by running $\mathsf{Sim}_{\mathsf{TPKE}}$ on the audit token which outputs the underlying message id to be sent to $\mathcal{F}_{\mathsf{TPKE}}$. It then finds the corresponding user by checking the mapping it created earlier. As part of the proof, $\mathcal{A}$ also attaches a signature that verifies under pk^{W}. This is sufficient for Sim to prepare aux and send $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, aux)$ to $\mathcal{F}$.

When an honest party obtains a verification badge, Sim must simulate the view of $\mathcal{A}$ which contains a proof π and audit token ψ with a signature that verifies with public key pk^{W}. Since Sim simulates the CA, it also knows the secret key and can issue credentials on arbitrary public keys. Thus when the ideal functionality announces that a pseudonym pk^{W} has been verified, Sim samples a random public-key pair as $\mathsf{pk}^{\mathsf{U}} = \mathsf{PRF}^{\mathsf{adr}}(\mathsf{sk}^{\mathsf{U}}, 0)$, $\mathsf{sk}^{\mathsf{U}} \leftarrow \{0,1\}^{\lambda}$, a random user-id id and creates a signature on $\mathsf{pk}^{\mathsf{U}}||id||e$ as described in Figure 4. In addition, it runs the simulator of the TPKE scheme $\mathsf{Sim}_{\mathsf{TPKE}}$ to simulate the adversary's view of ciphertexts.

Audit. When an audit of a pseudonym pk^{W} occurs, Sim receives the user(s) that requested a verification badge from the ideal functionality. Recall that when simulating Credential Verification, Sim ran $\mathsf{Sim}_{\mathsf{TPKE}}$ to simulate the adversary's view of ciphertexts. For each audit token that is decrypted, Sim runs $\mathsf{Sim}_{\mathsf{TPKE}}$ with the corresponding honest party's user-id id as input and simulates the adversary's view of decryption such that decrypted message is id.

Revocation. When a user is banned they appear on a public list $\mathcal{L}_B$ maintained by the ideal functionality. When this happens, Sim updates the revocation list of CA by faitfully following the protocol and adding the user-id of the corresponding user as described in Figure 11.

Argument for successful simulation. In the initial hybrid, the adversary is in the real world interacting with honest parties. The next hybrid is identical except that Sim runs the Setup for the NIAoK instead of the trusted party. Since this is done exactly as done in the real execution, this hybrid is indistinguishable from the previous hybrid.

In the next hybrid, Sim simulates the CA as described during credential generation and revocation. This is indistinguishable from the previous hybrid because Sim approves/revokes a credential only when done by the honest CA and the rest of the simulation is carried out in a manner identical to the real execution.

Now Sim simulates the honest auditors by simply simulating the trusted party who distributes keys and then responding to audit requests as described in the protocol. This hybrid is identically distributed to the previous hybrid as the simulator samples the keys in manner identical to the trusted party and the simulated auditors follow the protocol faithfully.

Next, Sim simulates the honest parties and their verification requests by using the simulated CA's public key to generate a credential for a random user-id and public key and running $\mathsf{Sim}_{\mathsf{TPKE}}$ to simulate ciphertexts in the audit tokens. In a real execution a single honest party could have obtained verification for multiple pseudonyms. Whereas in the simulation, Sim samples a fresh public key for each pk^{W} that it receives from the ideal functionality and treats each verification as coming from a different honest user. However, when an audit of an honest party occurs, Sim runs $\mathsf{Sim}_{\mathsf{TPKE}}$ to ensure the decrypted message is consistent with the id received from the ideal functionality as described earlier. Due to the zero-knowledge property of the NIAoK, and the fact that $\mathsf{Sim}_{\mathsf{TPKE}}$ is a good simulator for the TPKE scheme emulating Figure 9, this hybrid is computationally indistinguishable from the previous hybrid.

Finally, Sim simulates SmrtCont. When a malicious user U attempts to obtain a verification badge, Sim receives a proof π, pseudonym pk^{W} and audit token ψ as described in Figure 5. Recall that the ideal functionality runs a credential verification policy before verifying a pseudonym. Sim now has to prepare a request $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, aux)$ where $\mathcal{C}(\mathsf{pk}^{\mathsf{W}}, aux) = 1$. It can be seen that the constraints in Figure 5 capture the verification policy outlined earlier, therefore if π passes verification, then the verification policy is satisfied. However, Sim must still determine the user who submitted this request and then submit the verification request on behalf of that user[14]. This can be determined by running $\mathsf{Sim}_{\mathsf{TPKE}}$ and decrypting the audit token to obtain the id of the party that created the request.

