Arke: Scalable and Byzantine Fault Tolerant Privacy-Preserving Contact Discovery

Nicolas Mohnblatt
Geometry
nico@geometry.xyz

Alberto Sonnino
Mysten Labs & University College London
alberto@mystenlabs.com

Kobi Gurkan
Geometry
kobi@geometry.xyz

Philipp Jovanovic
University College London
p.jovanovic@ucl.ac.uk

***Abstract*—Contact discovery is a crucial component of social applications, facilitating interactions between registered contacts. This work introduces Arke, a novel contact discovery scheme that addresses the limitations of existing solutions in terms of privacy, scalability, and reliance on trusted third parties. Arke ensures the unlinkability of user interactions, mitigates enumeration attacks, and operates without single points of failure or trust. Notably, Arke is the first contact discovery system whose performance is independent of the total number of users and the first that can operate in a Byzantine setting. It achieves its privacy goals through an unlinkable handshake mechanism built on top of an identity-based non-interactive key exchange. By leveraging a custom distributed architecture, Arke forgoes the expense of consensus to achieve scalability while maintaining consistency in a Byzantine fault tolerant environment. Performance evaluations demonstrate that Arke provides enough throughput to support the needs of the most popular messaging applications while maintaining sub-second latencies in a large geo-distributed setting.**

I. Introduction

Contact discovery enables users of social applications, such as messengers, payment systems, or media-sharing platforms, to find and interact with their registered contacts [69]. This process allows bootstrapping social applications on top of an existing social graph, providing immediate value to the application. This is particularly effective when the social graph uses familiar and widely shared *identifiers* such as phone numbers, email addresses or usernames from popular platforms.

Current solutions have significant shortcomings in meeting several important expectations. Some fail to adequately protect users' privacy, exposing their underlying social relations either by design [88], [90] or when targeted by enumeration or crawling attacks [53], [60]. These solutions often rely on centralized parties [32], [59] or trusted hardware for privacy protection [70]. Finally, all these solutions express some form of dependency on the total number of users (either in latency, computation or storage) and may not be suitable for applications with billions of users[1].

Arke[2] is a novel contact discovery scheme that addresses the limitations found in existing systems. Arke ensures the unlinkability of user interactions and effectively mitigates enumeration attacks. It prioritizes user privacy by ensuring that no information about users, their messages, or their communication partners is revealed. Additionally, Arke enforces a bi-directional relationship requirement, meaning that users can only discover each other if they are mutually seeking contact. This approach prevents crawling attacks, setting it apart from traditional contact discovery schemes. Furthermore, Arke supports multiple applications sharing the same contact discovery infrastructure while maintaining independent security assumptions. Notably, Arke represents a significant advancement as the first privacy-preserving contact discovery system whose performance is independent of the total number of users in the system (often referred to as the database size). Moreover, Arke stands out as the first contact discovery system designed without any single points of failure or trust; Arke offers scalability in terms of throughput and extremely low latency despite the presence of a Byzantine adversary.

The Arke contact discovery protocol generalizes the construction of Chaum *et al.* [32], known as *UDM* (User Discovery with Minimal information disclosure). Implicit to the UDM architecture is the fact that a contact discovery scheme can be built by combining a *key exchange* and an *unlinkable handshake*. First, users run a key exchange to establish a shared secret. Then, using this secret, the users run the handshake protocol to establish an end-to-end encrypted channel, without revealing any connection details to third parties. Chaum *et al.* [32] realize both of these subprotocols with the help of centralized parties (the *Public-Key Manager* and *Encrypted ID Manager* respectively). Arke improves on these requirements. The key exchange is instantiated with a variant of the Boneh-Waters identity-based non-interactive key exchange (ID-NIKE) [17]. By utilizing distributed key generation [47] and blind threshold BLS signatures [13], we modify the original protocol to distribute the master secret key and enable oblivious and verifiable key issuance. We then present a custom unlinkable handshake protocol which only requires an untrusted (and potentially distributed) public bulletin board. The design of this handshake ensures that each system resource is mutated by at most a single user, eliminating the need for an expensive consensus protocol to maintain consistency in the distributed setting. Instead, Arke relies on a simpler and more efficient primitive based on Byzantine Consistent Broadcast (BCB) [23].

We implement and evaluate a prototype of Arke written in Rust on Amazon EC2 in a large geo-distributed wide-area network deployment. We show that after a short one-time offline phase taking only a couple of seconds, Arke supports over 1'500 users per second with a latency of less than 0.5 seconds even when the infrastructure is maintained by 50 authorities. Furthermore, Arke can maintain this throughput with sub-second latency even when up to a third of these

[1] WhatsApp, the most popular end-to-end encrypted messaging application, was reported to have 2.7 billion unique active users in June 2023 [28]

[2] In Greek mythology, Arke is the messenger of the Titans.

authorities fail.

