Beyond the Blockchain Address: Zero-Knowledge Address Abstraction

Abstract—Integrating traditional Internet (web2) identities with blockchain (web3) identities presents considerable obstacles. Conventional solutions typically employ a mapping strategy, linking web2 identities directly to specific blockchain addresses. However, this method can lead to complications such as fragmentation of identifiers across disparate networks.

To address these challenges, we propose a novel scheme, Address Abstraction (AA), that circumvents the need for direct mapping. AA scheme replaces the existing blockchain address system while maintaining essential properties including a unique identifier, immutability of requests, and privacy preservation. This capability allows users to interact with the blockchain via their web2 identities, irrespective of the specific blockchain address, thereby eliminating limitations tied to a blockchain-specific address system. This mechanism fosters the seamless integration of web2 identities within the web3, in addition, promotes cross-chain compatibility.

We also provide an application of AA, denoted as zero-knowledge Address Abstraction (zkAA). It mainly leverages the zero-knowledge proofs to ensure the properties of AA. zkAA has been implemented as a smart contract — compatible with any existing contract-enabled blockchains. Our evaluation of zkAA on Ethereum demonstrates its efficiency. The average cost for registering an abstracted identity is approximate $7.66, whereas publishing an abstracted transaction costs around $4.75. In contrast, on Polygon, the associated costs are markedly lower: $0.02 for registration and $0.01 for publication, as of January 13, 2023. This empirical evaluation substantiates the feasibility of our proposed solution.

Keywords—digital identity; identity management; blockchain; decentralized applications; zero-knowledge proofs;

I. INTRODUCTION

The rapid growth of blockchain technology has driven an explosion in the development of decentralized applications (dApps), whose core ethos rests on decentralization principles. Interestingly, despite their decentralized pursuit, certain dApps integrate centralized identification mechanisms to ensure efficient operation. These include the utilization of social login systems and Know-Your-Customer (KYC) procedures [63], [29], which are particularly crucial in fortifying the security infrastructure against harmful actions, such as Sybil attacks [12].

A. Related Work and Challenges

The centralized identification strategies stem from web2 companies and technologies. Unfortunately, previous attempts at integrating web2 identities into the web3 blockchain milieu have encountered numerous challenges.

A basic strategy to merge web2 and web3 identities is by creating a bi-directional mapping between them. Several approaches have been proposed to do this, most of which fundamentally depend on linking web2 identities to their corresponding blockchain addresses [63], [47], [56], [64], [33], [30], [35]. However, this approach necessitates meticulous management of secrets related to both web2 and web3 identities. This twofold responsibility obliges users to safeguard their passwords (web2) and private keys (web3), while service providers have to handle both identity types securely. The problem escalates in the context of multi-chain dApps, where managing multiple chain-specific addresses can result in fragmented digital identifiers, given each blockchain’s own address system.

Web3Auth employs Shamir’s Secret Sharing [51] and Threshold Cryptography [11] to divide the private key into several key shares, thereby enhancing private key security management. On the other hand, Ramper integrates a third-party Key Management System (KMS) and a Hardware Security Module (HSM) [49], [24]. Here, transactions gain authorization within a Trusted Execution Environment (TEE) [10], ensuring that Ramper neither retains nor accesses the private key at any point. Despite these advancements, they remain tethered to the blockchain-specific address system. As a result, even with a robust underlying system, the provision of non-uniform addresses across diverse blockchains remains a challenge.

Several studies have employed zero-knowledge proofs (ZKPs) to preserve privacy in systems. For instance, [47], [64], [33] leveraged ZKPs to ensure the confidentiality of credentials. Holonym [30] validates the hashes of web2s — JSON Web Tokens [27] — via ZKP. Notebook [35]
TABLE I: Comparing zkAA with existing web2-web3 identity integration approaches.

<table>
<thead>
<tr>
<th>Easy Secret Management</th>
<th>Unified Identifier</th>
<th>Data Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebctCom [65]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZEBRA [47]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>ZKID [56]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>BZIDMS [64]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>DA Luong [33]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Holonym [30]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Notebook [58]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Web3Auth [32]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Ramper [66]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>POAP [29]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SBT [57]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>zkAA (Ours)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1 Private data are stored in a separate private data repository.
2 A user needs to store share(s) instead of private key.

verifies credentials signed by third-party entities and other constraints via ZKP. Yet, these methods still rely on the original blockchain address as the identifier, thereby leading to potential management limitations.

B. Contributions

To address the aforementioned challenges, we propose a novel scheme termed Address Abstraction (AA), along with its implementation, zero-knowledge Address Abstraction (zkAA). The AA scheme enables the direct usage of web2 identities as blockchain identifiers, obviating the need for their mapping to blockchain-specific addresses. This not only eliminates private key management but also facilitates the utilization of a single identifier across various blockchains. Table I provides a detailed comparison between existing methods for integrating web2-web3 identities and the zkAA system proposed in this paper. Further, Table II carries out a comparative analysis of the AA and the existing blockchain address system, with a focus on Ethereum [61].

The paper’s primary goal is to introduce a cutting-edge approach to address abstraction in blockchain systems. The key contributions are as follows:

- We propose a novel identity system, the Address Abstraction (AA), devised to surmount the limitations of traditional chain-specific identifier and signature systems.
- The paper describes the implementation of AA, named zero-knowledge Address Abstraction (zkAA). This system leverages the zero-knowledge proofs to maintain privacy and practicality.
- We conduct an empirical evaluation of zkAA on Ethereum [61] and Polygon [53]. This analysis provides valuable insights into its performance and efficiency, underlining the utility of our proposed solution.
- Lastly, the paper illustrates the application of zkAA in the realm of decentralized applications, with particular emphasis on multi-chain and web2-web3 hybrid systems.

II. BACKGROUND

This section provides background knowledge for our proposed scheme. We place primary emphasis on blockchain technology, particularly platforms amenable to smart contracts, and succinctly discuss decentralized identity management methods prevalent in the blockchain.

A. Blockchain and Address System

1) Blockchain: Blockchain, a revolutionary construct for secure data storage and executing smart contracts [52], has drastically altered the digital landscape. Among its most prominent platforms is Ethereum [61], which uses the Ethereum Virtual Machine (EVM) to execute contracts across a global network of public nodes, with the Ether (ETH) cryptocurrency serving as transaction fees.

Our implementation, zkAA, utilizes contracts to register and employ web2 identities. Relying solely on EVM functionality, this implementation integrates effortlessly without necessitating a hardfork, thereby significantly enhancing its practicality.

2) Address System: Ethereum consists of two account categories: External Owned Accounts (EOAs) and Contract Accounts (CAs). EOAs are traditional blockchain accounts controlled via a private key that enables users to sign transactions, while CAs carry program codes and storage but lack a corresponding private key. In both cases, the address serves as the account’s identifier. However, different blockchains use different address calculation schemes, leading to unique addresses for each chain and consequently, a lack of unified identifiers in a multi-chain environment.

To mitigate this issue, we propose an innovative paradigm shift in the blockchain sphere: address abstraction. This concept disentangles the user identifier from the conventional blockchain address, transitioning it into an abstract, unified identifier not reliant on the specific blockchain address system. This strategy greatly simplifies identity management and improves compatibility across multiple chains, as it allows users to interact using a single credential.

Address abstraction can be perceived as a specialized application of account abstraction [58], [7], given its function in modifying the signature system linked with the account.

TABLE II: Comparing the Ethereum address with the abstracted address scheme (ours). "Pre-Reg" stands for "Pre-Registration" requirement, indicating that a registration process is required prior to the publication of transactions.

<table>
<thead>
<tr>
<th>Blockchain Address/Transaction</th>
<th>Abtracted Address/Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret</td>
<td>The secret key is denoted as</td>
</tr>
<tr>
<td></td>
<td>sk.</td>
</tr>
<tr>
<td>Issuing Authority</td>
<td>sks are user-generated, with</td>
</tr>
<tr>
<td></td>
<td>no centralized authority in-</td>
</tr>
<tr>
<td></td>
<td>volved.</td>
</tr>
<tr>
<td>Pre-Reg</td>
<td>No. Any randomly generated</td>
</tr>
<tr>
<td></td>
<td>256-bits number can</td>
</tr>
<tr>
<td></td>
<td>function as the sk without</td>
</tr>
<tr>
<td></td>
<td>any specific usage limita-</td>
</tr>
<tr>
<td></td>
<td>tions.</td>
</tr>
<tr>
<td>Chain-Agnostic</td>
<td>No. This is contingent upon</td>
</tr>
<tr>
<td></td>
<td>the underlying cryptography</td>
</tr>
<tr>
<td></td>
<td>and the blockchain protocol.</td>
</tr>
</tbody>
</table>
Section VI-D delves further into the integration of smart contracts within the address abstraction context, similar to the account abstraction proposition in EIP-4337 [7].

