Beyond the Blockchain Address: Zero-Knowledge Address Abstraction

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

Jeong Hyuk Lee
Hanyang University
Seoul, Republic of Korea
ahoo791@hanyang.ac.kr

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

Jung Hyun Chun
Hanyang University
Seoul, Republic of Korea
jungdev@hanyang.ac.kr

Hyeonmyeong Cho
UNIST
Ulsan, Republic of Korea
heyitsme@unist.ac.kr

MinGi Kim
Sungkyunkwan University
Seoul, Republic of Korea
lupin1001@skku.edu

Hyun Ki Cho
Sogang University
Seoul, Republic of Korea
blockchain@sogang.ac.kr

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

***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 [65], [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.

	Easy Secret Management	Unified Identifier	Data Privacy
WebttCom [65]	✓[†]	✗	✓[†]
ZEBRA [47]	✗	✗	✓ZKP
ZKBID [56]	✗	✗	✓ZKP
BZDIMS [64]	✗	✗	✓ZKP
DA Luong [33]	✗[‡]	✗	✓ZKP
Holonym [30]	✗	✗	✓ZKP
Notebook [35]	✗	✗	✓ZKP
Web3Auth [32]	✗[‡]	✗	Not Related
Ramper [46]	✓KMS/HSM	✗	Not Related
POAP [25]	✗	✗	✗
SBT [57]	✗	✗	✗/✓ZKP
zkAA (Ours)	✓	✓	✓ZKP

[†] Private data are stored in a separate private data repository.
[‡] 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.

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.

	Blockchain Address/Transaction	Abstracted Address/Transaction
Secret	The secret key is denoted as sk.	The certificate is represented as $cert$, akin to a JSON Web Token.
Issuing Authority	sks are user-generated, with no centralized authority involved.	$cert$s can be issued by trusted web2 certificate institutes.
Pre-Reg	**No**. Any randomly generated 256-bits number can function as the sk without any specific usage limitations.	**Yes**. The $cert$ must be checked once and then registered, for verifying its issue by the authenticating institute.
Chain-Agnostic	**No**. This is contingent upon the underlying cryptography and the blockchain protocol.	**Yes**. Identifiers remain consistent across multiple chains.

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.

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 $\mathcal{R}$. Given that $(io, w) \in \mathcal{R}$, 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 $\mathcal{A}$ signifies a non-uniform polynomial time adversary. The output of this adversary is designated as $y \leftarrow \mathcal{A}(x)$, where x represents the input and y is the output. An extractor, denoted by $\mathcal{X}_{\mathcal{A}}$, can compute a corresponding witness whenever the adversary $\mathcal{A}$ generates a valid argument.

We use the notation $\mathcal{M}$ to signify the complete set of executable messages (msgs), and $\mathcal{P}$ is used to denote the entire set of potential passwords (pwds). 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.

The ZKP includes three operations: $\mathsf{zkp.Setup}$, $\mathsf{zkp.Prove}$, and $\mathsf{zkp.Verify}$.

- $crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R})$: This operation takes a relation $\mathcal{R}$ as an argument and produces a common reference string crs, which serves as a public parameter for proof generation and verification processes.
- $\pi \leftarrow \mathsf{zkp.Prove}(crs, io; w)$: This operation generates a proof π given the crs, input/output io, and witness w.
- $0/1 \leftarrow \mathsf{zkp.Verify}(crs, io, \pi)$: This operation verifies the proof π using the crs, input/output io, and trivially proof π 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 $\mathcal{R}$, every $(io, w) \in \mathcal{R}$ satisfies the following.*

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

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 $\mathcal{A}$ there exists a PPT extractor $\mathcal{X}_{\mathcal{A}}$ satisfies the following.*

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

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 $\mathcal{A}$ and $(io, w) \in \mathcal{R}$.*

$$\Pr\left[\begin{array}{l} crs \leftarrow \mathsf{zkp.Setup}(\mathcal{R}); \\ \pi \leftarrow \mathsf{zkp.Prove}(crs, io; w) : \\ \mathcal{A}(crs, \tau, \pi) = 1 \end{array}\right] = \Pr\left[\begin{array}{l} (crs, \tau) \leftarrow \mathsf{zkp.SimSetup}(\mathcal{R}); \\ \pi \leftarrow \mathsf{zkp.SimProve}(crs, \tau, io) : \\ \mathcal{A}(crs, \tau, \pi) = 1 \end{array}\right] \quad \text{(Perfect Zero-Knowledge)}$$

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 `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.

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 sig used in this work comprises three operations: sig.KeyGen, sig.Sign, and sig.Verify.

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

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

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

$$\Pr\begin{bmatrix}(pk, sk) \leftarrow \texttt{sig.KeyGen()};\\ (m, \sigma) \leftarrow \mathcal{A}(pk):\\ \texttt{sig.Verify}(pk, m, \sigma) = 1 \wedge m \notin \mathcal{M}\end{bmatrix} \approx 0 \quad \text{(Unforgeability)}$$

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 $\mathcal{H}$ 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, $\mathcal{R}_R$ and $\mathcal{R}_P$ are defined as the relations symbolizing registration and publication, respectively. The registration relation $\mathcal{R}_R$ 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 $\mathcal{H}$, i.e., $id \leftarrow \mathcal{H}(cert)$. Alternatively, the publication relation $\mathcal{R}_P$ 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* (Setup, Certificate, RegisterProve, RegisterVerify, PublishProve, PublishVerify).

- $(crs_R, crs_P) \leftarrow$ Setup$(\mathcal{R}_R, \mathcal{R}_P)$: This function generates public parameters crs_R and crs_P for registration and publication, respectively, from the relations.

TABLE III: Comparison of properties between the Ethereum address and our proposed abstracted address scheme.

