EasyChair Preprint

№ 15829

Beyond the Blockchain Address: Zero-Knowledge Address Abstraction

Sanghyeon Park, Jeonghyuk Lee, Seunghwa Lee, Jung Hyun Chun, Hyeonmyeong Cho, Mingi Kim, Hyun Ki Cho and Soo-Mook Moon

February 14, 2025

Beyond the Blockchain Address: Zero-Knowledge Address Abstraction

Sanghyeon Park
Seoul National University
Seoul, Republic of Korea
lukepark@snu.ac.kr

Jeonghyuk Lee
CPLABS Incorporated
Gyeonggi, Republic of Korea
ljh@cplabs.io

Seunghwa Lee
Kookmin University
Seoul, Republic of Korea
ttyhgo@kookmin.ac.kr

Jung Hyun Chun
Hanyang University
Seoul, Republic of Korea

Hyeonmyeong Cho
CPLABS Incorporated
Gyeonggi, Republic of Korea

Mingi Kim
Sungkyunkwan University
Seoul, Republic of Korea

Hyun Ki Cho
CPLABS Incorporated
Gyeonggi, Republic of Korea

Soo-Mook Moon
Seoul National University
Seoul, Republic of Korea
smoon@snu.ac.kr

Abstract

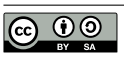
Merging Internet (web2) identities with blockchain (web3) identities is increasingly important for enhancing user experience and ensuring regulatory compliance. However, conventional solutions that map web2 identities to web3 accounts often lead to privacy concerns and fragmented identifiers across networks. To address these challenges, we propose a new identity scheme named *Address Abstraction* (AA), which redefines blockchain address and signing systems while preserving key properties: uniqueness, immutability, and privacy-preservation. This approach eliminates the limitations of chain-specific identity systems, enabling users to interact with multiple blockchains using their web2 certificates and unified identifiers. This chain-agnostic identifier also promotes cross-chain compatibility. We further present *Zero-Knowledge Address Abstraction* (zkAA), an implementation of AA that uses zero-knowledge proofs to uphold AA's core properties. Additionally, a proof aggregation technique combines multiple proofs into one, achieving approximately 5.5 times gas cost savings during verification in real-world scenarios. As of August 2024, zkAA with proof aggregation incurs an additional cost of only $0.66 per transaction on Ethereum.

CCS Concepts

• **Security and privacy** → **Authentication**; **Privacy-preserving protocols**; • **Computer systems organization** → *Distributed architectures.*

Keywords

Digital Identification, Zero-Knowledge Proofs, Blockchain.



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707839

ACM Reference Format:
Sanghyeon Park, Jeonghyuk Lee, Seunghwa Lee, Jung Hyun Chun, Hyeonmyeong Cho, Mingi Kim, Hyun Ki Cho, and Soo-Mook Moon. 2025. Beyond the Blockchain Address: Zero-Knowledge Address Abstraction . In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707839

1 Introduction

The rapid growth of various blockchains has introduced challenges in wallet setup and key management due to cryptography differences, particularly in address and signature systems. This results in inconsistent identifiers across chains, complicating user experience. Meanwhile, decentralized applications (dApps) often rely on conventional identification systems, such as social login [14], for efficiency and secure management. They often include Know-Your-Customer (KYC) processes, which are important for improving security against harmful activities and malicious users [8, 19].

Consequently, the need for integrating identity management across both multiple blockchains and traditional systems has become increasingly apparent. However, previous attempts at creating the unified identity have encountered numerous challenges.

1.1 Related Work and Challenges

Integrating web3 identities into web2 systems is relatively straightforward due to the centralized nature of web2. However, incorporating web2 identities into web3 presents challenges, such as identifier fragmentation and privacy concerns. A common strategy is creating a bi-directional mapping between different identifiers, that is, linking web2 authentication tokens to their corresponding web3 blockchain addresses [22, 36, 42, 44]. However, this approach merely links rather than unifying identities, requiring careful management of multiple credentials across web2 and web3. The issue is further compounded in multi-chain dApps, where managing multiple chain-specific addresses results in fragmented identifiers.

Zero-Knowledge Proofs (ZKPs) have been utilized in various studies to enhance privacy in linking. Holonym [20] uses ZKP to authenticate the possession of web2 credentials, specifically JSON

Web Tokens [18]. Notebook [24] employs ZKP to verify credentials signed by third-party entities. Nevertheless, these methods rely on the original blockchain address as the identifier, leading to potential management limitations. Sui zkLogin [39] and Aptos Keyless [2] link blockchain addresses with web2 credentials using ZKP, allowing credential-based control. However, these solutions depend on blockchain-specific cryptography, such as private key signing (though used ephemerally) and address schemes, and thus fail to offer a unified identifier across different webs.

Particle Network [28] and NEAR Protocol [27] offer chain abstraction [10] solutions, enabling accounts to interact across multiple blockchains. However, these systems essentially function as cross-chain message bridges, relying on each chain's cryptography and distinct identifiers, failing to provide a unified identifier. Furthermore, the limited list of chains supported by the service provider restricts users' ability to use their accounts or publish transactions on unsupported ones, thus diminishing user sovereignty.

1.2 Contributions

We propose the *Address Abstraction* (AA) scheme, along with its implementation, *zero-knowledge Address Abstraction* (zkAA), to address the aforementioned challenges—inconsistent identifiers across blockchains, fragmentation of user identities, and reliance on chain-specific cryptography. This presents an innovative method for managing identities, focusing on integration across multiple webs — web2s and web3s. The key features are as follows:

- In AA, a single credential suffices for all web2 and web3 networks, eliminating the need to manage multiple secrets.
- This chain-agnostic approach removes the need to map to chain-specific addresses, enabling a unified identifier across diverse networks and thereby enhancing interoperability. zkAA can operate on any blockchain supporting smart contracts capable of verifying zero-knowledge proofs.
- AA allows easy use of web2 features like recovery mechanisms and regulatory compliance within blockchains, without compromising the user's sovereign control over their credentials.
- zkAA employs zero-knowledge proofs to verify ownership of credential data (certificate) without exposing it.

We conduct an empirical evaluation of zkAA by implementing it on Ethereum [43] and layer-2 solutions like Arbitrum [31], Optimism [33], and Polygon [34]. Proof generation and on-chain verification are both fast and cost-effective, as shown in Section 6. We further explore the proof aggregation technique to improve cost-efficiency. By aggregating 15 proofs, zkAA incurred only an additional cost of $0.66 per transaction on Ethereum (and less than $0.01 on other chains), proving it a cost-effective alternative to traditional blockchain address and signature systems.

To facilitate further research, we have made the implementation and experimental code publicly available for replication at https://github.com/zkAA-onchain/.

2 Background

2.1 Accounts and Addresses

Given the widespread adoption of Ethereum [43], often used as a basis for many blockchain forks, we use it as the primary example to explain our approach. Ethereum categorizes accounts into two types: Externally Owned Accounts (EOAs) and Contract Accounts (CAs). EOAs are accounts governed by a private key, allowing users to sign transactions. CAs are accounts that contain programmatic logic and storage. Account abstraction [6] aims to blur the distinction between EOA and CA, enabling more advanced and customizable account behaviors. Regardless, an address remains the identifier for any account.

