ZEBRA: SNARK-based Anonymous Credentials for Practical, Private and Accountable On-chain Access Control

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

Restricting access to certified users is not only desirable for many blockchain applications, it is also legally mandated for decentralized finance (DeFi) applications to counter malicious actors. Existing solutions, however, are either (i) non-private, i.e., they reveal the link between users and their wallets to the authority granting credentials, or (ii) they introduce additional trust assumptions by relying on a decentralized oracle to verify anonymous credentials (ACs).

To remove additional trust in the latter approach, we propose verifying credentials on-chain in this work. We find that this approach has impractical costs with prior AC schemes, and propose a new AC scheme ZEBRA that crucially relies on zkSNARKs to provide efficient on-chain verification for the first time. In addition to the standard unlinkability property that provides privacy for users, ZEBRA also supports auditability, revocation, traceability, and theft detection, which adds accountability for malicious users and convenience for honest users to our access control solution. Even with these properties, ZEBRA reduces the gas cost incurred on the Ethereum Virtual Machine (EVM) by 14.3× when compared to Coconut [NDSS 2019], the state-of-the-art AC scheme for blockchains that only provides unlinkability. This improvement translates to a reduction in transaction fees from 176 USD to 12 USD on Ethereum in May 2023. Since 12 USD is still high for most applications, ZEBRA *further* drives down credential verification costs through batched verification. For a batch of 512 layer-1 and layer-2 wallets, the transaction fee on Ethereum is reduced to just 0.44 USD and 0.02 USD, respectively, which is comparable to the minimum transaction costs on Ethereum.

Keywords

Anonymous Credentials, zkSNARKs, Permissionless Blockchains, Account-based Model, Accountability, Access Control, Decentralized Finance (DeFi), Know-your-customer (KYC), Decentralized Voting, Proof-of-personhood

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 permissionless access in blockchains promotes decentralization through wider participation, there are blockchain (or on-chain) applications where it is desirable to restrict access to users and wallets that satisfy a particular predicate. For instance, decentralized voting and private NFT drops require single access per person, proof-of-personhood allows a single wallet per person, certain applications only allow access to 18+ users or users from a certain region, etc. More importantly, there are on-chain applications where access control is not only desirable, it is also legally mandated to counter illicit activities. One such example is the recently proposed guidelines from the Financial Action Task Force (FATF) that require decentralized finance (DeFi) applications to restrict their service to KYC-verified users [60]. The urgency of these regulations is further underscored by the US Treasury's recent sanctioning of all wallets on Ethereum [146] that interacted with a decentralized mixer called Tornado Cash [89]. Hence, a large number of popular on-chain applications can not operate without access control.

Prior Access Control Solutions. Majority of on-chain applications are hosted on *account-based* blockchains like Ethereum [146], Avalanche [120], BNB [40], Algorand [2], etc., and that's the setting we'll focus on in the rest of this paper. The existing access-control solutions on such chains lie in one of the following two categories:

- *Non-private* solutions [3, 24, 51, 94, 103, 117, 136, 137, 139, 141]: a certificate authority (CA) verifies the user and issues an on-chain credential to a user nominated wallet, where the credential can be thought of as a signature from the CA attesting to a user's attributes or access privileges. This solution, 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.
- *Decentralized oracle*-based solutions [111, 112, 115]: the CA issues an anonymous credential (AC) to the user instead, 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. This solution preserves privacy but the credential verification is done using a small committee of nodes, a.k.a., a decentralized oracle, which introduces *additional trust assumptions*[1] and reduces the integrity and availability of the system to that of the oracle[2].

Our Approach: On-chain Credential Verification. The oracle-based approach solves the privacy issue using anonymous credentials (ACs) but it introduces additional trust in the system. We propose a modification to this approach which removes this limitation: instead of using a decentralized oracle, we verify the credential *on-chain.* As a result, the oracle is no longer needed and the verification is performed with the same integrity and availability guarantees as the underlying blockchain.

[1] Even if the CA itself acts as the oracle, that still implies additional trust because the CA is *additionally* required to ensure the integrity and availability of credential verification, which is a stronger assumption than simply furnishing credentials that can even be done offline. There is also a *usability* problem, where access request to every single on-chain application goes through the CA, severely limiting the organizations that can act as CAs.

[2] Only recently, 600 million USD were lost from an exploit on the decentralized oracle of the Ethereum sidechain Ronin [97].

Insufficiency of Existing Anonymous Credentials. To achieve minimal prover costs, prior AC schemes rely on lightweight Non-interactive Zero-Knowledge (NIZK) arguments such as Sigma protocols [56]. A downside of using these arguments is that their verifier complexity is the same as their prover complexity, i.e., it is *linear* in the size of the verification predicate, a quantity that grows with the number of attributes and the properties supported by the AC scheme. While this is not a problem in the traditional setting where verification is typically done by a single machine, credential verification becomes quite expensive in the context of on-chain verification, where verification is performed by all nodes of the blockchain. To make things worse, the state-of-the-art AC schemes [29, 30, 48, 66, 82, 114, 122, 128, 151] rely on bilinear group operations, some of which are not efficiently supported on Ethereum Virtual Machine (EVM) [147], the de-facto runtime environment for account-based blockchains [98]. Consequently, on-chain verification of Coconut [128], the state-of-the-art AC scheme for blockchains, costs 4.2M gas for the simplest predicate on EVM (§ 7.2.2), which is equivalent to 176 USD on Ethereum in May 2023 [63].

zkSNARK-based Anonymous Credentials. In this work, we achieve practical on-chain verification by building an AC scheme on top of zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs), which addresses both sources of inefficiency in existing AC schemes: SNARKs exist that can be verified efficiently on EVM with verifier cost *independent* of the verification predicate complexity. Although this approach sacrifices prover efficiency to gain verifier efficiency, due to the recent advancements in zkSNARKs, the prover costs are still practical (§ 7.2.3), especially in our setting where credential verification is only required once per application and wallet.

Using SNARKs already brings down the on-chain verification cost by 14.3× to 12 USD, but this is still quite expensive for most applications and especially for L2 wallets (see § 2.1) where the transaction fees are on the order of a couple cents. To this end, we *further* reduce the credential verification cost through batched verification. Our 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 (§ 7.3.3).
- 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 (§ 7.3.1).

To this end, we propose a novel EVM-compatible solution with practical aggregator and user overhead (§ 5.5) that reduces and even *removes* per-user costs from batched verification of L1 and L2 wallets, respectively (§ 5.6). Since EVM support is standard in account-based blockchains [98], this solution is not tied to Ethereum and is widely applicable.

Beyond Unlinkability. So far, we have only discussed the *unlinkability* property of AC schemes which provides privacy to users. Most applications, especially with legally mandated access control, also require the following properties:

- *Auditability*: to ensure accountability for misbehaving users, auditability is needed to enable authorized auditors to identify the owner of a maliciously behaving wallet.
- *Revocation*: credentials are often lost or stolen, and credentials of malicious users also need to be revoked.
- *Traceability*: when a credential is revoked, the wallets that were verified with it also need to be repudiated. Traceability allows auditors to identify all wallets verified with the revoked credential and then *selectively* repudiate them.

Although existing AC schemes can provide all these properties, the state-of-the-art solution for traceability [57] requires the use of secure multiparty computation (MPC) [150] among the auditors which (i) limits the distribution of trust, (ii) requires all parties to be online at the same time, and (iii) restricts the number of wallets that can be verified with a credential (see § 5.4 for details). To address these limitations, we propose a solution that leverages SNARKs to enable traceability without MPC, and only requires *minimal interaction* among the auditors (§ 5.4). Morevoer, we observe that our tracing scheme also offers *credential theft detection* (Appendix A), which allows a user to detect unauthorized usages of its credential.

Our final AC scheme ZEBRA[3] offers practical on-chain verification and supports auditability, revocation, traceability, and theft detection. Due to the use of SNARKs, introducing the above-mentioned properties only marginally increases the verifier cost of ZEBRA. In contrast, just adding revocation to Coconut increases its on-chain verification cost by 3× (§ 7.2.1).

ZEBRA is practical. The single verification protocol requires 294K gas on EVM or 12 USD on Ethereum (§ 7.2.2). With batched verification, the user pays a *one-time* cost of 56.5K (7.9K, resp.) gas to register a credential with its L1 (L2, resp.) wallet, and then for a batch of 512 wallets, each subsequent credential verification costs just 10.6K (545, resp.) gas (§ 7.3.1). For perspective, the average gas usage per transaction on Ethereum is around 93K [47] in May 2023, and the minimum transaction gas cost is 21K and 500 for L1 and L2 [154] wallets, respectively. Finally, the user takes 2 seconds to generate a credential verification proof on a 2019 Macbook Pro (§ 7.2.3), and the aggregator can batch verification of 512 credentials within 1.5 minutes (§ 7.3.2) on a 64-core machine. Note that the all of these costs only have to be paid once per wallet and application.

In summary, we make the following contributions:

- We propose a design for credential verification in account-based blockchains that balances privacy and accountability (§ 2) without introducing additional trust assumptions like prior work.
- We propose a new AC scheme ZEBRA that provides practical on-chain verification for the first time (§ 7.2), and supports auditability, revocation, and traceability (§ 5.1-§ 5.4). Our tracing scheme avoids the use of MPC and does not suffer from limitations of prior work.
- We propose a novel and EVM-compatible solution for batching ZEBRA credentials (§ 5.5 & § 5.6). Compared to our solution, directly using prior proof recursion works leads to 11× higher

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

gas cost (§ 7.3.1), and either 6.6× higher aggregator latency or 11× higher user overhead (§ 7.3.3).
- We benchmark ZEBRA and demonstrate its practicality (§ 7).

2 System Overview

In this section, we first explain relevant blockchain terminology (§ 2.1), and then discuss the various entities (§ 2.2) and protocols (§ 2.3) in our system. Here we describe the simplest instantiation of our system; extensions are discussed in Appendix B. Finally, we describe the threat model in § 2.4.

2.1 (Informal) Blockchain Terminology

- *Wallet*: a wallet is a public-private key pair $(\mathsf{pk}^{\mathsf{W}}, \mathsf{sk}^{\mathsf{W}})$ used by the user to interact with the blockchain, i.e., sign and send transactions to it. The wallet's public key pk^{W} is the user's pseudonym/account on the blockchain, and a user can have many such wallets or pseudonyms.
- *Smart Contract*: a program that is executed on the blockchain (computer) when a wallet sends a transaction to it.
- *Ethereum Virtual Machine (EVM)*: the virtual machine that executes a smart contract on the blockchain. EVM is turing-complete if there's enough gas (see below) and is not tied to Ethereum. It is the defacto VM for smart contracts [98] and is supported by most account-based blockchains [40–42, 64, 79, 101, 116, 120, 138, 143].
- *Gas*: the cost (measured in gas units) paid by a user to execute a smart contract on EVM. It includes the cost of posting the transaction data on-chain and executing it. The price of gas varies and is set by the blockchain nodes.
- *Layer-1 (L1) solution*: a solution that is implemented entirely on the layer-1, i.e,. the blockchain itself.
- *L1 wallet*: L1 processes transactions from this wallet.
- *Layer-2 (L2) solution*: a scaling solution that offloads computation and storage to a more scalable layer-2 network that is built on top of the layer-1 blockchain. Importantly, we only consider L2 solutions that have *data-availability* on L1 [127], i.e., the state managed by the L2 network can be reconstructed entirely using the transaction summaries posted on L1.

  We specifically focus on zk-rollups [28] in this work where the L2 network is simply an *untrusted server*, which executes batches of transactions and proves to an L1 contract that it has done so correctly using a SNARK proof. Note that we do not trust the rollup server for either computation (due to SNARK proof) or storage (due to data-availability).
- *L2 wallet*: L2 processes transactions from this wallet.

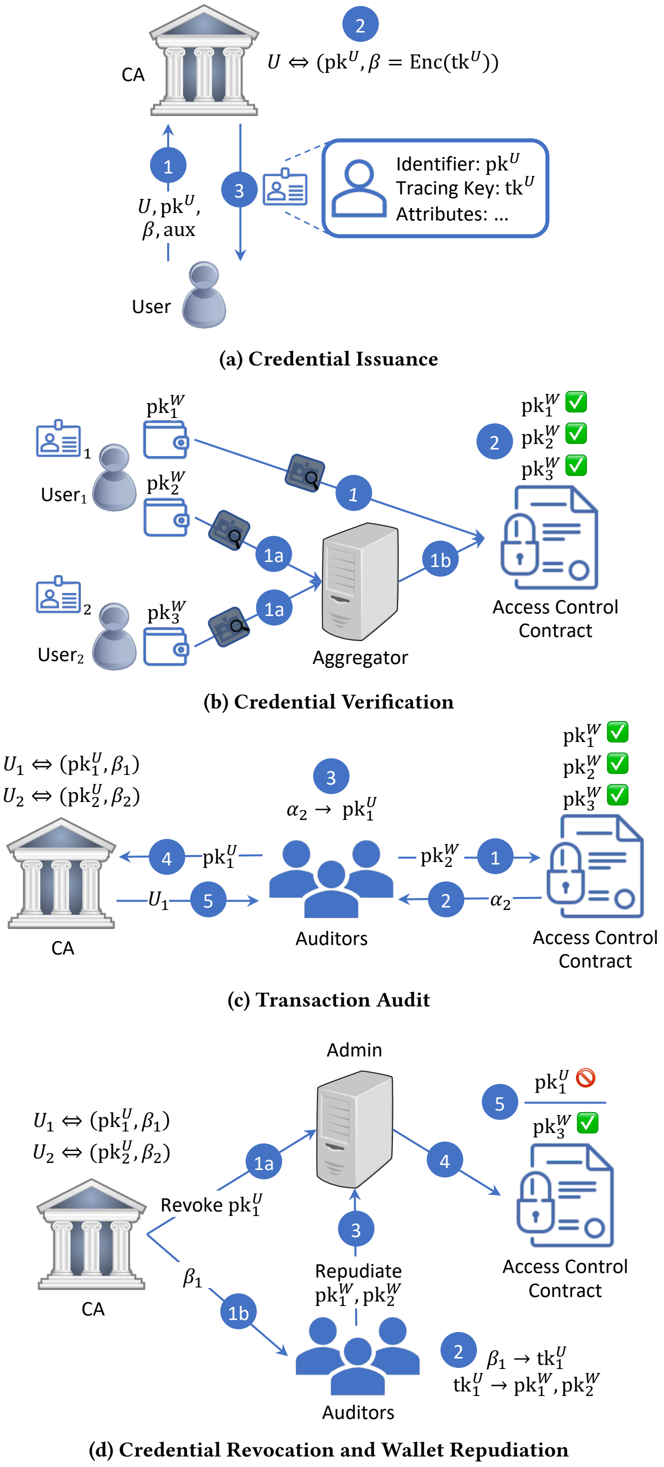
2.2 Entities

- *Access Control Contract*: a contract that grants access rights to wallets after verifying the validity of the credential used. It can be deployed as an L1 or L2 solution.
- *Application Contracts*: contracts that have the application logic and make queries to the access control contract to check if a wallet has appropriate access rights. They can be deployed on L1, or within the same L2 as the access control contract.
- *User*: a user owns multiple wallets and can use any of them to access applications provided it has the appropriate credential.
- *Certificate Authority (CA)*: the CA is an organization trusted to issue credentials with attributes to the user, and revoke them if needed. In Appendix B, we discuss how ZEBRA can be extended to support other issuance models with weaker trust assumptions.
- *Aggregator*: an *untrusted* party that batches credential verification transactions to reduce on-chain verification costs. For a rollup-based L2 solution, the rollup server is the aggregator.
- *Auditors*: a committee of n nodes that can do the following if $t+1$ of them agree to it: (i) audit a wallet to reveal the credential used to verify it, and (ii) track the usage of a credential to reveal the wallets verified using it. For simplicity, we assume a single chain and committee of auditors, and discuss the case where different chains have different sets of auditors in Appendix B.
- *System Admin*: the admin is a *logical entity* responsible for managing the access control contract, and its responsibilities include updating the revoked credential list and repudiating wallets. Since admin's actions are transparent and publicly verifiable, this role can be fulfilled by a *reputed but untrusted* organization, the CA, the auditors, a cryptographic proof, or any combination of them. In the definitions section (§ 4), we have the CA revoke the credentials and the auditors repudiate the wallets.