Let $\mathcal{K}_C$ denote the set of public keys for which $\mathcal{A}$ obtained a credential from CA, and $\mathcal{L}_H$ denote the tuples of pseudonym – user public keys pairs verified by the simulated honest parties. Note that a pseudonym can potentially appear multiple times in $\mathcal{L}_H$ in connection with (possibly repeated) public keys.

Claim 7. *For every verification request* $(\pi, \mathsf{pk}^{\mathsf{W}}, \psi)$ *submitted by* $\mathcal{A}$*, on input* ψ*,* $\mathsf{Sim}_{\mathsf{TPKE}}$ *either outputs a public key* $\mathsf{pk}^{\mathsf{U}} \in \mathcal{K}$ *or outputs* pk^{U} *such that* $(\mathsf{pk}^{\mathsf{W}}, \mathsf{pk}^{\mathsf{U}}) \in \mathcal{L}_H$.

Due to the proof of knowledge and soundness property of the NIAoK scheme, there exists an extractor that outputs the witness. From the EUF-CMA property, collision resistance of CRH, perfect correctness of the TPKE scheme and the security property of accumulators, the same id that results from decrypting the audit token must have been awarded a credential by a valid CA that has also not been revoked. Suppose $\mathsf{pk}^{\mathsf{U}} \notin \mathcal{K}$ and $(\mathsf{pk}^{\mathsf{W}}, \mathsf{pk}^{\mathsf{U}}) \notin \mathcal{L}_H$, then this implies that the extractor outputs a signature on $\mathsf{CRH}(\mathsf{pk}^{\mathsf{U}}, \mathsf{id}, .)$ for which CA has not issued a credential

14. Note that this user is not necessarily the same malicious user U who communicated with Sim as U' could have prepared a proof and U could have sent it on behalf of U'.

Suppose this was not true, then this implies that the extractor outputs two signatures σ, σ^{W} such that $\mathsf{SIG}^1.\mathsf{Verify}(\mathsf{vk}_g^{\mathsf{CA}}, \mathsf{CRH}(\mathsf{pk}^{\mathsf{U}}, \mathsf{id}, e), \sigma) = 1$ and $\mathsf{SIG}^2.\mathsf{Verify}(\mathsf{pk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}}, \sigma^{\mathsf{W}}) = 1$. Since $\mathsf{pk}^{\mathsf{U}} \notin \mathcal{K}$, there are two possibilities:

- $(\mathsf{pk}^{\mathsf{U}}, .) \notin \mathcal{L}_H$, in which case the adversary has produced SIG^1 on a pk^{U} which has never been signed by CA violating the EUF-CMA property.
- $(\mathsf{pk}^{\mathsf{U}}, .) \in \mathcal{L}_H$ but $(\mathsf{pk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}}) \notin \mathcal{L}_H$, in which case the adversary has produced SIG^2 on a wallet that verifies under a *simulated* honest party's public key, again, violating the EUF-CMA property.

2.2. Security against a semi-honest CA

Theorem 2. *The protocol in § 5 securely emulates the ideal functionality $\mathcal{F}$ (Figure 3) for any semi-honest PPT $\mathcal{A}$ corrupting the CA provided NIAoK and Public-Key Encryption schemes exist.*

We also guarantee security against a semi-honest CA, by constructing a simulator. Here Sim only needs to prepare the view of the CA for credential requests, which it can do by sampling a fresh public-key pair for each request that arrives from an honest user via the ideal functionality and then preparing a proof of knowledge of the secret key and attaching doc that was received. This is a good simulator as the public keys generated are indistinguishable from those generated by an honest party.

2.3. Privacy against colluding CA, Auditors and Users

Theorem 3. *The protocol in § 5 privately emulates the ideal functionality $\mathcal{F}$ (Figure 3) for any PPT malicious $\mathcal{A}$ corrupting the CA, a threshold number of auditors and an arbitrary number of users provided NIAoK and Public-Key Encryption schemes exist.*

Finally, we argue privacy (Definition 5) against an adversary $\mathcal{A}$ that corrupts the CA, up to t auditors and an arbitrary number of users. The view of the adversary consists of the public keys of honest parties on which credentials were issued, credential verification proofs sent by honest users to the smart contract along with the corresponding audit token. The CA has no input, the auditors have pseudonyms they wish to audit as input (since the functionality is reactive, the auditors can send their inputs over the course of the protocol) and users have pseudonyms they wish to obtain a verification badge for as input.