Table 1: Key Symbols

Symbol	Description
$\mathcal{R}_R, \mathcal{R}_P$	Relations for the registration and publication processes
crs_R, crs_P	Common Reference Strings for registration and publication
$\mathcal{A}, \mathcal{X}_{\mathcal{A}}$	Probabilistic Polynomial-Time adversary, Extractor for $\mathcal{A}$
$\mathcal{H}$	Cryptographic hash function
pk, sk	Public and secret keys of the institute
$cert$	Certificate issued by the institute
r	Random number sampled from the finite field[1] ($\mathbb{F}_p$)
id	Identifier calculated by $id \leftarrow \mathcal{H}(cert \|\| r)$
msg	Executable message
$\mathcal{M}$	Set of all executable messages
m	A unique sequence number for each msg per id
π_R, π_P	Proofs for registration and publication

Our Address Abstraction (AA) approach can be perceived as a specialized application of account abstraction, given its function in modifying the signature system linked to the account. However, beyond the scope of mere account abstraction, AA decouples user identifiers from blockchain addresses, enabling a transition to a consistent chain-agnostic identifier. This approach simplifies identity management across webs and enhances cross-chain compatibility by allowing users to interact through a single credential.

2.2 Decentralized Identifiers

Self-Sovereign Identity (SSI) allows individuals and entities to control their digital identities without relying on intermediaries. SSI's potential extends to fostering trust in digital interactions across diverse domains such as finance, healthcare, and supply chain management [35]. Decentralized Identifiers (DIDs) form the foundation, enabling verifiable identity management while minimizing data exposure [41]. Fundamentally, when credentials are treated as Verifiable Credentials (VCs) and their corresponding proofs as Verifiable Presentations (VPs), AA is closely aligned with the principles of DIDs. This demonstrates that zkAA, our implementation of the proposed AA scheme, can serve as a practical implementation of DIDs due to its low on-chain costs.

3 Cryptographic Preliminaries

The notations used in this paper are listed below, with key symbols summarized in Table 1. Let $\mathcal{R}$ represent a relation. Given that $(io, w) \in \mathcal{R}$, io refers to the public input/output, and w is the witness. The symbol crs signifies the public parameter known as the Common Reference String (CRS). We denote a Probabilistic Polynomial-Time (PPT) adversary by $\mathcal{A}$. The adversary's output is written as $y \leftarrow \mathcal{A}(x)$, where x is the input and y is the resulting

[1] In this paper, we use p = 21888242871839275222246405745257275088696311157297823662689037894645226208583, in alignment with the requirements of Ethereum's precompiled contracts [43].

output. An extractor, $\mathcal{X}_{\mathcal{A}}$, can compute a corresponding witness whenever the adversary $\mathcal{A}$ generates a valid argument.

Specifically, the symbols $\mathcal{R}_R$ and $\mathcal{R}_P$ denote the relations for the registration and publication processes in our scheme, respectively. $\mathcal{R}_R$ is represented as the tuple $((id, pk), cert||r)$, where id denotes the abstract identifier, pk is the institute's public key, $cert$ is the certificate issued by the institute, and r is a random value. The abstract identifier id is calculated by hashing the concatenation of $cert$ and r using the cryptographic hash function $\mathcal{H}$, i.e., $id \leftarrow \mathcal{H}(cert||r)$. Similarly, $\mathcal{R}_P$ is expressed as the tuple $((id, msg, m), cert||r)$, where msg represents the executable message the user intends to perform. The message's sequence number for each id, denoted by m, functions similarly to a nonce (a monotonically increasing counter) in traditional blockchain accounts. The symbol $\mathcal{M}$ refers to the complete set of executable messages.

The notation $x \xleftarrow{\$} X$ represents that x is uniformly sampled at random from the set X. The order-related notation $a \succ b$ implies that a cannot precede b. Lastly, we use the symbol $\perp$ to indicate an undefined or disregarded value.

3.1 Digital Signatures

Digital signatures are a fundamental component of digital identity and, nowadays, are widely used in technologies such as JSON Web Tokens (JWT) [18]. Their primary function is to verify the origin of data, thereby authenticating it.

The digital signature protocol sig used in this paper comprises three operations: sig.KeyGen, sig.Sign, and sig.Verify.

- $(pk, sk) \leftarrow \mathsf{sig.KeyGen}(\cdot)$: This operation generates a pair of keys; a secret key sk and a corresponding public key pk.
- $\sigma \leftarrow \mathsf{sig.Sign}(sk, q)$: This operation takes the secret key sk and a message q as inputs and outputs a signature σ.
- $0/1 \leftarrow \mathsf{sig.Verify}(pk, q, \sigma)$: This operation takes the public key pk, a message q, and a signature σ as inputs and outputs 1 if the signature is valid and 0 otherwise.

The digital signature scheme used in this work is unforgeable under chosen-message attacks:

Definition 3.1 (Unforgeability). The following must be satisfied for all PPT adversaries $\mathcal{A}$. Q is the set of message queries that $\mathcal{A}$ has already seen valid signatures for through sig.Sign from sk.

$$\Pr\left[\begin{array}{l}(pk, sk) \leftarrow \mathsf{sig.KeyGen}(\cdot); (q, \sigma) \leftarrow \mathcal{A}(pk) : \\ \mathsf{sig.Verify}(pk, q, \sigma) = 1 \wedge q \notin Q\end{array}\right] \approx 0$$

(Unforgeability)

Unforgeability denotes that it should be computationally infeasible for a probabilistic polynomial-time adversary to generate a valid signature for any message without access to the secret key.

3.2 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKPs) provide a method for verifying the authenticity of information without revealing underlying data. Since their inception, significant strides have been made in the field, notably the development of Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) [16]. zk-SNARKs offer succinct verification, making them well-suited for blockchain.

The ZKP typically involves three operations: zkp.Setup, zkp.Prove, and zkp.Verify.

- $crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R})$: This operation takes a relation $\mathcal{R}$ as input and produces a common reference string crs, which serves as a public parameter for the proof generation and verification.
- $\pi \leftarrow \mathsf{zkp.Prove}(crs, io; w)$: This operation generates a proof π given the crs, input/output io, and witness w.
- $0/1 \leftarrow \mathsf{zkp.Verify}(crs, io, \pi)$: The function outputs 1 for success and 0 for failure of the verification.

Our scheme employs the zk-SNARKs protocol that satisfies the properties outlined in Groth16 [16]:

Definition 3.2 (Completeness). For a relation $\mathcal{R}$, every $(io, w) \in \mathcal{R}$ satisfies the following:

$$\Pr\left[\begin{array}{l}crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R}); \pi \leftarrow \mathsf{zkp.Prove}(crs, io; w) : \\ \mathsf{zkp.Verify}(crs, io, \pi) = 1\end{array}\right] = 1$$

(Perfect Completeness)

This property implies that if a statement is true, a proof using witness w will always be accepted by an honest verifier.

Definition 3.3 (Knowledge Soundness). For all PPT adversaries $\mathcal{A}$, there exists a PPT extractor $\mathcal{X}_{\mathcal{A}}$ that satisfies the following:

$$\Pr\left[\begin{array}{l}crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R}); ((io, \pi); w) \leftarrow (\mathcal{A}||\mathcal{X}_{\mathcal{A}})(crs) : \\ \mathsf{zkp.Verify}(crs, io, \pi) = 1 \wedge (io, w) \notin \mathcal{R}\end{array}\right] \approx 0$$

(Computational Knowledge Soundness)

This property is defined by the requirement that it should be computationally infeasible for a prover to convince a verifier of the truth of a false statement without the knowledge of the witness.

Definition 3.4 (Zero-Knowledge). The following must be satisfied for all PPT adversaries $\mathcal{A}$ and $(io, w) \in \mathcal{R}$.

$$\Pr\left[\begin{array}{l}crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R}); \pi \leftarrow \mathsf{zkp.Prove}(crs, io; w) : \\ \mathcal{A}(crs, \pi) = 1\end{array}\right] =$$

$$\Pr\left[\begin{array}{l}(crs, \tau) \leftarrow \mathsf{zkp.SimSetup}(\mathcal{R}); \pi \leftarrow \mathsf{zkp.SimProve}(crs, \tau, io) : \\ \mathcal{A}(crs, \pi) = 1\end{array}\right]$$

(Perfect Zero-Knowledge)

This denotes that the distribution of a regular proof, generated with knowledge of the witness, is indistinguishable from that of a proof simulated by zkp.SimProve without the witness. The simulator utilizes the trapdoor τ, incorporated during the crs generation via zkp.SimSetup. This indicates that the proof does not leak any information aside from the truth of the statement.