2.3 Protocols

In this section, we provide a high-level description of protocols in ZEBRA using the example summarized in Figure 1.

- *Credential Issuance* (Figure 1a): ① user sends the following to the CA: its identity U, its public key pk^{U}, linking token β which is a threshold encryption of its tracing key tk^{U}, and some auxiliary information doc justifying the issuance of credential. ② CA stores the mapping from U to $(\mathsf{pk}^{\mathsf{U}}, \beta)$. ③ CA sends a credential with rich attributes to the user with pk^{U} as the identifier and tk^{U} as the tracing key. Importantly, this protocol does not reveal the tracing key to the CA.
- *Credential Verification* (Figure 1b): ① user 1 sends a credential verification transaction directly to the contract using its wallet $\mathsf{pk}^{\mathsf{W}}_1$. (1a) & (1b) user 1 and 2 batch credential verification of their wallets $\mathsf{pk}^{\mathsf{W}}_2$ and $\mathsf{pk}^{\mathsf{W}}_3$, respectively, with the help of an aggregator to minimize costs. Figure 1b shows batched verification as a one-shot protocol for sake of exposition, but it actually consists of two phases: a registration phase where credentials are registered on-chain that is cheaper than single verification, and a verification phase, where arbitrary many predicates can be proven w.r.t. the registered credential at a minimal cost. ② the access control contract verifies the credential verification transactions and grants access to the respective wallets.
- *Transaction Audit* (Figure 1c): ① & ② auditors retrieve the audit token α_2 for wallet $\mathsf{pk}^{\mathsf{W}}_2$ from the blockchain. ③ auditors collaboratively decrypt α_2 to learn $\mathsf{pk}^{\mathsf{U}}_1$, i.e., identifier of the credential used to verify $\mathsf{pk}^{\mathsf{W}}_2$. ④ auditors send $\mathsf{pk}^{\mathsf{U}}_1$ to the CA, and ⑤ the CA sends user's identity U_1 to the auditors.
- *Credential Revocation and Wallet Repudiation* (Figure 1d): CA sends (1a) identifier $\mathsf{pk}^{\mathsf{U}}_1$ of the revoked credential to the admin, and (1b) the corresponding linking token β_1 to the auditors. ② auditors collaboratively decrypt β_2 to learn tracing key of the revoked credential $\mathsf{tk}^{\mathsf{U}}_1$. Using $\mathsf{tk}^{\mathsf{U}}_1$, the auditors track wallets verified with the revoked credential, i.e,. $\{\mathsf{pk}^{\mathsf{W}}_1, \mathsf{pk}^{\mathsf{W}}_2\}$. ③ auditors


CA
2
$U \Leftrightarrow (\mathrm{pk}^U, \beta = \mathrm{Enc}(\mathrm{tk}^U))$
1
$U, \mathrm{pk}^U,$
β, aux
3
Identifier: pk^U
Tracing Key: tk^U
Attributes: ...
User

(a) Credential Issuance

(b) Credential Verification

(c) Transaction Audit

Admin
$U_1 \Leftrightarrow (\mathrm{pk}_1^U, \beta_1)$
$U_2 \Leftrightarrow (\mathrm{pk}_2^U, \beta_2)$
1a
Revoke pk_1^U
CA
β_1
1b
3
Repudiate
$\mathrm{pk}_1^W, \mathrm{pk}_2^W$
Auditors
4
5
pk_1^U 🚫
pk_3^W ✓
Access Control
Contract
2
$\beta_1 \rightarrow \mathrm{tk}_1^U$
$\mathrm{tk}_1^U \rightarrow \mathrm{pk}_1^W, \mathrm{pk}_2^W$

(d) Credential Revocation and Wallet Repudiation

Figure 1: System Overview.

send a request to repudiate these wallets to the admin. ④ admin sends a transaction to the access control contract to revoke credential $\mathrm{pk}_1^{\mathrm{U}}$ and repudiate wallets $\{\mathrm{pk}_1^{\mathrm{W}}, \mathrm{pk}_2^{\mathrm{W}}\}$. ⑤ the access control contract verifies the transaction, revokes the credential and repudiates the wallets.

2.4 Threat Model

We make the following assumptions about the entities:

- *User*: users are malicious and can act arbitrarily to get verified without holding a valid credential.
- *Certificate Authority (CA)*: the CA is trusted for integrity since it is an authorized and reputed organization, but even if it behaves arbitrarily, we prove that it cannot deanonymize users. In Appendix B, we discuss other issuance models for ZEBRA that minimize trust in CA.
- *Aggregator*: the aggregator is completely untrusted. The only malicious way it can affect the protocol is by censoring user's transactions. Since our L1 aggregators are stateless, users can easily switch to a different aggregator. In case of a censoring rollup server (i.e., an L2 aggregator), censorship-resistance techniques from zk-rollups [93, 154] are applicable.
- *Auditors*: as long at most t out of n auditors collude, the privacy of honest users is preserved. Even if all auditors collude, they still can not impersonate or blame an honest user. Note that it is relatively easier to find a large committee of auditors to sufficiently distribute trust since the auditors are called upon infrequently and our protocol requires minimal interaction among the auditors.
- *System Admin*: the admin is *not trusted* to post correct updates and since its operation is transparent, its malicious behaviour can either be prevented through a cryptographic proof or deterred through legal action.

3 Preliminaries

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

Collision Resistant PRF. A collision resistant PRF, CR-PRF : $\{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ satisfies the standard pseudorandomness property and in addition is collision resistant where it is computationally infeasible to find $(K, x) \neq (K^*, x^*)$ such that $\mathsf{CR\text{-}PRF}(K, x) = \mathsf{CR\text{-}PRF}(K^*, x^*)$.

3.1 zk-SNARKs

A zk-SNARK is a tuple of three algorithms:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{crs}_\mathcal{R}$: On input security parameter λ and relation $\mathcal{R}$, outputs a common reference string $\mathsf{crs}_\mathcal{R}$.
- $\mathsf{Prove}(\mathsf{crs}_\mathcal{R}, x, w) \rightarrow \pi$: On input $\mathsf{crs}_\mathcal{R}$ and a statement-witness pair $(x, w) \in \mathcal{R}$, outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}_\mathcal{R}, x, \pi) \rightarrow \{0,1\}$: 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 [76] and write zkSNARK when we demand perfect completeness, perfect zero-knowledge, and computational knowledge soundness from the argument system. We will sometimes use zkNIAoK to refer to zero-knowledge non-interactive arguments of knowledge where we do not demand succinctness. We will also demand that the proof is straight-line extractable [110]. This means that the extraction algorithm can, on input a valid proof and the list of random-oracle queries made by the adversary (prover), extract a witness with overwhelming probability without rewinding the prover.

We also use simdSNARK to denote a "data-parallel" SNARK [135] which takes as input multiple statement-witness pairs and outputs a single succinct proof π for all statements. Importantly, we relax the security definition here and require all the above properties except for zero-knowledge.

3.2 Digital Signature

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

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

3.3 Threshold Public-Key Encryption

We use threshold public key encryption (TPKE) satisfying the simulation based IND-CCA2 security notion defined in [39]. We also restrict the syntax of TPKE to consist of five algorithms:

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

3.4 Sparse Merkle Tree

Sparse Merkle trees [55] 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.
- $\mathsf{MProve}(k) \rightarrow P$: outputs a membership proof for key k.
- $\mathsf{MVerify}(\mathsf{rt}, k, v, P) \rightarrow \{0, 1\}$: 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 AC 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 a pseudonymous 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 other issuance models discussed in Appendix B that minimize trust.

In our concrete instantiation we use the following policy: *User must have been issued a credential by the CA and the credential must not been revoked.*

4.1 Security

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

- (Theorem 3) Simulation-based security against a fully malicious adversary corrupting up to t auditors and any number of users.
- (Theorem 4) Simulation-based security against a semi-honest CA that does not collude with any other party.
- (Theorem 5) 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 correctness guarantees for an honest party's output but ensure that the adversary does not learn any information about an honest party's inputs. We do this by introducing a weakened ideal functionality $\hat{\mathcal{F}}$ (Figure 8) that captures the privacy guarantees when the CA is malicious.

In Figure 2 we describe the ideal functionality involving a certificate authority CA, 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 first sending a message $(\mathsf{ReqCred}, \mathsf{doc}, \mathsf{tk}^\mathsf{U}, \mathsf{attr})$ to $\mathcal{F}$, where doc *justifies* the issue of a credential, tk^U is a tracing key (explained later), and attr are additional attributes in the credential. (U, doc) is then forwarded to the CA who decides whether to approve/reject a request. If the request is approved, the user U, tk^U, and attr are added to a list $\mathcal{L}$. Users interact with access controlled applications through pseudonyms which have been *verified.* To verify a pseudonym pk^W, a user U sends a nonce η. If the pseudonym has not been verified before, there exists $(U, \mathsf{tk}^\mathsf{U}, \mathsf{attr}) \in \mathcal{L}$, and the request satisfies the verification policy $\phi(\mathsf{attr}) = 1$, then the pseudonym is verified by adding $(\mathsf{pk}^\mathsf{W}, U)$ to a key-value database $\mathcal{D}$ where the keys (pseudonyms) are made public but the values (users) are hidden. The ideal functionality also publishes $(\eta, \mathsf{CR\text{-}PRF}(\mathsf{tk}^\mathsf{U}, \eta))$.

Finally, there are four main *maintenance* operations provided by the ideal functionality to read from the database $\mathcal{D}$ and handle updates after reports of suspicious behaviour, malicious users or credential theft amongst other things.

- Audit: Carried out by auditors, to identify the user who verified a particular pseudonym pk^W.
- Trace: Carried out by auditors in collaboration with CA to identify the list of all wallets verified by a particular user U.

- Revoke: Carried out by CA to revoke the credential awarded to a particular user U.
- Repudiate: Carried out by auditors to repudiate the verification of a pseudonym that has previously been verified. Pseudonyms can no longer be used after repudiation.

The ideal functionality (Figure 2) captures the desired notions of unforgeability of credentials, unlinkability of credential showings, and consistency of credentials. Our ideal functionality evaluates a cryptographic function PRF but this was unavoidable due to subtle issues in the proof. In particular PRFs are not programmable and hence the simulator cannot simulate audit tokens of honest parties. However, it can be seen that despite revealing this information, unlinkability is not affected as the output of a CR-PRF is pseudorandom when the secret key is hidden.

$\mathcal{F}$

Parties: Users $\{U_1, \ldots, U_N\}$, Certificate Authority CA and Auditors $(\mathsf{Aud}_1, \ldots, \mathsf{Aud}_n)$.

Parameters: Verification policy ϕ, pseudonym space $\mathcal{N}$, user space $\mathcal{U}$, and a collision resistant PRF CR-PRF.

- **Setup.** On input (Setup, ϕ) from the organization, publish a verification policy ϕ. Maintain a database $\mathcal{D}$ of tuples from the space $\mathcal{N} \times \mathcal{U}$, and a database of banned pseudonyms $\mathcal{D}_B$.
- **Credential Generation.** On input $(\mathsf{ReqCred}, \mathsf{doc}, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr})$ from user U,
  - If U has previously been issued a credential, return $\perp$.
  - Else, forward $(\mathsf{doc}, U, \mathsf{attr})$ to CA. On input $(\mathsf{AprCred}, U)$ from CA, add $(U, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr})$ to $\mathcal{L}$, the tuple of users who have been awarded credentials, their tracing key and their attributes. Send accept to U.
- **Pseudonym Verification.** On input $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, \eta)$ from user U,
  - If $(U, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr}) \in \mathcal{L}$, $\phi(\mathsf{attr}) = 1$, $(\mathsf{pk}^{\mathsf{W}}, \cdot) \notin \mathcal{D}$, and $\mathsf{pk}^{\mathsf{W}} \notin \mathcal{D}_B$, then publish $(\mathsf{pk}^{\mathsf{W}}, \eta, \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta))$ and add $(\mathsf{pk}^{\mathsf{W}}, U)$ to $\mathcal{D}$.
  - Else, return $\perp$ to U.
- **Audit.** On input $(\mathsf{Audit}, \mathsf{pk}^{\mathsf{W}})$ from $t+1$ auditors and CA,
  - If there exists an entry of the form $(\mathsf{pk}^{\mathsf{W}}, U) \in \mathcal{D}$, return U to all auditors.
  - If pk^{W} does not appear in $\mathcal{D}$, then return $\perp$ to auditors.
- **Trace.** On input (Trace, U) from $t+1$ auditors and CA,
  - If $(U, \mathsf{tk}^{\mathsf{U}}, \cdot) \in \mathcal{L}$, send tk^{U} to auditors.
  - If $(U, \cdot, \cdot) \notin \mathcal{L}$, then return $\perp$ to auditors.
- **Revoke.** On input (Revoke, U) from CA,
  - If $(U, \cdot, \cdot) \in \mathcal{L}$, remove the entry containing U from $\mathcal{L}$.
  - Else, return $\perp$ to CA.
- **Repudiate.** On input $(\mathsf{Repudiate}, \mathsf{pk}^{\mathsf{W}})$ from $t+1$ auditors, delete all records of the form $(\mathsf{pk}^{\mathsf{W}}, \cdot)$ from $\mathcal{D}$, and add pk^{W} to $\mathcal{D}_B$.