Contributions. This paper makes the following contributions:

- It generalizes the UDM construction of Chaum *et al.* [32] and presents Arke — a novel construction that is the first with performance independent of the total number of users in the system, and the first designed to operate in a Byzantine environment. It does so by introducing a threshold and oblivious variant of the Boneh-Waters ID-NIKE [17], as well as a custom unlinkable handshake.
- It proves the security and privacy guarantees of the system (left as an open question in the work of Chaum *et al.* [32]).
- It shows how Arke maintains consistency of a distributed key-value store without requiring consensus but instead using simpler and more efficient broadcast-based primitives.
- It provides a full implementation of Arke and a performance evaluation on a real geo-distributed environment under varying system loads and fault scenarios.
- It shows how existing blockchains can leverage Arke to build a privacy-preserving contact discovery service for their wallets, and how messaging services such as Signal [83], Telegram [87], and WhatsApp [91] can run Arke to allow users to privately discover each other's public keys.

II. System Overview

Arke enables Alice to discover a *message* msg_B from a sender Bob known only by his *identifier* id_B through the establishment of a shared cryptographic secret between them. An identifier is a public human-readable string unique to a user, such as a phone number, an email address, or a social media handle. Arke aims to be efficient and privacy-friendly by hiding the identifiers, messages, and relationships between users.

A. Actors

Arke is composed of the following actors.

Users. A *user*, Alice, owns a human-readable identifier id_A and a message (or payload) msg_A. She wishes to allow specific users to discover her message on the conditions that (i) Alice knows the other user's identifier and (ii) the other user knows Alice's identifier. Users wish to hide their relationships with other users from any observer.

Registration Authorities. A *registration authority* (RA) attests to the binding between users and their identifiers. A registration authority could be a social media service (e.g., Twitter) allowing the use of usernames as identifiers or a messaging service verifying a phone number, or any third party running an interactive protocol with the user to verify their identifiers (e.g., by sending them a text code). Identifiers always specify the registration authority that attested to them. As a result, multiple services (e.g., Signal [83], Telegram [87], WhatsApp [91], or any third-party service) can all use the user's phone number as an identifier by appending their unique RA domain, e.g., `phone_number@domain`. A registration authority can be a single entity or a distributed set of authorities. The concrete deployment structure is decided by the respective service designers/operators. For simplicity of presentation, we assume henceforth that a registration authority is a single entity.

Key-issuing Authorities. The *key-issuing authorities* (KAs) are a committee of n entities that share a threshold key (see Section IV). They are tasked with issuing private keys to users who present a valid proof of registration. Arke assumes that at most t key-issuing authorities are Byzantine (see Section II-D).

Storage Authorities. The Arke storage is operated by a set of $3f + 1$ independent *storage authorities* out of which at most f are Byzantine (see Section II-C). We present the storage authorities as independent entities but they may coincide with the key-issuing authorities (by setting $t = f$) or coincide with the maintainers of most existing blockchains (see Section V-B). In the general setting, storage authorities may enforce their own access control policy and only accept write requests from users registered with RAs of their choice.

B. Protocol Outline

Arke is divided into two phases: *(i)* a *setup phase* where users obtain a long-term private key over their identifier, and *(ii)* a *discovery phase* where users use their private keys to anonymously exchange messages with their contacts over an untrusted public message board. The setup phase is executed only once (or rarely) and the discovery phase is executed every time a user wishes to make her message discoverable or discover the message of a contact. Figure 1 provides an overview of Arke and the interactions between its actors.

Setup phase. Alice convinces a registration authority that she owns the identifier id_A and receives a signed attestation in return (❶). She then blinds her identifier and attestation to submit anonymous key-issuance requests to at least $t + 1$ key-issuing authorities. Upon verifying a request, each key-issuing authority blindly emits a share of Alice's private key. Finally, Alice locally combines the shares to obtain her long-term private key (❷).

Discovery phase. After running the setup phase, Alice wishes to signal to Bob that she has registered and optionally sends him a message. Using her long-term private key and Bob's identifier, Alice locally derives a shared secret with Bob (❸). From this shared secret, Alice can derive a label and a symmetric key used for encryption. She encrypts her message and writes the ciphertext and label to the distributed Arke store (❹). Bob can discover Alice's message by locally deriving the same shared secret (using his long-term private key and Alice's identifier) (❺) and reading the distributed Arke store (❻). Arke divides time in a sequence of epochs (e.g., lasting about 1 or 2 weeks). After a fixed number of epochs, the storage authorities delete the records of inactive users (see Section B-C).

