Hades: Practical Decentralized Identity with Full Accountability and Fine-grained Sybil-resistance

Ke Wang
wangk@pku.edu.cn
School of Computer Science, Peking University
Beijing, China

Jianbo Gao[*]
gaojianbo@pku.edu.cn
School of Computer Science, Peking University
Beijing, China
Peking University Chongqing Research Institute of Big Data
Chongqing, China

Qiao Wang
wangtsiao@stu.pku.edu.cn
School of Computer Science, Peking University
Beijing, China

Jiashuo Zhang
zhangjiashuo@pku.edu.cn
School of Computer Science, Peking University
Beijing, China

Yue Li
liyue_cs@pku.edu.cn
School of Computer Science, Peking University
Beijing, China

Zhi Guan[*]
guan@pku.edu.cn
National Engineering Research Center For Software Engineering, Peking University
Beijing, China

Zhong Chen
zhongchen@pku.edu.cn
School of Computer Science, Peking University
Beijing, China

ABSTRACT

Decentralized identity (DID), the idea of giving users complete control over their identity-related data, is being used to solve the privacy tension in the identity management of decentralized applications (Dapps). While existing approaches do an excellent job of solving the privacy tension, they have not adequately addressed the *accountability* and *Sybil-resistance* issues. Moreover, these approaches have a considerable gas overhead, making them impractical for Dapps.

We presented Hades, a novel practical DID system supporting full accountability and fine-grained Sybil-resistance while providing strong privacy properties. Hades supports three aspects of accountability, i.e., *auditability*, *traceability*, and *revocation*. Hades is the first DID system that supports accountability in all these three aspects. Hades is also the first DID system that supports *fine-grained Sybil-resistance*, enabling Dapps to customize personalized Sybil resistance strategies based on users' identity attributes. Hades can run efficiently on the Ethereum Virtual Machine (EVM). We implemented and evaluated Hades. The benchmarks showed that Hades has the lowest gas cost incurred on EVM as far as we know. Also, we presented a case study on attribute-associated fair NFT distribution ("airdrops") where all previous works failed, whereas we gave a solution leveraging Hades.

[*]Corresponding authors

ACSAC '23, December 04–08, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0886-2/23/12.
https://doi.org/10.1145/3627106.3627110

CCS CONCEPTS

• **Security and privacy** → **Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**; *Access control*.

KEYWORDS

Decentralized identity, Privacy, Sybil-resistance, Accountability, Dapp

ACM Reference Format:
Ke Wang, Jianbo Gao, Qiao Wang, Jiashuo Zhang, Yue Li, Zhi Guan, and Zhong Chen. 2023. Hades: Practical Decentralized Identity with Full Accountability and Fine-grained Sybil-resistance . In *Annual Computer Security Applications Conference (ACSAC '23), December 04–08, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3627106.3627110

1 INTRODUCTION

Blockchain technology has been the cornerstone of decentralized applications (Dapps). Benefiting from the openness and permissionless nature of Blockchain, Dapps have seen significant developments in fields such as Decentralized Finance (DeFi), Gaming, and Decentralized Social. However, due to the permissionless nature of blockchain, Dapps face challenges in implementing access control mechanisms based on users' authentic identities (e.g., the gender of the user). This restricts their utility in wider fields, especially those requiring real identity verification. Even in established fields, this limitation exposes Dapps to potential legal compliance risks. For instance, the Financial Action Task Force (FATF) stated that Dapps, including NFT marketplaces, DeFi protocols, and stablecoins, must adhere to Know Your Customer (KYC) compliance procedures [16]. Furthermore, due to the inability to determine whether on-chain addresses are associated with the same real-world identity, attackers can acquire disproportionate benefits by generating a multitude of addresses (Sybil attack). A naive solution to enable identity-based access control is to attach the user's wallet an on-chain credential issued by a Certificate Authority (CA) that certifies the user's identity. However, this solution suffers from a limitation: the openness of blockchain leads to users being exposed to a significant risk of privacy leakage. It is challenging to implement identity-based access control on the blockchain while preserving privacy.

To address this challenge, numerous solutions have been recently proposed, with the most promising ones being decentralized identities (DIDs) [22, 23, 28, 30] and anonymous credentials (ACs) [35, 40, 46]. The idea behind these works is to allow the user to unlinkably show that they possess a credential authenticating her/his identity without disclosing the original credential. While these protocols provide strong privacy-preserving properties, they encounter three notable limitations:

L1: Insufficiency of Supporting Accountability. Identity management ought to be *auditable*, *traceable* and *revocable*, as it is critical to identify individuals responsible for malicious behaviors, retrieve all activities of a suspect for investigations (such as anti-money laundering), and revoke certificates that are lost, stolen, or associated with malicious behaviors. *However, the privacy-preserving property makes supporting* traceability*,* auditability*, and* revocation *very challenging.* Though some of the previous works support *revocation* and *auditability* [35, 46], with some supporting *revocation* and *traceability* [28], none of them manage to fully support all accountability features. Furthermore, although CanDID [28] does support *traceability*, it comes at the compromise of privacy, as the linkability of pseudonyms is not invisible to the committee.

L2: Inability to resist Sybil attacks. Sybil-resistance is extremely useful in certain scenarios, such as anonymous voting, fair currency distribution ("airdrops"), etc. In the case of the Arbitrum airdrop, approximately 253 million Arbitrum (300 million USD) or 21.8% of the total tokens were taken by Sybil addresses. [45]. *Implementing Sybil resistance while ensuring unlinkability is challenging because the application cannot determine whether the access comes from the same user.* Few previous works support traceability. CanDID [28] is the state-of-the-art DID system to support Sybil-resistance, but at the cost of compromising unlinkability, where the committee knows the linkages among pseudonyms and which applications users have accessed. Moreover, the Sybil-resistance process requires the participation of the committee, which is quite burdensome. It only supports the 'unique per-user access' strategy, which is insufficient for increasingly complex application scenarios.

L3: Inefficiencies of running on the blockchain. Managing identity through smart contracts is desirable: the smart contracts of Dapps could directly call the identity management system, and the on-chain state could be included for identity management. *However, to ensure privacy, most previous works rely on complex cryptographic computations, resulting in enormous on-chain overhead.* Previous solutions could only revoke credentials but were unable to revoke the pseudonyms associated with those credentials. Since it's impossible to determine whether the associated credential has been revoked, the pseudonym must be re-verified with each use. Zebra [35] provides the first practical on-chain identity management scheme. To reduce on-chain overhead, it employs a validity period mechanism to reduce the times the pseudonym is re-verified. However, without a mechanism to identify pseudonyms associated with revoked credentials, a credential revocation compels revalidations of all pseudonyms.

To address the three limitations, we propose Hades, a practical decentralized identity system with full accountability support (i.e., *traceability*, *auditability*, *revocation*), lightweight fine-grained Sybil resistance support, and the low on-chain overhead, while providing strong privacy properties. Hades' identity services can be directly accessed by Dapp's on-chain contracts, providing Dapps with identity-based access control.

To reduce the on-chain overhead, we build Hades' privacy-preserving properties on top of zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs), because zk-SNARKs can be verified efficiently on Ethereum Virtual Machine (EVM) [44] and has sub-linear verification complexity. We focus on the overhead in the EVM because it hosts the most significant number of Dapps worldwide. Additionally, in Hades, pseudonyms only need to be verified once during their validity period. This is because Hades can revoke both credentials and their corresponding pseudonyms. If a pseudonym has not been revoked, its associated credential must also remain unrevoked, thus there's no necessity to re-verify the pseudonym.

Hades is decentralized in the sense that all its identity management operations rely on a committee, which is composed of multiple distinct entities. We employ threshold public-key encryption (TPKE) [5] to achieve decentralized accountability. The advantage lies in that the committee's participation is required only during the accountability process. This reduces the interactions and makes Hades more suitable for blockchain. Another challenge in accountability is how to trace unlinkable pseudonyms, that is, identify all pseudonyms belonging to a certain user. Our solution is to assign each pseudonym a unique trapdoor-linkable identifier, created using a user-specific secret trapdoor. With the knowledge of the secret trapdoor, all relevant pseudonyms can be traced by their identifiers. In Hades, as all pseudonyms of a user can be traced, the revocation of a credential does not instigate the revalidation of all pseudonyms.

The challenge of Sybil-resistance lies in determining whether a user has exceeded the number of access allowed for her/his identity, especially when users can hide their previous accesses and identity information under unlinkable pseudonyms. The idea to address this challenge is to attach each access a unique context-based access token, which a user can generate in limited numbers for a given context. Dapps can identify Sybil's access by detecting duplicate access tokens. We achieve lightweight Sybil-resistance without compromising privacy, since access tokens are secretly generated locally by the user and are unlinkable. Additionally, fine-grained Sybil-resistance can be achieved by customizing the access token generation scheme, where fine-grained means that Dapps can customize personalized Sybil-resistance strategies according to the user's identity attributes. To the best of our knowledge, Hades is the first identity system to support fine-grained Sybil-resistance without compromising unlinkability.