Figure 2: Ideal Functionality for an AC Scheme supporting Audits, Tracing, Revocation and Repudiation.

5 ZEBRA: Our Anonymous Credential Scheme

In this section, we present our anonymous credential scheme ZEBRA. We first start with a simple scheme that only provides privacy and then incrementally improve it to add *auditability*, *revocation* and *traceability*. In § 5.1-§ 5.4, we'll focus on adding functionality, and in § 5.5 & § 5.6, we'll improve the efficiency of the scheme. We defer the discussion on credential theft detection to Appendix A, and finally, in Appendix B, we discuss potential extensions to ZEBRA.

5.1 A Simple AC Scheme with Privacy

Let $(\mathsf{sk}^{\mathsf{CA}}, \mathsf{pk}^{\mathsf{CA}})$ and $(\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{U}})$ denote the signature key-pairs of CA and user U, respectively. Using zkSNARKs, an AC scheme that provides unlinkability is straightforward to construct: to issue a credential $\mathsf{cred}^{\mathsf{U}}$ with attributes attr for user U, the CA simply signs the user's public key and its attributes, i.e,. $\mathsf{cred}^{\mathsf{U}} = \mathsf{SIG.Sign}(\mathsf{pk}^{\mathsf{U}} \| \mathsf{attr})$. Now, user U can use $\mathsf{cred}^{\mathsf{U}}$ to acquire access privileges for its wallet pk^{W} in a privacy-preserving way by producing a zkSNARK proof π for the following statement:

"*Given CA's public key* $\mathsf{pk}^{\mathsf{CA}}$*, wallet address* pk^{W}*, and access-control predicate* ϕ*, I know credential* $\mathsf{cred}^{\mathsf{U}}$ *with attributes* attr *for public key* pk^{U}*, and signature* σ *such that*:

- $\mathsf{cred}^{\mathsf{U}}$ is a valid credential $\Leftrightarrow$ $\mathsf{cred}^{\mathsf{U}}$ is a valid signature w.r.t. $\mathsf{pk}^{\mathsf{CA}}$ on pk^{U} and attr: $\mathsf{SIG.Verify}(\mathsf{pk}^{\mathsf{CA}}, \mathsf{pk}^{\mathsf{U}} \| \mathsf{attr}, \mathsf{cred}^{\mathsf{U}}) = 1$.
- pk^{W} is *backed* by the owner of $\mathsf{cred}^{\mathsf{U}}$ $\Leftrightarrow$ σ is a valid signature w.r.t. pk^{U} on pk^{W}: $\mathsf{SIG.Verify}(\mathsf{pk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}}, \sigma) = 1$.
- Credential attributes satisfy the predicate: $\phi(\mathsf{attr}) = 1$."

Note that the public part of this statement has no information about the user's identity. The wallet pk^{W} then signs the proof π and posts it for verification on-chain to get access privileges.

5.2 Adding Auditability

In the scheme described in § 5.1, users get privacy because credential verification reveals no information about user's identity, but this also means there is no accountability in the system. In this section, we add *auditability* to our scheme to deter malicious behaviour, which allows identifying the owner of a misbehaving wallet. We take the standard approach [35] of adding an encryption of the user's identity pk^{U} as part of credential verification, which can only be decrypted by the committee of auditors if at least $t+1$-out-of-n auditors agree to decrypt. In particular, the user now creates a zkSNARK proof for the following statement during credential verification:

"*Given* $\mathsf{pk}^{\mathsf{CA}}$*,* pk^{W}*,* ϕ*, auditors' public key* pk^{A}*, and audit token* α*, I know* $\mathsf{cred}^{\mathsf{U}}$*,* attr*,* pk^{U}*,* σ*, and encryption randomness* ρ *such that*:

- All checks from § 5.1.
- α is a well-formed encryption of pk^{U} under pk^{A} using randomness ρ: $\alpha = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{pk}^{\mathsf{U}}; \rho)$."

Since the audit token α is also posted on-chain by pk^{W} during credential verification, the auditors can collaboratively learn the user's public key, and from there its identity with the help of CA.

5.3 Adding Revocation

Consider the following straightforward revocation strategy [36]: the CA maintains a list of revoked credentials on-chain, and in each credential verification, the user is also required to prove that its credential does not lie on the revocation list. Specifically, the revocation list stores a unique identifier for each revoked credential and is maintained in the form of an authenticated dictionary [16, 36, 55, 58, 90, 105, 108] that supports efficient proof of non-membership.

The unique identifier could be the credential owner's public key pk^{U}, or an attribute which is unique to each credential, whichever one has the smaller domain.

Concretely, we use a Sparse Merkle Tree (SMT) [55] to maintain the list of revoked credentials since it supports efficient updates and our AC scheme only requires non-membership proof for a single element. SMTs are well-suited for this use-case, and with SMTs, the proof of non-membership is already not a bottleneck in our AC scheme. With revocation, the zkSNARK proof for credential verification is now required to prove the following statement:

"*Given* $\mathsf{pk}^{\mathsf{CA}}$, pk^{W}, ϕ, pk^{A}, α, *and revocation list Merkle root* $\mathsf{rt}^{\mathsf{rl}}$, *I know* $\mathsf{cred}^{\mathsf{U}}$, attr, pk^{U}, σ, ρ, *and Merkle non-membership proof* P^{rl} *for* pk^{U} *such that*:

- All checks from § 5.1 and § 5.2.
- $\mathsf{cred}^{\mathsf{U}}$ is not revoked $\Leftrightarrow$ pk^{U} is *not* a member of Merkle tree with root $\mathsf{rt}^{\mathsf{rl}}$: $\mathsf{MT.NMVerify}(\mathsf{rt}^{\mathsf{rl}}, \mathsf{pk}^{\mathsf{U}}, P^{\mathsf{rl}}) = 1$."

While the above solution prevents users whose credentials have been revoked from verifying new wallets, they may already (and very likely) own wallets which have already been verified. A straw-man solution is to repudiate all wallets, and have everyone *re-verify* their wallets every time a credential is revoked. But this leads to a system that imposes penalties on honest users who pay a price (literally) every time a user misbehaves and makes the system susceptible to denial of service attacks. In the next section, we introduce *traceability*, which enables *efficient* repudiation of wallets verified using a credential that has since been revoked.

5.4 Adding Traceability

As discussed in the previous section, it is desirable to have the ability to identify all wallets that were verified using a particular credential. Unlike the straw-man solution in § 5.3, when the malicious user's credential is revoked, we will then be able to identify all wallets associated with the revoked credential and *selectively* repudiate their access. Crucially, this allows honest users to continue using the system *completely unaffected.*

Our starting point is the tracing protocol of [57] where users provide the CA with a linking token β during credential generation, which is basically an encryption of their secret key sk^{U} under the auditors' public key using a threshold public-key encryption scheme, i.e,. $\beta = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{sk}^{\mathsf{U}})$. When verifying wallets, the user also posts a tracing token $\gamma := \mathsf{CR\text{-}PRF}(\mathsf{sk}^{\mathsf{U}}, \eta)$ computed as the output of a collision resistant PRF on a nonce $\eta \in [0, \tau]$, where τ is a limit on the credential usage. To trace all wallets verified by a user using sk^{U}, any $t+1$ auditors can use secure multiparty computation (MPC) to first decrypt β and then compute $\Gamma = \{\gamma_i = \mathsf{CR\text{-}PRF}(\mathsf{sk}^{\mathsf{U}}, i)\}_{i \in [0,\tau]}$. The auditors now search for matches with the posted tracing tokens to identify all wallets verified using the corresponding key sk^{U}.

An astute reader may wonder why an expensive MPC that limits the distribution of trust is needed, when instead the auditors can *non-interactively* decrypt β to first recover sk^{U} by using an appropriate threshold encryption scheme [39] and then locally evaluate the PRF to obtain the tracing tokens. The use of MPC also requires a threshold number of auditors to be online at the same time and the communication is proportional to the credential usage limit τ, thereby limiting the number of wallets a user can verify as a large number would be infeasible with MPC.

There are two main reasons to avoid revealing sk^{U} in the clear to auditors: (i) a *rushing* malicious auditor can potentially impersonate the user in the time it takes for the auditors to revoke the credential, and (ii) a malicious auditor can query points outside the limited set of the weakly-robust[4] Dodis-Yampolskiy PRF [62], in which case, we can no longer invoke the guarantees of the weakly-robust PRFs in the security reduction.

To address the first issue, we use a separate *tracing key* tk^{U}, independent of sk^{U}, to compute the linking and tracing tokens. Specifically, the linking token β sent by the user to the CA during credential generation becomes $\beta = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{tk}^{\mathsf{U}})$, and the tracing token γ posted during credential verification becomes $\gamma := \eta \| \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta)$. Importantly, knowledge of tk^{U} by itself does not grant ownership of a credential, and thus, can not be used to impersonate users. To bind the user to a specific tk^{U} without revealing it to the CA, we introduce a hiding commitment to tk^{U} as an attribute to each credential.

We resolve the second issue by using a stronger, collision-resistant PRF (CR-PRF) (see § 3). We can do so because we use a general-purpose proof system to prove well-formedness of the tracing token, and hence, are not restricted to using the Dodis-Yampolskiy PRF. Importantly, the use of a CR-PRF also allows us to not enforce any limitation on η except for the fact that it must be a field element.

To trace wallets verified by a user, any $t+1$ auditors can now easily decrypt β to recover tk^{U} and then locally compute the set of wallets verified by this user as $\{\mathsf{pk}^{\mathsf{W}} \mid \tau = \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta) \wedge (\mathsf{pk}^{\mathsf{W}}, \cdot, \gamma = \eta \| \tau) \in \mathcal{T}\}$, where $\mathcal{T}$ denotes the set of wallet verification transactions. Indeed the work done to trace depends linearly on the number of wallets but this only involves symmetric key operations and even for 100 million wallets, it is estimated to take approximately 15 seconds with a single thread. We will see in Appendix A that this also enables credential theft detection without compromising ownership of one's credential.

Overall, with traceability, the credential verification zkSNARK proof statement is as follows:

"*Given* $\mathsf{pk}^{\mathsf{CA}}$, pk^{W}, ϕ, pk^{A}, α, $\mathsf{rt}^{\mathsf{rl}}$, *and tracing token* γ, *I know* $\mathsf{cred}^{\mathsf{U}}$, attr, pk^{U}, σ, ρ, P^{rl}, *tracing key* tk^{U}, *tracing nonce* η, *commitment* ζ, *and commitment randomness* ω *such that*:

- All checks from § 5.1, § 5.2, and § 5.3.
- ζ is a credential attribute.
- ζ opens to tk^{U} with randomness ω: $\zeta = \mathsf{Com}(\mathsf{tk}^{\mathsf{U}}; \omega)$.
- γ is well-formed: $\gamma = \eta \| \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta)$.

In Figure 3 we describe the full credential verification protocol.

We now turn our attention to credential generation, which is presented in detail in Figure 6. During credential generation, we require that the user send a zkSNARK proof π to the CA attesting to the well-formedness of the values it provides, namely, $(U, \mathsf{pk}^{\mathsf{U}}, \beta, \zeta)$. Crucially, this proof π not only protects against a malicious user, it also preserves unlinkability against a malicious CA. In particular, the CA can learn tk^{U} for any user U by simply providing its corresponding tracing token β to the auditors during the tracing

[4] A PRF is weakly-robust if in addition to the pseudorandomness property, it is computationally infeasible for an adversary given oracle access to $\mathsf{PRF}^{DY}(K, \cdot)$ on a *limited* set of queries Q, to find (K^*, x^*) and x such that $\mathsf{PRF}^{DY}(K^*, x^*) = \mathsf{PRF}^{DY}(K, x)$.

Credential Verification

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

- Sample $\rho \leftarrow \{0,1\}^{\lambda}$ and set $\alpha := \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{id}; \rho)$.
- Sign pk^{W} as $\sigma^{\mathsf{W}} := \mathsf{SIG.Sign}(\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}})$.
- Let $x = (\mathsf{pk}^{\mathsf{CA}}, \mathsf{pk}^{\mathsf{W}}, \phi, \mathsf{pk}^{\mathsf{A}}, \mathsf{rt}^{\mathsf{rl}}, \alpha, \gamma)$.
- Let $w = (\mathsf{cred}^{\mathsf{U}}, \mathsf{attr}, \mathsf{pk}^{\mathsf{U}}, \sigma, \rho, P^{\mathsf{rl}}, \eta, \zeta, \omega)$.
- Create a proof $\pi := \mathsf{zkSNARK.Prove}(\mathsf{crs}_{\mathsf{VER}}, x, w)$ for :

$$\begin{aligned}\mathsf{VER} = \Big\{ (x, w) \mid\ & \mathsf{SIG.Verify}(\mathsf{pk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{W}}, \sigma^{\mathsf{W}}) = 1 \\ & \wedge \mathsf{SIG.Verify}(\mathsf{pk}^{\mathsf{CA}}, \mathsf{pk}^{\mathsf{U}} \| \zeta \| \mathsf{attr}, \mathsf{cred}^{\mathsf{U}}) = 1 \\ & \wedge \phi(\mathsf{attr}) = 1 \\ & \wedge \alpha = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{pk}^{\mathsf{U}}; \rho) \\ & \wedge \mathsf{MT}^{\mathsf{rl}}.\mathsf{NMVerify}(\mathsf{rt}^{\mathsf{rl}}, \mathsf{pk}^{\mathsf{U}}, P^{\mathsf{rl}}) = 1 \\ & \wedge \zeta = \mathsf{Com}(\mathsf{tk}^{\mathsf{U}}; \omega) \\ & \wedge \gamma = \eta \| \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta) \Big\}\end{aligned}$$

$\mathsf{pk}^{\mathsf{W}} \rightarrow$ **Smart Contract**: Send π and $\mathsf{TX} = (\mathsf{Verify}, \mathsf{pk}^{\mathsf{W}}, \alpha, \gamma)$
Smart Contract: If $\mathsf{zkSNARK.Verify}(\mathsf{crs}_{\mathsf{VER}}, x, \pi) = 1$, approve access.

Figure 3: Credential Verification Protocol

of some user U^*, and consequently, break unlinkability for U. To prevent this, we ask the CA to additionally send the proof π from credential generation to the auditors, as the CA can not generate a valid proof for β and U^* without knowledge of tk^{U}.

5.5 Batching Credential Verification

So far, we have focused on adding desired functionality to our AC scheme. Although our AC scheme already has a much smaller gas cost compared to the existing AC schemes owing to the use of zkSNARKs (§ 7.2.2), it still costs 294K gas or 12 USD despite using the cheapest on-chain SNARK verifier [76]. The bottleneck is the on-chain verification of the zkSNARK proof (§ 7.2.2), and in this section, we discuss how users can significantly reduce this cost through proof recursion [15, 17, 46, 140] and the help of an *untrusted aggregator*.