C. Design Goals

Arke guarantees several system security, privacy, and performance properties.

System security properties. Arke maintains several systems security properties depending on which assumptions (Section II-D) hold. These security properties are formally defined and proved in Appendix C.

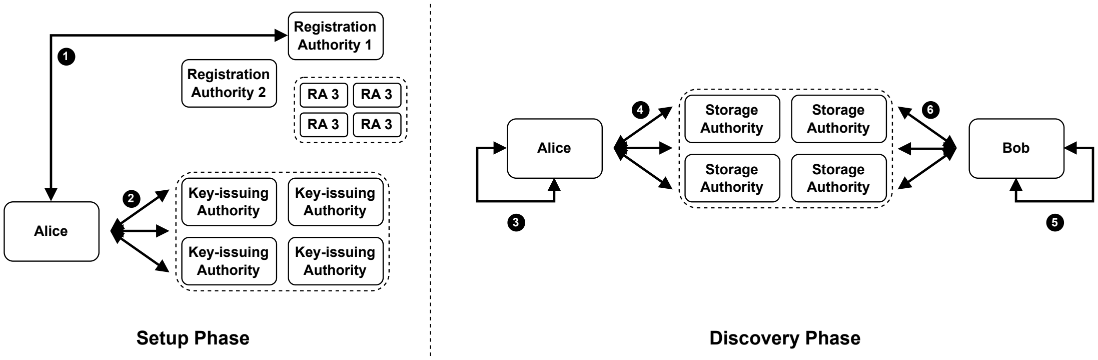

Fig. 1. *Arke* overview. During the setup phase, users run an (anonymous) identification procedure with a registration authority to obtain an attestation over their identifier (❶). They then use this attestation along with their blinded identifier to obtain a long-term private key by interacting with the key-issuing authorities (❷). During the discovery phase, users locally derive a shared secret with each contact (❸,❺) and use it to read and write the Arke distributed store and discover their messages (❹,❻).

- **Validity:** Alice can only update the Arke store by updating messages associated with her identifier id_A.
- **Write consistency:** No correct storage authorities hold conflicting records.
- **Read consistency:** No two read operations over the same label return a different ciphertext.
- **Write termination:** A correct user can eventually update the store to make its message discoverable.
- **Read termination:** A correct user can eventually read the store and learn the message associated with a user with a known identifier.

Privacy properties. Arke upholds the following privacy properties:

- **Anonymity**: The identities of active Arke users are kept hidden from the key-issuing authorities, storage authorities, and any third-party observer. Identities may also be hidden from the relevant registration authority if their authentication mechanism is anonymous. This mechanism is left at the discretion of each registration authority and is out of our design scope.
- **Confidentiality**: Messages exchanged over Arke are encrypted and recipient-anonymous.
- **Unlinkability**: None of the authorities or third-party observers can determine whether Alice and Bob have exchanged messages over Arke.
- **Selective discovery**: Users may choose whether or not to be discoverable by other users on a *per-user* basis. The default behavior is to remain hidden. This property contrasts with other contact discovery schemes where users make themselves discoverable to all, allowing crawling attacks as studied by Hagen *et al.* [53].

Performance properties. Arke also guarantees the following system and performance properties. Section VI demonstrates these properties through a thorough implementation and evaluation of Arke.

- **High-throughput:** Arke provides enough throughput to support multiple applications with billions of users each; we estimate that Arke can support the combined user base of WhatsApp, Facebook Messenger, Signal, and Telegram.
- **Low-latency:** Arke achieves sub-second latency even for large geo-distributed deployments.
- **Performance under (crash-)faults:** The performance (throughput and latency) of Arke is virtually unaffected by (crash-)faulty authorities. Note that evaluating a BFT system while experiencing Byzantine faults is an open research problem [9].
- **Bounded storage**: Storage is not growing linearly over time. Arke enables authorities to periodically purge their store entries. This property is proven as part of *consistency*.

Additional properties. Furthermore, Arke guarantees the following meta-properties:

- **Censorship resistance**: Correct users can always obtain private keys from the key-issuing authorities. Furthermore, correct users can write and read the Arke store despite the presence of Byzantine authorities. This property is proved as part of *write termination* and *read termination*.
- **Authorities Non-Interactivity**: Neither the Arke key-issuing authorities nor the storage authorities need to communicate with each other. This property allows for easier deployment and is crucial to integrate Arke into the Sui blockchain [71] (see Section V-B).