We implemented Hades with Solidity and Rust. Our performance evaluation demonstrates the practicality of Hades: the most gas cost scheme, pseudonym registration, costs 339K gas on EVM, three times the cost of a decentralized token swap (one of the most commonly occurring transaction types on the blockchain), and the proof generation time is 1.9s with a single thread. This is practical because a pseudonym only needs to be verified once within its validity period. Compared to Zebra [35], the state-of-the-art anonymous credentials scheme, our scheme has less gas consumption, and the proof generation scheme is three times faster.
To summarize, our main technical contributions are:

- We propose a practical DID scheme that supports full accountability and lightweight fine-grained Sybil-resistance for the first time while providing strong privacy properties. The scheme can work efficiently on EVM with the lowest on-chain overhead.
- We present an EVM-compatible identity management solution for Dapps. We implemented Hades using solidity and Rust and benchmarked it. The benchmarks showed that Hades has the lowest gas cost incurred on EVM as far as we know.
- We describe a case study on fair NFT distribution ("airdrops"), explain why all previous DID solutions fail in this context, and show how utilizing Hades to achieve differentiated Sybil-resistance and improved security without compromising privacy.

2 OVERVIEW

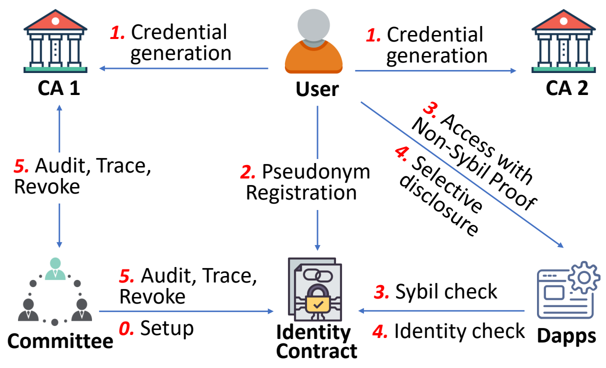

Figure 1: The architecture of Hades.

2.1 System Model

Hades is a decentralized identity system implemented as a series of off-chain programs and on-chain contracts. The architecture of Hades is illustrated in Fig. 1. There are five roles in the Hades system, namely:

a) Committee. The committee is a union of several distinct entities responsible for system management and identity accountability. The committee works in a threshold manner, i.e., all operations require the cooperation of at least $t + 1$ entities. We assume that the committee is *honest-majority*, i.e., up to t of the n committee members may corrupt, for $t < n/2$.

b) Certificate Authority (CA). CA is an authorized organization that authenticates and stores users' identity attributes, and assists the committee in identity accountability. Hades supports multiple CAs, allowing identity attributes to be authenticated by various CAs. each of which may authenticate different identity attributes. Distinct CAs might authenticate different sets of identity attributes. Supporting multiple CAs helps eliminate single-point failures, thereby improving system liveliness. We assume that the CA is *semi-honest*, i.e., honest but curious.

c) Identity Contract. The *Identity Contract* is a system contract that verifies, stores, and manages users' pseudonyms. Specifically, users can invoke *Identity Contract* to register their pseudonyms and bind them to certain identity attributes. Besides, *Identity Contract* also provides interfaces for identity checking, Sybil checking, and accountability.

d) Dapp. The Dapp is a series of smart contracts deployed on the blockchain. It can invoke the *Identity Contract* to perform identity checking and Sybil checking. We assume that the Dapp is malicious.

e) Users. Users interact with DApps using pseudonyms (*e.g.*, the address of the on-chain account), with proofs proving that their access complies with Dapps' access controls, such as identity proof and non-Sybil proof. We assume that users are malicious and can act arbitrarily to gain an advantage.

The overall workflow is as follows: 1) Users apply for credentials from CAs in Hades' CA list. 2) With the obtained credentials, users can register pseudonyms in the identity contract. 3) If needed, applications can create Sybil-resistance instances with customized access control policies. 4) Users access Dapps with non-Sybil proofs or proofs of identity attribute assertions. 5) If a user or pseudonym is found malicious, the committee will reveal or revoke the identity-related information and all associated pseudonyms.

2.2 System Goals

The system goals of Hades are:

- **Privacy-preserving.** When a user interacts with the Hades system, no identity-related or issuer information will be revealed except for the information the user chooses to expose. (selective disclosure) Pseudonyms should be owner-indistinguishable, except for those whose linkability is revealed by the owner. (selective linkability) Furthermore, the linkability revealing should not disclose any additional information.
- **Fine-grained Sybil-resistance.** Sybil-resistance should be sufficiently lightweight that only require the participation of users and Dapps. Dapps should be able to employ custom Sybil-resistance strategies according to users' identity attributes. Furthermore, no identity-related or linkability information should be revealed during the Sybil-resistance process.
- **Decentralized full accountability.** Identities can be audited, traced, and revoked if and only if more than a certain number of committee members agree.
- **Economy.** Hades should be able to run efficiently on the blockchain. The communication overhead between users and the Hades system should be sufficiently small, implying that the data submitted by users to the system must be compact, and the communication should be non-interactive. The computational overhead for users should also be adequately minimal, allowing

users to complete an interaction with the Hades within a few seconds using mobile devices.

To achieve the *privacy-preserving* goal, we first employ zero-knowledge proofs for credential verification. This process does not expose any information regarding the credential, including its content and issuer. Furthermore, we allow users to generate unlinkable pseudonyms using a single credential to achieve unlinkability. Each pseudonym is bound to a commitment to the user's identity attributes, and the user can employ zero-knowledge proofs to prove that their identity attributes satisfy specific assertions.

To achieve the *lightweight Sybil-resistance* goal, we designed an n−time access token generation scheme, where for a single Sybil-resistance instance initiated by a Dapp, users can generate up to n distinct unlinkable access tokens with the same credential or identity attribute. If a user provides duplicate identifiers, the Dapp can ascertain that the user is a Sybil. Moreover, the access tokens can be utilized to achieve *selective linkability*, as users can link their pseudonyms by providing the same access token.

We utilize threshold public-key encryption (TPKE) to achieve the *decentralized accountability* goal. To enable *auditability*, users are required to provide an audit string corresponding to each pseudonym they generated, which is a TPKE encryption of the identity-related information. A threshold number of committee members can collaborate to decrypt the audit string to reveal the identity-related information. To enable *traceability*, users are required to provide a trapdoor-linkable trace string for each pseudonym they registered, which can be linked with the knowledge of the trapdoor. Also, a TPKE encryption of the trapdoor is submitted to the CA during the credential application process, and the committee can decrypt it to identify all pseudonyms belonging to the user. The credentials and pseudonyms can be revoked by adding them to the revocation list of the *Identity Contract*.

To reduce the EVM gas cost, we employ zkSNARKs for verifying the correctness of credentials, identity assertions, and no-Sybil assertions, as zkSNARK proofs can be efficiently verified on the EVM. To further minimize the gas cost, we allow verified pseudonyms to remain valid throughout their validity period, eliminating the need for validity verification each time they are used. We also optimized the Hades protocol to be more zero-knowledge-proof friendly, ensuring that all zero-knowledge proofs in Hades cost less than 30K R1CS constraints. This allows each proof in Hades to be generated within 2s.

2.3 Security Model

Adversarial model: The committee is *honest-majority*, that is, the adversary may statically and actively corrupt up to t of the n committee members, for $t < n/2$. CAs are *semi-honest*, cannot be corrupted by the adversary. In addition, the adversary can corrupt any number of external entities, such as users and applications.

Security Properties: The Hades scheme must satisfy the following security properties.

- **Unforgeability:** Under the semi-honest CA assumption, it must be infeasible for a malicious user to forge honest users' pseudonyms or otherwise impersonate them, or to forge linkability between pseudonyms.
- **Privacy:** Under the honest-majority committee assumption, an adversary can learn about an honest user no more than the information the user explicitly presents.
- **Sybil-resistance:** An adversary cannot access an application using a larger number of pseudonyms than the number assigned to its credentials or identity attributes.

3 THE HADES CONSTRUCTION

We first introduce the notations of the cryptographic schemes used in Hades in Section 3.1. Then we introduce the Hades system in detail in five parts and present the security proof in Appendix A.

3.1 Notations

Zero-knowledge proofs: We use the notation introduced by Camenisch et al. [13] to present non-interactive zero-knowledge proofs used in Hades:

$$\texttt{NIZK}\ \{(x, y, ...) : \text{statements about } x, y, ...\}$$

which denotes to prove in zero-knowledge that the secret values $(x, y, ...)$ and all other public values satisfy the statements after the colon.

Signature schemes: A digital signature scheme is a triple $(\texttt{Gen}, \texttt{Sign}, \texttt{SigVerify})$ of algorithms running in expected polynomial time [27].

- $(pk, vk) \leftarrow \texttt{Gen}(1^\lambda)$: takes a security parameter λ as input and outputs a public key pk and a secret key vk.
- $\sigma \leftarrow \texttt{Sign}(sk, m)$: takes a message m and a secret key sk as input and outputs a signature σ.
- $\{0, 1\} \leftarrow \texttt{SigVerify}(pk, m, \sigma)$: takes public key pk, message m and signature σ as input and outputs 1 if σ is a valid signature for m w.r.t pk, otherwise outputs 0.