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 [15] 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 (§ 7.3.3). In more detail, prior works use the following approaches:

- Pairing-based [15, 81]: requires either a 2-chain (or cycle) of pairing-friendly elliptic curves which is not known[5] for BN254, or the use of expensive non-native arithmetic which leads to intractable aggregator overhead.

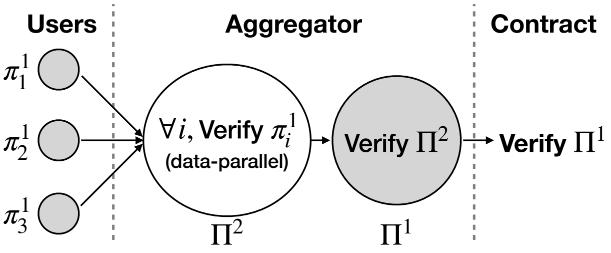


Figure 4: Batched verification workflow. Circles denote computations inside a SNARK and the circle labels denote the respective SNARK proofs. Grey circles represent pairing-based zkSNARK^1 **over BN254 and white circle represents the (data-parallel) dlog-based** simdSNARK^2 **over Grumpkin.**

- FRI-based[6] [118]: has a high recursion threshold and prover cost, which leads to high aggregator and user overhead.
- Accumulation-based [23, 26, 27, 88]: 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 § 7.3.

In this work, we adopt a *new approach* that relies on recursively verifying a discrete-log (dlog)-based SNARK. In general, dlog-based SNARKs are not suitable for recursion as their verifier complexity is at least $O(\sqrt{C})$ for a circuit of size C. In contrast, the verifier complexity of pairing-based SNARKs and FRI-based SNARKs is just $O(1)$ and $O(\log^2 C)$, respectively. Despite this limitation, we still manage to achieve a practical solution through careful use of the dlog-based SNARK and *two layers of recursion*. Before we discuss why our solution is practical, we first describe it in detail.

We use two SNARKs, namely, pairing-based zkSNARK^1 (over BN254) and dlog-based simdSNARK^2 (over Grumpkin [67]), and our batching solution (see Figure 4 for illustration) works as follows:

(1) A batch of N users independently create credential verification proofs $\{\pi_i^1\}_{i\in[N]}$ using zkSNARK^1, and send them along with transaction data $\{\mathsf{TX}_i\}_{i\in[N]}$ to the aggregator.
(2) First, the aggregator verifies the proofs $\{\pi_i^1\}_{i\in[N]}$ using simdSNARK^2 to output Π^2.
(3) Then, the aggregator verifies Π^2 using zkSNARK^1 to output Π^1, which is then sent to the contract for batched verification along with $\{\mathsf{TX}_i\}_{i\in[N]}$.
(4) 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 granting access to $\{\mathsf{pk}_i^{\mathsf{W}}\}_{i\in[N]}$.

Now, we discuss the efficiency benefits of our approach:

- Unlike the pairing-based approach, there are *no compatibility issues* as BN254 forms a cycle with (non-pairing-friendly) Grumpkin, and thus, simdSNARK^2 over Grumpkin can efficiently verify zkSNARK^1 proofs over BN254 and vice-versa.

[5] Even outside the context of EVM, the most efficient known 2-chain with 128-bit security is BLS12-377/BW6-761 [81], where BW6-761 is 6× slower than BN254 [102], albeit at a higher security level.

[6] Concurrent work on ZkEVM [65, 134] 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.

- Unlike the FRI-based approach, the users generate pairing-based zkSNARK^1 proofs with constant verifier complexity that impose *low recursion threshold and low prover overhead.*
- For N users and $O(U)$ cost of verifying a single zkSNARK^1 *user proof*, the *verifier complexity* of simdSNARK^2 is $O(\sqrt{N \cdot U})$, which corresponds to the circuit size proven by the aggregator within zkSNARK^1. Unlike the accumulation-based approach where the aggregator proves a circuit of size $O(U)$ within zkSNARK^1, $O(\sqrt{N \cdot U})$ is much better as $N \ll U$ in our setting. Importantly, this step is the aggregator runtime bottleneck for our solution as well as the accumulation-based solution.
- Since simdSNARK^2 is verifying N independent zkSNARK^1 proofs, i.e., a *data-parallel* (or SIMD) computation, the concrete cost for both its prover and verifier are improved. The latter helps further reduce the circuit size proven within zkSNARK^1.

As a result, we get a solution with zkSNARK^1 on-chain verification that has either 6.6× less aggregator overhead or 11× less user overhead compared to prior solutions (§ 7.3.3).

5.6 Reducing per-user Costs in Batching

Batching SNARK verification 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 ($\{\mathsf{TX}_i = (\mathsf{pk}_i^{\mathsf{W}}, \alpha_i, \gamma_i)\}_{i \in [N]}$), 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 posted on-chain. Even *ignoring* the SNARK verification completely, this places a lower bound of 22.3K gas per L1 wallet and 6.3K gas per L2 wallet (§ 7.3.1).

We minimize these lower bounds following the observation that user-specific inputs posted on-chain can be made *static* for a given pair of wallet and credential *without affecting privacy guarantees* in account-based blockchains. In particular, the use of a credential $\mathsf{cred}^{\mathsf{U}}$ can be divided into two phases:

- *Registration Phase*: a wallet pk^{W} pays a *one-time cost* to register or cache an audit and tracing token (α, γ) corresponding to $\mathsf{cred}^{\mathsf{U}}$ within the access control contract. Importantly, these tokens are published on L1 for *data availability*, but they are cached on either L1 or L2 depending on type of wallet.
- *Verification Phase*: subsequently, pk^{W} retrieves the tokens from the contract and pays much lower costs in accessing different applications by proving different predicates w.r.t. $\mathsf{cred}^{\mathsf{U}}$.

While caching removes the cost of posting the tokens and verifying a signature on them in subsequent batched verifications, there are still two problems that need to be addressed:

(1) *L1 storage is expensive*: given that the total length of tokens is 320-Bytes in our instantiation (§ 6), caching and retrieving these tokens on L1 costs 200K and 21K gas *per user*, respectively.
(2) *Number of SNARK inputs scales linearly with batch size*: SNARK input processing costs ≈ 9.7K gas per input in our concrete instantiation (§ 7.3.1) which lower bounds the batched verification of L2 wallets at ≈ 29K gas per user.

To address (1), instead of caching tokens directly, we instead cache a short 32-Byte commitment to them, which reduces the storage costs by 10×. Consequently, we need to change the credential verification statement as follows:

“*Given* $\mathsf{pk}^{\mathsf{CA}}$, pk^{W}, ϕ, pk^{A}, $\mathsf{rt}^{\mathsf{rl}}$, *(cached) token commitment* ν, *I know* $\mathsf{cred}^{\mathsf{U}}$, attr, pk^{U}, σ, ρ, P^{rl}, tk^{U}, η, ζ, ω, *audit token* α, *and tracing token* γ *such that*:
- All checks from § 5.1, § 5.2, § 5.3, and § 5.4.
- ν is a well-formed commitment to (α, γ): $\nu = \mathsf{CRH}(\alpha, \gamma)$, where CRH is collision-resistant hash (CRH) function.”

To address (2), we make the number of SNARK inputs independent of the batch size by moving user-specific *public inputs* in the batched verification statement to *private witnesses* using a CRH. In particular, the modified batched verification statement is as follows:

“*Given* $\mathsf{pk}^{\mathsf{CA}}$, ϕ, pk^{A}, $\mathsf{rt}^{\mathsf{rl}}$, *and commitment* H, *I know proofs* $\{\pi\}_{i \in [N]}$, *wallet addresses* $\{\mathsf{pk}_i^{\mathsf{W}}\}_{i \in [N]}$, *token commitments* $\{\nu_i\}_{i \in [N]}$, *and intermediate commitments* $\{H_i\}_{i \in [N]}$ *such that*:
- For each $i \in [N]$:
  - H_i is well-formed: $H_i = \mathsf{CRH}(\mathsf{pk}_i^{\mathsf{W}}, \nu_i)$.
  - π_i is a valid credential verification proof: $\mathsf{zkSNARK}^1.\mathsf{Verify}(\mathsf{crs}_{\mathsf{VER}}, (\mathsf{pk}^{\mathsf{CA}}, \mathsf{pk}_i^{\mathsf{W}}, \phi, \mathsf{pk}^{\mathsf{A}}, \mathsf{rt}^{\mathsf{rl}}, \tau, \nu_i), \pi_i^1) = 1$, where VER is the credential verification statement.
- H is well-formed: $H = \mathsf{CRH}(H_0, \ldots, H_{N-1})$.

The verification of this statement on-chain requires H, which is computed either on L1 or L2 depending on the wallet type.

Overall, our solution reduces the batched verification cost by up to 14.8× and 284× for L1 and L2 wallets, respectively (Figure 5).

6 Concrete Instantiation

In this section, we discuss how we concretely instantiate the primitives used in § 5 and our rationale behind these choices.

6.1 Core Protocols

Collision-Resistant Hash. We use Poseidon [75] to instantiate the correlation-resistant hash (CRH) in our scheme.

Collision-Resistant Pseudorandom Function. Like Zcash [14], we instantiate it with the SHA256 compression function.

zk-SNARK. We use Groth16 [76] instantiated over the BN254 curve [14, 123] 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 [69], 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 [142, 149]. We use Groth16 in our evaluation because it has much better development support.

Digital Signature. We instantiate the signature scheme used by CA with EdDSA [92] on BabyJubJub curve [144], which is efficient to verify within Groth16 instantiated over the BN254 curve [15].

The signature scheme used to sign user messages is instantiated with the simulation-extractable NIZKPoK+OWF signature scheme by Bellare [12], 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 G for a full proof. The one way function is instantiated with Poseidon [1].

Threshold Public Key Encryption. We instantiate TPKE with the threshold variant of CCA-2 secure Cramer-Shoup encryption described in [39] combined with the upgrade to non-interactive decryption via Key-Homomorphic PRFs from [22].

Sparse Merkle Tree. We instantiate the Sparse Merkle Tree with depth 254 and Poseidon hash.

6.2 Batched and L2 Verification

We instantiate zkSNARK[1] with pairing-based Groth16 [76] (over BN254) and simdSNARK[2] with discrete-log-based Spartan [125] (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 (§ 7.3), they can be further improved significantly using plonk arithmetization [68, 69] and its custom gates, specifically for non-native arithmetic and MSMs [149].

7 Evaluation

In this section, we evaluate ZEBRA and answer the following:

(1) For single credential verification, how does ZEBRA compare with prior AC schemes for blockchains in terms of:
  - verification and proof complexity (§ 7.2.1),
  - gas cost on EVM-compatible blockchains (§ 7.2.2), and
  - computational cost imposed on users (§ 7.2.3)?

(2) For batched credential verification:
  - What is the improvement in gas cost incurred per user with batched verification (§ 7.3.1)?
  - For a large enough batch, what is the computational and monetary overhead on ZEBRA's aggregator (§ 7.3.2)?
  - How does our proof batching solution compare with prior approaches to proof recursion (§ 7.3.3)?

Other aspects of ZEBRA like credential generation, revocation, tracing, and transaction audit are lightweight in terms of cryptographic tools and not time-sensitive.

For EVM-compatible blockchains, we use gas cost as our on-chain cost metric as it includes the cost of computation, memory, storage, as well as transaction size, and it is also the sole metric used by prior works (Coconut, BASS). The throughput of an EVM-compatible blockchain is also defined in terms of gas usage per second, and thus, the lower the gas cost of a transaction, the higher its throughput.

7.1 Implementation and Experimental Setup

We defer the implementation details to Appendix C and our code is available at https://github.com/deevashwer/zebra. The details of our experimental setup are as follows:

- EVM Gas Cost for Contracts: ganache v7.8.0 [130] and truffle suite v5.8.4 [129]
- User Overhead: 2019 MacBook Pro (2.4 GHz 8-Core Intel Core i9 processor, 16 GB RAM)
- Aggregator Overhead: `m6i.32xlarge` AWS instance (3.5 GHz Intel Xeon processor, 128 vCPUs, 512 GB RAM)

7.2 Credential Verification and Comparison

In this section, we evaluate ZEBRA's single credential verification and compare it with the following prior AC schemes for blockchains:

- Coconut [77, 128]: the state-of-the-art AC scheme; does not provide revocation, auditability or traceability.
- BASS [151]: a subsequent work that adds revocation to Coconut.
- Concordium's AC scheme [57]: a recent AC scheme in the context of permissioned blockchains that provides auditability and traceability, but not revocation.

We include BASS and Concordium to highlight the high cost of supporting properties beyond unlinkability in existing AC schemes. Table 1 summarizes the supported properties, verifier and proof complexity, estimated verifier runtime, proof size, and EVM gas cost for ZEBRA and the above-mentioned AC schemes. We estimate verifier time from the runtime of individual BN254 curve operations, which we benchmark using the `gnark` backend, i.e., the backend we use to generate user proofs. To estimate the gas cost of prior AC schemes, we used the costs from EIP-1108 [145] for $\mathbb{G}_1$ and pairing operations, and the benchmarks from Coconut's code [99] for $\mathbb{G}_2$ operations. For $\mathbb{G}_T$ operations, we assume the cost to be equal to that of $\mathbb{G}_2$ operations to favor prior works as there's no implementation available and our benchmarks show that $\mathbb{G}_T$ operations are 4× more expensive than $\mathbb{G}_2$ operations in BN254. As suggested by Coconut, we swapped the $\mathbb{G}_1$ and $\mathbb{G}_2$ operations for both Coconut and BASS to reduce their respective gas costs. Now, we first analyze the verifier and proof complexity.

7.2.1 Verification and Proof Complexity. Since the verifier and proof complexity of prior AC schemes is linear in the verification predicate, it scales linearly with the number of attributes m and gets significantly worse as we add more properties to the AC scheme. For instance, Table 1 shows that complexity of Coconut becomes 3× worse by just adding revocation support. Similarly, to support auditability and traceability, Concordium's AC scheme incurs a cost that even grows with the number of auditors.

In contrast, ZEBRA has a constant verifier complexity irrespective of the number of attributes and the properties supported. ZEBRA's proof complexity remained the same with revocation and only grew by 10 $\mathbb{F}_p$ elements to support auditability and traceability. Concretely, ZEBRA's verifier time and proof size are comparable to the simplest instantiation of Coconut that only provides unlinkability. Since any typical blockchain deployment requires more than just unlinkability, this shows that ZEBRA is the most efficient AC scheme for on-chain verification.