		Blockchain Address/Transaction	Abstracted Address/Transaction
Unique Identifier	Injectiveness	The same address should have the same sk.	The same id should have the same $cert$.
	Unforgeability	Transaction signing cannot be achieved without knowing the secret key sk.	id registration and msg publication cannot occur without knowing the $cert$.
	Correctness	A user with a valid sk always generates valid signatures.	A user with a valid $cert$ always generates valid proofs π.
Immutable Request	Chronicle	Each transaction order is set by account nonce.	Each msg order is set by an increasing value m.
	Tamper Resistance	A signature cannot be reused and the signed transaction cannot be modified.	Proofs π cannot be reused and the corresponding msg cannot be modified.
Privacy-Preserving		sk is not leaked during transaction signing and publishing.	$cert$ is not leaked during the registration and publications.

- $cert \leftarrow$ CERTIFICATE(pwd): This function allows users to obtain a certificate from a trusted authority by providing their web2 secret pwd. In response, the authority issues a certificate $cert$ signed with its secret key.
- $\pi_R \leftarrow$ REGISTERPROVE($crs_R, id, pk; cert$): This function generates a proof π_R, validating that the abstracted identifier id is derived from the certificate $cert$ (confirmed by $id \leftarrow \mathcal{H}(cert)$), and that the certificate is issued by a trusted authority associated with the public key pk.
- $0/1 \leftarrow$ REGISTERVERIFY(crs_R, id, pk, π_R): 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 π_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, π_P): This function enables the execution of a requested m-th abstracted transaction msg when provided with a valid proof π_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 $(0, \emptyset)$ 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.

$$\text{for all } (cert^x, cert^y),\ cert^x \neq cert^y \implies id^x \neq id^y \qquad \text{(Injectiveness)}$$

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 $\mathcal{A}$ there exists a PPT extractor $\mathcal{X}_\mathcal{A}$,

$$\Pr\begin{bmatrix} (crs_R, \perp) \leftarrow \text{SETUP}(\mathcal{R}_R, \mathcal{R}_P); \\ ((id_R, pk, \pi_R); cert_R) \leftarrow (\mathcal{A}||\mathcal{X}_\mathcal{A})(crs_R); \\ s_R \leftarrow \text{REGISTERVERIFY}(crs_R, id_R, pk, \pi_R) : \\ s_R = 1 \land ((id_R, pk), cert_R) \notin \mathcal{R}_R \end{bmatrix} \approx 0,$$

and

$$\Pr\begin{bmatrix} (crs_R, crs_P) \leftarrow \text{SETUP}(\mathcal{R}_R, \mathcal{R}_P); \\ cert \leftarrow \text{CERTIFICATE}(pwd); id \leftarrow \mathcal{H}(cert); \\ \pi_R \leftarrow \text{REGISTERPROVE}(crs_R, id, pk; cert); \\ s_R \leftarrow \text{REGISTERVERIFY}(crs_R, id, pk, \pi_R); \\ ((id_P, msg, m, \pi_P); cert_P) \leftarrow (\mathcal{A}||\mathcal{X}_\mathcal{A})(crs_P); \\ (s_P, \perp) \leftarrow \text{PUBLISHVERIFY}(crs_P, id_P, msg, m, \pi_P) : \\ s_R \land s_P = 1 \land ((id_P, msg, m), cert_P) \notin \mathcal{R}_P \end{bmatrix} \approx 0 \quad \text{(Unforgeability)}$$

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.

$$\Pr\begin{bmatrix} (crs_R, crs_P) \leftarrow \text{SETUP}(\mathcal{R}_R, \mathcal{R}_P); \\ cert \leftarrow \text{CERTIFICATE}(pwd); id \leftarrow \mathcal{H}(cert); \\ \pi_R \leftarrow \text{REGISTERPROVE}(crs_R, id, pk; cert); \\ s_R \leftarrow \text{REGISTERVERIFY}(crs_R, id, pk, \pi_R); \\ \pi_P \leftarrow \text{PUBLISHPROVE}(crs_P, id, msg, m; cert); \\ (s_P, \perp) \leftarrow \text{PUBLISHVERIFY}(crs_P, id, msg, m, \pi_P) : \\ s_R \land s_P = 1 \\ \land ((id, pk), cert) \in \mathcal{R}_R \land ((id, msg, m), cert) \in \mathcal{R}_P \end{bmatrix} = 1 \quad \text{(Correctness)}$$

A user with a valid $cert$ can always generate a proof π_R for registration and π_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 [34]. 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* (SETUP, CERTIFICATE, REGISTERPROVE, REGISTERVERIFY, PUBLISHPROVE, PUBLISHVERIFY) *is immutable, if the system ensures the constraints* Chronicle *and* Tamper Resistance.

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

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 $\mathcal{A}$,

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

and

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

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 $cert$, from public exposure, thereby preserving its confidentiality.

Definition V.4 (Privacy-Preserving)**.** *System* (SETUP, CERTIFICATE, REGISTERPROVE, REGISTERVERIFY, PUBLISHPROVE, PUBLISHVERIFY) *can be classified as privacy-preserving, if the secret cert associated with the identifier id remains undisclosed both prior to and following the execution of any procedure.*

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

$$\Pr\left[\begin{array}{l}(crs_R, \perp) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(pwd); id \leftarrow \mathcal{H}(cert);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert) :\\ \mathcal{A}(crs_R, \tau_R, \pi_R) = 1\end{array}\right] =$$
$$\Pr\left[\begin{array}{l}(crs_R, \perp, \tau_R, \perp) \leftarrow \textsc{SimSetup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(pwd); id \leftarrow \mathcal{H}(cert);\\ \pi_R \leftarrow \textsc{SimRegisterProve}(crs_R, \tau_R, id, pk) :\\ \mathcal{A}(crs_R, \tau_R, \pi_R) = 1\end{array}\right],$$

and

$$\Pr\left[\begin{array}{l}(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(pwd); id \leftarrow \mathcal{H}(cert);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert) :\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ \pi_P \leftarrow \textsc{PublishProve}(crs_P, id, msg, m; cert);\\ s_R = 1 \wedge \mathcal{A}(crs_P, \tau_P, \pi_P) = 1\end{array}\right] =$$
$$\Pr\left[\begin{array}{l}(crs_R, crs_P, \tau_R, \tau_P) \leftarrow \textsc{SimSetup}(\mathcal{R}_R, \mathcal{R}_P);\\ cert \leftarrow \textsc{Certificate}(pwd); id \leftarrow \mathcal{H}(cert);\\ \pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert) :\\ s_R \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R);\\ \pi_P \leftarrow \textsc{SimPublishProve}(crs_P, \tau_P, id, msg, m);\\ s_R = 1 \wedge \mathcal{A}(crs_P, \tau_P, \pi_P) = 1\end{array}\right]$$
$$\text{(Privacy-Preserving)}$$