Merkle trees: The Merkle tree allows a prover to commit to an arbitrary finite set S of values, and for any value x, reveal with a proof whether $x \in S$ or $x \notin S$ [29]. We present the Merkle tree operations used in the Hades system using the following notations:

- $\gamma \leftarrow \texttt{TreeRoot}(\tau)$: return the root γ of tree τ;
- $\texttt{TreeAdd}(v, \tau)$: add a value v in tree τ;
- $\texttt{TreeRm}(v, \tau)$: remove value v from tree τ;
- $\pi \leftarrow \texttt{InProof}(v, \tau)$: returns a membership proof of τ for v;
- $\pi \leftarrow \texttt{NotInProof}(v, \tau)$: returns a non-membership proof of τ for v;
- $\{0, 1\} \leftarrow \texttt{InVerify}(\pi, v, \gamma)$: outputs 1 if π is a valid proof that v is in τ s.t. $\texttt{TreeRoot}(\tau) \equiv \gamma$, otherwise outputs 0;
- $\{0, 1\} \leftarrow \texttt{NotInVerify}(\pi, v, \gamma)$: outputs 1 if π is a valid proof that v is not in τ s.t. $\texttt{TreeRoot}(\tau) \equiv \gamma$, otherwise outputs 0;

Threshold public-key encryption: Threshold public-key encryption (TPKE) allows a set of users to decrypt a ciphertext if a predetermined threshold of authorized users cooperates [33]. We use a threshold version of EC-ElGamal encryption [34] and present the TPKE operations using the following notations:

- $\{PK, (sk_1, ..., sk_n)\} \leftarrow \texttt{Setup}(1^\lambda, n, t)$: takes threshold t, set size n and security parameter λ, outputs the public key PK and secret keys $(sk_1, ..., sk_n)$;
- $c \leftarrow \texttt{Enc}(PK, (M_1, M_2))$: takes public key pk and message (M_1, M_2), outputs a cipher c;
- $d_i \leftarrow \texttt{Dec}(c, sk_i)$: outputs a partial decryption of c;

- $(M_1, M_2) \leftarrow \texttt{Comb}(\{d_i\}_{i\in S \text{ s.t. } |S|>t})$: takes $t+1$ partial decryption as input, outputs a decryption.

In Hades, each TPKE encryption is accompanied by a zero-knowledge proof to prove the correctness of the ciphertext, which can also prevent chosen ciphertext attacks (CCA) [14]. Notice that our approach doesn't require a validity validation for partial decryption. Consequently, to ensure successful decryption, a mechanism is needed to discern valid partial decryptions from the gathered ones, as active faulty committee members may provide faulty partial decryptions. Accordingly, we require that the message being encrypted follows a specific format, and forging a partial decryption to produce a plaintext matching this format is computationally impossible. Thus, the correctness of the decryption can be ensured by checking the format of the decrypted message. Due to the above two features, our TPKE scheme does not require the verification key and the encryption/decryption verification scheme included in [14] to ensure CCA security and active faulty decryption tolerance. Our TPKE scheme is derived from applying secret sharing to the EC-ElGamal encryption scheme [34], which makes our TPKE scheme remarkably simple and efficient.

Generalized Pedersen commitment: In Hades, a generalized version of Pedersen commitment scheme [32] is used to hide values of identity attributes into a commitment. We use the following notation to present this operation: $cm \leftarrow \texttt{Commit}(\{v_0, ..., v_n\})$.

3.2 System Setup

a) TPKE $\texttt{Setup}(1^\lambda, n, t)$: The Hades' committee C is consisted of n distinguished entities $(C_1, ..., C_n)$. To set up, the committee members run a distributed key generation protocol [25] to generate a key pair $(\mathsf{PK}^\mathsf{C}, \mathsf{sk}^\mathsf{C})$ (sk^C is unknown). In the end, committee member C_i receives sk^c which is a (t, n)-Shamir secret sharing of sk^c, where $i \in [1, n]$. The committee will publish PK^C on the *identity contract.*

b) Zk-SNARK trusted setup: For efficiency, Hades uses zk-SNARK as the zero-knowledge proof scheme since zk-SNARK is well supported by EVM. Currently, zk-SNARK schemes require a common reference string (CRS) to be constructed in a one-time setup for each statement [6]. To set up, the committee members run a multi-party computation protocol [6] to generate CRSs used in Hades.

c) Base points selection: Recall that we use *Generalized Pedersen commitment* to hide identity attributes, a typical commitment to n identity attributes is

$$cm \leftarrow \sum_{i=1}^{n} v_i G_i + rG$$

where v_i is an identity attribute value, and G_i is a base point of elliptic curve cryptography (ECC). Suppose Hades supports n identity attributes at the beginning, to set up, the committee members first run a multiparty coin toss scheme [3] to generate a pseudo-random number y_0, then run a base points selection scheme [38] to generate n base points $(\mathsf{G}_1, ..., \mathsf{G}_n)$ with y_0 as the seed. These base points will be published on the identity contract. Notice that, the number of identity attributes is not fixed, and new attributes can be added afterward, simply by adding new base points.

3.3 Credential Generation

Hades supports multiple CAs, where each CA's public key PK_i^A (i is the serial number) is recorded in the identity contract. Each CA may authenticate different identity attributes. The committee can add new CAs to the system or remove old ones.

Let $\mathbb{G}_p$ be a elliptic curve group of order p defined over the finite field $\mathbb{F}_q$ of order q, to obtain a credential, user U first generates two random numbers

$$sk^u \leftarrow_R \mathbb{Z}_p, \beta \leftarrow_R \mathbb{Z}_p \text{ s.t. } \texttt{Encode}(\beta) \in \mathbb{G}_p$$

where sk^u is the user's private key, β is the trapdoor for tracing, and $\texttt{Encode}()$ is a probabilistic encoding function that embeds a number into an ECC point [26]. User $\texttt{U}$ then calculate $PK^U \leftarrow sk^U G, B \leftarrow \beta G$ which will be sent to the CA.

Recall that, to enable traceability, the user must supply the CA with a *trace string* which is TPKE encryption of the trapdoor. In order to ensure that the ciphertext can still be decrypted correctly in the presence of active adversaries, we require the user to encrypt the trapdoor β together with the CA's public key PK_i^A, that is

$$\begin{aligned}\psi^t &\leftarrow \texttt{Enc}(\mathsf{PK}^\mathsf{C}, \texttt{Encode}(\beta), \mathsf{PK}_i^\mathsf{A}) \\ &= (kG, (k\mathsf{PK}^\mathsf{C}).\texttt{x} \cdot \texttt{Encode}(\beta), (k\mathsf{PK}^\mathsf{C}).\texttt{y} \cdot \mathsf{PK}_i^\mathsf{A})\end{aligned}$$

where k is a secret random number, PK^C is the TPKE public-key, $.\texttt{x}$ returns the x-coordinate of the point, and $.\texttt{y}$ returns the y-coordinate of the point. This achieves active faulty decryption tolerance because it's difficult for an adversary to forge a ciphertext or partial decryption such that it decrypts to a correct PK_i^A.

Furthermore, to ensure that the user submits a correct encryption of the trapdoor, the user is also required to provide the CA with a zero-knowledge proof of the ciphertext's validity, that is

$$\begin{aligned}\Pi^C \leftarrow &\texttt{NIZK}^1\{(\beta, k) : B = \beta G \\ &\wedge \psi^t = \texttt{Enc}(\mathsf{PK}^\mathsf{C}, \texttt{Encode}(\beta), \mathsf{PK}_i^\mathsf{A})\}.\end{aligned}$$

In summary, user $\texttt{U}$ submits (PK^U, B, ψ^t, π_c) with an *identity document* to the CA. The *identity document* contains the user's identity attributes to be authenticated $\{a_j\}_{j\in S}$ and the information required to verify these attributes. The CA conducts verification of the user's attributes along with some other checks. Once the verification is successful, the CA issues a credential to the user. The format of the credential is as follows:

$$\Gamma : (PK^U || B || A || e, \sigma)$$

where $A = \sum_{j\in S} a_j \mathsf{G}_j$ is the commitment of identity attribute values, e is the expiration time of the credential, $\sigma = \texttt{Sign}(\mathsf{sk}_i^\mathsf{A}, (PK^U||B||A||e))$ is a signature on $(PK^U||B||A||e)$ signed by the CA. The scheme of credential generation is shown in Fig. 2.

3.4 Pseudonym Registration

Utilizing credentials, users can register pseudonyms (i.e., Blockchain addresses) on the identity contract. Recall that, each credential has an expiration time e. The pseudonym should only be valid before its associated credential expires. Therefore, when registering a pseudonym, users are required to set an expiration time e_t for the pseudonym. To prevent pseudonyms from being linked via their expiration times, we allow users to set e_t to any value that is less than or equal to e, i.e., $e_t \leq e$.

$\mathsf{PK}^\mathsf{C}, \mathsf{G}, (\mathsf{G}_1, \ldots, \mathsf{G}_n), \mathsf{CRS}_1$		
User: $\{a_j\}_{j \in S}$		**CA**: sk_i^A
$sk^U \leftarrow_R \mathbb{Z}_p$		
$\beta \leftarrow_R \mathbb{Z}_p$: $\exists \alpha \in \mathbb{F}_q, (\alpha, \beta) \in \mathbb{G}_p$		
$PK^U \leftarrow x\mathsf{G}, B \leftarrow \beta\mathsf{G}$		
$\psi^t \leftarrow \mathsf{Enc}(\mathsf{PK}^\mathsf{C}, \mathsf{Encode}(\beta), \mathsf{PK}_i^\mathsf{A})$		
$\Pi^c \leftarrow \mathsf{NIZK}^1\{\ldots\}$[a]	$\xrightarrow[\{a_j\}_{j \in S}]{PK^U, B, \psi^t, \Pi^c}$	Verify identity, *abort* if failed Verify Π^c, *abort* if failed
		$A \leftarrow \sum_{j \in S} a_j G_j$
		Choose e
		$\sigma = \mathsf{Sign}(sk, (X \| B \| A \| e))$
Verify σ, *abort* if failed	$\xleftarrow{e, \sigma}$	Store: $(PK^U, B, e, \psi^t, \sigma, \{a_j\}_{j \in S})$
Store $(sk^U, \beta, e, \sigma, \{a_j\}_{j \in S})$		
$(sk^U, \beta, e, \sigma, \{a_j\}_{j \in S})$		$(PK^U, B, e, \psi^t, \sigma, \{a_j\}_{j \in S})$

[a] We omit the details for simplicity, which can be obtained in Section 3.3

Figure 2: The scheme of credential generation

In order to enable users to selectively disclose their identity attributes when accessing applications, users need to bind an identity commitment A_t to each pseudonym registered, which is a generalized Pedersen commitment that can perfectly hide the identity attributes:

$$A_t = A + r_t G, r_t \leftarrow_R \mathbb{Z}_p$$

where A is the commitment of identity attribute values contained in the credential, and r_t is a secret random number that differs each time the pseudonym is registered.