B. Decentralized Identity Managements

Decentralized identity management refers to the generating, storing, and managing of digital identities within a decentralized environment. Notable examples of decentralized identity management systems operating on blockchain include the Proof of Attendance Protocol (POAP) [25] and SoulBound Tokens (SBTs) [57]. POAP uses blockchain to create unique tokens representing personal experiences, while SBTs, non-transferable Non-Fungible Tokens (NFTs), represent the traits or achievements that make up a soul.

Despite these advancements, these solutions are rooted in the web3 identifier, also known as the blockchain address. In contrast, our novel identity system, address abstraction, allows direct application of the web2 identity on the blockchain, eliminating the necessity for mapping to a web3 identifier.

III. Notation

Throughout this paper, we employ the following notations. We represent a relation as $R$. Given that $(io, w) \in R$, we refer to $io$ as the public input/output, and $w$ is defined as the witness. The symbol $crs$ denotes a public parameter, referred to as the Common Reference String (CRS). The term $A$ signifies a non-uniform polynomial time adversary. The output of this adversary is designated as $y \leftarrow A(x)$, where $x$ represents the input and $y$ is the output. An extractor, denoted by $X_A$, can compute a corresponding witness whenever the adversary $A$ generates a valid argument.

We use the notation $M$ to signify the complete set of executable messages ($msgs$), and $P$ is used to denote the entire set of potential passwords ($pwd$s). The notation $x \overset{\$}{\leftarrow} X$ denotes that $x$ is a value randomly sampled from the set $X$. The order-related symbol, $a \succ b$ indicates that $a$ cannot precede $b$. Lastly, the symbol $\perp$ specifies that a value is undefined or not considered.

IV. Cryptographic Preliminaries

This section introduces the core cryptographic concepts that form the foundation of the zkAA architecture. Specifically, we focus on zero-knowledge proofs and digital signatures. Additionally, we elaborate on the primary security assumptions underpinning our proposed scheme.

A. Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs), first introduced in [20], provide a method for verifying the authenticity of information without disclosing the 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-SNARK) [21]. This technology allows for the succinct verification of information, rendering it an ideal choice for blockchain. ZoKrates [13], a toolkit designed for creating, compiling, and executing ZKP-based smart contracts on Ethereum, is a commonly used resource for implementing zero-knowledge systems.


- $crs \leftarrow zkp.Setup(R)$: This operation takes a relation $R$ as an argument and produces a common reference string $crs$, which serves as a public parameter for proof generation and verification processes.
- $\pi \leftarrow zkp.Prove(crs, io; w)$: This operation generates a proof $\pi$ given the $crs$, input/output $io$, and witness $w$.
- $0/1 \leftarrow zkp.Verify(crs, io, \pi)$: This operation verifies the proof $\pi$ using the $crs$, input/output $io$, and trivially proof $\pi$ as arguments. The function returns a binary outcome indicating whether the verification was successful (1) or not (0).

Our scheme employs the zk-SNARKs protocol that satisfies the properties outlined in [40], [3], [21], [17]:

**Definition IV.1 (Completeness).** For a relation $R$, every $(io, w) \in R$ satisfies the following.

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

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

**Definition IV.2 (Knowledge Soundness).** For all PPT adversaries $A$ there exists a PPT extractor $X_A$ satisfies the following.

$$\Pr \left[ \begin{array}{l} crs \leftarrow zkp.Setup(R); \\ ((io, \pi); w) \leftarrow (A||X_A)(crs); \\ zkp.Verify(crs, io, \pi) = 1 \wedge (io, w) \notin R \end{array} \right] \approx 0 \quad \text{(Computational Knowledge Soundness)}$$

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 IV.3 (Zero-Knowledge).** The following must be satisfied for all PPT adversaries $A$ and $(io, w) \in R$.

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

(Please note additional properties and theorems for completeness and soundness.)
This denotes that the distribution of a regular proof, generated with knowledge of the witness, is indistinguishable from the distribution of a proof simulated by \( \text{zkp.SimProve} \) without the witness. The simulator utilizes the trapdoor \( \tau \) incorporated during the \( \text{crs} \) generation via \( \text{zkp.SimSetup} \). This indicates that the proof does not leak any information aside from the truth of the statement.

**B. Digital Signatures**

Digital signatures form a cornerstone in digital identity verification, finding utility in technologies like JSON Web Tokens [27]. Their primary function lies in authenticating digital data, thereby vouching for its authenticity. Notably, digital signatures extend their reach to the burgeoning field of web3, particularly within blockchain technology, where they certify the authenticity and nonrepudiation of transactions, subsequently bolstering the overall reliability of the blockchain system. For instance, Ethereum [61] employs the Elliptic Curve Digital Signature Algorithm (ECDSA) [26] for transaction signing, while Cardano [23] and Algorand [19] utilize the Edwards-curve Digital Signature Algorithm (EdDSA) [28].

The digital signature protocol \( \text{sig} \) used in this work comprises three operations: \( \text{sig.KeyGen} \), \( \text{sig.Sign} \), and \( \text{sig.Verify} \).

1. \( (pk, sk) \leftarrow \text{sig.KeyGen}() \): This operation generates a pair of keys: a secret key \( sk \) and a corresponding public key \( pk \).
2. \( \sigma \leftarrow \text{sig.Sign}(sk, m) \): This operation takes the sender’s secret key \( sk \) and a message \( m \) as inputs, and produces a signature \( \sigma \).
3. \( 0/1 \leftarrow \text{sig.Verify}(pk, m, \sigma) \): This operation takes the sender’s public key \( pk \), message \( m \), and signature \( \sigma \) as inputs, and returns 1 if the signature is valid and 0 otherwise.

The digital signature scheme used in this work is unforgo-
able under chosen message attacks.

**Definition IV.4 (Unforgeability).** The following must be satisfied for all PPT adversaries \( A \). \( M \) is the set of message-queries that \( A \) has already seen valid signatures through \( \text{sig.Sign} \) from \( sk \).

\[
\begin{align*}
(pk, sk) & \leftarrow \text{sig.KeyGen}(); \\
(m, \sigma) & \leftarrow A(pk); \\
\Pr[\text{sig.Verify}(pk, m, \sigma) = 1 \land m \notin M] & \approx 0 \quad \text{(Unforgeability)}
\end{align*}
\]

An essential property of our digital signature scheme, unforgeability denotes that it should be computationally infeasible for an adversary to generate a valid signature for any message without access to the secret key.

**C. Security Assumptions**

For the reliable operation of our proposed system, we predicate its security on a number of fundamental assumptions:

- We rely on the security of the blockchain and the contracts deployed on it. We assume these to be robust against security vulnerabilities, including threats such as denial-of-service (DoS) attacks [9], [37]. Furthermore, the potential for chain reorganization through a fork is considered negligible within our assumptions. This assumption essentially guarantees the reliability of transaction execution and data storage within our system.
- We assume the usage of a hash function \( H \) that is resistant to collision and preimage attacks [36].
- We posit the secure and confidential transmission of web2 data, specifically, the certificate \( cert \) and sensitive data \( pwd \) (e.g., a password). We assume these data to be transferred over a secure channel.
- We assume that the public key \( pk \) of the certificate-issuing organization is available and verifiable by anyone.

**V. Definitions**

This section presents an innovative scheme of address abstraction, which proposes an alternative to the conventional blockchain address and transaction signing system.

The core concept of address abstraction is the use of the web2 certificate, denoted as \( cert \), to generate a unified and unique identifier, denoted as \( id \). This identifier is designed to be independent of the underlying cryptographic systems of the blockchain. The identifiers \( id \) and \( cert \) play pivotal roles in generating abstracted transactions, similar to how the original blockchain address and its associated private key are utilized in the signing of transactions.