D. Threat Model

We define the main assumptions under which Arke guarantees the properties of Section II-C.

Assumption 1: Correct registration authorities. Arke guarantees the security properties of Section II-C for identifiers attested by correct registration authorities. Indeed, a malicious RA could falsely issue attestations and impersonate any user it desires. Fortunately, recent work on authenticating web data has shown that privacy-preserving, untrusted and correct RAs can be realized in practice [96], [94], [31], [29], [65]. Some of these solutions are under active development at the time of publication of this work [89], [75]. Additionally, Arke mitigates the threat of malicious RAs by confining each RA to a unique domain (see Section II-A and Section IV-C).

Assumption 2: BFT authorities. Arke assumes a computationally bounded adversary that controls the network and can corrupt at most t key-issuing authorities (out of $2t+1$) and up to f (out of $3f+1$) storage authorities in every epoch. We say that authorities corrupted by the adversary are Byzantine or faulty and the rest are honest or correct. Byzantine authorities may act arbitrarily, while correct ones follow the protocol.

Assumption 3: Cryptography. The cryptographic schemes used in Arke assume the existence of a non-degenerate and efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for which the decisional bilinear Diffie-Hellman (DBDH) assumption holds. Hash functions are modeled as random oracles. Finally, we assume the existence of zero-knowledge non-interactive proofs (or arguments) of knowledge for NP relations.

Assumption 4: Network model. To capture real-world networks we assume that links between users and correct authorities are reliable (the authorities do not communicate with each other). That is, all messages among the correct authorities eventually arrive. We assume a known Δ and say that execution of a protocol is eventually synchronous if there is a global stabilization time (GST) after which all messages sent among honest parties are delivered within the network delay Δ time. An execution is synchronous if GST occurs at time 0, and asynchronous if GST never occurs. Arke assumes an eventually synchronous network. Finally, we assume that messages between users and storage authorities are anonymous. In practice, Arke requires that users query the storage via an anonymity network such as Tor, Nym or Loopix.

Assumption 5: Roughly synchronized clocks. Arke assumes that users have roughly synchronized clocks with the correct storage authorities.

Definition 1 (Roughly Synchronized Clocks)**.** *While a user is in epoch Epoch, correct authorities are either in epoch Epoch, $\mathsf{Epoch}-1$, or $\mathsf{Epoch}+1$. Also, users remain in the same epoch of each correct authority for a duration of at least 3Δ (where Δ is the bound on message propagation time during periods of synchrony introduced in assumption 4).*

III. Preliminaries

For a security parameter λ, let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order $q > 2^\lambda$ such that there exists an efficiently computable and non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We denote by g_1, g_2, and g_T the canonical generators of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively, and by $H : \{0,1\}^* \to \{0,1\}^l$, $H_1 : \{0,1\}^* \to \mathbb{G}_1$, and $H_2 : \{0,1\}^* \to \mathbb{G}_2$ hash functions. We treat H, H_1, and H_2 as random oracles.

A. Zero Knowledge Proofs

A zero-knowledge proof of knowledge (ZKPoK) is a tuple of algorithms, or protocols, that prove that an instance x and witness w are in a relation $\mathcal{R}$. Importantly, a ZKPoK allows the prover to prove that it *knows* the secret witness w; as opposed to simply proving the *existence* of the witness.

We make use of two types of ZKPoK. The first proves knowledge of the discrete logarithm of some public value y with respect to the canonical generator g. The second is a zk-SNARK[3] for generic NP relations. Note that although we could use the zk-SNARK to prove the discrete logarithm relation, the resulting protocol would be much more computationally expensive for the prover.

Schnorr DLOG. For a group $\mathbb{G}$ of prime order q, the Schnorr DLOG ZKPoK is a Σ-protocol for the relation

$$\mathcal{R}_{\mathsf{DLOG}} := \{((x,y),\alpha) : y = x^\alpha\}$$

where $x, y \in \mathbb{G}$ and $\alpha \in \mathbb{Z}_q$. It can be compiled into a non-interactive zero-knowledge proof (NIZK) using the Fiat-Shamir transform. We denote the resulting algorithms as:

- $\mathsf{DLOG.Prove}((x,y),\alpha) \to \pi$. Given an instance (x,y) and the corresponding witness α such that $((x,y),\alpha) \in \mathcal{R}$, output a proof π.
- $\mathsf{DLOG.Verify}((x,y),\pi) \to \{0,1\}$. Given the instance (x,y) and proof π, return 1 if the proof is valid and 0 otherwise.