In order to reveal the user's identity information when necessary, users need to provide an *audit string* for each of their pseudonyms, which is a TPKE encryption of the user's public key PK^U and issuer's public key PK_i^A. To prevent adversaries from reusing the registration data to register other pseudonyms, we encode the pseudonym into the audit string, which makes the audit string only valid for the pseudonym it was encoded with. The *audit string* is

$$\begin{aligned}\psi^a &\leftarrow \mathsf{Enc}(\mathsf{PK}^\mathsf{C}, PK^U, (\mathsf{PK}_i^\mathsf{A} + \xi^U \mathsf{G})) \\ &= (k\mathsf{G}, (k\mathsf{PK}^\mathsf{C}).\mathsf{x} \cdot PK^U, (k\mathsf{PK}^\mathsf{C}).\mathsf{y} \cdot (\mathsf{PK}_i^\mathsf{A} + \xi^U \mathsf{G}))\end{aligned}$$

where k is a secret random number, and ξ^U is the identifier of the pseudonym.

Furthermore, to make the pseudonym traceable, we require the user to generate k in the following way:

$$k = \mathsf{Hash}(\beta \| \mathsf{nonce}) \text{ s.t. } |\mathbb{P}| - w \leq \mathsf{nonce} \leq |\mathbb{P}|$$

where Hash is a collision-resistant hash function, β is the trapdoor of the credential, $|\mathbb{P}|$ is the number of pseudonyms registered in Hades, and w is the size of the sliding window which helps to prevent transaction validation failures caused by changes in the number of pseudonym registrations. The idea is simple. If the trapdoor of a credential is revealed, the CA or other entities can use the trapdoor to locally compute all values of eligible k and then all values of $k\mathsf{G}$ which are the first elements of the audit strings. Notice that, We limit the value of the nonce within the range of a sliding window. This ensures that during the tracing process, only $|\mathbb{P}| - |\mathbb{P}_0| + w$ computations of the k value are needed, where $|\mathbb{P}_0|$ is the number of pseudonyms when the credential is issued.

Overall, to register a pseudonym, the user needs to submit (e_t, A_t, ψ^a) to the identity contract. The user also needs to prove that the above information is correctly provided and that she/he is the owner of the credential with zero-knowledge proof. In addition, The user also needs to prove that the credential used has not been revoked and that the issuer is a member of Hades' CAs. To achieve this, the committee maintains two Merkle trees, the CA tree τ^c and the revocation tree τ^r, where the CA tree stores the public keys of Hades' CAs and the revocation tree stores the master public keys of all revoked credentials. If a new CA is added or a new credential is revoked, the committee will update these two trees and record the new roots on the contract. The zero-knowledge proof that the user needs to submit is

$$\begin{aligned}\Pi^P \leftarrow & \mathsf{NIZK}^2\{(sk^U, \beta, \Gamma, r_t, \mathsf{nonce}, \pi^r, \pi^c) : e^p \leq \Gamma.e \\ & \wedge k = \mathsf{Hash}(\beta \| \mathsf{nonce}) \wedge m_0 \leq \mathsf{nonce} \leq m_1 \\ & \wedge \psi^a \leftarrow \mathsf{Enc}(\mathsf{PK}^\mathsf{C}, PK^U, (\mathsf{PK}_i^\mathsf{A} + \xi^U \mathsf{G})) \text{ with } k \\ & \wedge A_t = \Gamma.A + r_t \mathsf{G} \\ & \wedge sk^U \mathsf{G} = \Gamma.PK^U \wedge \beta\mathsf{G} = \Gamma.B \\ & \wedge 1 = \mathsf{SigVerify}(\mathsf{PK}_i^\mathsf{A}, \Gamma.PK^U \| \Gamma.B \| \Gamma.A \| \Gamma.e, \Gamma.\sigma) \\ & \wedge 1 = \mathsf{InVerify}(\pi^c, \mathsf{PK}_i^\mathsf{A}, \mathsf{TreeRoot}(\tau^c)) \\ & \wedge 1 = \mathsf{NotInVerify}(\pi^r, PK^U, \mathsf{TreeRoot}(\tau^r))\}.\end{aligned}$$

The user initiates a transaction to submit (e_t, A_t, ψ^a) along with (Π^P, m_0, m_1) to the identity contract. The identity contract verifies Π^P and checks that $|\mathbb{P}| - w \leq m_0 \leq m_1 \leq |\mathbb{P}|$. Once the verification passes, the registration is accepted and (e_t, A_t, ψ^a) is recorded in the identity contract. Furthermore, to ensure that the pseudonym registration is approved by its owner, we require that the sender of the transaction must be the pseudonym that is being registered. The whole scheme of pseudonym registration is shown in Fig. 3.

3.5 Accountability

Audit. If a pseudonym has shown malicious behavior, its identity-related information can be revealed by a threshold number (t+1-out-of-n) of committee members. Recall that, to register a pseudonym, an *audit string* ψ^a is submitted to the identity contract, which is TPKE encryption of the owner's public key PK^U and the issuer's public key pk_i^A on PK^C (Section 3.4). To audit a pseudonym, the committee first retrieves the *audit string* ψ^a of the pseudonym from the identity contract, and then $t + 1$ of them can collaboratively decrypt ψ^a to recover PK^U and pk_i^A. By querying the CA identified by pk_i^A with PK^U, the identity information associated with the pseudonym can be revealed. Notice that, the user's public key PK^U is not publicly visible and can only be obtained by decrypting the associated audit string. Thus, the identity information of pseudonyms will not be revealed unless more than a threshold of committee members cooperate.

$\mathsf{PK^C}, \mathsf{G}, \tau_r, \tau_c, \mathsf{CRS}_2$		
User: $sk^U, \beta, \Gamma, \mathsf{pk}_i^{\mathsf{A}}, \xi^U$		**Identity Contract:**
Choose $e_s : e_t \leq e$		
$A_t \leftarrow A + r_t G, r_t \leftarrow_R \mathbb{Z}_p$		
Choose $m_0, m_1 : \lvert\mathbb{P}\rvert - w \leq m_0 \leq m_1 \leq \lvert\mathbb{P}\rvert$		
Choose $\mathtt{nonce} : m_0 \leq \mathtt{nonce} \leq m_1$		
$k \leftarrow \mathtt{Hash}(\beta \vert\vert \mathtt{nonce})$		
$\psi^a \leftarrow \mathtt{Enc}(\mathsf{PK^C}, PK^U, (\mathsf{PK}_i^{\mathsf{A}} + \xi^U \mathsf{G}))$ `with` k		
$\Pi^P \leftarrow \mathtt{NIZK}^2\{...\}$[a]	$\xrightarrow[\Pi^P, m_0, m_1, T_\xi]{A_t, e_t, \psi^a}$	Check: $\lvert\mathbb{P}'\rvert - w \leq m_0 \leq m_1 \leq \lvert\mathbb{P}'\rvert$ *abort* if failed
		$\xi^U \leftarrow T_\xi.\mathtt{sender()}$
		Verify Π^P, *abort* if failed
if $Res \equiv 0$, *abort*	$\xleftarrow{Res}$	Store $(\xi^U, A_t, e_t, \psi^a)$
Store $(\xi^U, e_t, r_t, \mathtt{nonce})$		
$(\xi^U, e_t, r_t, \mathtt{nonce})$		$(\xi^U, A_t, e_t, \psi^a)$

[a] We omit the details for simplicity, which can be obtained in Section 3.4

Figure 3: The scheme of pseudonym registration

Tracing. Recall that, when applying for a credential, the user is required to provide a *trace string* ψ^t to the issuer, which is TPKE encryption of the trapdoor β on $\mathsf{PK^C}$ (Section 3.3). To trace a user, the authority can request the CA that authenticated the user to submit the trace string ψ^t to the committee. $t+1$ members of the committee can collaboratively decrypt ψ^t to recover the trapdoor β. With the trapdoor β, the authority can locally computes $\lvert\mathbb{P}\rvert - \lvert\mathbb{P}_0\rvert + w$ points:

$$\mathcal{L} : \{C_0^i = \mathtt{Hash}(\beta \vert\vert \mathtt{nonce})\mathsf{G}, \text{ s.t. } \lvert\mathbb{P}_0\rvert - w \leq \mathtt{nonce} \leq \lvert\mathbb{P}\rvert\}$$

where $\lvert\mathbb{P}\rvert$ is the current number of pseudonyms and $\lvert\mathbb{P}_0\rvert$ is the number of pseudonyms when the user obtains the credential. Then, the authority identifies all pseudonyms among the registered pseudonyms whose *audit string*'s first element is in $\mathcal{L}$, which are all pseudonyms that belong to the user under investigation (Section 3.4). The trapdoor can only be retrieved through the collaboration of a supermajority of committee members, since the committee is honest-majority, tracing honest users is unfeasible.