A comparison between the Address Abstraction (AA) and Ethereum’s address schemes can be found in Table III. This comparison delves into several aspects such as the handling of secrets, ordering of messages, and security and privacy measures inherent to both schemes.

**A. Address Abstraction**

The Address Abstraction (AA) scheme is an abstraction over the inherent identifier system of the blockchain. It separates identity handling from the specific characteristics of the blockchain protocol, facilitating registration and utilization of identities independently of the blockchain’s unique features.

For the elucidation, \( R_B \) and \( R_P \) are defined as the relations symbolizing registration and publication, respectively. The registration relation \( R_B \) is expressed by the tuple \((id, pk, cert)\), where \( id \) signifies the abstracted identifier, \( pk \) is the public key assigned by a trusted certificate authority, and \( cert \) is the certificate issued by this authority. The identifier \( id \) is derived from \( cert \) via the hash function \( H \), i.e., \( id \leftarrow H(cert) \). Alternatively, the publication relation \( R_P \) comprises the tuple \((id, msg, m, cert)\), where \( msg \) is the abstracted transaction that the user intends to perform, and \( m \) is akin to the nonce in blockchain accounts, indicating the sequence of requests.

**Definition V.1 (Address Abstraction).** To govern the registration of an abstracted identity \( id \) and the publication of abstracted transaction \( msgs \), the system requires six functions (\( \text{Setup}, \text{Certificate}, \text{RegisterProve}, \text{RegisterVerify}, \text{PublishProve}, \text{PublishVerify} \)).

- \((crs_R, crs_P) \leftarrow \text{Setup}(R_B, R_P)\): This function generates public parameters \( crs_R \) and \( crs_P \) for registration and publication, respectively, from the relations.
Abstracted Address/Transaction
[0x0]cert
[0x0]Chronicle
[0x0]Each
[0x0]A user with a valid
[0x0]Injectiveness
[0x0]Each transaction order is set by account nonce.
B. Properties
(0
(Unique Identifier)
 oppose verifiability (Correctness).
\(\pi\) generates a proof \(\pi_R\), validating that the abstracted identifier \(id\) is derived from the certificate \(cert\) (confirmed by \(id \leftarrow H(cert)\)), and that the certificate is issued by a trusted authority associated with the public key \(pk\).
\(0/1 \leftarrow REGISTERPROVE(crs_R, id, pk; cert)\): This function enables the registration of a unique \(id\) upon successful verification, returning a binary outcome that denotes the validation status of the identifier \(id\). \(0\) represents invalidity, and \(1\) signifies validity.
\(\pi_p \leftarrow PUBLISHPROVE(crs_p, id, msg, m; cert)\): This function generates a proof \(\pi_p\), attesting that the user has a certificate \(cert\) and can calculate the corresponding hash \(id\). The \(id\) should be a previously registered identifier via the REGISTERVERIFY function. Within generating proof, also considers the abstracted transaction \(msg\) and the transaction sequence order \(m\) as constraints.
\((0/1, res) \leftarrow PUBLISHVERIFY(crs_p, id, msg, m, \pi_p)\): This function enables the execution of a requested \(m\)-th abstracted transaction \(msg\) when provided with a valid proof \(\pi_p\) related to an identifier \(id\). The result is denoted as \(res\). It includes a monotonically increasing counter \(m\) to ensure the chronological processing of submitted \(msg\). If successful, it returns \((1, res)\), or \((0, 0)\) otherwise.

B. Properties
The address abstraction must conform to three properties to ensure its functionality: **Unique Identifier** (V.2), **Immutable Request** (V.3), and **Privacy-Preserving** (V.4). Preserving these properties is vital for successfully transitioning from the traditional blockchain address to the proposed address abstraction scheme.

1) **Unique Identifier**: The Unique Identifier property assures the singularity of a user’s identity (Injectiveness), and is solely controlled by the user who owns the secret (Unforgeability), and is publicly verifiable (Correctness).

Definition V.2 (Unique Identifier). 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 \((cert^x, cert^y)\), \(cert^x \neq cert^y \implies id^x \neq id^y\) (Injectiveness)

Under the injectiveness condition, two distinct certificates would not map to the same identifier, ensuring that each certificate \(cert\) and its derived identifier \(id\) remain unique within the system.

for all PPT adversaries \(A\) there exists a PPT extractor \(\chi_A\),

\[
\Pr \left[ (crs_R, \bot) \leftarrow \text{Setup}(R_R, R_P); \right.
\]

\[
\left. ((id_R, pk, \pi_R); cert_R) \leftarrow (A||\chi_A)(crs_R); \right. \]

\[
\left. s_R \leftarrow \text{REGISTERPROVE}(crs_R, id_R, pk, \pi_R); \right. \]

\[
\left. ((id_R, msg, m, \pi_p); cert) \leftarrow (A||\chi_A)(crs_p); \right. \]

\[
\left. (s_p, \bot) \leftarrow \text{PUBLISHPROVE}(crs_p, id, msg, m, \pi_p); \right. \]

\[
\left. s_R \wedge s_p = 1 \land ((id_R, msg, m), cert) \notin R_P \right] \approx 0 \quad \text{(Unforgeability)}
\]

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 \(cert\). This ensures that valid identifier registration and abstracted transaction publications cannot be accomplished without the creator possessing the necessary secret.

\[
\Pr \left[ (crs_R, crs_p) \leftarrow \text{Setup}(R_R, R_P); \right. \]

\[
\left. cert \leftarrow \text{Certificate}(pwd); id \leftarrow H(cert); \right. \]

\[
\left. \pi_R \leftarrow \text{REGISTERPROVE}(crs_R, id, pk, cert); \right. \]

\[
\left. s_R \leftarrow \text{REGISTERVERIFY}(crs_R, id, pk, \pi_R); \right. \]

\[
\left. \pi_p \leftarrow \text{PUBLISHPROVE}(crs_p, id, msg, m, cert); \right. \]

\[
\left. (s_p, \bot) \leftarrow \text{PUBLISHVERIFY}(crs_p, id, msg, m, \pi_p); \right. \]

\[
\left. s_R \wedge s_p = 1 \land ((id, pk), cert) \in R_R \land ((id, msg, m), cert) \in R_P \right] \}

\[
= 1 \quad \text{(Correctness)}
\]

A user with a valid \(cert\) can always generate a proof \(\pi_p\) for registration and \(\pi_p\) for publication to validate to the verifier.
2) Immutable Request: This property guarantees that any proof-coupled request can solely execute the prescribed actions, and is impervious to modifications during transit. This characteristic is vital for mitigating the threat of front-running attacks. Such attacks are characterized by malicious entities taking advantage of transaction observability, often exploiting publicly accessible proofs via transactions in the mempool, then manipulating requests for their gain.

Definition V.3 (Immutable Request). Requests on the system (\text{Setup}, \text{Certificate}, \text{RegisterProve}, \text{RegisterVerify}, \text{PublishProve}, \text{PublishVerify}) is immutable, if the system ensures the constraints of \textbf{Chronicle} and \textbf{Tamper Resistance} for all \( m \geq 0, msg_{m+1} \succ msg_m \) (Chronicle).

It establishes a chronological sequence of the abstracted transactions in the system. Each message succeeds its predecessor in a strictly increasing order.

for all PPT adversaries \( \mathcal{A} \),

\[
\begin{align*}
\left[(crs_R, \bot) &\leftarrow \text{Setup}(R_R, R_P); \\
\right. \\
&(pwd, pwd^A) \leftarrow \mathcal{P} \text{ s.t. } pwd^A \neq pwd; \\
&\text{cert} \leftarrow \text{Certificate}(pwd); id \leftarrow \mathcal{H}(\text{cert}); \\
&\text{cert}^A \leftarrow \text{Certificate}(pwd^A); id^A \leftarrow \mathcal{H}(\text{cert}^A); \\
&\pi_R \leftarrow \text{RegisterProve}(crs_R, id, pk; \text{cert}); \\
&s_R \leftarrow \text{RegisterVerify}(crs_R, id^A, pk, \pi_R); \\
&s_R = 1 \land ((id^A, pk), \text{cert}) \notin R_R
\end{align*}
\]

and

\[
\begin{align*}
\left[(crs_R, crs_P) &\leftarrow \text{Setup}(R_R, R_P); \\
&\text{cert} \leftarrow \text{Certificate}(pwd); id \leftarrow \mathcal{H}(\text{cert}); \\
&\pi_R \leftarrow \text{RegisterProve}(crs_R, id, pk; \text{cert}); \\
&s_R \leftarrow \text{RegisterVerify}(crs_R, id, pk, \pi_R); \\
&(msg, msg^A) \leftarrow \mathcal{M} \text{ s.t. } msg^A \neq msg; \\
&\pi_P \leftarrow \text{PublishProve}(crs_P, id, msg, m; \text{cert}); \\
&(s_P, \bot) \leftarrow \text{PublishVerify}(crs_P, id, msg^A, m, \pi_P); \\
&s_R \land s_P = 1 \\
&\land ((id, pk), \text{cert}) \in R_R \land ((id, msg^A, m), \text{cert}) \notin R_P.
\end{align*}
\]

\( \approx 0 \) (Tamper Resistance)

This property indicates the robustness of the cryptographic system against unauthorized modifications. It asserts that the probability of an adversary successfully tampering with the registration of an identifier \( id \) or the publication of an abstracted transaction \( msg \), while simultaneously preserving the original proof, is negligible.