The proofs simulated by SIMREGISTERPROVE and SIMPUBLISHPROVE, which utilize the trapdoors τ_R and τ_P generated through SIMSETUP, are generated without a witness. The distributions of π_R and π_P remain indistinguishable whether the witness $cert$ 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.

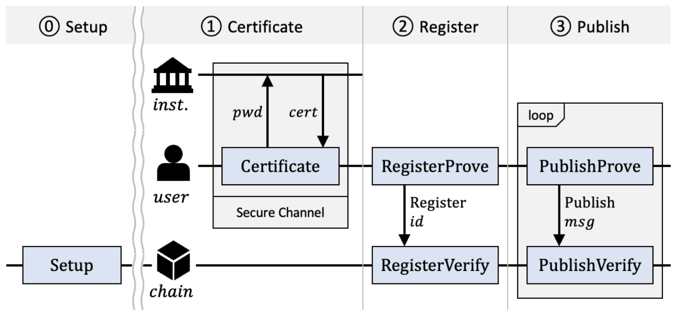

Fig. 1: The figure illustrates the process flow of zkAA. ⓪ Public parameters used in ZKP are generated. ① The user obtains a certificate, denoted as $cert$, from the institute through a secure channel. ② The user calculates the hash of the certificate as $id \leftarrow \mathcal{H}(cert)$, and proceeds to register it on the smart contract. ③ The user utilizes the registered id to publish abstracted transactions, denoted as msgs.

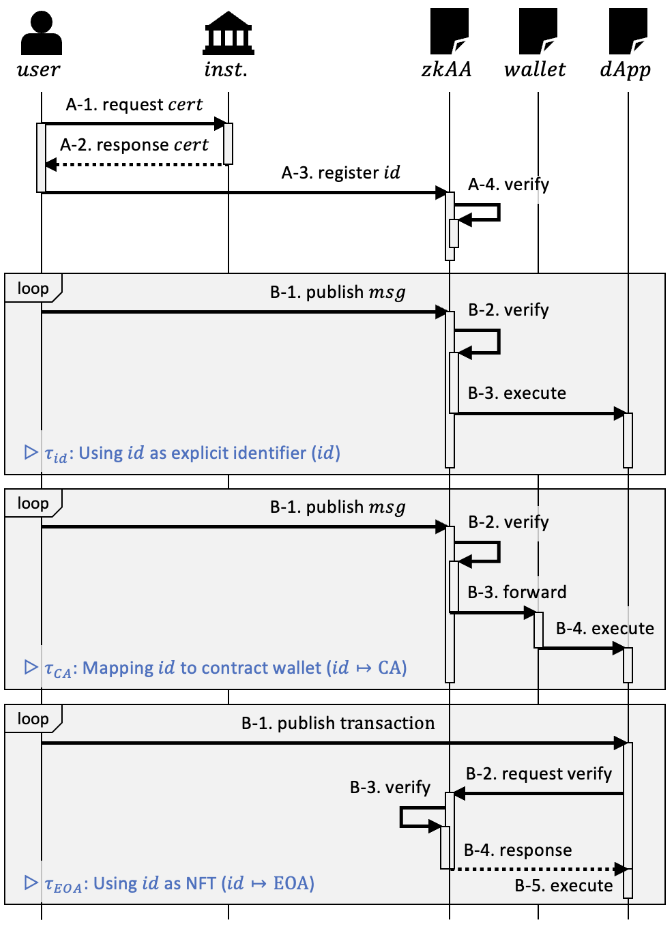

Fig. 2: Various explicit identities and their respective uses. The smart contracts zkAA, wallet, and dApp are deployed on the same blockchain.

Our implementation is based on Ethereum [61], thereby grounding the context of the message msg and the handling methodology of the identifier id within the Ethereum protocol's structure. Specifically, msg comprises four parameters: $msg \leftarrow (target, funcsig, calldata, value)$. Here, $funcsig$ denotes the function identifier that executes on the $target$ address with $calldata$ serving as arguments, and $value$ signifies the quantity of ETH involved. Though zkAA is presented in the context of Ethereum, it is designed with broader applicability in mind. It can be adapted to any blockchains that support the execution of smart contracts.

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.