For certain needs, such as regulatory requirements, users can proactively provide the trapdoor β to third parties to achieve *limited privacy*, that is, the user's linkability information is transparent to those who receive β, and hidden from others.

Revocation. The credentials or the pseudonyms can be revoked if $t + 1$ members of the committee agree. Furthermore, to reduce validation overhead, we make that the revocation of a certificate or pseudonym does not affect the validity of other users' pseudonyms. (1) To revoke a credential, the committee first adds the credential's public key (i.e., PK^U) into the revocation tree τ_r, and then submits the root of the new tree to the identity contract with signatures of $t+1$ members. From now on, proof of pseudonym registration using this credential will fail verification. (2) To revoke a pseudonym, $t + 1$ committee members update the status of the pseudonym to "revoked" in the identity contract. A pseudonym with a status of "revoked" will no longer be valid.

In general, unless the user voluntarily reveals her identity information, auditing, tracing, and revoking all require the cooperation of $t + 1$ committee members. So, the accountability is performed in a decentralized way. The scheme of accountability is shown in Fig. 4.

$\mathsf{G}, \mathsf{PK^C}, \tau_r, \mathtt{IC}$[a]		
Committee: $\{\mathsf{sk}_1^{\mathsf{C}}, ..., \mathsf{sk}_n^{\mathsf{C}}\}, t$		**CAs**: $\mathsf{DB_U}$[a]
Audit: ξ^U		
$\psi^a \leftarrow \mathtt{IC.info}(\xi^U)$		
$d_j^a \leftarrow \mathtt{Dec}(\psi^a, \mathsf{sk}_j^{\mathsf{C}})$		
$(PK^U, M_2) \leftarrow \mathtt{Comb}(\{d_j^a\}_{j \in S \text{ s.t. } \lvert S\rvert > t})$		
$\mathsf{PK}_i^{\mathsf{A}} = M_2 - \xi^U \mathsf{G}$	$\xrightarrow[\text{to CA}(\mathsf{PK}_i^{\mathsf{A}})]{PK^U}$	$\mathtt{info} \leftarrow \mathsf{DB_U}.\mathtt{info}(PK^U)$
Reveal `info`	$\xleftarrow{\mathtt{info}}$	
Trace: $\mathtt{ID^U}$		
$d_j^t \leftarrow \mathtt{Dec}(\psi^t, \mathsf{sk}_j^{\mathsf{C}})$	$\xleftarrow{\psi^t}$	$\psi^t \leftarrow \mathsf{DB_U}.\text{info}(\mathtt{ID^U})$
$(\beta, \mathsf{PK}_j^{\mathsf{A}}) \leftarrow \mathtt{Comb}(\{d_j^t\}_{j \in S \text{ s.t. } \lvert S\rvert > t})$		
Public ψ^t	$\xrightarrow{\beta}$	$n_0 \leftarrow \lvert\mathbb{P}_0\rvert - w, n \leftarrow \lvert\mathbb{P}\rvert$
		$C_0 \leftarrow \{\mathtt{Hash}(\beta, j)\}_{j \in [n_0, n]}$
Reveal $\{\xi^U\}$	$\xleftarrow[\text{if needed}]{\{\xi^U\}}$	$\{\xi^U\} \leftarrow \mathtt{IC.filter}(C_0)$
Revoke: ξ^U, Γ		
$\tau_r \leftarrow \mathtt{TreeAdd}(\Gamma.PK^U, \tau_r)$		
$\mathtt{IC.udateRoot}(\mathtt{TreeRoot}(\gamma_r))$		
$\mathtt{IC.revokePseudonym}(\xi^U)$	$\xrightarrow[\text{revoked}]{\Gamma}$	$\mathsf{DB_U}.\mathtt{revoke}(\Gamma)$

[a] `IC` is the identity contract; $\mathsf{DB_U}$ is the user information storage of CA.

Figure 4: The scheme of accountability

3.6 Sybil-resistance

We design an n-time access token generation scheme, which takes the credential's private key and Sybil-resistance instance ID as input, and outputs at most n distinct tokens. The scheme is

$$\varphi = \mathtt{Hash}(sk^U \vert\vert \zeta \vert\vert \mathtt{nonce}) \text{ s.t. } 0 \leq \mathtt{nonce} < \mathtt{limit}(\{a_i\})$$

where sk^U is the private key of the credential, ζ is the Sybil-resistance instance ID, $\{a_i\}$ is a set of identity attributes, $\mathtt{Hash}(...)$ is a collision-resistant hash function, and $\mathtt{limit}(...)$ is a function that can be customized by the application, which returns a quantity limit based on the identity attributes.

To implement Sybil resistance, the application first initializes a Sybil resistance instance with an instance ID ζ and a limit function

$\mathtt{limit}(...)$. Users accessing the application need to provide an access token generated with this instance ID ζ for each pseudonym they use. Given that a user can generate only a limited number of distinct tokens, she/he cannot carry out a Sybil attack by registering massive unlinkable pseudonyms. In addition, in order to ensure the correctness of the provided access token, the user needs to generate a zero-knowledge proof, that is

$$\begin{aligned}\Pi^S \leftarrow \mathtt{NIZK}^3\{(sk^U, \mathtt{nonce}, k, \{a_i\}, r) : PK^U &= sk^U \mathtt{G} \\ \wedge\ \psi^a &\leftarrow \mathtt{Enc}(\mathtt{PK}^\mathtt{C}, PK^U, (\mathtt{PK}^\mathtt{A}_i + \xi^U \mathtt{G})) \\ \wedge\ \varphi &= \mathtt{Hash}(sk^U || \zeta || \mathtt{nonce}) \\ \wedge\ 0 \le \mathtt{nonce} &< \mathtt{limit}(PK^U, \{a_i\}) \\ \wedge\ A_t &= \sum a_i \mathtt{G}_i + r\mathtt{G}\ \}\end{aligned}$$

where ψ^a is the pseudonym's *audit string* and A_t is the Pedersen commitment of identity attributes, both can be obtained from the identity contract. For efficiency, applications can design dedicated zero-knowledge proof circuits embedding their customized `limit` functions. For some commonly used `limit` functions, we will design general-purpose zero-knowledge proof circuits or provide circuit design templates. Finally, the application checks the access token for duplicates and verifies the proof (on-chain or off-chain). The scheme of Sybil-resistance is shown in Fig. 5.

Notice that, apart from the access token, users do not need to disclose any other information (such as the nonce) to the application. Moreover, only the pseudonym's owner can generate the access tokens, so the values of access tokens cannot be predicted by others. These characteristics ensure that the Sybil-resistance scheme is privacy-preserving. Furthermore, as an adversary cannot forge an access token that doesn't belong to her/him and the access token is collision-resistant, an adversary cannot have a user be denied access by the application by forging access for the user.

$\mathtt{G}, \mathtt{PK}^\mathtt{C}, \mathtt{CRS}_3, \mathtt{limit}(...), \zeta, \mathtt{IC}$[b]		
User: $sk^U, k, r, \{a_i\}$		**Application**:
$n \leftarrow \mathtt{limit}(\{a_i\})$		
$\mathtt{nonce} \leftarrow_R [0, n)$		
$\varphi = \mathtt{Hash}(sk^U \| \zeta \| \mathtt{nonce})$		
$\Pi^S \leftarrow \mathrm{NIZK}^3\{...\}$[a]	$\xrightarrow{\xi^U, \varphi, \Pi^S}$	$(\psi^a, A_t) \leftarrow \mathtt{IC.info}(\xi^U)$
		Verify Π^S, *abort* if failed
if $\mathtt{Res} \equiv 0$, abort	$\xleftarrow{\mathtt{Res}}$	Store (ξ^U, φ)
Store $(\xi^U, \mathtt{nonce}, \varphi)$		
$(\xi^U, \mathtt{nonce}, \varphi)$		(ξ^U, φ)

[a] We omit the details for simplicity, which can be obtained in Section 3.4
[b] IC is the identity contract.

Figure 5: The scheme of Sybil-resistance

Sybil-resistance over identity identifiers. In the Sybil-resistance mechanism discussed above, the Sybil-resistance process is conducted over credentials, considering each credential as a unique entity. This is not problematic under the single issuer model but may fail when users can apply credentials from multiple CAs. A plausible approach is to employ Sybil-resistance over identity identifiers, such as Social Security Numbers (SSN) issued by the US government for US residents. We will show how to achieve this approach using a threshold pseudorandom function (PRF).