7.2.2 EVM Gas Cost. A credential verification transaction in ZEBRA costs 294K gas, out of which, around 235K gas is for SNARK verification (including input processing), making it the bottleneck. Other than that, base transaction fee is 21K gas, signature verification on the tokens and wallet address takes 6K gas, updating the access map takes 20K gas, and finally, posting the proof, the tokens, and the signature on-chain requires another 10K gas; the remaining gas cost is due to miscellaneous factors. With the current average gas price of 23 Gwei and the price of Ethereum (1826.03 USD) on May 27, 2023[7], our credential verification would require 12.34 USD for 294K gas, which is reasonable for some users and applications.

Now, we compare the gas cost of ZEBRA with prior AC schemes. Table 1 shows that the gas cost of Coconut credential verification

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

Table 1: Comparison of ZEBRA's single verification with prior AC works in context of blockchains for m private attributes. The properties beyond unlinkability (see § 1) are abbreviated as follows: Rv (Revocation), Au (Auditability), and Tr (Traceability). The bilinear group is instantiated over the BN254 curves, where the scalar field size $|\mathbb{F}_p| = 32$ Bytes, $|\mathbb{G}_1| = 64$ Bytes, $|\mathbb{G}_2| = 128$ Bytes, $|\mathbb{G}_T| = 384$ Bytes. The verifier runtime is estimated by benchmarking BN254 curve operations with a single thread on MacBook Pro 2019. All metrics for Concordium [57] only represent a part of the credential verification computation.

Scheme	Properties			Proof Complexity	Verifier Complexity	Estimated Time (ms) / Proof Size (Bytes)	Gas Cost
	Rv	Au	Tr				
ZEBRA	✓	✓	✓	$\lvert\mathbb{G}_2\rvert + 2\lvert\mathbb{G}_1\rvert + 10\lvert\mathbb{F}_p\rvert$	3 exp-$\mathbb{G}_1$, 3 op-$\mathbb{G}_1$, 4 Pairings	1.54 / 576	294K
Coconut [128]	✗	✗	✗	$3\lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_1\rvert$ $+ (m+2)\lvert\mathbb{F}_p\rvert$	$(m+3)$ exp-$\mathbb{G}_1$, $(m+3)$ op-$\mathbb{G}_1$, 2 exp-$\mathbb{G}_2$, 2 op-$\mathbb{G}_2$, 2 Pairings	$1.44 + 0.08m$ / $512 + 32m$	4.2M $+ m \cdot 6$K
BASS [151]	✓	✗	✗	$\lvert\mathbb{G}_T\rvert + 5\lvert\mathbb{G}_2\rvert + 4\lvert\mathbb{G}_1\rvert$ $+ (m+8)\lvert\mathbb{F}_p\rvert$	$(m+5)$ exp-$\mathbb{G}_1$, $(m+4)$ op-$\mathbb{G}_1$, 2 exp-$\mathbb{G}_2$, 2 op-$\mathbb{G}_2$, 8 Pairings, 4 exp-$\mathbb{G}_T$, 5 op-$\mathbb{G}_T$	$5.77 + 0.08m$ / $1536 + 32m$	12.6M $+ m \cdot 6$K
Concordium [57] n: #Auditors (lower bound)	✗	✓	✓	$(m+3)\lvert\mathbb{G}_T\rvert$ $+ (7n+2)\lvert\mathbb{G}_1\rvert$ $+ (m+3)\lvert\mathbb{F}_p\rvert$	$8n$ exp-$\mathbb{G}_1$, $5n$ op-$\mathbb{G}_1$, $(m+6)$ Pairings, $(4m+13)$ exp-$\mathbb{G}_T$, $(3m+9)$ op-$\mathbb{G}_T$	$11.0 + 3.07m + 0.61n$ / $1376 + 416m + 448n$	26.57M $+ m \cdot 8.1$M $+ n \cdot 48$K

is at least 4.2M, which is 14× larger than ZEBRA and translates to 176.5 USD on Ethereum. An astute reader might wonder why Coconut's gas cost is much higher than ZEBRA's even though their estimated verifier time and proof size are comparable. This is due an EVM artefact as we alluded to in § 1: 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× compared to $\mathbb{G}_1$ operations.

7.2.3 User Overhead. As discussed in § 1, the prover time in prior AC schemes is minimal (i.e., in the order of tens to hundreds of milliseconds) as they rely on lightweight and customized NIZK arguments. In contrast, ZEBRA sacrifices prover efficiency to gain verifier efficiency by relying on general-purpose zkSNARKs. Although ZEBRA's prover is orders of magnitude slower than prior AC schemes, we found that it is still practical. In particular, we evaluated the SNARK proof generation overhead on the user, which is the bottleneck in generating a credential verification transaction. The total R1CS constraints in our credential verification circuit are 386K, which requires 2s to prove on a Macbook Pro 2019 using 16 threads. Most of the constraints are due to the use of SHA256, a single call of which costs 60K constraints on a 64-Byte input. While this is already practical for laptops, especially given that these proofs need to be generated once per application and wallet, it can be made more accessible to weak-client devices like smartphones. For instance, we can (i) replace SHA256 with a SNARK-friendly hash function like Poseidon [75], (ii) delegate proof generation in a secure manner [45, 70, 107], and (iii) use plonk arithmetization instead of R1CS to reduce SHA256 constraints [149].

7.3 Batched Verification

7.3.1 Gas Costs. Figure 5 compares the gas cost per user for five kinds of credential verification: (i) ZEBRA's single verification as evaluated in § 7.2 (Single), (ii) L1 batched verification without 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 token caching (§ 5.6), 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 27.8× and requires just 14.1K and 10.6K gas per user for a batch of 64 and 512 users, respectively. For a batch of 512 users, this translates to just 0.44 USD on Ethereum. Without caching, the gas cost improvement is just 2×, which is largely due to expensive SNARK input processing with Groth16 which costs around 9.7K gas per 32-Byte input. Even ignoring the cost from Groth16 entirely, the gas cost without caching would still be around 22.3K. This shows that our caching technique is essential for batching with Groth16, and it leads to at least 2× 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 539× for a batch of 512 users. For the same, the gas cost is just 545, which costs 0.02 USD and is comparable with the L2 transaction gas costs for ZkSync [154] and Loopring [93], the most cost-efficient L2 solutions [104]. 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 6.3K from just posting the tokens on-chain. This is at least 11.5× worse than ZEBRA, demonstrating that caching is crucial for batching L2 wallets.

Previously we assumed that tokens were already cached, and now we report the gas costs for caching tokens, a.k.a., registering the credential. The gas cost for caching is just 56.5K and 7.9K for L1 and L2 wallets, respectively, given a batch of 512 users. This translates to just 2.37 USD and 0.33 USD on Ethereum, respectively, and only has to be paid once per credential and wallet.

7.3.2 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 50 seconds for 64 users, which already leads to a good amortization of gas costs. For a better amortization with 512 users, the total runtime grows sub-linearly to around 1.5 minutes, which is still quite practical given credential verification is required once per

Figure 5: 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.

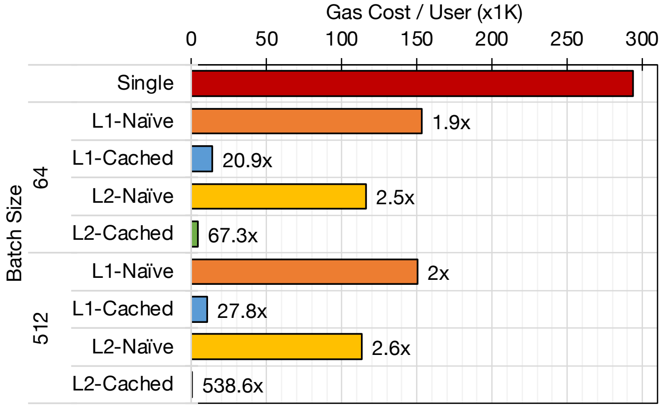


Table 2: Aggregator overhead for N users. Times are in seconds (s) and R1CS constraints are in millions (M).

SNARK	Metric	$N = 64$	$N = 512$
Spartan (data-parallel)	Time	14.6 s	33.9 s
	#Constraints	7M	57M
Groth16	Time	34.7 s	50.4 s
	#Constraints	65M	105M
Total Aggregator Time		49.4 s	84.4 s

application. 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 does not affect the total runtime significantly, and thus, the rollup server overhead for L2 batching is similar to the L1 batching overhead.

Now, we analyze the monetary cost of batched verification. The `m6i.32xlarge` we rented from AWS has a spot price of 2.3437 USD/hour (US East - Ohio). We can batch around 21.8K users an hour with this instance, 512 at a time. Thus, the compute cost incurred by the aggregator per user is just 0.000107 USD. This cost is negligible compared to the minimum price of gas cost per user we achieve which is 0.02 USD, underscoring the significance of minimizing gas costs.

7.3.3 Cost of Prior Recursion Approaches. We discussed why prior approaches to recursion are not suitable in our setting in § 5.5, 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 of Spartan is better than Groth16 [125].

First, we have the pairing-based approach that requires use of non-native arithmetic which introduces ≈ 1000× overhead in R1CS [6]. 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 [118], resulting in 2^{29} constraints for $N = 512$. Finally, the cost of just the decider in accumulation-based recursion is $> 2^{13} \cdot n$ [7], where n is the number of constraints in each accumulated instance. In our setting, $n \approx 2^{17}$ (see Spartan constraints in Table 2), and thus, the decider cost is $> 2^{30}$ constraints.

In contrast, the total overhead in our solution is just $2^{27.27}$ constraints ($2^{26.65}$ within Groth16), which is at least 212× and 6.6× better than pairing and accumulation-based approach, respectively. Similarly, the FRI-based approach is also 3.3× worse, and it additionally imposes an 11× higher prover overhead on users (Figure 7 in [125]). Thus, prior approaches to recursion either increase aggregator overhead by 6.6× or the user overhead by 11×.

8 Related Work

Anonymous Credentials. Following the initial work of Chaum [44], there has been a long line of work [11, 21, 25, 30, 30, 35, 37, 48, 66, 78, 82, 122, 133] with successively more efficient and expressive anonymous credentials that have been widely deployed in a number of real-world applications [4, 31, 59, 109]. Today, we have credentials that can be used a limited number of times [9, 25, 32], revoked [33, 34, 36], audited [35], traced [57], delegated [10, 18, 29, 43, 54], updated [19, 52], and issued by a decentralized organization [71, 77, 121, 128]. Recent years have also witnessed a synergy between ACs and blockchains. ACs are being used in permissioned blockchains for identity management [5, 20, 57, 126], and blockchains are being leveraged as a verifiable data registry to improve off-chain certificates [50, 86, 106, 119].

Private On-chain Access Control. Like ZEBRA, several concurrent works, namely, iden3 [85], Polygon ID [119] and Semaphore [124], enable a private on-chain access control solution using zkSNARKs. Thus, they can achieve similar gas cost as ZEBRA's single verification by instantiating the underlying zk-SNARK with Groth16 [76] or PLONK [69]. However, unlike ZEBRA, they don't support auditability, traceability, and batching, which are necessary for accountability, efficient revocation, and practical verification costs.

Similarly, Espresso Systems's CAPE [131, 132] is a concurrent work that also relies on zkSNARKs to enable anonymous asset transfer on Ethereum with configurable policies per asset. One such policy can be ownership of a valid credential. However, unlike our solution, CAPE works in the UTxO model, and does not support batching, revocation and traceability.

Other Works. Azeroth [87] provides privacy-preserving transactions with auditing. In a similar vein, there are works adding auditability and regulation to existing privacy-preserving cryptocurrencies like ZCash [72] and Monero [91], and central bank digital currencies (CBDCs) [148]. Finally, zkLedger [100] proposed privacy-preserving ledgers that can be used by banks to settle cross-organization transactions while also allowing third-party auditing.

References

[1] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*. Springer.

[2] Algorand. 2022. https://www.algorand.com/.

[3] Ian Allison. 2021. Aave's Push for Institutional DeFi Gets Second KYC Provider Proposal. https://www.coindesk.com/business/2021/12/03/aaves-push-for-institutional-defi-gets-second-kyc-provider-proposal/. *CoinDesk* (2021).

[4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys*.

[5] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. 2020. Privacy-Preserving Auditable Token Payments in a Permissioned Blockchain System *(AFT '20)*. Association for Computing Machinery.

[6] arkworks rs. 2021. nonnative. https://github.com/arkworks-rs/nonnative.

[7] arkworks rs. 2021. r1cs-std. https://github.com/arkworks-rs/r1cs-std.

[8] arkworks rs. 2022. arkworks-rs. https://github.com/arkworks-rs.

[9] Foteini Baldimtsi and Anna Lysyanskaya. 2013. Anonymous credentials light. In *CCS*. ACM.

[10] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO*. Springer.

[11] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. 2008. P-signatures and noninteractive anonymous credentials. In *TCC*. Springer.

[12] Mihir Bellare. 2021. Lectures on NIZKs: A Concrete Security Treatment. https://cseweb.ucsd.edu/~mihir/cse208-Wi20/main.pdf

[13] Mihir Bellare, Sarah Meiklejohn, and Susan Thomson. 2014. Key-versatile signatures and applications: RKA, KDM and joint enc/sig. In *EUROCRYPT*. Springer.

[14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE S&P*. IEEE.

[15] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. In *CRYPTO*. Springer.

[16] Josh Benaloh and Michael de Mare. 1993. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT*. Springer.

[17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*. ACM.

[18] Johannes Blömer and Jan Bobolz. 2018. Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures. In *ACNS (Lecture Notes in Computer Science)*. Springer.

[19] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. 2019. Updatable Anonymous Credentials and Applications to Incentive Systems. In *CCS*. ACM.

[20] Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. 2021. Anonymous Transactions with Revocation and Auditing in Hyperledger Fabric. *CANS* (2021).

[21] Dan Boneh and Xavier Boyen. 2004. Short Signatures Without Random Oracles. In *EUROCRYPT*. Springer.

[22] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. 2013. Key Homomorphic PRFs and Their Applications. In *CRYPTO (1) (Lecture Notes in Computer Science, Vol. 8042)*. Springer, 410–428.

[23] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Recursive Proof Composition without a Trusted Setup. Cryptology ePrint Archive, Report 2019/1021.

[24] Pelle Braendgaard. 2019. EIP-1812: Ethereum Verifiable Claims. https://github.com/ethereum/EIPs/pull/1812

[25] Stefan Brands and Frédéric Légaré. 2002. Digital Identity Management based on Digital Credentials. In *GI Jahrestagung (LNI)*. GI.

[26] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. 2021. Proof-Carrying Data Without Succinct Arguments. In *CRYPTO*. Springer.

[27] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. 2020. Recursive Proof Composition from Accumulation Schemes. In *TCC*. Springer.

[28] Vitalik Buterin. 2018. On-chain scaling to potentially 500 tx/sec through mass tx validation. (2018). https://ethresear.ch/t/on-chain-scaling-topotentially-500-tx-sec-through-mass-tx-validation

[29] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. 2017. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In *CCS*.