- $(crs_R, crs_P) \leftarrow \textsc{Setup}(\mathcal{R}_R, \mathcal{R}_P)$: The setups generate the public parameters $crs_R \leftarrow \mathsf{zkp.Setup}(\mathcal{R}_R)$ and $crs_P \leftarrow \mathsf{zkp.Setup}(\mathcal{R}_P)$.
- $cert \leftarrow \textsc{Certificate}(pwd)$: 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 `sig.Sign` of EdDSA [28]. The user derives the identifier id from the $cert$ by invoking the hash function $\mathcal{H}$, SHA256 [45].
- $\pi_R \leftarrow \textsc{RegisterProve}(crs_R, id, pk; cert)$: A proof π_R is produced via $\pi_R \leftarrow \mathsf{zkp.Prove}(crs_R, (id, pk); cert)$ to validate two claims: Firstly, that the identifier id is computed as $id = \mathcal{H}(cert)$. Secondly, the $cert$ has been signed using the secret key correlated to the public key pk, thereby passing `sig.Verify` validation.
- $0/1 \leftarrow \textsc{RegisterVerify}(crs_R, id, pk, \pi_R)$: The user registers the identifier id and proof π_R on the blockchain, where the zkAA contract applies $\mathsf{zkp.Verify}(crs_R, (id, pk), \pi_R)$ 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 $nonce[id] == 0$, indicative of an unregistered state. Should the id have been registered previously, its corresponding $nonce[id]$ would satisfy the condition $nonce[id] \geq 1$, 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].
- $\pi_P \leftarrow \textsc{PublishProve}(crs_P, id, msg, m; cert)$: The user generates $\pi_P \leftarrow \mathsf{zkp.Prove}(crs_P, (id, \mathcal{H}(msg), m); cert)$, 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, $\mathcal{H}(msg)$ is employed rather than the raw msg to neglect concerns about msg's length.
- $(0/1, res) \leftarrow \textsc{PublishVerify}(crs_P, id, msg, m, \pi_P)$: In alignment with the PUBLISHPROVE function, the contract verifies π_P through $\mathsf{zkp.Verify}(crs_P, (id, \mathcal{H}(msg), m), \pi_P)$ 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 REGISTERVERIFY function. The sequence number m acts as a counter analogous to an account's nonce. Hence the value should exceed the current $nonce[id]$ by more than 1. If π_P, msg, and its sequence m pass validation, the intended message is executed, yielding a result labeled as res. After successful execution, $nonce[id]$ 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 `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 with 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].

Algorithm 1 ⓪ SETUP

```
    [zkAA Smart Contract]

 1: function SETUP($\mathcal{R}_R$, $\mathcal{R}_P$)
 2:     $crs_R \leftarrow$ zkp.Setup($\mathcal{R}_R$)
 3:     $crs_P \leftarrow$ zkp.Setup($\mathcal{R}_P$)
 4:     return $(crs_R, crs_P)$
```

Algorithm 2 ① CERTIFICATE

```
 1: institute has the pair of $(pk, sk) \leftarrow$ sig.KeyGen()
    ----------------------------------------
    [zkAA Client]

 2: function CERTIFICATE($pwd$)
 3:     $cert \leftarrow$ call ISSUE($pwd$)
 4:     return $cert$
    ----------------------------------------
    [Institute]

 5: procedure ISSUE($pwd$)
 6:     construct $cert_{header}$ with the type and algorithm
 7:     construct $cert_{payload}$ with claims by $pwd$
 8:     $cert_{sig} \leftarrow$ sig.Sign($sk$, $(cert_{header}||cert_{payload})$)
 9:     $cert = (cert_{header}||cert_{payload}||cert_{sig})$
10:     return $cert$
```

B. Algorithms of zkAA

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

0) Setup Phase: **`Smart Contract Side`**: The primary objective of the function SETUP is to generate two Common Reference Strings (CRSs) crs_R and 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 zkAA system. The first CRS, crs_R, is utilized in the registration circuit, playing a significant role in the user registration process. The second CRS, crs_P, is leveraged to facilitate the publication of abstracted transactions.

1) Certificate Phase: **`Client Side`**: The user makes a request for a $cert$ from the institute using their password (pwd). **`Institute Side`**: Then the institute, through the ISSUE procedure, constructs a $cert_{header}$, specifying the `type` and hashing `algorithm`, followed by the $cert_{payload}$ with the appropriate `claims` based on the provided pwd. Afterward, the institute generates a $cert_{sig}$ by signing the combination of encoded $cert_{header}$ and $cert_{payload}$ with its secret key (sk). The final $cert$ is then assembled by concatenating $cert_{header}$, $cert_{payload}$, and $cert_{sig}$, 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 pwd and $cert$.

2) Register Phase: **`Client Side`**: At this stage, the client generates a proof π_R that authenticates the user's identifier. Following this, the client initiates the REGISTER transaction, thereby commencing the registration process. **`Smart Contract Side`**: Upon receipt of the REGISTER transaction, the REGISTERVERIFY 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 τ. For further details regarding τ, refer to Section VI-D.

3) Publish Phase: **`Client Side`**: The client constructs a proof, noted as π_P, which affirms the legitimacy of the user's identifier associated with a specific message. Subsequently, the client initiates a PUBLISH transaction. **`Smart Contract Side`**: Based on the registration type τ, the PUBLISH transaction either calls the PUBLISHVERIFY function or executes the message via a normal transaction. The PUBLISHVERIFY 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.

Algorithm 3 ② REGISTER

```
1: pre-defined registration type $\tau \in \{\tau_{id}, \tau_{CA}, \tau_{EOA}\}$
2: $id \leftarrow \mathcal{H}(cert)$

[zkAA Client]

3: function REGISTERPROVE($crs_R$, $id$, $pk$; $cert$)
4:     $\pi_R \leftarrow$ zkp.prove$(crs_R, (id, pk); cert)$
5:     initiate REGISTER($crs_R$, $id$, $pk$, $\pi_R$)

6: transaction REGISTER($crs_R$, $id$, $pk$, $\pi_R$)
7:     call REGISTERVERIFY($crs_R$, $id$, $pk$, $\pi_R$)

[zkAA Smart Contract]

8: function REGISTERVERIFY($crs_R$, $id$, $pk$, $\pi_R$)
9:     assert($nonce[id] == 0$)
10:    assert(zkp.Verify$(crs_R, (id, pk), \pi_R)$)
11:    $nonce[id] \leftarrow 1$
12:    switch $\tau$ do
13:        case $\tau_{id}$ then return 1                      ▷ ($id$)
14:        case $\tau_{CA}$ then                               ▷ ($id \mapsto$ CA)
15:            create contract wallet controlled by $id$
16:            map $id$ into address of contract wallet CA
17:            return 1
18:        case $\tau_{EOA}$ then                              ▷ ($id \mapsto$ EOA)
19:            create NFT contains $id$
20:            give ownership to msg.sender
21:            return 1
```