The Unforgeability ensures the authenticity, asserting that only users with the correct secret can create valid registrations and transactions. On the other hand, Tamper Resistance, maintains the integrity of the registered identifiers and published transactions, asserting that once a proof has been published, it cannot be altered by any adversary.

3) Privacy-Preserving: This property pertains to the capacity of a system to safeguard user personal information, denoted as \( \text{cert} \), from public exposure, thereby preserving its confidentiality.

Definition V.4 (Privacy-Preserving). System (\text{Setup}, \text{Certificate}, \text{RegisterProve}, \text{RegisterVerify}, \text{PublishProve}, \text{PublishVerify}) can be classified as privacy-preserving, if the secret \( \text{cert} \) associated with the identifier \( id \) remains undiscovered both prior to and following the execution of any procedure.

for all PPT adversaries \( \mathcal{A} \),

\[
\begin{align*}
\left[\left[(crs_R, \bot) \leftarrow \text{Setup}(R_R, R_P); \\
\right. \\
&\text{cert} \leftarrow \text{Certificate}(pwd); id \leftarrow \mathcal{H}(\text{cert}); \\
&\pi_R \leftarrow \text{RegisterProve}(crs_R, id, pk; \text{cert}); \\
&s_R \leftarrow \text{RegisterVerify}(crs_R, id^A, pk, \pi_R); \\
&s_R = 1 \land \mathcal{A}(crs_R, \pi_R, \pi_R) = 1
\end{align*}
\]

and

\[
\begin{align*}
\left[\left[(crs_R, crs_P) \leftarrow \text{Setup}(R_R, R_P); \\
&\text{cert} \leftarrow \text{Certificate}(pwd); id \leftarrow \mathcal{H}(\text{cert}); \\
&\pi_R \leftarrow \text{RegisterProve}(crs_R, id, pk; \text{cert}); \\
&s_R \leftarrow \text{RegisterVerify}(crs_R, id, pk, \pi_R); \\
&s_R \leftarrow \text{RegisterVerify}(crs_R, id, pk, \pi_R); \\
&\pi_P \leftarrow \text{PublishProve}(crs_P, id, msg, m; \text{cert}); \\
&s_R = 1 \land \mathcal{A}(crs_P, \tau_P, \pi_P) = 1
\end{align*}
\]

The proofs simulated by \text{SimRegisterProve} and \text{SimPublishProve}, which utilize the trapdoors \( \tau_R \) and \( \tau_P \) generated through \text{SimSetup}, are generated without a witness. The distributions of \( \pi_R \) and \( \pi_P \) remain indistinguishable whether the witness \( \text{cert} \) is known or not.

VI. ZK ADDRESS ABSTRACTION

We introduce an implementation of Address Abstraction, named zero-knowledge Address Abstraction (zkAA), which leverages zk-SNARKs to preserve user confidentiality while enabling the registration and utilization of identities on a blockchain. The application of zk-SNARKs engenders zkAA’s efficiency and succinctness, thereby rendering it suitable for integration within smart contracts. In zkAA, proofs are verified on the contract, providing an incorruptible method for managing and verifying user identities.
A Design of zkAA

The zkAA design is a concrete framework consisting of functions in the domain of address abstraction: SETUP, CERTIFICATE, REGISTERPROVE, REGISTERVERIFY, PUBLISHPROVE, PUBLISHVERIFY.

- \((crs_R, crs_P) \leftarrow \text{SETUP}(R_R, R_P)\): The setups generate the public parameters \(crs_R \leftarrow \text{zkp.Setup}(R_R)\) and \(crs_P \leftarrow \text{zkp.Setup}(R_P)\).
- \(cert \leftarrow \text{CERTIFICATE}(pwd)\): A user obtains a JSON Web Token (JWT) \([27]\), referred to as a cert, from an institution using their personal web2 data, denoted as pwd. This institution authorizes the cert by applying its secret key \(sk\) via \(\text{sig.Sign}\) of EdDSA \([28]\). The user derives the identifier \(id\) from the cert by invoking the hash function \(H, SHA256\) \([45]\).
- \(\pi_R \leftarrow \text{REGISTERPROVE}(crs_R, id, pk; cert)\): A proof \(\pi_R\) is produced via \(\pi_R \leftarrow \text{zkp.Prove}(crs_R, (id, pk); cert)\) to validate two claims: Firstly, that the identifier \(id\) is computed as \(id = H(cert)\). Secondly, the cert has been signed using the secret key correlated to the public key \(pk\), thereby passing \(\text{sig.Verify}\) validation.
- \(0/1 \leftarrow \text{REGISTERVERIFY}(crs_R, id, pk, \pi_R)\): The user registers the identifier \(id\) and proof \(\pi_R\) on the blockchain, where the zkAA contract applies \(\text{zkp.Verify}(crs_R, (id, pk), \pi_R)\) to check their validity. Upon successful verification and confirmation of the identifier \(id\) as being previously unregistered, it is appended to the contract’s map data structure of valid identifiers. The determination of registration status relies on the evaluation of \(\text{nonce}[id] == 0\), indicative of an unregistered state. Should the \(id\) have been registered previously, its corresponding \(\text{nonce}[id]\) would satisfy the condition \(\text{nonce}[id] \geq 1\), since the nonce for a newly registered \(id\) begins at 1. It imitates the nonce configuration used in contract accounts according to EIP-161 \([59]\).
- \(\pi_p \leftarrow \text{PUBLISHPROVE}(crs_P, id, msg, m; cert)\): The user generates \(\pi_p \leftarrow \text{zkp.Prove}(crs_P, (id, H(msg), m); cert)\), asserting that the identifier \(id\) is obtained from the cert, without revealing cert. This proof is distinctly associated with a specific message \(msg\), thwarting its potential reuse and mitigating potential attacks\([34]\). For efficiency, \(H(msg)\) is employed rather than the raw \(msg\) to neglect concerns about \(msg\)’s length.
- \((0/1, res) \leftarrow \text{PUBLISHVERIFY}(crs_P, id, msg, m, \pi_p)\): In alignment with the \text{PUBLISHPROVE} function, the contract verifies \(\pi_p\) through \(\text{zkp.Verify}(crs_P, (id, H(msg), m), \pi_p)\) to confirm that the sender is the legitimate owner of the
identity denoted by \(id\) for the specific message \(msg\). The identifier \(id\) is expected to have been previously registered using the \texttt{REGISTER} function. The sequence number \(m\) acts as a counter analogous to an account’s nonce. Hence the value should exceed the current \(nonce[\text{id}]\) by more than 1. If \(\pi_p, \text{msg}\), and its sequence \(m\) pass validation, the intended message is executed, yielding a result labeled as \(res\). After successful execution, \(nonce[\text{id}]\) is updated to match the value of \(m\).

The necessity of performing registration prior to publication arises from the complexity of the registration process. This process involves the usage of EdDSA \texttt{sig.Verify} logic, which in turn can result in a lengthier generation time for the proofs, as depicted in Section VII. Consequently, our protocol follows a sequential approach by first registering the identifier \(id\) and subsequently utilizing it in the subsequent phases. The latter merely checks that \(id\) matches the hashed value of the certificate \(cert\).