In all of our constructions we implicitly assume that a user can submit a credential verification request anonymously such that the adversary does not know which user submitted the request. This can be achieved through standard techniques of onion routing. If this were not the case, a CA could trivially map users to their pseudonyms.

Now, observe that the public key pair of each honest party is sampled uniformly at random, independent of the input. During credential verification, due to the zero-knowledge property of the NIAoK scheme, the proof π does not reveal any information about the credential being used cred and due to the CCA property of TPKE scheme, the adversary cannot distinguish between $\mathsf{TPKE.Enc}_{\mathsf{pk}}^{\mathsf{A}}(\mathsf{pk}_0^{\mathsf{W}})$ and $\mathsf{TPKE.Enc}_{\mathsf{pk}}^{\mathsf{A}}(\mathsf{pk}_1^{\mathsf{W}})$ for two different pseudonyms $\mathsf{pk}_0^{\mathsf{W}}, \mathsf{pk}_1^{\mathsf{W}}$. The above statement is not immediate as we use a simulation based definition (Appendix A). However, this was shown to imply the indistinguishability notion of Threshold Public-Key Encryption found in [108]. Thus for any two series of inputs $\{\vec{x}_\kappa^1\}$, $\{\vec{x}_\kappa^1\}$ which consist of pseudonyms to obtain verification badges for and pseudonyms that are to be audited $\{\mathsf{View}_{\mathcal{A}}^{\Pi}(\vec{x}_\kappa^1)\} \approx_c \{\mathsf{View}_{\mathcal{A}}^{\Pi}(\vec{x}_\kappa^2)\}$.

3. EUF-CMA from Weak-SE NIZKs + OWF

It is know that standard simulation extractability combined with one way functions can be used to create Strongly Unforgeable signatures under Chosen Message Attacks (SUF-CMA) [18], [61], [19]. In this section we show how to build an Existentially Unforgeable Signature scheme secure against Chosen Message Attacks (EUF-CMA) using a weak Simulation-Extractable NIAoK (weak-SE NIAoK) combined with One Way Functions. We begin by recalling the weak-SE property of NIAoKs from [80]. Here, the setup outputs an additional trapdoor τ which facilitates the creation of proofs for false statements. The definitions of knowledge soundness and zero knowledge in Appendix A are adapted accordingly.

Definition 8 (Weak Simulation-Extractability)**.** *A Non-Interactive Argument system is weak simulation extractable for a relation $\mathcal{R}$, if for every non-uniform PPT adversary $\mathcal{A}$ there exists a PPT extractor $\varepsilon_{\mathcal{A}}$ such that*

$$\Pr\left[\begin{array}{c|c} (\mathsf{crs}, \tau) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) & \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_{\mathsf{crs},\tau}}(\mathsf{crs}) & \wedge (x, w) \notin \mathcal{R} \\ w \leftarrow \varepsilon_{\mathcal{A}} & \wedge\, x \notin Q \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{S}_{\mathsf{crs},\tau}(x)$ is a simulator oracle that calls $\mathsf{Sim}(\mathsf{crs}, \tau, x)$ internally, and also records x in a list of queries Q.

In standard (strong) Simulation-Extractability the adversary must not be able to produce a new proof on a statement it has previously queried whereas here, the adversary needs to produce a proof on an entirely new statement that has not been previously queried.

Our construction of EUF-CMA signatures is identical to the construction of SUF-CMA signatures found in [18], except that we use weak-SE NIAoKs in place of SE NIAoKs. For completeness, we describe the scheme in Figure 10.