Let $\mathtt{PRF}(\mathtt{sk}^\mathtt{C}_\mathtt{prf}, x)$ be a threshold pseudorandom function. To set up, the committee members execute a distributed key generation protocol to generate $(\mathtt{PK}^\mathtt{C}_\mathtt{prf}, \mathtt{sk}^\mathtt{C}_\mathtt{prf})$, and each member holds $\mathtt{sk}^\mathtt{C}_\mathtt{prf,i}$, secret shares of $\mathtt{sk}^\mathtt{C}_\mathtt{prf}$. Suppose a user possesses an SSN $\mathtt{ID}^\mathtt{U}$ and holds a credential that verifies this SSN. The user sends the secret shares $[\mathtt{ID}^\mathtt{U}]$ of $\mathtt{ID}^\mathtt{U}$ to the committee, accompanied by a zero-knowledge proof, indicating that she/he knows the private key of a credential that has authenticated $\mathtt{ID}^\mathtt{U}$. The committee verifies the received proofs and executes an MPC protocol to compute $\mathtt{ID}^\mathtt{U}_\mathtt{prf} = \mathtt{PRF}([\mathtt{sk}^\mathtt{C}_\mathtt{prf}], [\mathtt{ID}^\mathtt{U}])$. With $\mathtt{ID}^\mathtt{U}_\mathtt{prf}$, the user can generate access token as

$$\varphi = \mathtt{ID}^\mathtt{U}_\mathtt{prf}\mathtt{Hash}(\mathtt{ID}^\mathtt{U} || \zeta || \mathtt{nonce}) \text{ s.t. } 0 \le \mathtt{nonce} < \mathtt{limit}(\{a_i\}).$$

The aforementioned scheme is privacy-preserving, as only the owner of the $\mathtt{ID}^\mathtt{U}$ can obtain $\mathtt{ID}^\mathtt{U}_\mathtt{prf}$ through the committee, thereby keeping $\mathtt{ID}^\mathtt{U}_\mathtt{prf}$ confidential from others. Although the committee members might learn about $\mathtt{ID}^\mathtt{U}_\mathtt{prf}$ during the PRF computation process, they remain unaware of $\mathtt{ID}^\mathtt{U}$ and thus cannot predict the user's access token.

3.7 Selective Disclosure & Selective Linkability

Selective Disclosure. Recall that when a user registers a pseudonym, she/he needs to submit a Pedersen commitment, A_t, of her/his identity attribute values, which is recorded in the identity contract. To meet access control requirements, a user may need to prove to the application that her/his identity attributes meet certain assertions, such as being over 18 years old. To achieve this, the user can provide a zero-knowledge proof to the Dapp to prove that its identity attribute values committed to A_t satisfy the given assertions without revealing the values, as shown below.

$$\Pi^D \leftarrow \mathtt{NIZK}^4\{\{a_1, ..., a_n\}, r) : A_t = \sum_{i=1}^{n} a_i G_i + rG$$
$$\wedge \text{ statements about } a_i \text{ for } 1 \le i \le n\}$$

Applications can design dedicated zero-knowledge proof circuits in line with their access control rules, which offers high efficiency and flexibility. For ease of use, Hades provides a relatively general zero-knowledge proof scheme, which allows users to prove that their attribute values are within a certain range. It's worth noting that we specify 0 to represent an empty value, so adding new attributes will not affect the old identities, and the old attribute commitments are still valid, with empty values on the new attributes.
Selective Linkability. Users can use the access token generation scheme to prove the linkability of their pseudonyms without revealing identity-related information. If two pseudonyms are registered using the same credential, then given the same instance ID, the owner will certainly be able to generate an identical access token for them. Moreover, due to the collision resistance of the hash function, it is computationally difficult to generate an identical *access token* for pseudonyms registered using different credentials. To prove

the linkability of pseudonyms, the owner can generate an identical *access token* for the pseudonyms using the same context. Since the *access token* generation does not reveal any identity-related information, this selective linkability scheme is privacy-preserving.

Selective linkability is very useful in pseudonym replacement and pseudonym revocation. If a pseudonym's private key is lost, the owner can replace this pseudonym with a new one by proving to the application that the new pseudonym is linkable to the old pseudonym. In case a certificate is stolen and an adversary has registered pseudonyms using it, the owner can request the committee to revoke these pseudonyms by proving to the committee that these pseudonyms are linkable to her/his own, without having to disclose identity information to the committee.

4 IMPLEMENTATION

We implemented Hades using Rust and Solidity and published the code on GitHub as an open-source project[1]. The SDK is written in Rust, which can be used for setting up systems, issuing credentials, generating zero-knowledge proofs, auditing, tracing and revoking identities, etc. The identity contract is written in Solidity, which is responsible for verifying zero-knowledge proofs and recording the status information of users and CAs. The identity contract can be deployed on EVM-powered Blockchains. In the rest of this section, we will describe our implementation in detail.

4.1 Cryptographic Schemes

Cryptographic hash function. We use Poseidon [17] to instantiate the cryptographic hash function used in the Hades system. Poseidon is a ZK-friendly hash function that just costs 240 constraints hashing two field elements.

Zero-knowledge proof. Since the Byzantium fork, Ethereum (and other EVM-powered Blockchains) has supported efficient performing pairing checks and elliptic curve operations on the alt_bn128 (a.k.a. BN254) curve through pre-compiled contracts [36] [7]. For efficient verification of zero-knowledge proofs, we use Groth16 [18] over the alt_bn128 to instantiate the zero-knowledge proof scheme used in the Hades system. Since we build the Hades system on the assumption that the majority of the committee is honest, it is not trivial to run the trusted setup required by Groth16 among the committee, using an honest-majority MPC protocol.

Elliptic curve cryptography. All the elliptic curve operations used in the schemes of Hades, except for the zk-SNARK scheme, are performed on Baby-jubjub curve [43], an elliptic curve designed to work efficiently with zk-SNARKs. Baby-jubjub is defined over a finite field $\mathbb{F}_r$ which has the same order with alt_bn128 curve. This means that the curve operations on Baby-jubjub can be performed directly inside the zk-SNARK circuit, which will cost very few constraints. We use EdDSA [24] on Baby-jubjub curve to instantiate the signature scheme used in credential generation (Section 3.3).

Ordered Merkle tree. We use the Ordered Merkle tree to instantiate the Merkle tree scheme used in Hades, which is a complete and full binary tree with a fixed height. The elements in the tree are placed on the leaf nodes sequentially in ascending order, starting from the leftmost leaf node. To prove that an element e does not exist in the tree is to prove that there are two adjacent leaf nodes whose elements are e_i, e_j that satisfy $e_i < e < e_j$. We use the Ordered Merkle tree because it doesn't have the element collision problem, which means that we can store any 2^{40} numbers of 256-bit length in a tree of height 40 without worrying that some numbers cannot be stored due to collisions.

4.2 Implementation Details

We present the zk-SNARK statements in *Circom* [4], which is a domain-specific language for defining arithmetic circuits and can compile statements into R1CS constraints. We implemented the zk-SNARK scheme in Rust using the *ark-circom* library and implemented the Baby-jubjub curve operations, EdDSA scheme, and Poseidon hash scheme in Rust using the *babyjubjub-rs* library. Also, we implemented the interaction scheme of EVM-powered chains in Rust using the *ethers-rs* library.

We implemented the identity contract in Solidity and use the pre-compiled contracts for performing zk-SNARK proof verification. To verify a zk-SNARK proof, the contract needs to perform n point additions, n point multiplications, and 4 pairing checks on the alt_128 curve, where n is the number of public inputs. Notice that, a point addition costs 150 gas, a point multiplication cost 6000 gas, and 4 pairing checks cost 181,000 gas, and the gas cost for a zk-SNARK proof verification is

$$g_v = 181000 + n(150 + 6000)$$

where n is the number of public inputs. So, we can reduce the gas cost by reducing the number of public inputs. To reduce the number of public inputs, we: (1) compress each point (x, y) in the public inputs into a symbol and an integer $(\{0, 1\}, y)$ and encode all the symbols into an integer. This reduces the number of inputs for points in half. (2) Commit the values (i.e., $\mathsf{PK}^{\mathsf{C}}, \gamma_r, \gamma_c$) controlled by the committee into an integer, and this will reduce 2 inputs.

5 CASE STUDY

In this section, we present a case study on fair NFT distribution ("airdrops") and show how to leverage Hades to achieve fine-grained Sybil-resistance and improved security without compromising privacy. Suppose a scenario where an organization wants to limit the number of NFT airdrops a user can get based on her identity attributes (e.g., the number of Twitter followers). At the same time, in order to achieve privacy-preserving, the identity information of airdrop recipients, the number of airdrops received by each user, and the linkage information among airdrop recipients are required to be hidden. In addition, to achieve improved security, in the event of a user violating the rules (such as illegally trading NFT), all airdrops received by the user should be identified and confiscated.

We show that all previous works [28, 35, 40, 46] fail in this scenario. Firstly, none of these works, except Candid [28], support traceability. While CanDID does support traceability, due to its lack of auditing capabilities, it can only trace a given real identity, but cannot trace a given pseudonym. This implies that for a user violating the rules, none of these works are able to trace all the airdrops received by her/him. Secondly, only CanDID supports Sybil-resistance, but it has two limitations: (1) to achieve sybil resistance, users need to apply for non-duplicated context-based credentials from the CanDID committee, which will expose the

[1]https://github.com/didnet/Hades

airdrops that users participate in. (2) Only one context-based credential per context can be issued in CanDID, this means it is hard to configure individualized airdrop reception limits for each user. Although this challenge can be addressed by setting up multiple different contexts, this will make the recipient linkable and the identity information leaked. So, the solution using CanDID is not privacy-preserving.

We will show how to use Hades to achieve Sybil-resistance for the scenario in the aforementioned example with privacy-preserving and improved security. Suppose there is a CA in Hades that can authenticate the number of Twitter followers of a user, and for each Twitter user, the CA will only issue one credential. The organization instantiates a Sybil-resistance instance with an instance id ζ and a limit function `limit()` which takes the user's identity attributes as input and returns 2 if the user's Twitter followers count exceeds 100, otherwise returns 1. To receive airdrops, users first apply for credentials from the CA that authenticates the number of their Twitter followers (skip if they have previously applied). Then, users utilize the acquired credentials to register pseudonyms and claim airdrops using these pseudonyms. Furthermore, users must generate an access token φ for each access to the airdrop, using the method discussed in section 3.6. As the number of distinct access tokens that a user can generate will not exceed the value specified by the limit function, i.e., users with more than 100 Twitter followers can generate at most two distinct access tokens, while the rest can only generate one. The airdrop application can disburse airdrops solely to previously unseen access tokens. This ensures users with over 100 followers can receive up to two airdrops, while the rest can receive a maximum of one. If the NFT holder has a violation, the organization can find out her/his identity through *auditing*, and then find out all her pseudonyms through *tracing*, and then find out all the NFTs she holds, and confiscate these NFTs. This is done in a decentralized way as it requires the cooperation of $t + 1$ committee members.