Moreover, by distinctly separating the registration and publication, our smart contract gains the capability to effectively manage the registration statuses of entities. This separation affords the flexibility to impose various conditions and criteria. For instance, the system can enforce time-based expiration conditions, ensure compliance with regulatory requirements such as the Office of Foreign Assets Control (OFAC) regulations [54], [41], and compliance with data protection standards such as the General Data Protection Regulation (GDPR) [55].

\subsection{Algorithms of \texttt{zkAA}}

Figure 1 offers an overview of \texttt{zkAA}, highlighting the distinct phases, each of which is reliant on the successful execution of the preceding one. The detailed \texttt{SETUP}, \texttt{CERTIFICATE}, \texttt{REGISTER} and \texttt{PUBLISH} phase’s processes are written in Algorithms 1, 2, 3, and 4.

\subsubsection{Setup Phase: \texttt{Smart Contract} Side}
The primary objective of the function \texttt{SETUP} is to generate two Common Reference Strings (CRSs) \texttt{crs}_R and \texttt{crs}_P. Each CRS comprises a unique pair of a proving key and a verification key, whose generation is governed by a specific circuit. The two CRSs serve distinct purposes within the \texttt{zkAA} system. The first CRS, \texttt{crs}_R, is utilized in the registration circuit, playing a significant role in the user registration process. The second CRS, \texttt{crs}_P, is leveraged to facilitate the publication of abstracted transactions.

\subsubsection{Certificate Phase: \texttt{Client} Side}
The user makes a request for a \texttt{cert} from the institute using their password (pwd). \texttt{Institute} Side: Then the institute, through the \texttt{ISSUE} procedure, constructs a \texttt{cert}_{\text{header}}, specifying the \texttt{type} and hashing \texttt{algorithm}, followed by the \texttt{cert}_{\text{payload}} with the appropriate \texttt{claims} based on the provided \texttt{pwd}. Afterward, the institute generates a \texttt{cert}_{\text{sig}} by signing the combination of encoded \texttt{cert}_{\text{header}} and \texttt{cert}_{\text{payload}} with its secret key (sk). The final \texttt{cert} is then assembled by concatenating \texttt{cert}_{\text{header}}, \texttt{cert}_{\text{payload}}, and \texttt{cert}_{\text{sig}}, in the form of a JWT. Upon successful authentication, the requested certificate is issued to the user. This process transpires off-chain between the user and the institute, through a secure channel, thus ensuring the confidentiality and integrity of the \texttt{pwd} and \texttt{cert}.

\subsubsection{Register Phase: \texttt{Client} Side}
At this stage, the client generates a proof \(\pi_p\) that authenticates the user’s identifier. Following this, the client initiates the \texttt{REGISTER} transaction, thereby commencing the registration process. \texttt{Smart Contract} Side: Upon receipt of the \texttt{REGISTER} transaction, the \texttt{REGISTER} function examines the proof’s validity and the identifier’s uniqueness. If the checks pass, the contract assigns a nonce of 1 to the identifier and conducts actions in accordance with the registration type, denoted by \(\tau\). For further details regarding \(\tau\), refer to Section VI-D.

\subsubsection{Publish Phase: \texttt{Client} Side}
The client constructs a proof, noted as \(\pi_p\), which affirms the legitimacy of the user’s identifier associated with a specific message. Subsequently, the client initiates a \texttt{PUBLISH} transaction. \texttt{Smart Contract} Side: Based on the registration type \(\tau\), the \texttt{PUBLISH} transaction either calls the \texttt{PUBLISH} function or executes the message via a normal transaction. The \texttt{PUBLISH} function scrutinizes the proof, confirms the registration status of the identifier, and checks that the sequence number \(m\) is incremented correctly. Upon successful validation, the smart contract proceeds with the message execution, returning the result along with a success status.

\begin{algorithm}
\begin{algorithmic}[1]
\caption{\texttt{SETUP}}
\Statex [zkAA Smart Contract]
\Function{\texttt{SETUP}}{\texttt{\tau}_R, \texttt{\tau}_P}
\State \texttt{crs}_R \leftarrow \texttt{zkp.Setup}(\texttt{\tau}_R)
\State \texttt{crs}_P \leftarrow \texttt{zkp.Setup}(\texttt{\tau}_P)
\Return \texttt{(crs}_R, \texttt{crs}_P)\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\begin{algorithmic}[1]
\caption{\texttt{CERTIFICATE}}
\Statex [zkAA Client]
\Function{\texttt{CERTIFICATE}}{pwd}
\State \texttt{cert} \leftarrow \texttt{call ISSUE}(pwd)
\State \Return cert\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\begin{algorithmic}[1]
\caption{\texttt{ISSUE}}
\Statex [Institute]
\Procedure{\texttt{ISSUE}}{pwd}
\State \texttt{construct} \texttt{cert}_{\text{header}} \texttt{with the type and algorithm}
\State \texttt{construct} \texttt{cert}_{\text{payload}} \texttt{with claims by pwd}
\State \texttt{cert}_{\text{sig}} \leftarrow \texttt{sig.Sign}(sk, (\texttt{cert}_{\text{header}}|\texttt{cert}_{\text{payload}}))
\State \texttt{cert} = (\texttt{cert}_{\text{header}}|\texttt{cert}_{\text{payload}}|\texttt{cert}_{\text{sig}})
\State \Return cert\end{algorithmic}
\end{algorithm}
C. Properties of zkAA

The zkAA process is designed to ensure three fundamental properties of address abstraction, namely Unique Identifier, Immutable Requests, and Privacy-Preserving. The following is a brief exposition proving that the zkAA protocol indeed meets these requisite properties. Detailed security proofs can be found in Appendix A.

- **Unique Identifier.** The zkAA protocol produces a unique identifier, id, representing a user’s identity. The employed hash function H is resistant to collision and preimage attacks, thereby ensuring injectivity under our security assumptions. The protocol’s privacy-preserving characteristic, made possible by zk-SNARK, guarantees that neither registration nor publication can take place without the knowledge of the cert. This effectively guarantees unforgeability. Additionally, generating valid proofs and verifying them is consistent, thereby ensuring correctness.

- **Immutable Request.** The user request, represented as msg, is included in the proof np, enhancing its resistance to tampering. The sequential number n functions as a nonce, preserving the sequence of abstracted transactions by ensuring that they execute in the correct order without modifications or omissions. This feature ensures the transaction chronology.

- **Privacy-Preserving.** Privacy is a fundamental aspect of the zkAA protocol, enabled by zk-SNARK. The protocol ensures that public information does not reveal the original cert, thereby safeguarding user privacy.

D. Explicit Identifier

This paper proposes the use of cert-based identifiers for improved identity management. By hashing a cert to create the identifier, i.e., H(cert), we provide a unified solution to handle the variability in address systems across different chains. However, existing decentralized applications (dApps) that are primarily built around the address system may not be directly compatible with our approach.

To resolve this issue, we introduce the concept of explicit identifiers. Except for the type τid, which does not utilize

**Algorithm 3 REGISTER**

1: pre-defined registration type τ ∈ {τid, τCA, τEOA}
2: id ← H(cert)

[zkAA Client]

3: function REGISTERPROVE(crsR, id, pk; cert)
4: πR ← zkpprove(crsR, (id, pk); cert)
5: initiate REGISTER(crsR, id, pk, πR)

6: transaction REGISTER(crsR, id, pk, πR)
7: call REGISTERVERIFY(crsR, id, pk, πR)

[zkAA Smart Contract]

8: function REGISTERVERIFY(crsR, id, pk, πR)
9: assert(nonce[id] == 0)
10: assert(zkp.Verify(crsR, (id, pk), πR))
11: nonce[id] ← 1
12: switch τ do
13: case τid then return 1 ▷ (id)
14: case τCA then ▷ (id → CA)
15: create contract wallet controlled by id
16: map id into address of contract wallet CA
17: return 1
18: case τEOA then ▷ (id → EOA)
19: create NFT contains id
20: give ownership to msg.sender
21: return 1

**Algorithm 4 PUBLISH**

1: pre-defined registration type τ ∈ {τid, τCA, τEOA}
2: id ← H(cert)
3: msg ← (target, funcsig, calldata, value)

[zkAA Client]

4: function PUBLISHPROVE(crsP, id, msg, m; cert)
5: πP ← zkpprove(crsP, (id, H(msg), m); cert)
6: initiate PUBLISH(crsP, id, msg, m, πP)