Theorem 9. *The protocol described in Figure 10 is an EUF-CMA signature scheme assuming the existence of weak-SE NIAoKs.*

Proof. To prove security of the above scheme we provide a reduction from an adversary $\mathcal{A}_{\mathsf{EUF}}$ that violates the EUF-CMA property of the NIAoK based signature scheme to an adversary $\mathcal{A}_{\mathsf{owf}}$ that inverts one way functions with non-negligible property. The EUF-CMA game between a challenger C and adversary $\mathcal{A}_{\mathsf{EUF}}$ is defined as follows:

NIAoK based EUF-CMA signature

Parameters: A weak-SE NIAoK NIAoK, a family of one-way functions $\mathsf{F} : \{0,1\}^{k(\lambda)} \times \{0,1\}^{d(\lambda)}$, message space $\mathcal{M}$ and relation $\mathcal{R} := \{((K, Y, m), \mathsf{sk}) \mid Y = F(K, \mathsf{sk})\}$.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$:
  - $K \leftarrow \{0,1\}^{k(\lambda)}$; $\mathsf{sk} \leftarrow \{0,1\}^{d(\lambda)}$; $Y \leftarrow F(K, \mathsf{sk})$.
  - $\mathsf{crs} \leftarrow \mathsf{NIAoK.Setup}(1^\lambda, \mathcal{R})$; $\mathsf{vk} \leftarrow (K, Y)$.
  - Return $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{vk}, \mathsf{sk}, m) \to \sigma$:
  - Return $\sigma \leftarrow \mathsf{NIAoK.Prove}(\mathsf{crs}, (K, Y, m), \mathsf{sk}))$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to \{0,1\}$:
  - Return $\mathsf{NIAoK.Verify}(\mathsf{crs}, (K, Y, m), \sigma)$.

Figure 10: EUF-CMA signature scheme from weak Simulation-Extractable NIZK and one way functions

1) C sends $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ to $\mathcal{A}_{\mathsf{EUF}}$.
2) $\mathcal{A}_{\mathsf{EUF}}$ asks C for signatures on message m_i.
3) C responds with $\sigma_i = \mathsf{Sign}(\mathsf{vk}, \mathsf{sk}, m_i)$ and adds m to the list of queries Q.
4) Repeat steps 2 and 3 as long as $\mathcal{A}_{\mathsf{EUF}}$ desires.
5) $\mathcal{A}_{\mathsf{EUF}}$ provides a final answer (m^*, σ^*).

$\mathcal{A}_{\mathsf{EUF}}$ wins the game if $m^* \notin Q$ and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$. The advantage of $\mathcal{A}_{\mathsf{EUF}}$ is defined as the probability that $\mathcal{A}_{\mathsf{EUF}}$ wins the game. For a protocol to be secure the advantage of every PPT adversary $\mathcal{A}_{\mathsf{EUF}}$ in the above game must be negligible.

Suppose there exists a PPT adversary $\mathcal{A}_{\mathsf{EUF}}$ that has non-negligible advantage ϵ in the EUF-CMA game when instantiated with the scheme in Figure 10, then we give below an adversary $\mathcal{A}_{\mathsf{owf}}$ that can successfully invert a one way function with probability negligibly close to ϵ.

1) C_{owf} samples a one way function and sends a challenge (K, Y) to $\mathcal{A}_{\mathsf{owf}}$ computed as $Y = F(K, X)$ where $X \leftarrow \{0,1\}^{d(\lambda)}$.
2) $\mathcal{A}_{\mathsf{owf}}$ generates $(\mathsf{crs}, \tau) \leftarrow \mathsf{NIAoK.Setup}(1^\lambda, \mathcal{R})$
3) $\mathcal{A}_{\mathsf{owf}}$ sets $\mathsf{vk} := (K, Y)$.
4) $\mathcal{A}_{\mathsf{owf}}$ internally runs $\mathcal{A}_{\mathsf{EUF}}$ with vk and crs computed as above.
5) $\mathcal{A}_{\mathsf{owf}}$ creates signatures as $\sigma \leftarrow \mathsf{NIAoK.Sim}(\mathsf{crs}, \tau, (K, Y, m))$ in response to queries made by $\mathcal{A}_{\mathsf{EUF}}$.
6) $\mathcal{A}_{\mathsf{EUF}}$ outputs (m^*, σ^*) where $m^* \notin Q$ and verification passes $\mathsf{NIAoK.Verify}(\mathsf{crs}, (K, Y, m), \sigma)$ with non-negligible probability.