[30] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. 2015. Composable and Modular Anonymous Credentials: Definitions and Practical Constructions. In *ASIACRYPT*. Springer.

[31] Jan Camenisch and Els Van Herreweghen. 2002. Design and implementation of the *idemix* anonymous credential system. In *CCS*. ACM.

[32] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS*. ACM.

[33] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *PKC*. Springer.

[34] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2010. Solving Revocation with Efficient Update of Anonymous Credentials. In *SCN*. Springer.

[35] Jan Camenisch and Anna Lysyanskaya. 2001. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*. Springer.

[36] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO*. Springer.

[37] Jan Camenisch and Anna Lysyanskaya. 2004. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*. Springer.

[38] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*. IEEE.

[39] Ran Canetti and Shafi Goldwasser. 1999. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*. Springer.

[40] BNB Smart Chain. 2022. https://www.bnbchain.org/en/smartChain.

[41] Gnosis Chain. 2022. https://www.gnosis.io/.

[42] HECO Chain. 2022. https://www.hecochain.com/en-us/.

[43] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2014. Malleable Signatures: New Definitions and Delegatable Anonymous Credentials. In *CSF*. IEEE.

[44] David Chaum. 1985. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM* (1985).

[45] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. 2023. Eos: Efficient Private Delegation of zkSNARK Provers. In *USENIX Security Symposium*.

[46] Alessandro Chiesa and Eran Tromer. 2010. Proof-Carrying Data and Hearsay Arguments from Signature Cards. In *ICS*. Tsinghua University Press.

[47] Coin Metrics Network Chart. 2022. https://charts.coinmetrics.io/.

[48] Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. 2022. Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes. In *PKC*. Springer.

[49] ConsenSys. 2021. gnark. https://github.com/ConsenSys/gnark.

[50] World Wide Web Consortium et al. 2019. Verifiable Credentials Data Model v1.1. *W3C First Public Working Draft, https://www.w3.org/TR/vc-data-model/* (2019).

[51] Ben Cooper. 2021. Announcing Alkemi Network & KYC-Chain Partnership. https://medium.com/alkemi/announcing-alkemi-network-kyc-chain-partnership-f87aa1f27700. *Medium* (2021).

[52] Scott E. Coull, Matthew Green, and Susan Hohenberger. 2009. Controlling Access to an Oblivious Database Using Stateful Anonymous Credentials. In *PKC*. Springer.

[53] Ronald Cramer and Victor Shoup. 1998. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*. Springer.

[54] Elizabeth C Crites and Anna Lysyanskaya. 2019. Delegatable anonymous credentials from mercurial signatures. In *CT-RSA*. Springer.

[55] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. 2016. Efficient sparse merkle trees. In *NordSec*. Springer.

[56] Ivan Damgård. 2002. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science* (2002), 84.

[57] Ivan Damgård, Chaya Ganesh, Hamidreza Khoshakhlagh, Claudio Orlandi, and Luisa Siniscalchi. 2021. Balancing Privacy and Accountability in Blockchain Identity Management. In *CT-RSA*. Springer.

[58] Ivan Damgård and Nikos Triandopoulos. 2008. Supporting Non-membership Proofs with Bilinear-map Accumulators. *IACR Cryptol. ePrint Arch.* (2008), 538.

[59] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PETS* (2018).

[60] Nikhilesh De. 2021. State of Crypto: FATF's New Guidance Takes Aim at DeFi. https://www.coindesk.com/policy/2021/03/30/state-of-crypto-fatfs-new-guidance-takes-aim-at-defi/. *CoinDesk* (2021).

[61] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. 2010. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*. Springer.

[62] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *Public Key Cryptography (Lecture Notes in Computer Science, Vol. 3386)*. Springer, 416–431.

[63] Ethereum Gas Charts. 2022. https://ethereumprice.org/gas/.

[64] Fantom. 2022. https://fantom.foundation/.

[65] Brendan Farmer. 2022. Introducing Plonky2. https://blog.polygon.technology/introducing-plonky2/. *Polygon Blog* (2022).

[66] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2019. Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials. *J. Cryptol.* (2019).

[67] Ariel Gabizon, Zac Williamson, and Tom Walton-Pocock. 2021. Aztec Yellow Paper. https://hackmd.io/@aztec-network/ByzgNxBfd. *hackmd* (2021).

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

[69] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953.

[70] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. 2023. zkSaaS: Zero-Knowledge SNARKs as a Service. In *USENIX Security Symposium*.

[71] Christina Garman, Matthew Green, and Ian Miers. 2013. Decentralized anonymous credentials. Cryptology ePrint Archive, Report 2013/622. (2013).

[72] Christina Garman, Matthew Green, and Ian Miers. 2016. Accountable Privacy for Decentralized Anonymous Payments. In *Financial Cryptography*. Springer.

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

[74] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. 2020. Storing and Retrieving Secrets on a Blockchain. Cryptology ePrint Archive, Report 2020/504.

[75] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security Symposium*. USENIX Association, 519–535.

[76] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*. Springer.

[77] Harry Halpin. 2020. Nym credentials: Privacy-preserving decentralized identity with blockchains. In *CVCBT*. IEEE.

[78] Lucjan Hanzlik and Daniel Slamanig. 2021. With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In *CCS*.

[79] Harmony. 2022. https://www.harmony.one/.

[80] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*. Springer, 339–352.

[81] Youssef El Housni and Aurore Guillevic. 2020. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. Cryptology ePrint Archive, Paper 2020/351.

[82] Chloé Hébant and David Pointcheval. 2020. Traceable Constant-Size Multi-Authority Credentials. Cryptology ePrint Archive, Report 2020/657.

[83] iden3. 2023. Library of basic circuits for circom. https://github.com/iden3/circomlib.

[84] iden3. 2023. zkSnark circuit compiler. https://github.com/iden3/circom.

[85] iden3 On-chain Verification Contracts. 2022. https://github.com/iden3/contracts/tree/master/contracts/validators.

[86] iden3.io. 2022. https://iden3.io/.

[87] Gweonho Jeong, Nuri Lee, Jihye Kim, and Hyunok Oh. 2022. Azeroth: Auditable Zero-knowledge Transactions in Smart Contracts. Cryptology ePrint Archive, Report 2022/211.

[88] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2021. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. Cryptology ePrint Archive, Report 2021/370.

[89] Ravie Lakshmanan. 2022. U.S. Sanctions Virtual Currency Mixer Tornado Cash for Alleged Use in Laundering. https://thehackernews.com/2022/08/us-sanctions-virtual-currency-mixer.html. *The Hacker News* (2022).

[90] Jiangtao Li, Ninghui Li, and Rui Xue. 2007. Universal accumulators with efficient nonmembership proofs. In *International Conference on Applied Cryptography and Network Security*. Springer.

[91] Yannan Li, Guomin Yang, Willy Susilo, Yong Yu, Man Ho Au, and Dongxi Liu. 2021. Traceable Monero: Anonymous Cryptocurrency with Enhanced Accountability. *IEEE Trans. Dependable Secur. Comput.* (2021).

[92] Ilari Liusvaara and Simon Josefsson. 2017. Edwards-curve digital signature algorithm (EdDSA). *IETF* (2017).

[93] Loopring. 2022. Loopring. *URL: https://loopring.org/* (2022).

[94] Lukso. 2022. https://www.lukso.network/.

[95] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. 2021. CanDID: Can-do decentralized identity with legacy compatibility, Sybil-resistance, and accountability. In *IEEE S&P*. IEEE.

[96] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. CHURP: dynamic-committee proactive secret sharing. In *CCS*. 2369–2386.

[97] Michael McSweeney. 2022. Axie Infinity's Ethereum sidechain Ronin hit by $600 million exploit. https://www.theblockcrypto.com/post/139761/axie-infinitys-ethereum-sidechain-ronin-hit-by-600-million-exploit. *The Block Crypto* (2022).

[98] MetisDAO. 2021. EVM Equivalence vs. EVM Compatibility. https://metisdao.medium.com/evm-equivalence-vs-evm-compatibility-199bd66f455d. *Medium* (2021).

[99] musalbas. 2018. coconut-ethereum. https://github.com/musalbas/coconut-ethereum.

[100] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In *NSDI*. USENIX Association.

[101] Celo Network. 2022. https://celo.org/.

[102] Celer Network. 2023. The Pantheon of Zero Knowledge Proof Development Frameworks. https://blog.celer.network/2023/03/01/the-pantheon-of-zero-knowledge-proof-development-frameworks/. *Celer Network Blog* (2023).

[103] Brian Newar. 2022. Aave launches its permissioned pool Aave Arc, with 30 institutions set to join. https://cointelegraph.com/news/aave-launches-its-permissioned-pool-aave-arc-with-30-institutions-set-to-join. *Cointelegraph* (2022).

[104] Bitpush News. 2022. Which Layer 2 Rollup for Ethereum is the Best? https://bitpushnews.medium.com/which-layer-2-rollup-for-ethereum-is-the-best-f7ac047c1ac6. *Medium* (2022).

[105] Kobbi Nissim and Moni Naor. 1998. Certificate Revocation and Certificate Update. In *USENIX Security Symposium*. USENIX Association.

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

[107] Alex Ozdemir and Dan Boneh. 2022. Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. In *USENIX Security Symposium*. USENIX Association, 4291–4308.

[108] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. 2011. Optimal Verification of Operations on Dynamic Sets. In *CRYPTO (Lecture Notes in Computer Science, Vol. 6841)*. Springer, 91–110.

[109] Christian Paquin. 2011. U-prove technology overview v1. 1. *Microsoft Corporation Draft Revision* 1 (2011).

[110] Rafael Pass. 2003. Simulation in Quasi-Polynomial Time, and Its Application to Protocol Composition. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 2656)*. Springer, 160–176.

[111] Pieter Pauwels. 2021. zkKYC: A solution concept for KYC without knowing your customer, leveraging self-sovereign identity and zero-knowledge proofs. Cryptology ePrint Archive, Report 2021/907.

[112] Pieter Pauwels, Joni Pirovich, Peter Braunz, and Jack Deeb. 2022. zkKYC in DeFi: An approach for implementing the zkKYC solution concept in Decentralized Finance. Cryptology ePrint Archive, Report 2022/321.

[113] Alexey Pertsev, Roman Semenov, and Roman Storm. 2019. Tornado Cash Privacy Solution Version 1.4. (2019).

[114] David Pointcheval and Olivier Sanders. 2016. Short randomizable signatures. In *CT-RSA*. Springer.

[115] Polkadex. 2022. What is Decentralized KYC and why Polkadex is implementing it. https://medium.com/polkadex/what-is-decentralized-kyc-and-why-polkadex-is-implementing-it-88f01c4c3e9a. *Medium* (2022).

[116] Polygon. 2022. https://polygon.technology/.

[117] Chaitanya Potti and Partha Bhattacharya. 2018. EIP-1261: Membership Verification Token. https://github.com/ethereum/eips/issues/1261

[118] Fractal Protocol. 2021. Fractal Protocol. https://protocol.fractal.id/

[119] Andjela Radmilac. 2022. A look at Polygon ID, a new zk-proof based Web3 identity solution. https://cryptoslate.com/polygons-new-zk-proof-based-web3-identity-service/. *CryptoSlate* (2022).

[120] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and probabilistic leaderless BFT consensus through metastability. *arXiv preprint arXiv:1906.08936* (2019).

[121] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. 2022. zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. Cryptology ePrint Archive, Paper 2022/878.

[122] Olivier Sanders. 2020. Efficient Redactable Signature and Application to Anonymous Credentials. In *PKC*. Springer.

[123] scipr lab. 2020. libsnark. https://github.com/scipr-lab/libsnark.

[124] Semaphore V2. 2022. https://semaphore.appliedzkp.org/.

[125] Srinath T. V. Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO*. Springer.

[126] Wei Shao, Chunfu Jia, Yunkai Xu, Kefan Qiu, Yan Gao, and Yituo He. 2020. Attrichain: Decentralized traceable anonymous identities in privacy-preserving permissioned blockchain. *Computers & Security* (2020).

[127] Corwin Smith, Alex Beckett, Paul Wackerow, and AlehNat. 2023. Data Availability. https://ethereum.org/en/developers/docs/data-availability/. *Ethereum Docs* (2023).

[128] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. 2019. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. In *NDSS*. The Internet Society.

[129] Truffle Suite. 2018. Truffle Suite - Your Ethereum Swiss Army Knife. *URL: http://truffleframework.com/* (2018).

[130] Truffle Suite. 2022. Ganache. https://github.com/trufflesuite/ganache.

[131] Espresso Systems. 2022. CAPE Overview. https://docs.cape.tech/espresso-systems/cape/overview.

[132] Espresso Systems. 2022. Specification: Configurable Asset Privacy. https://github.com/EspressoSystems/cap/blob/main/cap-specification.pdf. *Github* (2022).

[133] Syh-Yuan Tan and Thomas Groß. 2020. MoniPoly - An Expressive q-SDH-Based Anonymous Attribute-Based Credential System. In *ASIACRYPT*. Springer.

[134] Polygon Team. 2022. The Future is Now for Ethereum Scaling: Introducing Polygon zkEVM. https://blog.polygon.technology/the-future-is-now-for-ethereum-scaling-introducing-polygon-zkevm/. *Polygon Blog* (2022).

[135] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO (2) (Lecture Notes in Computer Science, Vol. 8043)*. Springer, 71–89.

[136] Andrew Thurman. 2021. Aave Proposal Enlists Fireblocks to Aid DeFi Protocol's Mainstream Finance Push. https://www.coindesk.com/tech/2021/09/27/aave-proposal-enlists-fireblocks-to-aid-defi-lenders-mainstream-finance-push/. *CoinDesk* (2021).

[137] Joel Torstensson. 2017. EIP-780: Ethereum Claims Registry. https://github.com/ethereum/EIPs/issues/780

[138] TRON. 2022. https://tron.network/.

[139] Matan Tsuberi, Ben Kaufman, Adam Levi, and Oren Sokolowsky. 2018. EIP-1480: Access Control Standard. https://github.com/ethereum/EIPs/issues/1481

[140] Paul Valiant. 2008. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. In *TCC (Lecture Notes in Computer Science)*. Springer.

[141] Fabian Vogelsteller. 2017. EIP-735: Claim Holder. https://github.com/ethereum/eips/issues/735

[142] Thomas Walton-Pocock. 2019. PLONK Benchmarks I — 2.5x faster than Groth16 on MiMC. https://medium.com/aztec-protocol/plonk-benchmarks-2-5x-faster-than-groth16-on-mimc-9e1009f96dfe. *Medium* (2019).

[143] Tracy Wang. 2022. Algorand Pushes for Ethereum Compatibility With $20M Incentive Program. https://www.coindesk.com/tech/2022/02/18/algorand-pushes-for-ethereum-compatibility-with-20m-incentive-program/. *CoinDesk* (2022).