7: transaction PUBLISH(crsP, id, msg, m, πP)
8: if τ ∈ {τid, τCA} then ▷ (id) or (id → CA)
9: call PUBLISHVERIFY(crsP, id, msg, m, πP)
10: else ▷ (id → EOA)
11: execute msg through normal transaction

[zkAA Smart Contract]

12: transaction PUBLISHVERIFY(crsP, id, msg, m, πP)
13: assert(nonce[id] ≥ 1)
14: assert(m == nonce[id] + 1)
15: assert(zkp.Verify(crsP, (id, H(msg), m), πP))
16: nonce[id] ← m
17: switch τ do
18: case τid then ▷ (id)
19: res ← execute msg with identifier id
20: return (1, res)
21: case τCA then ▷ (id → CA)
22: res ← call ExecuteMsg(msg)
23: return (1, res)

[Contract Wallet]

24: function EXECUTEMSG(msg)
25: assert(msg.sender == address(zkAA contract))
26: res ← execute msg
27: return res
the explicit identifier, explicit identifiers facilitate the usage of dApps ($\tau_{CA}$) or enhance efficiency ($\tau_{EOA}$). As outlined in Algorithm 3 and 4, functions $\text{REGISTER}, \text{VERIFY}, \text{PUBLISH}, \text{VERIFY}$, and the transaction $\text{PUBLISH}$ of zkAA encompass a conditional branch based on the explicit identifier type $\tau$. This $\tau$ is predefined during contract deployment, hence, it is not required as an input parameter. Figure 2 illustrates the flow of explicit identifier usage for each type.

1) $\tau_{id}$ ($id$): In this method, $id \leftarrow H(cert)$ itself acts as the explicit identifier. While this method may not be compatible with traditional dApps (since the $msg.sender$ is always the address of zkAA, necessitating specific dApp logic to distinguish users), it facilitates simpler secret management, as $\pi_R$ and $\pi_P$ are generated solely based on the $cert$ without involving the blockchain address.

2) $\tau_{CA}$ ($id \mapsto CA$): During the registration, as depicted in line 14 of Algorithm 3, a new contract wallet is created and linked to the $id$. Here, the explicit identifier is the contract wallet’s address (CA). This method supports compatibility with existing dApps that rely on the traditional address system. For forwarding a request $msg$ to a target, the contract wallet must have a $\text{ExecuteMsg}(msg)$ function, which can be called by the zkAA contract as shown in Algorithm 3 line 22. This approach mirrors Account Abstraction [7] as allowing users to utilize contract wallets containing codes.

3) $\tau_{EOA}$ ($id \mapsto EOA$): In this variant, a user acquires a Non-Fungible Token (NFT) [14] that represents the $id$. It links the $id$ to a specific Externally Owned Account (EOA) by ownership. This method reflects the traditional mapping approach employed for integrating web2 identities into web3. This approach permits users to interact with dApps using normal transactions instead of abstracted transactions containing proof. However, it requires the execution of NFT ownership verification logic, potentially introducing an additional sequence within the dApp, depicted as B-2 under $\tau_{EOA}$ in Figure 2. This method offers a balance between cost and convenience, with normal transactions costing less than abstracted transactions as they do not necessitate proof verification. Nevertheless, it also shares the drawbacks of the mapping approach in achieving seamless dApps integration, as discussed in Section I-A.

### VII. Experiments

To evaluate the effectiveness and efficiency of our proposed zero-knowledge address abstraction implementation, we conducted experiments to measure the gas costs for verification on Ethereum [61] and Polygon [53], as well as the time required to generate proofs. The average gas price on Ethereum was 14 gwei (14 $\times$ 10$^{-9}$ ETH), with an ETH price of 1412.49 USD. The average gas price on Polygon was 51.6 gwei (51.6 $\times$ 10$^{-9}$ MATIC), with a MATIC price of 0.91 USD. These measurements were as of January 13, 2023.

Experiments were carried out on a local machine — an Apple M1 Pro with 16GB of memory.

#### A. Implementations

We utilized EdDSA [2] to sign the certificate $cert$. To generate the signature, we used the SHA512 [45] digest of the $cert$. The resultant signature was then used to compute the identifier $id$ from $cert$, applying the SHA256 [45] hash function via the 1024-bit padded. The registration phase involved the verification of EdDSA and SHA256 through the proof $\pi_R$, whereas the publication phase used the proof $\pi_P$ to verify SHA256 on the $cert$.

We implemented the proposed scheme using ZoKrates [13], based on the Groth16 [21]. The ALT-BN128 curve was utilized for efficient on-chain verification via pre-compiled contracts in Ethereum at addresses 0x6 for ADD, 0x7 for MUL, and 0x8 for pairing check [48], [6]. The contract was optimized using Solidity 0.8.17 version with the option (200 runs). Our implementation consisted of two circuits: the registration with 169188 constraints and the publication with 75945 constraints.

In an effort to contribute to the research community, we have made the implementation and experimental codes publicly available on GitHub [7] for replication and further examination.

#### B. Verification Costs

Table IV examines the costs associated with on-chain verification. Deploying the registration verification contract requires 1410472 gas, which amounts to approximately $27.89 on Ethereum. Similarly, the publication verification contract deployment necessitates 921902 gas, costing around $18.23 on Ethereum. Despite its initial cost, it is a one-time expenditure.

The verification of the identifier $id$ incurs costs during registration and publication. The average gas cost for registering is 387494, translating to about $7.66 on Ethereum. The publishing cost is 239953 gas, approximately $4.75 on Ethereum. The verification cost during publication is reasonably economical.

#### TABLE IV: Gas cost on Ethereum and Polygon (gas/$)

<table>
<thead>
<tr>
<th>Methods</th>
<th>min</th>
<th>max</th>
<th>avg (sd) USD (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Registration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>verifyTx</td>
<td>387456</td>
<td>387516</td>
<td>387494 $7.6626</td>
</tr>
<tr>
<td>Deploy</td>
<td>-</td>
<td>-</td>
<td>1410472 $27.8919</td>
</tr>
<tr>
<td>Publication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>verifyTx</td>
<td>239906</td>
<td>239966</td>
<td>239953 $4.7450</td>
</tr>
<tr>
<td>Deploy</td>
<td>-</td>
<td>-</td>
<td>921902 $18.2305</td>
</tr>
</tbody>
</table>

#### TABLE V: Time taken for generating proofs (sec)

<table>
<thead>
<tr>
<th>Methods</th>
<th>min</th>
<th>max</th>
<th>avg (sd)</th>
<th>med</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Registration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>witness</td>
<td>3.7511</td>
<td>3.9998</td>
<td>3.831 (0.0474)</td>
<td>3.8300</td>
</tr>
<tr>
<td>proof</td>
<td>5.4607</td>
<td>11.8173</td>
<td>5.7365 (0.6323)</td>
<td>5.6448</td>
</tr>
<tr>
<td>Publication</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>witness</td>
<td>1.7636</td>
<td>1.9181</td>
<td>0.8132 (0.0285)</td>
<td>1.8098</td>
</tr>
<tr>
<td>proof</td>
<td>2.5685</td>
<td>2.9915</td>
<td>2.6759 (0.0640)</td>
<td>2.6668</td>
</tr>
</tbody>
</table>
particularly on cost-efficient blockchains like Polygon, with $0.02 and $0.01 for registration and publication, respectively.

C. Overhead on Generating Proofs

Table V showcases the average time required to generate witnesses and proofs. During the registration phase, the process of creating both witness and proof took an average of 9.5746 seconds. Conversely, for the publication of abstracted transactions, it took an average of 3.4891 seconds. The proof generation of registration takes longer than publication as it involves more logic about validating the EdDSA signature.

While this time overhead is noticeable, it remains within an acceptable limit, given that Ethereum’s block interval is approximately 12 seconds, providing sufficient computation and broadcasting time.

VIII. Applications

The Address Abstraction (AA) scheme we proposed unveils a wealth of potential applications. By decoupling the identifier from the private key, AA streamlines identity management across both web2 and web3 platforms, facilitating seamless user interactions, as illustrated in Figure 3.

A. Web2-Web3 Hybrid Applications

Hybrid applications that amalgamate the advantages of both traditional web2 and emerging web3 systems have emerged, particularly in resource-demanding sectors such as Artificial Intelligence (AI) and gaming. These applications marry the high-performance and user-friendly aspects of centralized systems with the robust security and decentralization offered by blockchain, resulting in an enhanced user experience.