Algorithm 4 ③ PUBLISH

```
1: pre-defined registration type $\tau \in \{\tau_{id}, \tau_{CA}, \tau_{EOA}\}$
2: $id \leftarrow \mathcal{H}(cert)$
3: $msg \leftarrow (target, funcsig, calldata, value)$

[zkAA Client]

4: function PUBLISHPROVE($crs_P$, $id$, $msg$, $m$; $cert$)
5:     $\pi_P \leftarrow$ zkp.Prove$(crs_P, (id, \mathcal{H}(msg), m); cert)$
6:     initiate PUBLISH($crs_P$, $id$, $msg$, $m$, $\pi_P$)

7: transaction PUBLISH($crs_P$, $id$, $msg$, $m$, $\pi_P$)
8:     if $\tau \in \{\tau_{id}, \tau_{CA}\}$ then              ▷ ($id$) or ($id \mapsto CA$)
9:         call PUBLISHVERIFY($crs_P$, $id$, $msg$, $m$, $\pi_P$)
10:    else                                                  ▷ ($id \mapsto$ EOA)
11:        execute $msg$ through normal transaction

[zkAA Smart Contract]

12: transaction PUBLISHVERIFY($crs_P$, $id$, $msg$, $m$, $\pi_P$)
13:     assert($nonce[id] \geq 1$)
14:     assert($m == nonce[id] + 1$)
15:     assert(zkp.Verify$(crs_P, (id, \mathcal{H}(msg), m), \pi_P)$)
16:     $nonce[id] \leftarrow m$
17:     switch $\tau$ do
18:         case $\tau_{id}$ then                               ▷ ($id$)
19:             $res \leftarrow$ execute $msg$ with identifier $id$
20:             return $(1, res)$
21:         case $\tau_{CA}$ then                               ▷ ($id \mapsto$ CA)
22:             $res \leftarrow$ call ExecuteMsg($msg$)
23:             return $(1, res)$

[Contract Wallet]

24: function EXECUTEMSG($msg$)
25:     assert(msg.sender == address(zkAA contract))
26:     $res \leftarrow$ execute $msg$
27:     return $res$
```

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 $\mathcal{H}$ 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 π_P, enhancing its resistance to tampering. The sequential number m 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., $\mathcal{H}(cert)$, 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 τ_{id}, which does not utilize the explicit identifier, explicit identifiers facilitate the usage of dApps (τ_{CA}) or enhance efficiency (τ_{EOA}). As outlined in Algorithm 3 and 4, functions REGISTERVERIFY, PUBLISHVERITY, and the transaction PUBLISH of zkAA encompass a conditional branch based on the explicit identifier type τ. This τ 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) τ_{id} (id): In this method, $id \leftarrow \mathcal{H}(cert)$ 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 π_R and π_P are generated solely based on the $cert$ without involving the blockchain address.

2) τ_{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 `ExecuteMsg`(msg) function, which can be called by the zkAA contract as shown in Algorithm 4, line 22. This approach mirrors Account Abstraction [7] as allowing users to utilize contract wallets containing codes.

3) τ_{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 τ_{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 ($14 * 10^{-}9$ ETH), with an ETH price of 1412.49 USD. The average gas price on Polygon was 51.6 gwei ($51.6 * 10^{-}9$ 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.

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

Methods	min	max	avg	USD (avg)	
				Ethereum	Polygon
Registration					
verifyTx	387456	387516	387494	$ 7.6626	$ 0.0182
Deploy	-	-	1410472	$ 27.8919	$ 0.0662
Publication					
verifyTx	239906	239966	239953	$ 4.7450	$ 0.0113
Deploy	-	-	921902	$ 18.2305	$ 0.0433

TABLE V: Time taken for generating proofs (sec)

Methods	min	max	avg (sd)	med
Registration				
witness	3.7511	3.9998	3.8381 (0.0474)	3.8300
proof	5.4607	11.8173	5.7365 (0.6323)	5.6348
Publication				
witness	1.7636	1.9181	0.8132 (0.0285)	1.8098
proof	2.5685	2.9915	2.6759 (0.0640)	2.6668

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 π_R, whereas the publication phase used the proof π_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[1] 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,

[1]https://github.com/lukepark327/zkAA

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.

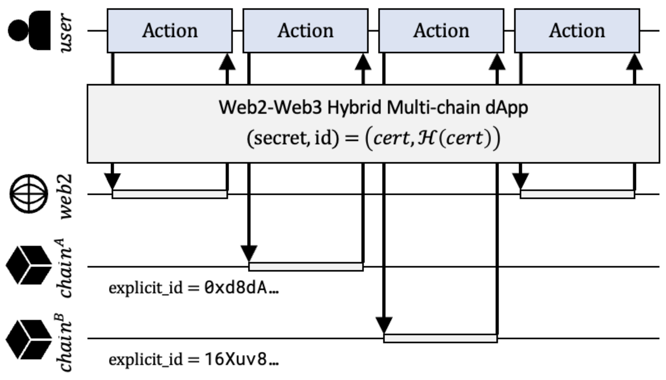

Fig. 3: The invariant pair (secret, identifier) facilitates seamless usability for web2-web3 hybrid and multi-chain dApps.

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 emergent web3 systems are has emerged, particularly in resource-demanding sectors such as Artificial Intelligence (AI) [42] 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 identities 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 incompatible address systems employed by various blockchains, such as Ethereum [61], Cosmos [31], and Polkadot [60], 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], [8] 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.

References

[1] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.

[2] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[3] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 315–333. Springer, 2013.

[4] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[5] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 677–706. Springer, 2020.

[6] Vitalik Buterin and Christian Reitwiessner. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-197, 2017.

[7] Vitalik Buterin, Yoav Weiss, Kristof Gazso, Namra Patel, Dror Tirosh, Shahaf Nacson, and Tjaden Hess. EIP-4337: Account Abstraction Using Alt Mempool [DRAFT]. https://eips.ethereum.org/EIPS/eip-4337, 2021.

[8] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2075–2092, 2019.

[9] Raj Chaganti, Rajendra V Boppana, Vinayakumar Ravi, Kashif Munir, Mubarak Almutairi, Furqan Rustam, Ernesto Lee, and Imran Ashraf. A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges. *IEEE Access*, 2022.