Now $\mathcal{A}_{\mathsf{owf}}$ runs the extractor $\varepsilon_{\mathcal{A}_{\mathsf{owf}}}$ and obtains X^*. From the weak-SE property of the NIAoK, the probability that $((K, Y, m^*), X') \notin \mathcal{R}$ is negligible. Thus, $F(K, X') = Y$ with probability negligibly close to ϵ. Hence by contradiction, Figure 10 is a signature scheme satisfying EUF-CMA security. □

Credential Revocation

Input: $\cup_g \{\mathsf{id}_{g,i} = g \| u_i\}_i$

CA with group-id g (CA_g):

- $\forall i, \mathsf{MT}^{\mathsf{rl}}_g.\mathsf{Add}(\mathsf{id}_{g,i}, 1)$ and set $\mathsf{rt}^{\mathsf{rl}}_g := \mathsf{MT}^{\mathsf{rl}}_g.\mathsf{Root}$.
- Sign the revocation root and the current epoch E: $\sigma_{\mathsf{rv},g} := \mathsf{SIG}^1.\mathsf{Sign}(\mathsf{sk}^{\mathsf{CA}}_g, \mathsf{CRH}(\mathsf{rt}^{\mathsf{rl}}_g, E))$.
- Sign the revoked ids: $\sigma_{\mathsf{id},g} := \mathsf{SIG}^1.\mathsf{Sign}(\mathsf{sk}^{\mathsf{CA}}_g, \mathsf{CRH}(\{\mathsf{id}_{g,i}\}_i))$.

$\mathsf{CA}_g \to$ **Coordinator**: $\mathsf{rt}^{\mathsf{rl}}_g, \sigma_{\mathsf{rv},g}, \{\mathsf{id}_{g,i}\}_i, \sigma_{\mathsf{id},g}$

Coordinator:

- Let old roots be $\{\mathsf{rt}^{\mathsf{rl}}_{\mathsf{old},\mathsf{g}}\}_g$ and $\mathsf{rt}^{\mathsf{rr}}_{\mathsf{old}}$.
- $\forall g$, set $U[g] := 1$ if $\mathsf{rt}^{\mathsf{rl}}_{\mathsf{old},\mathsf{g}} \neq \mathsf{rt}^{\mathsf{rl}}_g$.
- $\forall g$, $\mathsf{MT}^{\mathsf{rr}}.\mathsf{Add}(g, \mathsf{rt}^{\mathsf{rl}}_g)$ and set $\mathsf{rt}^{\mathsf{rr}} := \mathsf{MT}^{\mathsf{rr}}.\mathsf{Root}$.
- Let $\mathsf{x} = (\mathsf{rt}^{\mathsf{vk}}, \mathsf{rt}^{\mathsf{rr}}, \mathsf{rt}^{\mathsf{rr}}_{\mathsf{old}}, E)$.
- Let $\mathsf{w} = (\{\mathsf{vk}^{\mathsf{CA}}_g, \mathsf{rt}^{\mathsf{rl}}_g, \mathsf{rt}^{\mathsf{rl}}_{\mathsf{old},\mathsf{g}}, \sigma_{\mathsf{rv},g}\}_g, U)$.
- Create proof $\pi := \mathsf{SNARK.Prove}(\mathsf{crs}_{\mathsf{rvc}}, \mathsf{x}, \mathsf{w})$, where rvc is defined as follows:

$$
\begin{aligned}
\mathsf{rvc} = \Big\{ \; & (\mathsf{x}, \mathsf{w}) \mid \\
& \forall g \in G : \\
& \quad \mathsf{MT}^{\mathsf{vk}}.\mathsf{Add}(g, \mathsf{vk}^{\mathsf{CA}}_g); \mathsf{MT}^{\mathsf{rr}}_{\mathsf{old}}.\mathsf{Add}(g, \mathsf{rt}^{\mathsf{rl}}_{\mathsf{old},\mathsf{g}}) \\
& \quad \text{if } U[g] = 1, \mathsf{rt}_g := \mathsf{rt}^{\mathsf{rl}}_g, \text{ else } \mathsf{rt}_g := \mathsf{rt}^{\mathsf{rl}}_{\mathsf{old},\mathsf{g}} \\
& \quad \mathsf{MT}^{\mathsf{rr}}.\mathsf{Add}(g, \mathsf{rt}_g) \\
& \quad \text{if } U[g] = 1 : \\
& \qquad \mathsf{SIG}^1.\mathsf{Verify}(\mathsf{vk}^{\mathsf{CA}}_g, \mathsf{CRH}(\mathsf{rt}^{\mathsf{rl}}_g, E), \sigma_{\mathsf{rv},g}) = 1 \\
& \wedge \mathsf{MT}^{\mathsf{vk}}.\mathsf{Root} = \mathsf{rt}^{\mathsf{vk}} \\
& \wedge \mathsf{MT}^{\mathsf{rr}}_{\mathsf{old}}.\mathsf{Root} = \mathsf{rt}^{\mathsf{rr}}_{\mathsf{old}} \\
& \wedge \mathsf{MT}^{\mathsf{rr}}.\mathsf{Root} = \mathsf{rt}^{\mathsf{rr}} \; \Big\}
\end{aligned}
$$