The above implementation is privacy-preserving because the `nonce` used by the recipient to generate the access token is not exposed, so it is impossible to infer the recipient's airdrop-taking caps, and then the identity attributes. It is also security-improved, as organizations can not only freeze NTFs associated with malicious behavior but also find out the identity of the adversary and freeze all NFTs under her/his name.

6 EVALUATION

We evaluated our implementation of Hades on an Ubuntu 2204 instance, hosted via Windows Subsystem for Linux (WSL2) on a Microsoft Windows 11 operating system. The test machine was equipped with an Intel Core i9-13900K@3.0GHz 16-Core (8P+16E) CPU and 64 GB of RAM. The identity contract was deployed on BSC Testnet.

We evaluated the generation time of zero-knowledge proofs in Hades, which are the most time-consuming operations, and the results are shown in Table 1. It shows that the pseudonym registration proof has the highest overhead, with a time consumption of 1.9s and a constraint consumption of 27k, which is one-third of ZEBRA's credential verification [35]. Note that these benchmarks are done on a normal PC using a single thread, and the time overhead of running them with a browser plugin or mobile application does not increase significantly. In addition, we can generate the proofs in parallel [31], which will greatly reduce the time overhead of zero-knowledge proof generation.

Table 1: Benchmark of Hades' client

Operation	#Constraints	Time[ms]
credential generation	3, 907	195
pseudonym registration	31, 951	614
Sybil-resistance	4, 291	245
selective disclosure	15, 856	564

*All the benchmarks are performed with a single thread.
†We use a CA tree of height 31 and a revocation tree of height 41.
‡We use a commonly used Sybil-resistance strategy where the `limit` function returns a fixed value. The selective disclosure scheme is to prove that the user's attribute values are within a certain range.

We evaluated the operations of the identity contract, and the results are shown in Table 2. Pseudonym verification has the largest gas consumption of 339, 978, and it performs 4 paring checks, 7 point additions, and 7 point multiplications, which cost 224K gas, and stores 3 256-bit values, which costs 60k gas. Access token verification has a gas consumption of 248, 514, and it performs 4 paring checks, 4 point additions, and 4 point multiplications, which cost 205K gas. Selective disclosure has a gas consumption of 239, 857, and it performs 4 paring checks, 3 point additions, and 3 point multiplications, which cost 199k gas. Pseudonym revocation will incur a gas cost of 2, 500 for each pseudonym revoked.

Compared with decentralized finance (DeFi), the most popular and DID-demanding applications in the blockchain, it costs around 110k gas to exchange a token (measured in Uniswap v2 [1]), and the cost of pseudonym verification is just three times that. Since a pseudonym only needs to be verified once, the gas cost of our system is practical.

Table 2: Benchmark of Hades' contract

Operation	gas cost	input data size
pseudonym registration	339, 978	320 bytes
Sybil-resistance	248, 514	224 bytes
pseudonym revocation	$\sim$ 2, 500/pr[a]	32 bytes/pr[a]
selective disclosure	232, 857	192 bytes

[a]The symbol '/pr' means 'per pseudonym revocation'.

We compared Hades' pseudonym verification (the highest gas-cost operation in Hades) with credential verification of ZEBRA [35], Coconut [40] and BASS [46] in gas cost. The results are shown in Table 3, and it shows that our scheme has the lowest gas cost.

We also compared Hades with other identity management protocols such as ZEBRA [35], Coconut [40], BASS [46] and CanDID [28], in terms of selective linkability, selective disclosure, audit, trace,

Table 3: Comparison with prior works in gas cost

Technique	Gas cost	One-time cost
Hades	339 K	Yes
ZEBRA [35]	360 K	Yes
Coconut [40]	2, 150 K	No
BASS [46]	1, 585 K	No

revocation, and Sybil-resistance. The results are shown in Table 4, it turns out that Hades is the first DID system that has implemented all of the above features. Hades is also the first DID system supporting lightweight, fine-grained Sybil-resistance, where applications can customize personalized Sybil-resistance strategies specific to the identity attributes of users and can independently perform Sybil verification without burdening users to apply for a non-duplicate one-time credential for each application.

Table 4: Comparison with prior works in functionality

Property	Selective-disclosure	Selective-linkability	Audit-ability	Trace-ability	Revo-cation	Sybil-resistance
Hades	●	●	●	●	●	●
ZEBRA	○	◐	●	○	●	○
Coconut	●	◐	○	○	○	○
BASS	○	◐	●	○	●	○
CanDID	●	◐	○	●	●	◐

†●: supports, ◐: partially supports ○: does not support.
‡Partial support for selective linkability indicates only supporting unlinkability; partial support for Sybil-resistance refers to supporting Sybil-resistance with a single and fixed strategy.

7 RELATED WORK

Anonymous Credentials. The concept was first introduced by Chaum [15], allowing a user to make assertions about identity without revealing additional information. Following that, a long line of work has been presented with new properties, such as efficiency [12, 20], conditional anonymity revocation [10], decentralized audit [39], conditional linkability [2], selective traceability [21], traceability [8], n-times anonymous [8], multiple issuance [10], threshold issuance [37, 40], revocation [9, 11], Sybil-resistance [28], legacy-compatiblity [28] and Blockchain-friendliness [20, 35, 46]. These works are mainly built based on blind signatures, group signatures, zero-knowledge proofs, or MPC. Hades can be regarded as an anonymous credential system with decentralized audit, traceability, revocation, Sybil-resistance, and Blockchain-friendliness, implemented by not exposing the credential during the verification process. The approach used in [8] to trace e-tokens and achieve n-times anonymous authentication is close to our approach. To the best of our knowledge, Hades is the first practical system supporting fine-grained Sybil-resistance.

Decentralized Identity (DID). The standard proposed by W3C [42] is the most widely used standard of DID. Several works [19, 22, 23, 28, 30] that meet this standard have been presented. However, some of them [19, 22, 28] are not designed for permission-less Blockchains and therefore cannot run efficiently on Ethereum. Some of them [19, 23, 30] do not support accountability, which makes them fail in many scenarios. The credential/claim schemas defined by W3C [42] can be easily embedded into our credentials, thus making Hades comply with W3C DID standards, without changing the properties of Hades.

8 CONCLUSION

We presented Hades, a practical decentralized identity system that supports full accountability and fine-grained Sybil-resistance. Hades emerges as the first DID system encouraging fine-grained Sybil-resistance through a lightweight solution. Hades is the first practical DID system capable of supporting large-scale pseudonym tracing. Compared to previous approaches, most of the tracing computations in Hades can be performed locally on a single node, which is highly efficient. Hades is also the first DID system that supports selective linkability, allowing users to selectively disclose the linkability of their pseudonyms without revealing identity-related information. We implemented Hades, and our evaluation shows that Hades has the lowest gas cost incurred on EVM and is suitable for mobile devices and web plugins.

We address two future works. First, in Hades, the tracing scheme reveals all pseudonyms of a user, which can be overly burdensome for certain scenarios. Hades can address this by employing context-based *trace string*s to achieve selective traceability so that only pseudonyms of the same context will be traced. Second, we can further reduce the gas cost per verification by batch verification, like [35].

ACKNOWLEDGMENTS

This work is supported by National Key Research and Development Program of China (2022YFB2703203), National Natural Science Foundation of China (62202011, 62172010), Beijing Municipal Natural Science Foundation (M21040), and MoST Science and Technology Innovation Project of Xiong'an (2022XAGG0115).

REFERENCES

[1] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. Uniswap v2 core, 2020. *URL: https://uniswap. org/whitepaper. pdf* (2020).
[2] Foteini Baldimtsi and Anna Lysyanskaya. 2013. Anonymous Credentials Light. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &; Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 1087–1098. https://doi.org/10.1145/2508859.2516687
[3] Amos Beimel, Eran Omri, and Ilan Orlov. 2010. Protocols for Multiparty Coin Toss with Dishonest Majority. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 538–557.
[4] Marta Belles-Munoz, Miguel Isabel, Jose Luis Munoz-Tapia, Albert Rubio, and Jordi Baylina. 2022. Circom: A Circuit Description Language for Building Zero-knowledge Applications. *IEEE Transactions on Dependable and Secure Computing* 01 (2022), 1–18.
[5] Dan Boneh, Xavier Boyen, and Shai Halevi. 2006. Chosen Ciphertext Secure Public Key Threshold Encryption Without Random Oracles. In *Topics in Cryptology – CT-RSA 2006*, David Pointcheval (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 226–243.
[6] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Paper 2017/1050. https://eprint.iacr.org/2017/1050 https://eprint.iacr.org/2017/1050.
[7] Vitalik Buterin and Christian Reitwiessner. 2017. EIP 197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. *Ethereum Improvement Proposals, Ethereum, Zug, Switzerland, Tech. Rep* 197 (2017).
[8] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to Win the Clonewars: Efficient Periodic n-Times Anonymous Authentication. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '06)*.

Association for Computing Machinery, New York, NY, USA, 201–210. https://doi.org/10.1145/1180405.1180431

[9] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *Public Key Cryptography – PKC 2009*, Stanisław Jarecki and Gene Tsudik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 481–500.