[10] Somnath Chakrabarti, Brandon Baker, and Mona Vij. Intel sgx enabled key manager service with openstack barbican. *arXiv preprint arXiv:1712.07694*, 2017.

[11] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[12] John R Douceur. The sybil attack. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*, pages 251–260. Springer, 2002.

[13] Jacob Eberhardt and Stefan Tai. Zokrates-scalable privacy-preserving off-chain computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091. IEEE, 2018.

[14] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. ERC-721: Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-161, 2018.

[15] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1304–1316, 2016.

[16] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019.

[17] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: practical snark aggregation. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 203–229. Springer, 2022.

[18] Christina Garman, Matthew Green, and Ian Miers. 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*, pages 81–98. Springer, 2017.

[19] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[20] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *STOC*, pages 291–304. ACM, 1985.

[21] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

[22] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Advances in Cryptology - CRYPTO 2017*, volume 10402, pages 581–612. Springer, 2017.

[23] Charles Hoskinson. Why we are building cardano. *IOHK (accessed 18 December 2017) https://whycardano. com*, 2017.

[24] IBM. IBM Cloud HSM. https://www.ibm.com/cloud/hardware-securitymodule, accessed May 26, 2023.

[25] POAP Inc. Proof of Attendance Protocol. https://poap.xyz/, accessed May 26, 2023.

[26] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[27] M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). https://www.rfc-editor.org/rfc/rfc7519, May 2015.

[28] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (eddsa). Technical report, 2017.

[29] Nikolaos Kapsoulis, Alexandros Psychas, Georgios Palaiokrassas, Achilleas Marinakis, Antonios Litke, and Theodora Varvarigou. Know your customer (kyc) implementation with smart contracts on a privacy-oriented decentralized architecture. *Future Internet*, 12(2):41, 2020.

[30] Nanak Nihal Khalsa, Caleb Tuttle, Lily Hansen-Gillis, Kushal Kahar, and Shady El Damaty. Holonym: A decentralized zero-knowledge smart identity bridge. 2022.

[31] Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, page 27, 2019.

[32] Torus Labs Pte Ltd. Web3Auth. https://web3auth.io/, accessed May 26, 2023.

[33] Duc Anh Luong and Jong Hwan Park. Privacy-preserving identity management system on blockchain using zk-snark. *IEEE Access*, 11:1840–1853, 2023.

[34] Jerry W Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.

[35] Nathaniel Masfen-Yan, Solal Afota, Dhruv Mangtani, and Sacha Arroues-Paykin. Notebook: A zero-knowledge identity infrastructure layer. accessed May 26, 2023.

[36] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[37] Michael Mirkin, Yan Ji, Jonathan Pang, Ariah Klages-Mundt, Ittay Eyal, and Ari Juels. Bdos: Blockchain denial-of-service. In *Proceedings of the 2020 ACM SIGSAC conference on Computer and Communications Security*, pages 601–619, 2020.

[38] Multichain. Multichain. https://multichain.org/, accessed May 26, 2023.

[39] Jose L Muñoz-Tapia, Marta Belles, Miguel Isabel, Albert Rubio, and Jordi Baylina. Circom: A robust and scalable language for building complex zero-knowledge circuits. 2022.

[40] Anca Nitulescu. zk-snarks: a gentle introduction, 2020.

[41] Office of Foreign Assets Control. FAQs 1095. https://ofac.treasury.gov/faqs/1095, 2022.

[42] Sanghyeon Park, Junmo Lee, and Soo-Mook Moon. A blockchain-based platform for reliable inference and training of large-scale models. *arXiv preprint arXiv:2305.04062*, 2023.

[43] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[44] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO'91: Proceedings*, pages 129–140. Springer, 2001.

[45] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in context*, pages 1–18, 2008.

[46] Inc Ramper. ramper. https://www.ramper.xyz/, accessed May 26, 2023.

[47] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. *Cryptology ePrint Archive*, 2022.

[48] Christian Reitwiessner. EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-196, 2017.

[49] Amazon Web Services. AWS CloudHSM. https://aws.amazon.com/cloudhsm, accessed May 26, 2023.

[50] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, page 704–737, Berlin, Heidelberg, 2020. Springer-Verlag.

[51] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[52] Nick Szabo et al. Smart contracts. http://www.fon.hum.5uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994.

[53] Polygon technology. Polygon. https://polygon.technology/, accessed May 26, 2023.

[54] U.S. DEPARTMENT OF THE TREASURY. Treasury Designates DPRK Weapons Representatives. https://home.treasury.gov/news/press-releases/jy1087, 2022.

[55] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[56] Taotao Wang, Shengli Zhang, and Soung Chang Liew. Linking souls to humans with zkbid: Accountable anonymous blockchain accounts for web 3.0 decentralized identity. *arXiv preprint arXiv:2301.02102*, 2023.

[57] E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized society: Finding web3's soul. *Available at SSRN 4105763*, 2022.

[58] Sam Wilson, Ansgar Dietrichs, Matt Garnett, and Micah Zoltu. EIP-3074: AUTH and AUTHCALL opcodes [DRAFT]. https://eips.ethereum.org/EIPS/eip-3074, 2020.

[59] Gavin Wood. State trie clearing (invariant-preserving alternative). https://eips.ethereum.org/EIPS/eip-161, 2016.

[60] Gavin Wood. Polkadot Whitepaper. https://polkadot.network/PolkaDotPaper.pdf, accessed May 26, 2023.

[61] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[62] Wormhole. Potral. https://www.portalbridge.com/, accessed May 26, 2023.

[63] Piyush Yadav and Raj Chandak. Transforming the know your customer (kyc) process using blockchain. In *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, pages 1–5. IEEE, 2019.

[64] Xiaohui Yang and Wenjie Li. A zero-knowledge-proof-based digital identity management scheme in blockchain. *Computers & Security*, 99:102050, 2020.