[144] Barry WhiteHat, Marta Belles, and Jordi Baylina. 2020. EIP-2494: Baby Jubjub elliptic curve. https://github.com/ethereum/EIPs/pull/2494.

[145] Zachary Williamson and Antonio Salazar Cardozo. 2018. EIP-1108: Reduce alt-bn128 precompile gas costs. https://github.com/ethereum/EIPs/pull/1108.

[146] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

[147] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

[148] Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. 2019. PRCash: Fast, Private and Regulated Transactions for Digital Currencies. In *Financial Cryptography*. Springer.

[149] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. 2022. VERI-ZEXE: Decentralized Private Computation with Universal Setup. Cryptology ePrint Archive, Paper 2022/802.

[150] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*. IEEE Computer Society, 160–164.

[151] Yong Yu, Yanqi Zhao, Yannan Li, Xiaojiang Du, Lianhai Wang, and Mohsen Guizani. 2019. Blockchain-based anonymous authentication with selective revocation for smart industrial applications. *IEEE Trans. Industr. Inform.* (2019).

[152] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. ACM. In *CCS*.

[153] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. 2020. Deco: Liberating web data using decentralized oracles for tls. In *CCS*. ACM.

[154] ZKSync. 2022. ZKSync. *URL: https://zksync.io/* (2022).

A Credential Theft Detection

Credential theft detection allows a user to detect unauthorized usage of its credential, which implies that the credential has been stolen. Such an ability is possible in the first place due to the transparent nature of the blockchain. We observe that the scheme described so far already enables theft detection: since the user also knows the tracing key tk^{U} for its credential, it can trace all the wallets that have used its credential using the same strategy as in § 5.4. Moreover, since tk^{U} does not grant ownership of the credential, users with weak devices can also share it with a service to do the tracing on their behalf, at the cost of losing privacy to this service. We argue that this is still better than non-private solutions where the privacy is lost to the CA for the following reasons: (i) users are not forced to trust a particular institution to protect their privacy, and (ii) anyone with access to compute resources equivalent to a commodity grade laptop can start a tracing service.

Credential Generation

Parameters: SIG defines a key-pair relation $\mathcal{R}$.

User U:

- Sample $(\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{U}}) := \mathsf{SIG.Gen}(1^{\lambda})$ and $\mathsf{tk}^{\mathsf{U}} \leftarrow \{0,1\}^{\lambda}$.
- Sample $\psi \in \{0,1\}^{\lambda}$ and set $\beta = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{tk}^{\mathsf{U}}; \psi)$.
- Sample $\omega \in \{0,1\}^{\lambda}$ and set $\zeta = \mathsf{Com}(\mathsf{tk}^{\mathsf{U}}; \omega)$.
- Let $x = (\mathsf{pk}^{\mathsf{A}}, \mathsf{pk}^{\mathsf{U}}, \beta, \zeta, U)$ and $w = (\mathsf{sk}^{\mathsf{U}}, \mathsf{tk}^{\mathsf{U}}, \psi, \omega)$.
- Create a proof $\pi := \mathsf{zkSNARK.Prove}(\mathsf{crs}_{\mathsf{GEN}}, x, w)$ for :

$$\begin{aligned}\mathsf{GEN} = \Big\{ (x, w) \mid\ & \beta = \mathsf{TPKE.Enc}(\mathsf{pk}^{\mathsf{A}}, \mathsf{tk}^{\mathsf{U}}; \psi) \\ & \wedge (\mathsf{sk}^{\mathsf{U}}, \mathsf{pk}^{\mathsf{U}}) \in \mathcal{R} \\ & \wedge \zeta = \mathsf{Com.Open}(\zeta, \mathsf{tk}^{\mathsf{U}}; \omega) = 1 \Big\}\end{aligned}$$

$U \longrightarrow \mathsf{CA}$: $\mathsf{pk}^{\mathsf{U}}, \beta, \zeta, \mathsf{doc}, \mathsf{attr}$

Cerification Authority CA:

- Validate doc w.r.t. issuance policy and attributes attr.
- If $\mathsf{zkSNARK.Verify}(\mathsf{crs}_{\mathsf{GEN}}, (\mathsf{pk}^{\mathsf{A}}, \mathsf{pk}^{\mathsf{U}}, \beta, \zeta, U), \pi) = 1$:
  - Create a credential $\mathsf{cred}^{\mathsf{U}} = \mathsf{SIG.Sign}(\mathsf{pk}^{\mathsf{U}} \| \zeta \| \mathsf{attr})$.
  - Store mapping from $U \longrightarrow (\mathsf{pk}^{\mathsf{U}}, \beta, \mathsf{doc})$.

$\mathsf{CA} \longrightarrow U$: $\mathsf{cred}^{\mathsf{U}}$

Figure 6: Credential Generation Protocol

After detecting credential theft, the user informs the CA that its credential has been stolen, and then the CA revokes this credential and grants a new one to the user[8]. In addition, the user also provides the CA with tk^{U}, using which all wallets that were verified with the revoked credential are traced and repudiated. Note that the user loses its privacy in this process, but we argue that privacy was already compromised when the credential was stolen since the adversary learns tk^{U}.

B Extensions to ZEBRA

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

- We've considered a simple issuance model where a CA is trusted to issue credentials (Figure 6). Credential issuance is orthogonal to our work and our credential generation protocol can easily be extended to support different issuance models as our CA does not store any private state and we can easily support different flavors of signature schemes due to the use of general-purpose zkSNARKs. Thus, ZEBRA's credential issuance can be made decentralized [71, 121], legacy-compatible [95, 152, 153], threshold/hierarchical [20, 29, 128], etc.
- We observe that ZEBRA can provide a lot of flexibility in limiting the use of credential. Each application can *independently* set its own credential usage limit *after the credential has already been issued* without any changes. This is achieved by having each application choose a *disjoint* range of tracing nonces $\{\eta_i\}$ for which the tracing token is valid, and reject transactions with

[8] Appropriate validation checks and rate-limiting must be applied here to prevent denial of service but we view this as tangential to the focus of this work.

repeated tracing tokens[9]. This is especially important for applications like decentralized voting and proof-of-personhood, where the credential should only be used once *per application*.

- We consider a simple layout for our credentials, where the attributes are stored as leaves of a Merkle tree. It is straightforward to extend our scheme to support the credential/claim schemas defined by W3C [50] and iden3 [86, 119] for interoperability.
- In our simple scheme, we focus on the single chain setting, but in practice, we imagine applications on multiple chains that rely on the same credential but have different sets of auditors. In such a setting, it is important that auditors of a chain can only audit transactions and track credential usage within that chain. The former is straightforward to support by having the audit token encrypted under the appropriate public key. For the latter, each chain could require that the users prove that they used $\mathsf{KDF}(\mathsf{tk}^{\mathsf{U}})$ as the tracing key, where KDF is a *domain-separated* key-derivation function (KDF) specific to that chain. This takes care of audits and tracing within a chain, but we also need to trace wallets across all chains to handle credential revocation. To this end, we introduce a linking committee, which can be a *subset* of all auditors, provided that it is larger than the sets of auditors per chain. At the time of credential issuance, the user now provides the CA with an encryption of its tracing key under the public key of the linking committee.
- 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, 80, 96] to protect against *mobile* adversaries that can eventually corrupt all parties over time.

C Implementation

We benchmark[10] the following aspects of ZEBRA:

- *Credential Verification Contracts*: we implement the credential verification contracts in Solidity for single, batched-L1 and batched-L2 verification. Each contract requires a Groth16 verification key that is circuit-dependent to verify Groth16 proofs on-chain. For the sake of benchmarking, we use verification keys of dummy circuits that have the same number of public inputs as the actual circuits. This gives us accurate gas costs as Groth16 verification only depends on the number of public inputs and is independent of the circuit.
- *User Overhead*: we need the number of R1CS constraints in the credential verification circuit to benchmark the Groth16 proof generation overhead, which is the bottleneck for user computation. All primitives we use in § 6.1 except the TPKE scheme are already implemented in `circomlib` [83]. We additionally implement the Cramer-Shoup TPKE scheme [39] in `circom` [84], and then using the constraints for the primitives, we infer the constraints for each sub-circuit of the credential verification circuit, and from that, its total constraints.
- *Aggregator Overhead*: aggregator computation requires the following components: (i) Groth16 prover and constraints for its verifier over the BN254 curve, and (ii) (data-parallel) Spartan prover and constraints for its verifier over the Grumpkin curve. `arkworks` [8] already had the implementation of (i) and we implemented (ii) in `arkworks`. We also optimize constraints for multi-scalar multiplications (MSMs) in `arkworks` by 2.05× (Appendix D). Our `arkworks` implementation benchmarks the time to generate a Spartan proof that verifies the specified number of user proofs, and outputs the number of R1CS constraints required to verify the Spartan proof. These constraints are then used to benchmark the Groth16 proof generation overhead.

Given the total constraints for each Groth16 circuit, we benchmark it with `gnark` [49] as it has the fastest Groth16 prover [102].

D 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× from 3840ℓ to 1877ℓ.

E 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

[9] This can be done efficiently using a Merkle tree, similar to how Tornado Cash [113] and ZCash [14] reject repeated nullifiers.

[10] Implementation URL: https://github.com/deevashwer/zebra

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 [73] against a static adversary corrupting parties at the beginning of the protocol. 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.

The real execution. 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.

The ideal 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 1. *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, $\mathsf{REAL}_{\Pi,\mathcal{A}}[1^\lambda, \vec{x}, z, M] \approx_c \mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}[1^\lambda, \vec{x}, z, M]$.

zkSNARK. 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 $\mathcal{R}_C = \{(x, w) \in \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l : C(x, w) = 0^l\}$ and a language $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists\, w \in \mathbb{F}^h, C(x, w) = 0^l\}$. A zkSNARK for $\mathbb{F}$ is a triple of PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda, C) \to (\mathsf{crs}, \mathsf{td})$: Takes as input a security parameter and circuit description C, and outputs a common reference string crs and trapdoor td.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: 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 indicating whether verification has passed or failed.
- $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, x) \to \pi$: Takes as input crs, trapdoor td and statement x and outputs a *simulated* proof π.

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

- *Perfect Completeness.* A zkSNARK is perfectly complete if for any $(x, w) \in \mathcal{R}_C$, we have

$$\Pr\left[\begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, C) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array} : \mathsf{Verify}(\mathsf{crs}, x, w) = 1\right] = 1$$

- *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 $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, C)$, $(x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ 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:

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

- *Weak Simulation-Extractability.* For every PPT adversary $\mathcal{A}$, there exists a PPT extractor ε such that

$$\Pr\left[\begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, \cdot)}(\mathsf{crs}) \\ w \leftarrow \varepsilon \end{array} : \begin{array}{r} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\, (x, w) \notin \mathcal{R} \\ \wedge\, x \notin Q \end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where $\mathcal{A}$ has oracle access to $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, \cdot)$, and Q is a list of queries made by the adversary.

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 pk, $\mathsf{Sign}(\mathsf{sk}, m)$ outputs a signature σ on the message m, and $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ outputs either 1 to indicate that σ is a valid signature on m, or 0 otherwise.

Definition 2. *A signature scheme* (Gen, Sign, Verify) *is existentially unforgeable under chosen message attacks if for any PPT adversary $\mathcal{A}$, the following probability is negligible*

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

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 [39]. However, we restrict protocols Π_{TPKE} for TPKE to consist five PPT algorithms (Setup, Enc, Dec, Verify, Combine) as defined below:

- $\mathsf{Setup}(1^{\kappa}, n, t) \longrightarrow \{\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\}$: Takes as input a security parameter and positive integers n, t and outputs a public, verification key and secret keys with threshold $t + 1$.
- $\mathsf{Enc}(\mathsf{pk}, m; \rho) \longrightarrow \mathsf{ct}$: Takes as input the public key pk, a message m and randomness ρ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{ct}, sk_i) \longrightarrow m_i$: 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.
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, m_i) \longrightarrow \{0, 1\}$: 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.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \{m_i\}_{i \in S \subseteq [n]}) \longrightarrow m$ 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 1[11].

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

Parties: Encrypting user E and Servers $(S_1, \ldots, S_n)$.

Parameters: Space of receipts C, number of servers n and threshold $0 < t \leq n$.

- **Setup.** Adversary specifies a distribution Γ over 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 m to the servers. Else, send $\perp$ to the servers.

Figure 7: Ideal Functionality for Threshold Public-Key Encryption scheme.

Canetti and Goldwasser show that the Cramer-Shoup cryptosystem [53] can be modified by having servers prove correctness of partial decryptions using zero knowledge proofs to achieve the above definition.

F 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 (smart contract) does not have any private state and is trusted to faithfully follow the protocol.

We provide a proof sketch for the security of our scheme by describing a simulator S that interacts with the ideal functionality (Figure 2) such that the transcript of $\mathcal{A}$ interacting with honest parties in the real world is computationally indistinguishable from the transcript produced when $\mathcal{A}$ interacts with the simulator. We use a simple verification policy: *User must have been issued a credential by* CA *and the credential must not been revoked.* But this can easily be extended to any arbitrary policy expressed as an arithmetic circuit depending on the needs of the application.

F.1 Security against a malicious adversary corrupting auditors and users

Theorem 3. *ZEBRA securely emulates the ideal functionality $\mathcal{F}$ (Figure 2) for any PPT malicious adversary $\mathcal{A}$, corrupting t auditors, and an arbitrary number of users, provided straight-line simulation-extractable zkNIAoK, collision resitant PRFs, and adaptive CCA (CCA2) secure Public-Key Encryption schemes exist in the Random Oracle Model.*

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. We now describe the simulator.

Setup. Sim runs the TPKE simulator $\mathsf{Sim}_{\mathsf{TPKE}}$ to simulate the view of $\mathcal{A}$ during the setup phase of TPKE $\mathcal{F}_{\mathsf{TPKE}}$ which contains the public key and secret keys of malicious parties $\{\mathsf{pk}^{\mathsf{A}}, \{\mathsf{sk}^{\mathsf{A}}_i\}_{i \in S}\}$. The simulator also runs $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIAoK.Setup}(1^{\lambda}, \cdot)$ for the relevant circuits and publishes the crs. Next, it simulates the CA by sampling $(\mathsf{pk}^{\mathsf{CA}}, \mathsf{sk}^{\mathsf{CA}}) \leftarrow \mathsf{Sig.Gen}(1^{\lambda})$ and publishing $\mathsf{pk}^{\mathsf{CA}}$.