3.3 Proof Aggregation

Proof aggregation is an optimization technique that combines multiple zero-knowledge proofs into a single proof. This reduces the verification overhead, as the verifier only needs to check a single proof rather than verifying each proof individually.

Our implementation utilizes recursive aggregation [11, 21], using recursive circuits that can verify both new statements and previously aggregated proofs. Specifically, we leverage PLONK [12], a zk-SNARK system with a universal trusted setup, which makes it well-suited for efficient proof aggregation.

3.4 Security Assumptions

The reliable operation of our proposed system is based on the following security assumptions:

- The hash function $\mathcal{H}$ is modeled as a Random Oracle [4], meaning it produces outputs that are indistinguishable from random values for each unique input.
- The blockchain and deployed smart contracts are secure against attacks like denial-of-service [25]. Moreover, the possibility of chain reorganization is considered negligible. This ensures the reliable transaction execution and data access.
- The public key pk of the certificate-issuing institute is publicly available by any entity, as it is recorded in a smart contract.
- The transmission of web2 secrets (the certificate $cert$) between the institute and the client is secure and confidential.

4 Address Abstraction

The core concept of Address Abstraction (AA) scheme is the use of a certificate issued by a web2 institute, denoted as $cert$, to generate a unified and unique identifier, $id \leftarrow \mathcal{H}(cert||r)$. The secret $cert$ and the corresponding abstract identifier id play key roles in generating abstract transactions, msgs, similar to how a private key and its corresponding blockchain address are used for transaction signing.

Using a random number r in generating id effectively prevents the risk of rainbow table attacks [30]. Furthermore, by integrating this user-selected r, the secret of the identity $(cert||r)$ is kept confidential, even from the issuer of the $cert$, thereby ensuring that it remains exclusively known to the user.

The AA scheme consists of two key processes: *registration* and *publication*. During the registration phase, the validity of the id is verified and recorded on a smart contract. After this initial step, subsequent transactions (publications of msg) can be processed efficiently by referencing the stored validity status, instead of repeated id validity verifications for each msg. This distinction also enables diverse management of registration statuses, such as expirations, adherence to regulations like the Office of Foreign Assets Control (OFAC) [32], and compliance with standards like the General Data Protection Regulation (GDPR) [40], by allowing the contract to dynamically update validity statuses to reflect changing conditions.

4.1 Definitions

Definition 4.1 (Address Abstraction). To govern the registration of an abstract identifier id and the publication of abstract transaction msgs, the system requires six functions (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify).

- $(crs_R, crs_P) \leftarrow \text{Setup}(\mathcal{R}_R, \mathcal{R}_P)$: This function generates the public parameters crs_R and crs_P for registration and publication.
- $cert \leftarrow \text{Certificate}(\cdot)$: This function enables users to obtain a certificate $cert$ from an institute, which is signed using the issuer's secret key sk.
- $\pi_R \leftarrow \text{RegisterProve}(crs_R, id, pk; cert||r)$: This function generates a proof π_R to verify that the abstract identifier id is derived from the certificate $cert$ and user-selected randomness r (i.e., $id \leftarrow \mathcal{H}(cert||r)$), and that the $cert$ was issued by the institute associated with the public key pk.

Table 2: Comparative Analysis of Properties

		Blockchain Address/Transaction	Abstract id/msg
Uniqueness	Injectiveness	The same address has the same private key.	The same id has the same $cert$ and r.
	Unforgeability	Transaction signing cannot be achieved without knowing the private key.	id registration and msg publication cannot occur without knowing the $cert||r$.
	Correctness	A user with a valid private key always generates legit signatures.	A user with valid $cert$ and r always generates legitimate proofs πs.
Immutability	Tamper Resistance	A signature is non-reusable and the signed transaction cannot be modified.	Proof π cannot be reused and the corresponding msg cannot be modified.
	Chronicle	Each transaction has its own order, controlled by the account nonce.	Each msg has its own order, by monotone-increasing value m.
Privacy-Preservation		The private key is not leaked before/after publishing transactions.	$cert$ and r are not leaked before/after the registration and publications.

- $0/1 \leftarrow \text{RegisterVerify}(crs_R, id, pk, \pi_R)$: This function enables a user to register a unique id following successful verification. The inputs include the public parameters crs_R, the abstract identity id, the public key pk, and a proof π_R. The function returns a binary outcome indicating the validation status of the identifier id, where 0 represents invalidity and 1 indicates validity.
- $\pi_P \leftarrow \text{PublishProve}(crs_P, id, msg, m; cert||r)$: This function generates a proof π_P that attests the user possesses a valid $cert$ and r, enabling them to compute id. It also considers the abstract transaction msg and the transaction sequence order m as constraints.
- $(0/1, res) \leftarrow \text{PublishVerify}(crs_P, id, msg, m, \pi_P)$: This function executes the requested m-th abstract transaction msg given a valid proof π_P for an identifier id, based on the crs_P. The id should correspond with the previously registered identifier via the RegisterVerify function. Additionally, it requires a monotonically increasing counter m to ensure the chronological processing of msg. The result of the execution is denoted as res. If successful, it returns $(1, res)$; otherwise, it returns $(0, \emptyset)$.

4.2 Properties

Address Abstraction must satisfy three properties to function as an alternative to the chain's inherent identity system: **Uniqueness** (4.2), **Immutability** (4.3), and **Privacy-Preservation** (4.4). Table 2 presents a comparison between our scheme and traditional blockchain address/transaction schemes, especially Ethereum [43].

Definition 4.2 (Uniqueness). System (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify) has unique identifier id^k for $user^k$, if id^k meets the Injectiveness, Unforgeability, and Correctness.

for all $((cert||r)^x, (cert||r)^y)$,

$$(cert||r)^x \neq (cert||r)^y \implies id^x \neq id^y \quad \text{(Injectiveness)}$$

Under the injectiveness condition, two distinct certificate-and-randoms will not map to the same identifier, ensuring that each $cert||r$ and its derived identifier id are unique within the system.

for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\chi_{\mathcal{A}}$,

$$\Pr\left[\begin{array}{l}(crs_R, \perp) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ ((id_R, pk, \pi_R); (cert||r)_R) \leftarrow (\mathcal{A}||\mathcal{X}_{\mathcal{A}})(crs_R);\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id_R, pk, \pi_R) :\\ s_R = 1 \wedge ((id_R, pk), (cert||r)_R) \notin \mathcal{R}_R\end{array}\right] \approx 0, \text{ and}$$

$$\Pr\left[\begin{array}{l}(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r);\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ ((id_P, msg, m, \pi_P); (cert||r)_P) \leftarrow (\mathcal{A}||\mathcal{X}_{\mathcal{A}})(crs_P);\\ (s_P, \perp) \leftarrow \textsc{PublishVerify}(crs_P, id_P, msg, m, \pi_P) :\\ s_R = 1 \wedge s_P = 1 \wedge ((id_P, msg, m), (cert||r)_P) \notin \mathcal{R}_P\end{array}\right] \approx 0$$

(Unforgeability)

The unforgeability property confirms the legitimacy of the identifier *id*, the authenticity of the *m*-th message *msg*, and asserts the knowledge of the related secret $cert||r$. This ensures that valid registrations and publications can only be made by those possessing the required secrets.

$$\Pr\left[\begin{array}{l}(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r);\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ \pi_P \leftarrow \textsc{PublishProve}(crs_P, id, msg, m; cert||r);\\ (s_P, \perp) \leftarrow \textsc{PublishVerify}(crs_P, id, msg, m, \pi_P) :\\ s_R = 1 \wedge s_P = 1 \wedge ((id, pk), cert||r) \in \mathcal{R}_R\\ \wedge ((id, msg, m), cert||r) \in \mathcal{R}_P\end{array}\right] = 1$$

(Correctness)

A user with a valid secret $cert||r$ can always generate a proof π_R for registration and π_P for all executable messages ($msg \in \mathcal{M}$) and a monotonically increasing counter *m*, to validate to the verifier.