[65] Guangsheng Yu, Xu Wang, Qin Wang, Tingting Bi, Yifei Dong, Ren Ping Liu, Nektarios Georgalas, and Andrew Reeves. Towards web3 applications: Easing the access and transition. *arXiv preprint arXiv:2210.05903*, 2022.

Appendix

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

A. Ideal-world Execution

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

⓪ At the function IDEALSETUP in Algorithm 5, the trusted party $\mathcal{T}$ is responsible for initializing the foundational data structures of the protocol. Specifically, $\mathcal{T}$ creates three empty private tables, namely $\texttt{PrivTab}_c$, $\texttt{PrivTab}_r$, and $\texttt{PrivTab}_p$, alongside two empty public tables, $\texttt{PubTab}_r$ and $\texttt{PubTab}_p$. The tables are structured as key-value pairs. In addition, keys and values can be independently accessed using the $\texttt{Tab.keys}$ and $\texttt{Tab.values}$ functions, respectively. We assume that only $\mathcal{T}$ holds the authority to write values into the append-only tables. Access to the private tables is strictly limited to $\mathcal{T}$, even read operations are not permissible for any other party. Conversely, the public tables are accessible for reading to all participants.

① At the function IDEALCERTIFICATE in Algorithm 6, a participant send their password pwd to $\mathcal{T}$. Upon receipt, $\mathcal{T}$ verifies its presence in the $\texttt{PrivTab}_c$. If the password is already present, $\mathcal{T}$ retrieves and provides the associated unique identifier and certificate to the participant. However, if the password is unrecognized, $\mathcal{T}$ generates a unique identifier and certificate, stores this data in the $\texttt{PrivTab}_c$, and communicates this information to the participant.

② Algorithm 7 is bifurcated into two separate functions: IDEALREGISTERPROVE and IDEALREGISTERVERIFY. The IDEALREGISTERPROVE function enables participants to obtain a proof, designated as π_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, $\mathcal{T}$. Upon receipt of these components, $\mathcal{T}$ executes a validation process to authenticate the id and $cert$. Once these components are confirmed as legitimate, $\mathcal{T}$ then forwards π_R back to the participant. The IDEALREGISTERVERIFY function allows participants to register their id by sending id, pk, and π_R to $\mathcal{T}$. $\mathcal{T}$ verifies their validity and checks their previous registration status. If these components pass the verification process, the registration is successful and reflected in the $\texttt{PubTab}_r$.

③ Algorithm 8 encompasses two functions: IDEALPUBLISHPROVE and IDEALPUBLISHVERIFY. In the IDEALPUB-

Algorithm 5 ⓪ IDEALSETUP

```
1: function IdealSetup(·)
2:     $\mathcal{T}$ creates empty private table $\texttt{PrivTab}_c$
3:     $\mathcal{T}$ creates empty private table $\texttt{PrivTab}_r$
4:     $\mathcal{T}$ creates empty private table $\texttt{PrivTab}_p$
5:     $\mathcal{T}$ creates empty public table $\texttt{PubTab}_r$
6:     $\mathcal{T}$ creates empty public table $\texttt{PubTab}_p$
```

Algorithm 6 ① IDEALCERTIFICATE

```
1: function IdealCertificate($pwd$)
2:     Participant sends $pwd$ to $\mathcal{T}$
3:     if $pwd \in \texttt{PrivTab}_c\texttt{.keys}$ then
4:         $\mathcal{T}$ selects $(id, cert)$ from $\texttt{PrivTab}_c$ by $pwd$
5:     else
6:         $\mathcal{T}$ generates a unique random value $id$
7:         $\mathcal{T}$ generates another unique random value $cert$
8:         $\mathcal{T}$ inserts $(id, cert)$ in $\texttt{PrivTab}_c$ by $pwd$
9:     $\mathcal{T}$ sends $(id, cert)$ to participant
```

Algorithm 7 ② IDEALREGISTER

```
function IDEALREGISTERPROVE($id, pk, cert$)
    Participant sends $(id, pk, cert)$ to $\mathcal{T}$
    if $(id, cert) \notin \texttt{PrivTab}_c.\texttt{value}$ then
        abort                                    ▷ invalid $id$ and/or $cert$
    if $(id, pk, cert) \in \texttt{PrivTab}_r.\texttt{keys}$ then
        $\mathcal{T}$ selects $\pi_R$ from $\texttt{PrivTab}_r$ by $(id, pk, cert)$
    else
        $\mathcal{T}$ generates a unique random value $\pi_R$
        $\mathcal{T}$ inserts $\pi_R$ in $\texttt{PrivTab}_r$ by $(id, pk, cert)$
    $\mathcal{T}$ sends $\pi_R$ to participant

function IDEALREGISTERVERIFY($id, pk, \pi_R$)
    Participant sends $(id, pk, \pi_R)$ to $\mathcal{T}$
    if $(id, pk) \in \texttt{PubTab}_r.\texttt{keys}$ then
        $\mathcal{T}$ sends 0 to participant          ▷ already registered
    else if $\pi_R \notin \texttt{PrivTab}_r.\texttt{value}$ then
        $\mathcal{T}$ sends 0 to participant               ▷ invalid $\pi_R$
    else
        $\mathcal{T}$ inserts $(\pi_R, 1)$ in $\texttt{PubTab}_r$ by $(id, pk)$
        $\mathcal{T}$ sends 1 to participant
```