However, one notable hurdle in realizing these hybrid applications is the fragmented user identity management due to the distinct identifier systems of web2 and web3. AA overcomes this impediment, ensuring a consistent identifier representation across both systems. Based on the same certificate $cert$, AA eradicates the need for users and service providers to manage multiple addresses across various networks, thereby simplifying identity management.

B. Multi-chain Decentralized Applications

Managing user identities across multiple blockchains is traditionally a formidable challenge, largely due to the incompatibility of address systems employed by various blockchains, such as Ethereum, Cosmos, and Polkadot, among others. This incompatibility often leads to complications when trying to identify users across different decentralized applications.

AA presents a viable solution to this challenge. It establishes a consistent, blockchain-agnostic identifier by leveraging a non-revealing certificate $cert$ and its hash value $\mathcal{H}(cert)$, thereby eliminating the complications posed by heterogeneous address systems. This approach not only enables interoperability among different blockchains, but also paves the way for developing decentralized cross-chain applications while ensuring secure user identification. For example, in conventional asset transfer procedures between different blockchain networks, which rely on a bridge like [62], [38], users are required to input a receiver’s address. Given the potential incompatibility of the address with the current blockchain, this process can result in input errors, heightening the risk of asset loss. The AA scheme circumvents this issue by providing each user with a consistent identifier, mitigating the risk of address input errors and reducing the need for managing multiple private keys for various blockchains.

IX. Conclusion

This paper introduces a novel solution to identity management in blockchain: the Address Abstraction (AA) scheme. It enables users to securely unify their web2 identities with their web3 presence, eliminating the need to store additional secrets and effectively protecting sensitive information.

Our proposed implementation of AA, zero-knowledge Address Abstraction (zkAA), stands out from the previous works through its unique advantages. Users are only required to maintain their web2 identities, thereby negating the necessity for managing their private keys. The same identifier can be applied across various blockchains, negating the need for managing multiple addresses. A seamless integration between web2 and web3 services is achieved, owing to the utilization of the same secret.

Looking to the future, there are multiple potential improvements to the zkAA implementation. Current zk-SNARK tools such as [13], [39] typically utilize Quadratic Arithmetic Programs (QAPs) [21], [22], [16], [43], which necessitate a trusted setup. However, emerging research in zero-knowledge proofs [50], [1], [5], [4] aims to eliminate this requirement, bolstering decentralization and trust. We plan to incorporate these developments into zkAA, with the goal of obviating the SETUP phase. Moreover, the incorporation of commit-and-prove (CP) schemes [15], [9] into zkAA is a promising future direction. CP-SNARKs facilitate hash verification through pre-uploaded commitments such as Pedersen commitments [44], potentially enhancing proof generation efficiency at the expense of a slight increase in verification costs. We intend to explore CP-SNARKs in future work, evaluating their potential benefits and trade-offs.
Algorithm 5 \textcircled{5} \textsc{IdealSetup}

\begin{enumerate}
\item \textbf{function} \textsc{IdealSetup}() \textbf{end function}
\item $T$ creates empty private table $\text{PrivTab}_p$
\item $T$ creates empty private table $\text{PrivTab}_r$
\item $T$ creates empty private table $\text{PrivTab}_p$
\item $T$ creates empty public table $\text{PubTab}_r$
\item $T$ creates empty public table $\text{PubTab}_p$
\end{enumerate}

Algorithm 6 \textcircled{6} \textsc{IdealCertificate}

\begin{enumerate}
\item \textbf{function} \textsc{IdealCertificate}(pwd) \textbf{end function}
\item Participant sends $pwd$ to $T$
\item If $pwd \in \text{PrivTab}_c$ then
\item $T$ selects $(id, cert)$ from $\text{PrivTab}_c$ by $pwd$
\item $T$ generates a unique random value $id$
\item $T$ generates another unique random value $cert$
\item $T$ inserts $(id, cert)$ in $\text{PrivTab}_c$ by $pwd$
\item $T$ sends $(id, cert)$ to participant
\end{enumerate}

APPENDIX

The security model adheres to the simulation-based definition proposed by [18]. In this section, the ideal-world execution, denoted as $\text{Ideal}_{I,S}$, and the real world execution, denoted as $\text{Real}_{AA,A}$, for Address Abstraction are defined.

A. Ideal-world Execution

\textbf{Definition A.1} ($\text{Ideal}_{I,S}$). $\text{Ideal}_{I,S}$ denotes the ideal implementation of the AA scheme with the trusted party $T$ and simulator $S$, wherein all cryptographic operations are substituted to $T$. The simulator $S$ in ideal-world execution is assumed to be incapable of forging the function output generated by $T$ and distinguishing the data — hash value and signature — from a unique random value. In the ideal-world execution, participants only interact with the $T$ through an interface outlined in Algorithms \textcircled{5} \textcircled{6} \textcircled{7} and \textcircled{8}

\textcircled{1} At the function $\textsc{IdealSetup}$ in Algorithm \textcircled{5}, the trusted party $T$ is responsible for initializing the foundational data structures of the protocol. Specifically, $T$ creates three empty private tables, namely $\text{PrivTab}_p$, $\text{PrivTab}_r$, and $\text{PrivTab}_p$, alongside two empty public tables, $\text{PubTab}_r$ and $\text{PubTab}_p$. The tables are structured as key-value pairs. In addition, keys and values can be independently accessed using the $\text{Tab}.keys$ and $\text{Tab}.values$ functions, respectively. We assume that only $T$ holds the authority to write values into the append-only tables. Access to the private tables is strictly limited to $T$, even read operations are not permissible for any other party. Conversely, the public tables are accessible for reading to all participants.

\textcircled{2} Algorithm \textcircled{7} is bifurcated into two separate functions: $\textsc{IdealRegisterProve}$ and $\textsc{IdealRegisterVerify}$. The $\textsc{IdealRegisterProve}$ function enables participants to obtain a proof, designated as $\pi_R$. This is achieved by sending their unique identifier $id$, the publicly-known public key of an institution $pk$, and a certificate $cert$ to the trusted entity, $T$. Upon receipt of these components, $T$ executes a validation process to authenticate the $id$ and $cert$. Once these components are confirmed as legitimate, $T$ then forwards $\pi_R$ back to the participant. The $\textsc{IdealRegisterVerify}$ function allows participants to register their $id$ by sending $id$, $pk$, and $\pi_R$ to $T$. $T$ verifies their validity and checks their previous registration status. If these components pass the verification process, the registration is successful and reflected in the $\text{PubTab}_p$.

\textcircled{3} Algorithm \textcircled{8} encompasses two functions: $\textsc{IdealPublishProve}$ and $\textsc{IdealPublishVerify}$. In the $\textsc{IdealPublishProve}$ function, the participant sends their $pwd$ to $T$. Upon receipt, $T$ verifies its presence in the $\text{PrivTab}_c$. If the password is already present, $T$ retrieves and provides the associated unique identifier and certificate to the participant. However, if the password is unrecognized, $T$ generates a unique identifier and certificate, stores this data in the $\text{PrivTab}_c$, and communicates this information to the participant.
Algorithm 7  IdealRegister

1: function IdealRegisterProve(id, pk, cert)
2: Participant sends (id, pk, cert) to T
3: if (id, cert) ∉ PrivTab_r.value then
4: abort ⇔ invalid id and/or cert
5: if (id, pk, cert) ∈ PrivTab_r.keys then
6: T selects π_R from PrivTab_r by (id, pk, cert)
7: else
8: T generates a unique random value π_R
9: T inserts π_R in PrivTab_r by (id, pk, cert)
10: T sends π_R to participant

Algorithm 8  IdealPublish

1: function IdealPublishProve(id, msg, m, cert)
2: Participant sends (id, msg, m, cert) to T
3: if (id, cert) ∉ PrivTab_r.value then
4: abort ⇔ invalid id and/or cert
5: k ← (id, msg, m, cert)
6: if k ∉ PrivTab_p.keys then
7: T selects π_p from PrivTab_p by k
8: else
9: T generates a unique random value π_p
10: T inserts π_p in PrivTab_p by k
11: T sends π_p to participant