Definition 4.3 (Immutability). Requests on the system (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify) is immutable, if the system ensures the Tamper Resistance and Chronicle.

for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}(crs_R, \perp) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ (r, r^{\mathcal{A}}) \xleftarrow{\$} \mathbb{F}_p \text{ s.t. } r^{\mathcal{A}} \neq r;\\ cert \leftarrow \textsc{Certificate}(\cdot); id \leftarrow \mathcal{H}(cert||r);\\ cert^{\mathcal{A}} \leftarrow \textsc{Certificate}(\cdot); id^{\mathcal{A}} \leftarrow \mathcal{H}(cert^{\mathcal{A}}||r^{\mathcal{A}});\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r);\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id^{\mathcal{A}}, pk, \pi_R) :\\ s_R = 1 \wedge ((id^{\mathcal{A}}, pk), cert||r) \notin \mathcal{R}_R\end{array}\right] \approx 0, \text{ and}$$

$$\Pr\left[\begin{array}{l}(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r);\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ (msg, msg^{\mathcal{A}}) \xleftarrow{\$} \mathcal{M} \text{ s.t. } msg^{\mathcal{A}} \neq msg;\\ \pi_P \leftarrow \textsc{PublishProve}(crs_P, id, msg, m; cert||r);\\ (s_P, \perp) \leftarrow \textsc{PublishVerify}(crs_P, id, msg^{\mathcal{A}}, m, \pi_P) :\\ s_R \wedge s_P = 1 \wedge ((id, pk), cert||r) \in \mathcal{R}_R\\ \wedge ((id, msg^{\mathcal{A}}, m), cert||r) \notin \mathcal{R}_P\end{array}\right] \approx 0$$

(Tamper Resistance)

This indicates the robustness of the cryptographic system against unauthorized modifications, ensuring that the probability of an adversary tampering with the registration of an *id* or the publication of a *msg*, while using the original proofs, is negligible.

$$\text{for all } m \geq 0, msg_{m+1} \succ msg_m \qquad \text{(Chronicle)}$$

Under the chronicle property, each message succeeds its predecessor in a strictly increasing order.

Definition 4.4 (Privacy-Preservation). System (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify) preserves privacy if the secret $cert||r$ associated with the identifier *id* remains undisclosed before and after any function execution.

for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}(crs_R, \perp) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r) :\\ \mathcal{A}(crs_R, \pi_R) = 1\end{array}\right] =$$

$$\Pr\left[\begin{array}{l}(crs_R, \perp, \tau_R, \perp) \leftarrow \textsc{SimSetup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{SimRegisterProve}(crs_R, \tau_R, id, pk) :\\ \mathcal{A}(crs_R, \pi_R) = 1\end{array}\right], \text{ and}$$

$$\Pr\left[\begin{array}{l}(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r) :\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ \pi_P \leftarrow \textsc{PublishProve}(crs_P, id, msg, m; cert||r);\\ s_R = 1 \wedge \mathcal{A}(crs_P, \pi_P) = 1\end{array}\right] =$$

$$\Pr\left[\begin{array}{l}(crs_R, crs_P, \tau_R, \tau_P) \leftarrow \textsc{SimSetup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(\cdot); r \xleftarrow{\$} \mathbb{F}_p; id \leftarrow \mathcal{H}(cert||r);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert||r) :\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ \pi_P \leftarrow \textsc{SimPublishProve}(crs_P, \tau_P, id, msg, m);\\ s_R = 1 \wedge \mathcal{A}(crs_P, \pi_P) = 1\end{array}\right]$$

(Privacy-Preservation)

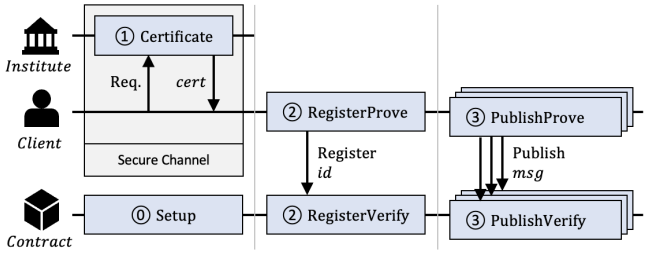


Figure 1: The process flow of zkAA. ⓪ Public parameters are generated. ① The user obtains a certificate *cert* from the institute. ② The user calculates the identifier as $id \leftarrow \mathcal{H}(cert||r)$, and proceeds to register it on the smart contract. ③ The user utilizes the registered *id* to publish abstract transaction *msg*s.

This property relates to the system's ability to protect the user's personal information $cert||r$ from exposure, ensuring confidentiality. SimRegisterProve and SimPublishProve use the trapdoors τ_R and τ_P from SimSetup to simulate proofs, which are generated without a witness. Under the privacy-preservation property, the distributions of simulated proofs π_R and π_P are indistinguishable from real ones, regardless of whether the witness $cert||r$ is known.

5 Zero-Knowledge Address Abstraction

We present a concrete implementation of Address Abstraction (AA), named zero-knowledge Address Abstraction (zkAA), which utilizes zk-SNARKs to offer a unified identity across webs while preserving user confidentiality during identity registration and publication.

5.1 Design of zkAA

Figure 1 presents the zkAA (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify) overview. The details of each function are specified in Algorithms 1, 2, 3, and 4.

- $(crs_R, crs_P) \leftarrow$ Setup$(\mathcal{R}_R, \mathcal{R}_P)$: This step generate the public parameters $crs_R \leftarrow$ zkp.Setup$(\mathcal{R}_R)$ and $crs_P \leftarrow$ zkp.Setup$(\mathcal{R}_P)$.
- $cert \leftarrow$ Certificate$(\cdot)$: A user obtains a certificate *cert* from the institution. The signed certificate *cert* is generated by the issuer using sig.Sign to sign the `claims` (i.e., attributes). The user then computes the identifier *id* from *cert* via the hash function $\mathcal{H}$, combined with a user-defined random number r.
- $\pi_R \leftarrow$ RegisterProve$(crs_R, id, pk; cert||r)$: The proof π_R is generated as $\pi_R \leftarrow$ zkp.Prove$(crs_R, (id, pk); cert||r)$, which proves: (i) *id* is derived from $id = \mathcal{H}(cert||r)$; and (ii) *cert* is signed with the secret key *sk* corresponding to the institution's public key *pk*, and so verifiable via sig.Verify.
- $0/1 \leftarrow$ RegisterVerify(crs_R, id, pk, π_R): The user submits *id* and π_R, then the contract invokes zkp.Verify$(crs_R, (id, pk), \pi_R)$ to validate them. If *id* is valid and not yet registered, it is then marked as registered in the contract by updating the *nonce* mapping to 1 for the given *id*. Since the default value of *nonce* is 0, any value greater than 0 indicates the *id* is already registered.
- $\pi_P \leftarrow$ PublishProve$(crs_P, id, msg, m; cert||r)$: The user generates a proof $\pi_P \leftarrow$ zkp.Prove$(crs_P, (id, \mathcal{H}(msg), m); cert||r)$, attesting that *id* is derived from $cert||r$ without revealing *cert* or r. The proof is tied to the message *msg* and the sequence number m, preventing its reuse. In the context of a transaction, the message contains the recipient address `target`, the function identifier `funcsig`, the encoded inputs `calldata`, and the cryptocurrency amount `value`. To improve efficiency, $\mathcal{H}(msg)$ is used instead of the raw *msg* to handle cases with large message sizes.
- $(0/1, res) \leftarrow$ PublishVerify$(crs_P, id, msg, m, \pi_P)$: The contract verifies π_P through zkp.Verify$(crs_P, (id, \mathcal{H}(msg), m), \pi_P)$ to validate the sender's ownership for the message *msg*. The identifier *id* must have been registered previously. The sequence number m serves as a counter, similar to an account nonce. If π_P, *msg*, and m are valid, the message is executed, producing a result *res*.