Algorithm 8 ③ IDEALPUBLISH

```
function IDEALPUBLISHPROVE($id, msg, m, cert$)
    Participant sends $(id, msg, m, cert)$ to $\mathcal{T}$
    if $(id, cert) \notin \texttt{PrivTab}_c.\texttt{value}$ then
        abort                                    ▷ invalid $id$ and/or $cert$
    $k \leftarrow (id, msg, m, cert)$
    if $k \in \texttt{PrivTab}_p.\texttt{keys}$ then
        $\mathcal{T}$ selects $\pi_P$ from $\texttt{PrivTab}_p$ by $k$
    else
        $\mathcal{T}$ generates a unique random value $\pi_P$
        $\mathcal{T}$ inserts $\pi_P$ in $\texttt{PrivTab}_p$ by $k$
    $\mathcal{T}$ sends $\pi_P$ to participant

function IDEALPUBLISHVERIFY($id, msg, m, \pi_P$)
    Participant sends $(id, msg, m, \pi_P)$ to $\mathcal{T}$
    $k_r \leftarrow (id, pk)$
    if $k_r \notin \texttt{PubTab}_r.\texttt{keys}$ then
        $\mathcal{T}$ sends $(0, \perp)$ to participant    ▷ unregistered $id$
    else
        $\mathcal{T}$ selects $(\pi_R, nonce)$ from $\texttt{PubTab}_r$ by $k_r$
        if $nonce < 1$ then
            $\mathcal{T}$ sends $(0, \perp)$ to participant  ▷ invalid $nonce$
        else if $m \neq nonce + 1$ then
            $\mathcal{T}$ sends $(0, \perp)$ to participant      ▷ invalid $m$
        else
            $k_p \leftarrow (id, msg, m)$
            $v_r \leftarrow (\pi_R, nonce + 1)$
            $res \leftarrow$ execute $msg$ with identifier $id$
            if $k_p \in \texttt{PubTab}_p.\texttt{keys}$ then
                $\mathcal{T}$ sends $(0, \perp)$ to participant  ▷ passed $m$
            else if $\pi_P \notin \texttt{PrivTab}_p.\texttt{value}$ then
                $\mathcal{T}$ sends $(0, \perp)$ to participant ▷ invalid $\pi_P$
            else
                $\mathcal{T}$ inserts $\pi_P$ in $\texttt{PubTab}_p$ by $k_p$
                $\mathcal{T}$ updates $\texttt{PubTab}_r$ with $k_r$ and $v_r$
                $\mathcal{T}$ sends $(1, res)$ to participant
```

LISHPROVE function, $\mathcal{T}$ generates a proof π_P for publication of a m-th message msg then sends it to the participant, provided the id and $cert$ is valid. The IDEALPUBLISHVERIFY function enables participants to confirm the success of their publication, which includes their id, msg, message sequence number m, and π_P. Moreover, the participant must be registered prior to the publication. The function then executes msg and returns the result res to the participant, along with the publication status. If these components pass the verification process, the publication is successful and reflected in the tables, $\texttt{PubTab}_r$ and $\texttt{PubTab}_p$.

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 V-B.

- **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 $\mathcal{T}$, 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 $\mathcal{T}$.
- **Correctness.** The algorithms' correctness is validated through the functionality of public and private tables that operate precisely as intended. Additionally, by definition, the $\mathcal{S}$ is incapable of forging the function output generated by $\mathcal{T}$.
- **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 $\mathcal{T}$.
- **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 $\mathcal{T}$, 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 ($\texttt{Real}_{\text{zkAA},\mathcal{A}}$). *$\texttt{Real}_{zkAA,\mathcal{A}}$ designates the real-world implementation of zkAA with the probabilistic polynomial time (PPT) adversaries $\mathcal{A}$. The actions of honest participants align precisely with Algorithms 1, 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 $\texttt{Real}_{zkAA,\mathcal{A}}$ is said to securely emulate the ideal-world execution $\texttt{Ideal}_{T,\mathcal{S}}$ if the following is satisfied:*

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

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

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 $\mathcal{A}$ in real-world execution can forge inputs/outputs and extract $cert$ no more effectively than the simulator $\mathcal{S}$, and all distributions in real-world execution are computationally identical to the ideal-world execution. This signifies that it is improbable for the adversary $\mathcal{A}$ to forge and extract in real-world execution if an attack in the ideal-world execution is deemed impossible.

- $\Game_0$. This game corresponds to a real-world execution experiment.
- $\Game_1$. In this game, we substitute the proofs π_R and π_P 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 $\Game_1 \approx \Game_0$.
- $\Game_2$. 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 $\Game_2 \approx \Game_1$.
- $\Game_3$. In this game, we substitute all output of the hash function $\mathcal{H}$ with a unique random value. By applying Lemma A.3, we demonstrate that if the hash function is secure, then $\Game_3 \approx \Game_2$.
- $\Game_4$. 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 $\Game_4 \approx \Game_3$.
- $\Game_5$. In this game, we substitute all data storage with the trusted and verifiable storage provided by $\mathcal{T}$. By applying Lemma A.5, we demonstrate that if the probability of fork of the chain is negligible, then $\Game_5 \approx \Game_4$.

In conclusion, through the proposed hybrid games, we demonstrate that $\Game_5 \approx \Game_0$. 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 $\mathcal{A}$, if a simulator on zk-SNARK exists, then the advantage of distinguishing $\Game_0$ and $\Game_1$ 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 $\Game_0$ and $\Game_1$ is negligibly small. $\square$

Lemma A.2. *For all PPT adversaries $\mathcal{A}$, if zk-SNARK is knowledge extractable, then the advantage of distinguishing $\Game_1$ and $\Game_2$ is negligible, as is the extraction failure rate.*

Proof. As outlined in Section IV-A, an extractor $\mathcal{X}_{\mathcal{A}}$ exists in zk-SNARK and will fail with a negligible probability. This assures that the distinction between $\Game_1$ and $\Game_2$ is negligibly small. $\square$

Lemma A.3. *For all PPT adversaries $\mathcal{A}$, the advantage of distinguishing $\Game_2$ and $\Game_3$ 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 $\mathcal{A}$ detecting the difference between the hash output and the substitution into a unique random value is negligible. $\square$

Lemma A.4. *For all PPT adversaries $\mathcal{A}$, if the signature scheme has unforgeability, then the advantage of distinguishing $\Game_3$ and $\Game_4$ 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 $\mathcal{A}$ can detect the difference between the signature and the substituted unique random value is negligible. $\square$

Lemma A.5. *If the data storage is reliable and durable, then the advantage of distinguishing $\Game_4$ and $\Game_5$ 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 $\mathcal{T}$ is negligible. $\square$