[10] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *Advances in Cryptology — EUROCRYPT 2001*, Birgit Pfitzmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 93–118.

[11] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology — CRYPTO 2002*, Moti Yung (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 61–76.

[12] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology – CRYPTO 2004*, Matt Franklin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 56–72.

[13] Jan Camenisch and Markus Stadler. 1997. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science* 260 (1997).

[14] Ran Canetti and Shafi Goldwasser. 1999. An Efficient threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack (Extended Abstract). In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–106.

[15] David Chaum. 1985. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* 28, 10 (oct 1985), 1030–1044. https://doi.org/10.1145/4372.4373

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

[17] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for {Zero-Knowledge} Proof Systems. In *30th USENIX Security Symposium (USENIX Security 21)*. 519–535.

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

[19] Harry Halpin. 2020. Vision: A Critique of Immunity Passports and W3C Decentralized Identifiers. In *Security Standardisation Research*, Thyla van der Merwe, Chris Mitchell, and Maryam Mehrnezhad (Eds.). Springer International Publishing, Cham, 148–168.

[20] Lucjan Hanzlik and Daniel Slamanig. 2021. With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2004–2023. https://doi.org/10.1145/3460120.3484582

[21] Chloé Hébant and David Pointcheval. 2022. Traceable Constant-Size Multi-authority Credentials. In *Security and Cryptography for Networks*, Clemente Galdi and Stanislaw Jarecki (Eds.). Springer International Publishing, Cham, 411–434.

[22] Hyperledger. 2019. *Hyperledger Indy.* Technical Report. https://hyperledger-indy.readthedocs.io/en/latest/

[23] Iden3. 2022. *Iden3 Docs.* Technical Report. https://docs.iden3.io/

[24] S. Josefsson and I. Liusvaara. 2017. *Edwards-Curve Digital Signature Algorithm (EdDSA).* Technical Report. https://doi.org/10.17487/RFC8032

[25] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. Cryptology ePrint Archive, Paper 2012/377. https://eprint.iacr.org/2012/377 https://eprint.iacr.org/2012/377.

[26] Neal Koblitz. 1987. Elliptic curve cryptosystems. *Mathematics of computation* 48, 177 (1987), 203–209.

[27] Philip MacKenzie and Michael K. Reiter. 2001. Two-Party Generation of DSA Signatures. In *Advances in Cryptology — CRYPTO 2001*, Joe Kilian (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 137–154.

[28] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. 2021. CanDID: Can-Do Decentralized Identity with Legacy Compatibility, Sybil-Resistance, and Accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1348–1366. https://doi.org/10.1109/SP40001.2021.00038

[29] Ralph C Merkle. 1988. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.

[30] Nitin Naik and Paul Jenkins. 2020. uPort Open-Source Identity Management System: An Assessment of Self-Sovereign Identity and User-Centric Data Platform Built on Blockchain. In *2020 IEEE International Symposium on Systems Engineering (ISSE)*. 1–7. https://doi.org/10.1109/ISSE49799.2020.9272223

[31] Ning Ni and Yongxin Zhu. 2023. Enabling zero knowledge proof by accelerating zk-SNARK kernels on GPU. *J. Parallel and Distrib. Comput.* 173 (2023), 20–31. https://doi.org/10.1016/j.jpdc.2022.10.009

[32] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 129–140.

[33] Bo Qin, Qianhong Wu, Lei Zhang, and Josep Domingo-Ferrer. 2010. Threshold Public-Key Encryption with Adaptive Security and Short Ciphertexts. In *Information and Communications Security*, Miguel Soriano, Sihan Qing, and Javier López (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–76.

[34] Kefa Rabah. 2005. Elliptic curve elgamal encryption and signature schemes. *Information Technology Journal* 4, 3 (2005), 299–306.

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

[36] Christian Reitwiessner. 2017. EIP 196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. *Ethereum Improvement Proposals, Ethereum, Zug, Switzerland, Tech. Rep* 196 (2017).

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

[38] Moumita Roy, Nabamita Deb, and Amar Jyoti Kumar. 2014. Point generation and base point selection in ECC: An overview. *International Journal of Advanced Research in Computer and Communication Engineering* 3, 5 (2014), 6711–6713.

[39] Wei Shao, Chunfu Jia, Yunkai Xu, Kefan Qiu, Yan Gao, and Yituo He. 2020. AttriChain: Decentralized traceable anonymous identities in privacy-preserving permissioned blockchain. *Computers & Security* 99 (2020), 102069. https://doi.org/10.1016/j.cose.2020.102069

[40] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, and George Danezis. 2019. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. *NDSS* (2019).

[41] Yiannis Tsiounis and Moti Yung. 1998. On the security of ElGamal based encryption. In *Public Key Cryptography*, Hideki Imai and Yuliang Zheng (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 117–134.

[42] W3C. 2022. *Decentralized Identifiers (DIDs) v1.0: Core architecture, data model, and representations.* Technical Report. https://www.w3.org/TR/did-core/

[43] Barry WhiteHat, Jordi Baylina, and Marta Bellés. 2020. Baby Jubjub elliptic curve. *Ethereum Improvement Proposal, EIP-2494* 29 (2020).

[44] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[45] X-explore and WuBlockchain. 2023. *Advanced Analysis For Arbitrum Airdrop.* Technical Report. https://mirror.xyz/x-explore.eth/AFroG11e24I6S1oDvTitNdQSDh8lN5bz9VZAink8lZ4

[46] Yong Yu, Yanqi Zhao, Yannan Li, Xiaojiang Du, Lianhai Wang, and Mohsen Guizani. 2020. Blockchain-Based Anonymous Authentication With Selective Revocation for Smart Industrial Applications. *IEEE Transactions on Industrial Informatics* 16, 5 (2020), 3290–3300. https://doi.org/10.1109/TII.2019.2944678

A SKETCH OF SECURITY PROOFS

This appendix sketches the security proofs of the Hades scheme described in Section 3.

a) Unforgeability: There are two possible ways for an adversary to forge a pseudonym: (1) forge a proof Π^P without a credential or with a credential issued by a CA not in Hades' CA list; (2) corrupt the CA to obtain a credential that does not match her/his identity. The soundness property of the zero-knowledge proof ensures the impossibility of the scenario (1). For scenario (2), Since the CA is semi-honest, it cannot be corrupted by the adversary to issue a credential that does not match the adversary's identity.

b) Sybil-resistance: There are two possible ways for an adversary to break the Sybil-resistance: (1) forging an access token that does not belong to her/him; (2) generating more access tokens than the limit allows. The soundness property of the zero-knowledge proof ensures the impossibility of these two ways.

c) Privacy: There are two possible ways for an adversary to reveal identity-related information of pseudonyms: (1) by opening the Pedersen commitment A_t; 2) by decrypting the audit string ψ^a. The perfect hiding property of the Pedersen commitment scheme guarantees that (1) is impossible. For scenario (2), since the committee is honest-majority (i.e., up to t of the n committee members can be corrupted, for $t < n/2$), if the adversary succeeds in decrypting ψ^a, it implies that at most t committee members are required to cooperate to decrypt ψ^a, violating the soundness property of the $(t, n)-$TPKE.

d) Unlinkability: There are three possible ways for an adversary to link pseudonyms: (1) through linking the *audit string*s ; (2) through linking the access tokens; (3) through linking the identity commitments. The IND-CPA property [41] of TPKE ensures the impossibility of the scenario (1). The hiding and pseudorandom properties of the Hash function ensure the impossibility of the scenario (2). The perfect hiding property of the Pedersen commitment scheme ensures the impossibility of the scenario (3).

e) Liveness: There are two possible ways for an adversary to disrupt the system's liveness: (1) Instigate the revocation of the user's pseudonyms or credentials; (2) Forge the user's access token to cause the user to be detected as a Sybil. The semi-honest assumption of the CA and the honest majority assumption of the committee ensure the impossibility of the scenario (1). The collision resistance property of the hash function and the soundness property of the zero-knowledge proof ensure the impossibility of the scenario (2).

f) Audit: There are three possible ways for an adversary to disable the audit: (1) Provide forged *audit string*s when registering pseudonyms; (2) Corrupt committee members to prevent the committee from decrypting the *audit string*; (3) Corrupt the CA to prevent it from providing identity-related information. The soundness property of the zero-knowledge proof ensures the impossibility of the scenario (1). Given that the adversary can corrupt up to t committee members, i.e., a minimum of $n - t$ members will engage in decryption honestly, and as $t < n/2$, therefore $n - t > t$, the rest of the honest members can still complete the decryption, making scenario (2) impossible. Since the CA is semi-honest and cannot be corrupted, scenario (3) is also impossible.

g) Trace: There are three possible ways for an adversary to make the trace fail: (1) provide a forged *trace string* or forged *audit string*; (2) Corrupt committee members to prevent the committee from decrypting the *audit string* or *trace string*; (3) Corrupt the CA to prevent it from providing the *trace string*. The soundness property of the zero-knowledge proof ensures the impossibility of the scenario (1). For scenario (2), we have shown that it is impossible to prevent decryption by corrupting committee members. Since the CA is semi-honest and cannot be corrupted, scenario (3) is also impossible.

h) Revocation: There are two possible ways for an adversary to make the revocation fail: (1) Registering new pseudonyms with a revoked credential; (2) Preventing their pseudonyms from being traced. The soundness property of the zero-knowledge proof ensures the impossibility of the scenario (1). For scenario (2), we have already proven that blocking tracing is impossible.