Algorithm 1 ⓪ Setup

function Setup$(\mathcal{R}_R, \mathcal{R}_P)$
  $crs_R \leftarrow$ zkp.Setup$(\mathcal{R}_R)$; $crs_P \leftarrow$ zkp.Setup$(\mathcal{R}_P)$
  return (crs_R, crs_P)

Algorithm 2 ① Certificate

institute has the pair of $(pk, sk) \leftarrow$ sig.KeyGen$(\cdot)$
function Certificate$(\cdot)$ ▷ [Institute]
  $cert \leftarrow$ sig.Sign$(sk,$ `claims`$)$; **return** *cert*

Algorithm 3 ② Register

$id \leftarrow \mathcal{H}(cert)$
function RegisterProve$(crs_R, id, pk; cert)$ ▷ [Client]
  $\pi_R \leftarrow$ zkp.Prove$(crs_R, (id, pk); cert)$; **return** π_R
function RegisterVerify(crs_R, id, pk, π_R) ▷ [Contract]
  assert$(nonce[id] == 0)$
  assert(zkp.Verify$(crs_R, (id, pk), \pi_R) == 1)$
  $nonce[id] \leftarrow 1$; **return** 1

Algorithm 4 ③ Publish

$id \leftarrow \mathcal{H}(cert)$; $msg \leftarrow$ (`target`, `funcsig`, `calldata`, `value`)
function PublishProve$(crs_P, id, msg, m; cert)$ ▷ [Client]
  $\pi_P \leftarrow$ zkp.Prove$(crs_P, (id, \mathcal{H}(msg), m); cert)$; **return** π_P
function PublishVerify$(crs_P, id, msg, m, \pi_P)$ ▷ [Contract]
  assert$(nonce[id] \geq 1$ and $m == nonce[id] + 1)$
  assert(zkp.Verify$(crs_P, (id, \mathcal{H}(msg), m), \pi_P) == 1)$
  $nonce[id] \leftarrow m$; $res \leftarrow$ **execute** *msg*; **return** $(1, res)$

5.2 Properties of zkAA

The zkAA satisfies AA's three core properties: **Uniqueness**, **Immutability**, and **Privacy-Preservation**. A detailed proof is provided in Section 5.3.

- **Uniqueness.** zkAA generates a unique identifier, *id*, for user identity. Given the random oracle assumption on $\mathcal{H}$, it ensures injectiveness. zk-SNARK guarantees that the registration and publication operations require knowledge of $cert||r$, ensuring unforgeability. Additionally, the protocol ensures the consistent generation of valid proofs due to use of zk-SNARKs and contracts, ensuring correctness.

- **Immutability.** The use of *msg* within proof π_P generation ensures tamper resistance. The sequential number m acts as a nonce, preserving the chronology of abstract transactions.
- **Privacy-Preservation.** zkAA leverages zk-SNARKs to protect user privacy, revealing only hashed values and keeping the original $cert||r$ confidential.

5.3 Security Proof

The security model adheres to the simulation-based definition [13]. In this section, we define the ideal-world execution $\texttt{Ideal}_{\mathcal{T},\mathcal{S}}$ and the real-world execution $\texttt{Real}_{\text{zkAA},\mathcal{A}}$ for Address Abstraction (AA). Algorithms 5, 6, 7, and 8 corresponding to Setup, Certificate, Register, and Publish in the ideal-world execution, along with their descriptions, are provided in Appendix A.

Definition 5.1 ($\texttt{Ideal}_{\mathcal{T},\mathcal{S}}$). $\texttt{Ideal}_{\mathcal{T},\mathcal{S}}$ denotes the ideal implementation of the AA scheme with the trusted party $\mathcal{T}$ and simulator $\mathcal{S}$, wherein all storage and operations are performed by $\mathcal{T}$. The simulator $\mathcal{S}$ in ideal-world execution is assumed to be incapable of forging the function outputs generated by $\mathcal{T}$ and distinguishing any data from random values. In the ideal-world execution, participants interact only with $\mathcal{T}$ through the interface outlined in Algorithms 5, 6, 7, and 8.

Definition 5.2 ($\texttt{Real}_{zkAA,\mathcal{A}}$). $\texttt{Real}_{\text{zkAA},\mathcal{A}}$ designates our real-world implementation of AA, zero-knowledge Address Abstraction (zkAA), with the PPT adversaries $\mathcal{A}$. The actions of honest participants align precisely with Algorithms 1, 2, 3, and 4.

We assert that the probability of an adversary compromising the real-world execution is, at most, equivalent to the probability of an adversary compromising the ideal-world execution.

Definition 5.3. The real-world execution $\texttt{Real}_{\text{zkAA},\mathcal{A}}$ is said to securely emulate the ideal-world execution $\texttt{Ideal}_{\mathcal{T},\mathcal{S}}$ if the following is satisfied:

for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ s.t.

for any PPT distinguisher $\mathcal{D}$,

$$\Pr\left[\mathcal{D}(\texttt{Ideal}_{\mathcal{T},\mathcal{S}}(\cdot)) = 1\right] \approx \Pr\left[\mathcal{D}(\texttt{Real}_{\text{zkAA},\mathcal{A}}(\cdot)) = 1\right]$$

Theorem 5.4. *Under the assumptions that zk-SNARKs are zero-knowledge and extractable, digital signatures are unforgeable under chosen-message attacks, hash functions behave like random oracles, and smart contracts function as trustworthy computation engines and data storage; zkAA fulfills the security requirement stated in Definition 5.3.*

Proof. We use the hybrid game approach. We define a sequence of games $\mathfrak{d}_0$ to $\mathfrak{d}_5$, modifying the execution step by step while arguing that differences in the adversary's view are negligible under our cryptographic assumptions.

- $\mathfrak{d}_0$: The real-world execution experiment.
- $\mathfrak{d}_1$: Replace proofs from honest participants with simulated ones using zkp.SimProve. Since they are computationally indistinguishable from real proofs, then $\mathfrak{d}_0 \approx \mathfrak{d}_1$.
- $\mathfrak{d}_2$: Run the knowledge extractor on the proofs, and abort if extraction fails. Given the extractor's negligible failure rate, the adversary cannot distinguish this change, so $\mathfrak{d}_1 \approx \mathfrak{d}_2$.

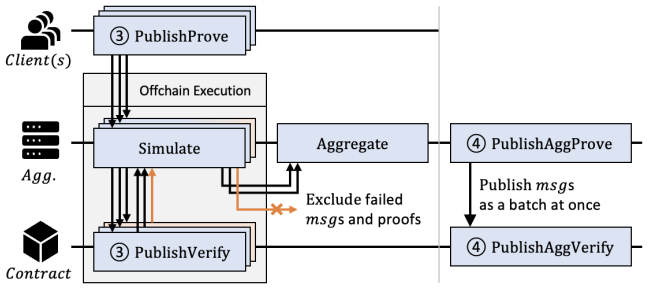


Figure 2: The zkAA proof aggregation process. ③ Clients generate proofs, then the aggregator simulates them off-chain. The aggregator collects and aggregates the proofs that successfully passed the off-chain execution. ④ The aggregated proof and the batch of valid *msg*s are then submitted to the smart contract for publication.