Coordinator $\to$ **Smart Contract** (periodically):

- $\pi, \mathsf{rt}^{\mathsf{rr}}, \forall g, (\mathsf{rt}^{\mathsf{rl}}_g, \sigma_{\mathsf{rv},g}, \{\mathsf{id}_{g,i}\}_i, \sigma_{\mathsf{id},g})$.

Smart Contract:

- If $\mathsf{SNARK.Verify}(\mathsf{crs}_{\mathsf{rvc}}, \mathsf{x}, \pi) = 1$:
  - Update revocation root from $\mathsf{rt}^{\mathsf{rr}}_{\mathsf{old}}$ to $\mathsf{rt}^{\mathsf{rr}}$.
  - Increment current epoch number E.

Figure 11: Credential Revocation Protocol

4. Optimized Multi-scalar Multiplication Constraints

A multi-scalar multiplication (MSM) is defined as follows: compute $g = \sum_{i \in [\ell]} s_i \cdot g_i$, where scalars $s_i \in \mathbb{F}$ are field elements and bases $g_i \in \mathbb{G}$ and result $g \in \mathbb{G}$ are group elements. The constraints for a multi-scalar multiplication (MSM) are naïvely implemented as follows: (i) the scalars are converted to bits (971ℓ constraints), and (ii) each base is independently multiplied by the corresponding scalar bits using the double-and-add algorithm and added to the result (2869ℓ constraints). We make changes to both steps to optimize the constraints.

First, if the scalars are computed by the verifier, then we precompute them and include their bit representation in the proof. Now, the verifier uses the scalars bits from the proof for the MSM and checks their consistency with the computed scalars (301ℓ constraints). Otherwise if the scalars are supplied by the proof itself, then we simply provide their

bit representation directly instead.

Second, we scalar-multiply all bases together. In each step of scalar-multiplication using double-and-add, the base is selectively added to the accumulator based on the scalar bit, and then the accumulator is doubled. Our optimized solution hoists the doubling step and performs it once per bit for all bases, which costs 1576ℓ constraints.

Overall, these two optimizations improve the constraints for MSM by $2.05\times$ from 3840ℓ to 1877ℓ.

5. Credential Revocation Evaluation

Credential revocation involves two entities: the CAs and the coordinator, and we discuss their performance overheads in this section. First, we look at the overhead on each CA to update its revocation list and post updates to the coordinator. The runtime to revoke 2^{10} credentials is just 110 ms with a single thread, and grows to 242 seconds for 2^{20} revoked credentials at a quasi-linear rate. Even with a single thread, the runtime is practical for the CA that has to do this once per epoch, and can be made much better with multi-threading. Note that the users also have to expend similar computation to update their local revocation tree before creating a non-revocation proof. For up to 2^{12} revoked IDs, it takes the user a minimal runtime of 440 ms to update its tree and create a non-revocation proof with a single thread, and requires a download of just around 20 KiB. We expect a significant improvement in runtimes for larger batches of revoked IDs with multithreading, and the communication for even 2^{20} revoked IDs is just around 5 MiB. In practice, we expect that most users will simply ask an untrusted service for the non-revocation path. As we discuss in § 5.1.2, this does not reveal any information that links the wallet to a user, and it is easy for the client to verify the non-revocation path w.r.t. the current root.

Next, we consider the performance overhead of the coordinator with 16 threads. The runtime and constraints of the coordinator grow linearly with the number of CAs, and it takes just 10 (2.6, resp.) seconds and around 2.8 M (0.69 M, resp.) constraints for 256 (64, resp.) CAs. Finally, we evaluate the gas cost the coordinator incurs to update the revocation root at the end of an epoch. The gas cost is just 271K for the SNARK verification, and an additional $3072G + 80R$ gas for storing signed revocation roots and revoked IDs on the blockchain with the current gas cost of 16 gas per Byte, where G is the number of CAs and R is the number of revoked IDs.