12: function IdealPublishVerify(id, msg, m, π_p)
13: Participant sends (id, msg, m, π_p) to T
14: k_r ← (id, pk)
15: if k_r ∉ PubTab_r.keys then
16: T sends (0, ⊥) to participant ⇔ unregistered id
17: else
18: T selects (π_R, nonce) from PubTab_r by k_r
19: if nonce < 1 then
20: T sends (0, ⊥) to participant ⇔ invalid nonce
21: else if m ≠ nonce + 1 then
22: T sends (0, ⊥) to participant ⇔ invalid m
23: k_p ← (id, msg, m)
24: v_r ← (π_R, nonce + 1)
25: res ← execute msg with identifier id
26: if k_p ∈ PubTab_p.keys then
27: T sends (0, ⊥) to participant ⇔ passed m
28: else if π_p ∉ PrivTab_p.value then
29: T sends (0, ⊥) to participant ⇔ invalid π_p
30: else
31: T inserts π_p in PubTab_p by k_p
32: T updates PubTab_r with k_r and v_r
33: T sends (1, res) to participant

The ideal-world execution (IdealSetup, IdealCertificate, IdealRegisterProve, IdealRegisterVerify, IdealPublishProve, IdealPublishVerify) satisfies the properties of Injectiveness, Unforgeability, Correctness, Chronicle, Tamper resistance, and Privacy-Preserving. These properties contribute to establishing three fundamental properties of address abstraction, Unique Identifier, Immutable Requests, and Privacy-Preserving, as defined in Section 11B.

- Injectiveness. Within this system’s context, injectiveness is secured through the generation of unique random values assigned as identifiers (id) and certificates (cert).
- Unforgeability. Unforgeability guarantees that only legitimate participants have the capability to register and publish. Consequently, only legitimate participants are eligible to receive proofs. In Algorithms 7 and 8, the trusted third party, denoted as T, generates unique proofs linked specifically to individual credentials. These proofs cannot be forged without accessing the original data securely stored in private tables accessible only to T.
- Correctness. The algorithms’ correctness is validated through the functionality of public and private tables that operate precisely as intended. Additionally, by definition, the S is incapable of forging the function output generated by T.
- Chronicle. The IdealPublishVerify algorithm supports this principle by requiring the message sequence order m to equal the nonce + 1, thereby enforcing the chronological order of msgs.
- Tamper Resistance. Tamper resistance, which assures that an adversary cannot alter the message, is achieved through data segregation into private and public tables. In addition, write permissions are exclusively reserved for T.
- Privacy-Preserving. This property ensures the user’s private information is not disclosed during the protocol execution.
Privacy is directly preserved by the private tables. The fact that all communication is intermediated by \( T \), performing only predefined actions, ensures that users’ private information is never exposed to other participants.

### B. Real-world Execution

Conversely, in the real-world execution, participants engage with zkAA through algorithms as detailed in Section VI-A.

**Definition A.2** (Real\(_{zkAA,A}\)). Real\(_{zkAA,A}\) designates the real-world implementation of zkAA with the probabilistic polynomial time (PPT) adversaries \( A \). The actions of honest participants align precisely with Algorithms \( 7 \), \( 2 \), \( 3 \), and \( 4 \).

**C. Security Proof**

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

**Definition A.3.** The real-world execution Real\(_{zkAA,A}\) is said to securely emulate the ideal-world execution Ideal\(_{\tau,S}\) if the following is satisfied:

\[
\text{for all PPT adversaries } A \text{ there exists a simulator } S \text{ s.t. for any PPT distinguisher } D,
\Pr[D(\text{Ideal}_{\tau,S}(\cdot)) = 1] \approx \Pr[D(\text{Real}_{zkAA,A}(\cdot)) = 1]
\]

**Theorem A.1.** Considering zk-SNARK, signature scheme, collision-resistant hash primitives, and the assumption of the smart contract functioning as a trustworthy computation engine and data storage, zkAA fulfills the security requirement stated in Definition A.3.

We prove Theorem A.1 by employing the hybrid game methodology, which demonstrates that the distribution of the simulator is computationally indistinguishable from the distribution of real-world experiments. In essence, we show that the adversary \( A \) in real-world execution can forge inputs/outputs and extract \textit{cert} no more effectively than the simulator \( S \), and all distributions in real-world execution are computationally identical to the ideal-world execution. This signifies that it is improbable for the adversary \( A \) to forge and extract in real-world execution if an attack in the ideal-world execution is deemed impossible.

\( \circ_0 \). This game corresponds to a real-world execution experiment.

\( \circ_1 \). In this game, we substitute the proofs \( \pi_P \) and \( \pi_F \) from honest participants with simulated proofs generated by the simulator. By applying Lemma A.1, we demonstrate that if the proof system exhibits computationally knowledge-soundness, then \( \circ_1 \approx \circ_0 \).

\( \circ_2 \). In this game, we run the knowledge extractor on the function output and abort if the extraction fails. By applying Lemma A.2, we demonstrate that the extractor fails with negligible probability, then \( \circ_2 \approx \circ_1 \).

\( \circ_3 \). In this game, we substitute all output of the hash function \( H \) with a unique random value. By applying Lemma A.3, we demonstrate that if the hash function is secure, then \( \circ_3 \approx \circ_2 \).

\( \circ_4 \). In this game, we substitute all signatures with a random unique value. By applying Lemma A.4, we demonstrate that if the signature scheme satisfies unforgeability, then \( \circ_4 \approx \circ_3 \).

\( \circ_5 \). In this game, we substitute all data storage with the trusted and verifiable storage provided by \( T \). By applying Lemma A.5, we demonstrate that if the probability of fork of the chain is negligible, then \( \circ_5 \approx \circ_4 \).

In conclusion, through the proposed hybrid games, we demonstrate that \( \circ_5 \approx \circ_0 \). This implies that our real-world execution, or the zkAA implementation, is identical to the ideal implementation of the AA scheme.

**Lemma A.1.** For all PPT adversaries \( A \), if a simulator on zk-SNARK exists, then the advantage of distinguishing \( \circ_0 \) and \( \circ_1 \) is negligible, as is the simulation failure rate.

**Proof.** As outlined in Section IV-A, a simulator zkp.SimProve exists in zk-SNARK and will fail with a negligible probability. This assures that the distinction between \( \circ_0 \) and \( \circ_1 \) is negligibly small.

**Lemma A.2.** For all PPT adversaries \( A \), if zk-SNARK is knowledge extractable, then the advantage of distinguishing \( \circ_1 \) and \( \circ_2 \) is negligible, as is the extraction failure rate.

**Proof.** As outlined in Section IV-A, an extractor \( \lambda_4 \) exists in zk-SNARK and will fail with a negligible probability. This assures that the distinction between \( \circ_1 \) and \( \circ_2 \) is negligibly small.

**Lemma A.3.** For all PPT adversaries \( A \), the advantage of distinguishing \( \circ_2 \) and \( \circ_3 \) is negligible.

**Proof.** As outlined in Section IV-C, we utilize a hash algorithm resistant to collision and preimage attacks. In the real zkAA implementation, we utilize SHA512 and SHA256 [45], which satisfy these requirements. This assures that the likelihood of an adversary \( A \) detecting the difference between the hash output and the substitution into a unique random value is negligible.

**Lemma A.4.** For all PPT adversaries \( A \), if the signature scheme has unforgeability, then the advantage of distinguishing \( \circ_3 \) and \( \circ_4 \) is negligible.

**Proof.** As outlined in Section IV-B, we employ a digital signature scheme that satisfies the unforgeability. In the real zkAA implementation, we utilize EdDSA [28], which meets this requirement. This assures that the likelihood that an adversary \( A \) can detect the difference between the signature and the substituted unique random value is negligible.

**Lemma A.5.** If the data storage is reliable and durable, then the advantage of distinguishing \( \circ_4 \) and \( \circ_5 \) is negligible.

**Proof.** As outlined in Section IV-B, we use the blockchain and the contracts deployed on it with robustness against attacks.
and reorganization. In the real zkAA implementation, we can use any contract-executable blockchains which satisfy these conditions. Although Ethereum [61] can be reorganized since the finality of blocks is given after several block confirmations, it is not high as much and can be practically negligible as many dApps do. On the other hand, we can use instant-finality blockchains, which mean that forks never occur, such as Cosmos [31] or Algorand [19], which gives us a zero fork-probability. Therefore the probability that anyone can recognize the difference between the smart contract and the storage by $T$ is negligible.