- $\mathfrak{d}_3$: Model the hash function as a random oracle, assigning unique random values to all hash outputs. Under this model, the adversary cannot differentiate between the hash outputs and random values, implying $\mathfrak{d}_2 \approx \mathfrak{d}_3$.
- $\mathfrak{d}_4$: Substitute the signatures with unique random values. Since these signatures are kept secret and the adversary cannot forge valid ones, they cannot distinguish between the real signatures and the random values. It follows that $\mathfrak{d}_3 \approx \mathfrak{d}_4$.
- $\mathfrak{d}_5$: Emulate all data storage and computations with those provided by the trusted party $\mathcal{T}$, modeling the ideal-world execution. Since we assume the blockchain is robust, and thus participants cannot observe differences, we have $\mathfrak{d}_4 \approx \mathfrak{d}_5$.

Therefore, we conclude that $\mathfrak{d}_0 \approx \mathfrak{d}_5$, implying that our real-world execution, zkAA, securely emulates the ideal implementation of the AA scheme. □

We show that the adversary in real-world execution can forge inputs/outputs and extract secrets no more effectively than in the ideal-world execution, and all distributions are computationally indistinguishable. This signifies that it is infeasible for the $\mathcal{A}$ to forge or extract in real-world execution if such an attack in the ideal-world execution is deemed impossible.

5.4 Optimization Using Proof Aggregation

To enhance verification efficiency, we employ proof aggregation. Figure 2 illustrates the aggregation process, which begins after the completion of steps ⓪ to ② in Figure 1. Here, multiple clients' proofs (π_Ps) are combined into a single aggregated proof (π_{agg}), which is verified in a single on-chain operation.

The aggregation is managed by an *Aggregator*, a centralized yet trustless entity. The cryptographic guarantees of zk-SNARKs ensure that the aggregator cannot alter the proofs.

- $\pi_P \leftarrow \text{PublishProve}(crs, id, msg, m; cert||r)$: Clients generate proofs and transmit them, along with *id*, *msg*, and *m*, to the aggregator, rather than directly interacting with the contract.
- $(0/1, \perp) \leftarrow \text{PublishVerify}(crs, id, msg, m, \pi_P)$: The aggregator simulates the PublishVerify process off-chain to validate each proof π_P along with its associated message *msg* and the sequence

Table 3: Experimental results on zkAA

		min [s]	max [s]	avg (sd) [s]
Registration	witness	0.096	0.105	0.099 (0.002)
	proof	0.717	0.804	0.762 (0.016)
Publication	witness	0.045	0.053	0.047 (0.001)
	proof	0.437	0.512	0.473 (0.015)

(a) Proof generation time

	gas	Ethereum	Arbitrum	Optimism	Polygon
Registration	209823	$3.71	$0.04	$0.06	$0.01
Publication	203066	$3.60	$0.04	$0.05	$0.01

(b) Proof verification costs

m. Since this occurs off-chain, it avoids on-chain state transitions and transaction costs. Valid proofs are batched for aggregation, while invalid ones are discarded, preventing potential attacks that could exploit the time-consuming aggregation process.

- $\pi_{agg} \leftarrow$ PublishAggProve($crs, ids, msgs, ms, \pi_P s$): The aggregator recursively combines individual proofs π_Ps into a single aggregated proof π_{agg}. Each recursive step generates a proof that verifies both the correctness of the current computation and the validity of prior proofs. By iteratively applying this process, multiple proofs can be aggregated into one succinct proof.
- $(0/1, ress) \leftarrow$ PublishAggVerify($crs, ids, msgs, ms, \pi_{agg}$): The contract verifies the aggregated proof π_{agg}. If valid, each message msg is executed in sequence, and the results are collected as $ress$.

We integrate the additional optimization steps PublishAggProve and PublishAggVerify without modifying the original zkAA implementation. This preserves all AA properties and ensures that the security guarantees of zkAA remain intact.

6 Experiments

To evaluate the efficiency and practicality of our implementation, we conducted experiments measuring proof generation time and gas overheads (blockchain transaction fees) across multiple networks.

As of August 2024, the average gas price on Ethereum [43] was 6.70 gwei (1 gwei = 10^{-9} ETH), with ETH priced at $2642.46 [9]. In comparison, Ethereum scaling solutions Arbitrum [31] and Optimism [33] had average gas prices of 0.07 gwei and 0.10 gwei, respectively [23]. Polygon [34] had an average gas price of 117.78 gwei, with POL (formerly MATIC) priced at $0.45.

6.1 Implementations

We utilized JSON Web Token (JWT) [18] as the certificate *cert*, with its signature generated using EdDSA [5], which integrates the zero-knowledge-friendly Poseidon hash [15]. The identifier *id* was then derived from *cert* using Poseidon hash as well.

zkAA was implemented in Circom [26], based on the Groth16 proof system [16].[2] Our implementation consisted of two circuits: a registration circuit with 4504 constraints and a publication circuit with 298 constraints. The contracts were compiled using Solidity version 0.8.23 with the optimization option set to $2^{32} - 1$ runs.

[2]Though Groth16's malleability [3, 17] does not harm zkAA, it can also be implemented with GM17 [17] or PLONK [12] (as in Section 6.4), readily avoiding this limitation.

Table 4: PLONK implementation of zkAA publication

	avg time (sd) [s]	peak mem. [GB]
Single (No Agg.)	0.58 (0.025)	0.14
3 Proofs Agg.	223.18 (0.355)	35.06
7 Proofs Agg.	442.92 (0.415)	66.30
15 Proofs Agg.	883.37 (3.109)	124.05

(a) Aggregated proof generation time and peak memory usage

	gas (per proof)	Ethereum	Arbitrum	Optimism	Polygon
Single (No Agg.)	376802 (376802.00)	$6.67 ($6.67)	$0.07 ($0.07)	$0.10 ($0.10)	$0.02 ($0.02)
3 Proofs Agg.	439678 (146559.33)	$7.78 ($2.59)	$0.08 ($0.03)	$0.12 ($0.04)	$0.02 ($0.01)
7 Proofs Agg.	466635 (66662.14)	$8.26 ($1.18)	$0.09 ($0.01)	$0.12 ($0.02)	$0.02 ($0.00)
15 Proofs Agg.	558138 (37209.20)	$9.88 ($0.66)	$0.10 ($0.01)	$0.15 ($0.01)	$0.03 ($0.00)

(b) Aggregated proof verification costs

6.2 Overhead in Generating Proofs

Table 3a shows the average time required to generate witnesses and proofs. To follow client sovereignty, the experiments were conducted on hardware that reflects what real users would use — a local Apple M1 Pro machine with 16GB of RAM. Each test was repeated 100 times to obtain averaged results.

Generating the witness and proof took an average of 0.86 seconds for registration and 0.52 seconds for publication. Although both times are brief compared to the block interval, registration is about 1.65 times slower due to the inclusion of EdDSA signature verification. Since registration is a one-time process per identifier and publication occurs repeatedly afterward, this two-step approach is more efficient than verifying the signature with every transaction.

6.3 Verification Costs

Table 3b examines the costs associated with on-chain verification. On Ethereum, registration costs $3.71 in gas fees, while publication costs $3.60. More cost-efficient chains like Polygon reduce these costs to approximately $0.01 for both registration and publication.