Credential Generation. When Sim receives a request for a credential from a corrupt user U containing $(\mathsf{pk}^{\mathsf{U}}, \beta, \zeta, \mathsf{doc}, \mathsf{attr}, \pi)$ (Figure 6), it checks if π verifies successfully. If so, it then decrypts β which is possible because Sim knows $t + 1$ simulated auditor secret keys to learn tk^{U}. Sim then sends $(\mathsf{ReqCred}, \mathsf{doc}, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr})$ to $\mathcal{F}$ on behalf of U. If Sim receives accept as response from $\mathcal{F}$ indicating the credential was approved, then Sim creates a credential using the secret key of the simulated CA by faithfully following the protocol (Figure 6) and sends it to U. Otherwise Sim sends $\perp$ to U. During this time Sim also stores a user–key mapping between the corrupted party U and its public key pk^{U} and tracing key tk^{U}. This is possible because Sim knows $t + 1$ simulated auditor secret keys and can hence decrypt β.

Pseudonym Verification. When a corrupt user U attempts to verify a pseudonym, it sends a proof π along with an audit token α, tracing token γ and wallet address pk^{W}, Sim simulates SmrtCont by faithfully following the protocol. That is, Sim adds pk^{W} to the

[11] In the original paper the authors actually define and construct protocols satisfying the stronger Universally Composable notion of security [38]

public list of verified pseudonyms, if the proof provided passes verification.

Sim then extracts pk^{U} by running $\mathsf{Sim}_{\mathsf{TPKE}}$ on the audit token which outputs the underlying message pk^{U} to be sent to $\mathcal{F}_{\mathsf{TPKE}}$. It then finds the corresponding user by checking the user–key mapping it created earlier. This is sufficient for Sim to send $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, \eta)$ to $\mathcal{F}$ on behalf of U.

When an honest party verifies a pseudonym, Sim must simulate the view of $\mathcal{A}$ which contains a proof π, audit token α, and tracking token γ. Note that Sim does not know the identity of the honest party that requested verification. Care must be taken as when an honest party is audited, the tracing tokens of all wallets verified by an honest user must be consistent with the tracing key and the audit token must decrypt to the corresponding public key.

When the ideal functionality announces that a pseudonym pk^{W} has been verified by an honest party, Sim receives γ. Next, it runs the simulator of the TPKE scheme $\mathsf{Sim}_{\mathsf{TPKE}}$ to compute α which is used to simulate the adversary's view of the audit token. The final component to be created is a proof π as described in Figure 3, which is again simulated using $\mathsf{SimProve}(x, \mathsf{td})$ as it does not know the witness attributes attr used by the honest party which includes tk^{U} used to compute γ. In particular the statement is $x = (\mathsf{pk}^{\mathsf{CA}}, \mathsf{pk}^{\mathsf{W}}, \phi, \mathsf{pk}^{\mathsf{A}}, \mathsf{rt}^{\mathsf{rl}}, \alpha, \gamma)$, where α and γ are computed as above.

Audit. When an audit of a pseudonym pk^{W} occurs, Sim forwards all messages from corrupt auditors to $\mathcal{F}$ and receives the user(s) that requested a verification of pk^{W} from the ideal functionality (in some canonical ordering). Sim now needs to simulate the view of $\mathcal{A}$ during threshold decryption of audit tokens. For all of the tokens created by corrupt parties, Sim follows the protocol faithfully as the tokens indeed contain the corresponding party's public key. For the tokens corresponding to honest parties, Sim first samples a fresh key pair $(\mathsf{pk}^{\mathsf{U}}, \mathsf{sk}^{\mathsf{U}})$ for each user U that has not previously appeared in an audit. In all future audits, the same key pair will be used for that particular user. Finally, Sim runs $\mathsf{Sim}_{\mathsf{TPKE}}$ with the corresponding honest party's public key pk^{U} (which it sampled) as input so as to obtain the same as the decrypted message.

Trace. When a user's pseudonyms are being traced, Sim forwards all messages from auditors to the ideal functionality and receives tk^{U} in return which it forwards to the corrupt auditors.

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 faithfully following the protocol and adding the public key of the corresponding user to the merkle tree $\mathsf{MT}^{\mathsf{rl}}$.

Repudiation. When a pseudonym pk^{W} is repudiated, Sim forwards forwards all messages from auditors to the ideal functionality.

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 to obtain the td instead of the trusted party. This hybrid is computationally indistinguishable from the previous hybrid.

In the next hybrid, Sim simulates the CA as described during credential generation and revocation. This is computationally 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 simulating the trusted party who distributes keys and then responding to audit requests as described in the protocol. This hybrid is computationally indistinguishable from 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 as described earlier. First note that even though Sim simulates proofs for statements $\{x_1, \ldots, x_k\}$ (say), the Weak Simulation-Extractability property of the NIAoK guarantees that for any proof provided by the adversary for a statement $x \notin \{x_1, \ldots, x_k\}$, the PPT extractor $\varepsilon_{\mathcal{A}}$ outputs a valid witness. Indeed, the adversary may be able to create new proofs for statements $x \in \{x_1, \ldots, x_k\}$ such that verification passes but the extractor does now output a valid witness. However, this is not an issue as Sim only simulates proofs to verify a pseudonym pk^{W} that has already been verified by an honest party. $\mathcal{A}$ can re-submit verification requests for pk^{W} but it will simply be rejected by the smart contract as it is is either already verified or repudiated. 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 7, this hybrid is computationally indistinguishable from the previous hybrid.

Finally, Sim simulates SmrtCont. It can be seen that the constraints in Figure 3 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[12]. This can be determined by running $\mathsf{Sim}_{\mathsf{TPKE}}$ and decrypting the audit token to obtain pk^{U}. Due to the proof of knowledge and soundness property of the NIAoK scheme, there exists an extractor that outputs the witness. We require that this extractor can extract from polynomially many instances in polynomial time. This is not always true, as if rewinding is involved the simulator may run in time exponential in the number of proofs. However, we demand that the NIAoK has a straight-line extractor and this is true for the groth16 and Spartan proof systems in the Algebraic Group Model. This ensures that the extractor and hence the simulator will run in polynomial time. From the EUF-CMA property, collision resistance of CRH, perfect correctness of the TPKE scheme and the security property of accumulators, the same pk^{U} that results from decrypting the audit token must have been awarded a credential by CA that has also not been revoked.

F.2 Security against a semi-honest CA

Theorem 4. *ZEBRA securely emulates the ideal functionality $\mathcal{F}$ (Figure 2) for any PPT semi-honest adversary $\mathcal{A}$, corrupting the CA, provided straight-line simulation-extractable zkNIAoK, collision resitant PRFs, and adaptive CCA (CCA2) secure Public-Key Encryption schemes exist in the Random Oracle Model.*

[12] 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'.

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 and tracing key for each request that arrives from an honest user via the ideal functionality and then preparing a proof as done in Figure 6 and attaching doc, attr that were received from $\mathcal{F}$. This is a good simulator as the keys generated are indistinguishable from those generated by an honest party in a real execution of the protocol.

F.3 Privacy against colluding CA, Auditors and Users

Finally, we argue privacy against an adversary $\mathcal{A}$ that corrupts the CA, up to t auditors and an arbitrary number of users. To do so, we introduce a weakened ideal functionality $\hat{\mathcal{F}}$ which fully captures the capabilities of a malicious CA. An important point to note here is that the CA is the only party who has a mapping between users and their public keys. Therefore, a malicious CA could frame honest users as the proponents of fraudulent transactions. However, even a malicious CA cannot violate the *privacy* of transactions in our system viz. the adversary cannot identify the user who verified a pseudonym without the help of a threshold number of auditors.

Importantly, we note that there are no longer meaningful security guarantees that can be captured for malicious users. Hence, we enforce that the weaker ideal functionality's (Figure 8) correctness and privacy guarantees of credential generation, pseudonym verification, auditing, tracing, revocation and repudiation only apply on honest users. When the adversary (malicious user) interacts with the ideal functionality we provide no guarantees. On top of this, we weaken audits and tracing as follows:

- During audits, the ideal functionality now takes as input a user U' from the adversary which is sent to auditors instead of the user U such that $(\mathsf{pk}^{\mathsf{W}}, U) \in \mathcal{D}$.
- Similarly, during tracing, the adversary is given the option to replace the tracing key with another key of it's own choice.

Note that despite the adversary being more powerful now, there is no way for the adversary to identify the user who verified a pseudonym without an audit for which it requires the help of a threshold number of auditors. We will now provide a proof sketch for why our scheme securely emulates this ideal functionality.

Theorem 5. *ZEBRA securely emulates the ideal functionality $\hat{\mathcal{F}}$ (Figure 8) for any PPT malicious adversary $\mathcal{A}$, corrupting the CA, t auditors, and an arbitrary number of users, provided straight-line simulation-extractable zkNIAoK, collision resitant PRFs, and adaptive CCA (CCA2) secure Public-Key Encryption schemes exist in the Random Oracle Model.*

$\hat{\mathcal{F}}$

Parties: Users $\{U_1, \ldots, U_N\}$, Certificate Authority CA and Auditors $(\mathsf{Aud}_1, \ldots, \mathsf{Aud}_n)$.

Parameters: Verification policy ϕ, pseudonym space $\mathcal{N}$, user space $\mathcal{U}$, and a collision resistant PRF CR-PRF.

- **Setup.** On input (Setup, ϕ) from the organization, publish a verification policy ϕ. Maintain a database $\mathcal{D}$ of tuples from the space $\mathcal{N} \times \mathcal{U}$, and a database of banned pseudonyms $\mathcal{D}_B$.
- **Credential Generation.** On input $(\mathsf{ReqCred}, \mathsf{doc}, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr})$ from an honest user U, forward $(\mathsf{doc}, U, \mathsf{attr})$ to CA. On input $(\mathsf{AprCred}, U)$ from CA, add $(U, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr})$ to $\mathcal{L}$, the tuple of users who have been awarded credentials, their tracing key and their attributes. Send accept to U.
- **Pseudonym Verification.** On input $(\mathsf{ReqVer}, \mathsf{pk}^{\mathsf{W}}, \eta)$ from an honest user U, if $(U, \mathsf{tk}^{\mathsf{U}}, \mathsf{attr}) \in \mathcal{L}$, $\phi(\mathsf{attr}) = 1$, $(\mathsf{pk}^{\mathsf{W}}, \cdot) \notin \mathcal{D}$, and $\mathsf{pk}^{\mathsf{W}} \notin \mathcal{D}_B$, then publish $(\mathsf{pk}^{\mathsf{W}}, \eta, \mathsf{CR\text{-}PRF}(\mathsf{tk}^{\mathsf{U}}, \eta))$ and add $(\mathsf{pk}^{\mathsf{W}}, U)$ to $\mathcal{D}$. The adversary can choose to censor by rejecting valid requests.
- **Audit.** On input $(\mathsf{Audit}, \mathsf{pk}^{\mathsf{W}})$ from $t+1$ auditors and CA, if there exists an entry of the form $(\mathsf{pk}^{\mathsf{W}}, U) \in \mathcal{D}$, return U to the adversary. The adversary then responds with any user U' (possibly U) of its choice (or $\perp$) which is then sent to all auditors.
- **Trace.** On input (Trace, U) of an honest party U, from $t+1$ auditors and CA, the adversary receives U and can respond in three possible ways:
  - Choose any $\mathsf{tk}^{\mathsf{U}'}$, which is then sent to auditors.
  - If $(U, \mathsf{tk}^{\mathsf{U}}, \cdot) \in \mathcal{L}$, then the auditor can choose to have auditors receive the honest party U's tk^{U}.
  - Send $\perp$ to auditors.
- **Revoke.** On input (Revoke, U) of an honest user U from CA,
  - If $(U, \cdot, \cdot) \in \mathcal{L}$, remove the entry containing U from $\mathcal{L}$.
  - Else, return $\perp$ to CA.
- **Repudiate.** On input $(\mathsf{Repudiate}, \mathsf{pk}^{\mathsf{W}})$ from $t+1$ auditors, delete all records of the form $(\mathsf{pk}^{\mathsf{W}}, \cdot)$ from $\mathcal{D}$, and add pk^{W} to $\mathcal{D}_B$.

Figure 8: Ideal Functionality for an Anonymous Credential Scheme supporting Audits, Tracing, Revocation and Repudiation with a malicious CA.

Note that the simulator longer need to extract the inputs of malicious users. However, it does need to extract inputs from a malicious CA. The simulator simulates the honest auditors and users as done previously. During credential generation, if and only if the CA issues a valid signature, the simulator forwards an approve credential message to the ideal functionality. Next, during the audit phase, the simulates auditors decrypting the audit token to recover the public key of the user and send this to the adversary. The adversary can now choose to respond with any user of it's choice or not respond at all which the simulator forwards to the ideal functionality.

When tracing occurs, we need to argue that the only power of the adversary is to either allow all auditors to learn the tk^{U} corresponding to the honest party being traced or choose some $\mathsf{tk}^{\mathsf{U}'}$ which will be revealed instead. During the simulation, the malicious CA will supply all information it has pertaining to user U, which includes the tracing token β, public key pk^{U}, and the proof sent by the user as outlined in § 5.4. All auditors verify this proof before decrypting β. Now note that the simulator has simulated all the honest users and auditors and therefore knows can always decrypt the ciphertext. Furthermore, since the adversary provides a proof of knowledge of randomness used during encryption and the cramer-shoup system is perfectly correct, the ciphertext is valid with overwhelming probability. Now the simulator can simply decrypt to recover the tracing key $\mathsf{tk}^{\tilde{\mathsf{U}}}$. Note that this $\mathsf{tk}^{\tilde{\mathsf{U}}}$ is either the tracing key chosen by the simulator for the honest party U or something chosen by the adversary. Note that it can't have been a *related-key* because we use a CCA2 secure encryption scheme and the proof is weakly simulation extractable. When simulating the honest party, recall that the simulator ran $\mathsf{Sim}_{\mathsf{TPKE}}$ to simulate the honest party's ciphertexts in such a way that they can be *equivocated* later. Sim now runs $\mathsf{Sim}_{\mathsf{TPKE}}$ with the corresponding honest party's tracing key tk^{U} (which it received from the ideal functionality) as input in order to convince the adversary that the decrypted message was tk^{U}. Importantly, such a simulation strategy will only work if the ciphertext retains semantic security until the user is traced. This holds true because we use a CCA2 secure encryption scheme, despite being able query the decryption oracle on any ciphertexts the adversary sees.

Revocation can be handled by relying on the extractor of the zkNIAoK to extract the user being revoked and then sending this to the ideal functionality. Repudiation can be handled as done previously.

G 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) [12, 13, 61]. 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.

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 [12], except that we use weak-SE NIAoKs in place of SE NIAoKs. For completeness, we describe the scheme in Figure 9.

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

Theorem 6. *The protocol described in Figure 9 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:

(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 9, 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 $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIAoK.Setup}(1^\lambda, \mathcal{R})$
(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 $\sigma \leftarrow \mathsf{NIAoK.Sim}(\mathsf{crs}, \mathsf{td}, (K, Y, m))$ 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 9 is a signature scheme satisfying EUF-CMA security. □