In a hypothetical scenario, the NFT marketplace OpenSea [29], one of the most transaction-intensive web3 services, could leverage zkAA to integrate NFT collections across multiple chains and addresses. It could also support features like account recovery and regulatory compliance. As of August 2024, adopting zkAA would add an estimated monthly cost of $23.01 for Ethereum users and $0.06 for Polygon users, based on 788,393 NFTs sold and 123,358 active traders [37].

6.4 Costs of Proof Aggregation

We implemented an optimized zkAA using Circom with PLONK [12]. Aggregator was run on an AMD 5950X processor (32 threads) and 128GB RAM, testing the aggregation of 3, 7, and 15 proofs (see Table 4). The tests were repeated 10 times to compute the average.

Aggregating more proofs requires higher resources, particularly RAM. Aggregating 15 proofs required 124.05GB of RAM and 15 minutes. Although proof generation is time-intensive, this can be adjusted to balance time constraints (Table 4a) and the number of

Algorithm 5 ⓪ IdealSetup

function IdealSetup(·)
  $\mathcal{T}$ creates empty private table $\texttt{PrivTab}_c$
  $\mathcal{T}$ creates empty public tables $\texttt{PubTab}_r$, $\texttt{PubTab}_p$, $\texttt{PubTab}_n$

Algorithm 6 ① IdealCertificate

function IdealCertificate(·)
  $\mathcal{T}$ generates unique random values id and $cert$
  $\mathcal{T}$ inserts $cert$ in $\texttt{PrivTab}_c$ with id as key
  $\mathcal{T}$ sends $(id, cert)$ to participant

proof aggregated (Table 4b). GPU acceleration or other optimizations can be used to significantly reduce generation time [1, 7].

As shown in Table 4b, the per-proof cost decreases as more proofs are aggregated. Verifying 15 aggregated proofs consumed 558,138 gas, averaging 37,209.20 gas per proof, highlighting significant cost savings. With proof aggregation, users on OpenSea would incur an additional cost of $4.22 per month on Ethereum (a reduction of approximately 5.5 times) and less than $0.07 on scaling solutions.

7 Conclusion

Address Abstraction (AA) provides users with a unified identifier for interacting across blockchains and traditional webs, eliminating the need for multiple addresses and certificates. This approach enhances web2 and web3 integration, facilitates seamless cross-chain interactions, and improves the user experience. For example, *zero-knowledge Address Abstraction* (zkAA) can be integrated into decentralized applications (dApps) like NFT marketplaces and DeFi, enabling web2 logins and supporting regulatory compliance.

One concern is that, since zkAA is built on smart contracts, users require private keys to interact with the blockchain (e.g., to pay transaction fees). However, this issue can be easily addressed using several techniques [6, 38], which allow transactions to be published by others (thus enabling fees to be paid by third parties). As AA separates signatures from addresses, it aligns with delegation mechanisms, making integration with these solutions straightforward.

A Ideal-World Execution

⓪ In the IdealSetup function (Algorithm 5), the trusted party $\mathcal{T}$ initializes the core data storage. It creates a private table, $\texttt{PrivTab}_c$ (accessible only to $\mathcal{T}$), and three append-only public tables, $\texttt{PubTab}_r$, $\texttt{PubTab}_p$, and $\texttt{PubTab}_n$ (readable by all but writable only by $\mathcal{T}$). All tables are structured as key-value pairs.

① In the IdealCertificate function (Algorithm 6), $\mathcal{T}$ issues an identifier id and a certificate $cert$ to the participant, storing them in $\texttt{PrivTab}_c$ with id as the key and $cert$ as the value. Since $cert$ is a random value itself, the additional random number r can be omitted in subsequent processes without loss of generality.

② In the IdealRegisterProve function (Algorithm 7), participants submit their id, the institution's public key pk, and the certificate $cert$ to $\mathcal{T}$ to obtain a proof π_R. $\mathcal{T}$ validates whether id and $cert$ are valid (exist in $\texttt{PrivTab}_c$). If valid, $\mathcal{T}$ randomly generates π_R, stores (id, pk) as the key and π_R as the value in $\texttt{PubTab}_r$, and then returns π_R to the participant. In IdealRegisterVerify, $\mathcal{T}$ verifies the registration using π_R and checks the nonce in $\texttt{PubTab}_n$ to ensure no prior registration. If this is the participant's first registration, the nonce for (id, pk) is set to 1.

③ In the IdealPublishProve function (Algorithm 8), $\mathcal{T}$ generates a proof π_P for the publication of the m-th message msg, provided the participant's id and $cert$ are valid. If valid, $\mathcal{T}$ generates π_P, stores (id, msg, m) as the key and π_P as the value in $\texttt{PubTab}_p$, and sends π_P to the participant. In the IdealPublishVerify

Algorithm 7 ② IdealRegister

function IdealRegisterProve($id, pk, cert$)
  Participant sends $(id, pk, cert)$ to $\mathcal{T}$
  if $\texttt{PrivTab}_c$.get(id) $\neq cert$ **then** abort ▷ invalid id, $cert$
  if $(id, pk) \in \texttt{PubTab}_r$.keys **then**
    $\mathcal{T}$ selects π_R from $\texttt{PubTab}_r$ with (id, pk) as key
  else
    $\mathcal{T}$ generates a unique random value π_R
    $\mathcal{T}$ inserts π_R in $\texttt{PubTab}_r$ with (id, pk) as key
  $\mathcal{T}$ sends π_R to participant
function IdealRegisterVerify(id, pk, π_R)
  Participant sends (id, pk, π_R) to $\mathcal{T}$
  if $(id, pk) \in \texttt{PubTab}_n$.keys **then**
    $\mathcal{T}$ sends 0 to participant ▷ already registered
  else if $\texttt{PubTab}_r$.get((id, pk)) $\neq \pi_R$ **then**
    $\mathcal{T}$ sends 0 to participant ▷ invalid π_R
  else
    $\mathcal{T}$ inserts 1 in $\texttt{PubTab}_n$ with (id, pk) as key
    $\mathcal{T}$ sends 1 to participant

Algorithm 8 ③ IdealPublish

function IdealPublishProve($id, msg, m, cert$)
  Participant sends $(id, msg, m, cert)$ to $\mathcal{T}$
  if $\texttt{PrivTab}_c$.get(id) $\neq cert$ **then** abort ▷ invalid id, $cert$
  if $(id, msg, m) \in \texttt{PubTab}_p$.keys **then**
    $\mathcal{T}$ selects π_P from $\texttt{PubTab}_p$ with (id, msg, m) as key
  else
    $\mathcal{T}$ generates a unique random value π_P
    $\mathcal{T}$ inserts π_P in $\texttt{PubTab}_p$ with (id, msg, m) as key
  $\mathcal{T}$ sends π_P to participant
  The public key pk is known to $\mathcal{T}$.
function IdealPublishVerify(id, msg, m, π_P)
  Participant sends (id, msg, m, π_P) to $\mathcal{T}$
  if $(id, pk) \notin \texttt{PubTab}_n$.keys **then**
    $\mathcal{T}$ sends $(0, \perp)$ to participant ▷ unregistered id
  else
    $\mathcal{T}$ selects $nonce$ from $\texttt{PubTab}_n$ with (id, pk) as key
    if $nonce < 1$ or $m \neq nonce + 1$ **then**
      $\mathcal{T}$ sends $(0, \perp)$ to participant ▷ invalid $nonce$, m
    else if $\texttt{PubTab}_p$.get((id, msg, m)) $\neq \pi_P$ **then**
      $\mathcal{T}$ sends $(0, \perp)$ to participant ▷ invalid π_P
    else
      $\mathcal{T}$ updates $\texttt{PubTab}_n$ with (id, pk) as key and m as value
      $res \leftarrow$ **execute** msg; $\mathcal{T}$ sends $(1, res)$ to participant

function, participants verify the publication by submitting their id, msg, m, and π_P. The participant must already be registered, and m must be exactly one greater than the current value in PubTab_n. The function then executes msg, returns the result res to the participant, along with the publication success status 1. Once the publication is completed, PubTab_n is updated by m.

The ideal-world execution (IdealSetup, IdealCertificate, IdealRegisterProve, IdealRegisterVerify, IdealPublishProve, IdealPublishVerify) satisfies the properties:

- **Injectiveness.** Each id and $cert$ is uniquely generated and stored, ensuring that no two distinct $cert$s map to the same id.
- **Unforgeability.** Only participants with the correct id and $cert$ can successfully register and publish messages. The proofs are securely generated and stored by $\mathcal{T}$, making them unforgeable.
- **Correctness.** The functions ensure that any participant with valid credentials can successfully register and publish messages in the correct sequence.
- **Tamper Resistance.** By restricting write access to only $\mathcal{T}$, the system and data are robust against unauthorized modifications.
- **Chronicle.** The functions enforce the correct message sequence, requiring the sequence number m to increment in order correctly.
- **Privacy-Preservation.** Participants' private data ($cert$, id) is never exposed or shared; it is securely stored and used only for verification purposes within $\mathcal{T}$.

Acknowledgments

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government Ministry of Science and ICT (MSIT) (No. 2021-0-00518, 40%), (No. 2021-0-00180, 10%), and (No. 2021-0-00136, 10%), by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00208245, 10%), by the ITRC (Information Technology Research Center) support program supervised by the IITP (IITP-2021-0-01835, 20%), and by the IITP under artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081, 10%).

References

[1] Miguel Ambrona et al. 2023. aPlonK: Aggregated PlonK from Multi-polynomial Commitment Schemes. In *International Workshop on Security*. Springer, 195–213.
[2] Aptos. 2024. Aptos Keyless. https://aptos.dev/en/build/guides/aptos-keyless. Accessed: Sep. 19, 2024.
[3] Karim Baghery, Zaira Pindado, and Carla Ràfols. 2020. Simulation extractable versions of Groth's zk-SNARK revisited. In *International Conference on Cryptology and Network Security*. Springer, 453–461.
[4] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 62–73.
[5] Daniel J Bernstein et al. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* 2, 2 (2012), 77–89.
[6] Vitalik Buterin et al. 2021. EIP-4337: Account Abstraction Using Alt Mempool. https://eips.ethereum.org/EIPS/eip-4337. Accessed: Sep. 19, 2024.
[7] Electric Coin Company. 2020. The Halo2 zero-knowledge proving system. https://github.com/zcash/halo2 Accessed: Sep. 19, 2024.
[8] John R Douceur. 2002. The sybil attack. In *Peer-to-Peer Systems: First International Workshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer, 251–260.
[9] Etherscan. 2024. Etherscan. https://etherscan.io/charts. Accessed: Sep. 19, 2024.
[10] Everclear. 2023. Chain Abstraction - Connext Network. https://www.connext.network/chain-abstraction. Accessed: Sep. 19, 2024.
[11] FluiDex. 2022. Plonkit. https://github.com/fluidex/plonkit. Accessed: Sep. 19, 2024.
[12] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* 2019 (2019), 953.
[13] Christina Garman, Matthew Green, and Ian Miers. 2017. Accountable privacy for decentralized anonymous payments. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*. Springer, 81–98.
[14] K Goings and P Abel. 2013. The Value of Social Login. Solving the Engagement Gap. Insights from Consumer Research.
[15] Lorenzo Grassi et al. 2021. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*. 519–535.
[16] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.
[17] Jens Groth and Mary Maller. 2017. Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs. Cryptology ePrint Archive, Paper 2017/540. https://eprint.iacr.org/2017/540
[18] M. Jones, J. Bradley, and N. Sakimura. 2015. JSON Web Token (JWT). https://www.rfc-editor.org/rfc/rfc7519. Accessed: Sep. 19, 2024.
[19] Nikolaos Kapsoulis et al. 2020. Know your customer (KYC) implementation with smart contracts on a privacy-oriented decentralized architecture. *Future Internet* 12, 2 (2020), 41.
[20] Nanak Nihal Khalsa et al. 2022. Holonym: A Decentralized Zero-Knowledge Smart Identity Bridge.
[21] Matter Labs. 2022. Recursive Aggregation Circuit. https://github.com/matter-labs/recursive_aggregation_circuit. Accessed: Sep. 19, 2024.
[22] Duc Anh Luong and Jong Hwan Park. 2023. Privacy-Preserving Identity Management System on Blockchain Using Zk-SNARK. *IEEE Access* 11 (2023), 1840–1853.
[23] Marcov. 2024. Transactions Fee - Totals. https://dune.com/queries/3686777/6201461. Accessed: Sep. 19, 2024.
[24] Nathaniel Masfen-Yan et al. 2022. Notebook: A Zero-Knowledge Identity Infrastructure Layer.
[25] Michael Mirkin, Yan Ji, Jonathan Pang, Ariah Klages-Mundt, Ittay Eyal, and Ari Juels. 2020. BDoS: Blockchain denial-of-service. In *Proceedings of the 2020 ACM SIGSAC conference on Computer and Communications Security*. 601–619.
[26] Jose L Muñoz-Tapia et al. 2022. CIRCOM: A Robust and Scalable Language for Building Complex Zero-Knowledge Circuits. *TechRxiv* (2022).
[27] Near.org. 2024. Near. https://near.org/. Accessed: Sep. 19, 2024.
[28] Particle Network. 2024. Particle Network. https://particle.network/. Accessed: Sep. 19, 2024.
[29] Ozone Networks. 2018. OpenSea. https://opensea.io/. Accessed: Sep. 19, 2024.
[30] Philippe Oechslin. 2003. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*. Springer, 617–630.
[31] Offchain Labs. 2020. Arbitrum. https://arbitrum.io/. Accessed: Sep. 19, 2024.
[32] Office of Foreign Assets Control. 2022. FAQs 1095. https://ofac.treasury.gov/faqs/1095. Accessed: Sep. 19, 2024.
[33] Optimism Foundation. 2020. Optimism. https://www.optimism.io/. Accessed: Sep. 19, 2024.
[34] Polygon Labs. 2024. Polygon. https://polygon.technology/. Accessed: Sep. 19, 2024.
[35] Alex Preukschat and Drummond Reed. 2021. *Self-sovereign identity*. Manning Publications.
[36] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. 2022. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. *Cryptology ePrint Archive* (2022).
[37] rchen8. 2024. OpenSea. https://dune.com/rchen8/opensea. Accessed: Sep. 19, 2024.
[38] Ronan Sandford et al. 2020. ERC-2771: Secure Protocol for Native Meta Transactions. https://eips.ethereum.org/EIPS/eip-2771. Accessed: Sep. 19, 2024.
[39] Sui. 2023. zkLogin. https://sui.io/zklogin. Accessed: Sep. 19, 2024.
[40] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.
[41] W3C. 2022. Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-core/. Accessed: Sep. 19, 2024.
[42] Taotao Wang, Shengli Zhang, and Soung Chang Liew. 2023. Linking Souls to Humans with ZKBID: Accountable Anonymous Blockchain Accounts for Web 3.0 Decentralized Identity. *arXiv preprint arXiv:2301.02102* (2023).
[43] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* (2014).
[44] Guangsheng Yu, Xu Wang, Qin Wang, Tingting Bi, Yifei Dong, Ren Ping Liu, Nektarios Georgalas, and Andrew Reeves. 2022. Towards Web3 Applications: Easing the Access and Transition. *arXiv preprint arXiv:2210.05903* (2022).