zk-SNARK for Hash Pre-images. A SNARK is defined as a quadruple of algorithms $\Pi_{\mathcal{R}}$:

- $\mathsf{Setup}(\lambda) \to (\mathsf{crs}, \mathsf{td})$. The Setup algorithm produces a *common reference string* crs and a *trapdoor* td.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$. Given the common reference string and an instance-witness pair $(x,w) \in \mathcal{R}$, output a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$. Given the common reference string, instance, and proof, return 1 if the proof is valid and 0 otherwise.
- $\mathsf{Simulate}(\mathsf{crs}, \mathsf{td}, x) \to \pi$. Using the common reference string and the trapdoor, produce a proof for the instance x *without knowledge of a corresponding witness*.

The main security properties of a SNARK are *perfect completeness* and *knowledge soundness*. Perfect completeness states that a prover that knows a valid witness for the instance x will always be able to produce an accepting proof. Knowledge soundness states that if a proof was accepted, then it holds with overwhelming probability that the prover knew a valid witness. A SNARK is said to be *zero-knowledge* if proofs produced by Prove and $\mathsf{Simulate}$ have (almost) identical probability distributions. We use the acronym *zk-SNARK* to specify that a SNARK upholds the zero-knowledge property. We use a zk-SNARK to keep users' identities private while still attesting

[3] Zero-knowledge succinct non-interactive argument of knowledge

that hashed values are correct. Let id be an identifier and $\alpha \in \mathbb{Z}_q$ a blinding factor, we define the relation $\mathcal{R}_{\mathsf{ID}}$ as:

$$\mathcal{R}_{\mathsf{ID}} := \left\{ \left(\widehat{\mathsf{id}}, (\mathsf{id}, \alpha)\right) : \widehat{\mathsf{id}} = (H_1(\mathsf{id})^{\alpha}, H_2(\mathsf{id})^{\alpha}) \right\}$$

For our benchmarks, we instantiate the zk-SNARK for $\mathcal{R}_{\mathsf{ID}}$ using Groth16 [51].

B. Distributed Key Generation

A distributed key generation (DKG) protocol allows n participants to jointly compute shares of a master secret without needing to compute, reconstruct or store this secret. The DKG can be parametrized with respect to a threshold t: any subset of at least $t + 1$ participants can perform actions that would normally require knowledge of the secret key; on the other hand, any smaller subsets cannot.

DKGs and security. Pedersen [77] is the first to propose a DKG scheme. While the Pedersen-DKG is attractive for its efficiency and simplicity, Gennaro *et al.* [47] show that a rushing adversary can influence the probability distribution of the master secret. Such an adversary would gain some a-priori knowledge on the secret key. Consequently, the Pedersen-DKG cannot be used as a stand-in replacement for a generic trusted key generation. Nevertheless, the Pedersen-DKG can be shown secure for certain applications: Gennaro *et al.* [48] demonstrate the unforgeability of Schnorr signatures under the Pedersen-DKG; Gurkan *et al.* [52] show that Pedersen-DKG is *security-preserving* for a large class of protocols, including BLS signatures and El-Gamal encryption. They obtain this latter result by introducing the notions of *key-expressable DKGs* and *rekeyability*, both of which are summarized below. In Section IV, we leverage these notions to show that Arke's cryptographic primitives remain secure when instantiated with efficient but weakly-secure DKGs such as the Pedersen-DKG.

Key-expressable DKGs. The notion of *key-expressability* [52] captures the a-priori knowledge gained by the adversary of Gennaro *et al.* [47]. It describes a weaker security requirement than Gennaro *et al.*'s [47] *correctness* and *secrecy*. A key-expressable DKG does not output a uniformly distributed public key pk_A. Instead it outputs a public key

$$\mathsf{pk} = f(\alpha, \mathsf{pk}_A, \mathsf{pk}_B) = (\mathsf{pk}_A)^{\alpha}\mathsf{pk}_B$$

where $\mathsf{pk}_A = g^{\mathsf{sk}_A}$ for a uniformly distributed sk_A, and α, pk_B are attacker-controlled values. Gurkan *et al.* [52] show that Pedersen-DKG is a key-expressable DKG.

Rekeyability. Informally, a protocol is said to be rekeyable if it is possible to transform objects (ciphertexts, signatures, etc.) that were formed using one set of keys into equivalent objects formed under a related set of keys. For example, a BLS signature under key sk_1, $\sigma = H_1(m)^{\mathsf{sk}_1}$, can be transformed into a signature under the key $\alpha\mathsf{sk}_1 + \mathsf{sk}_2$ by computing $\sigma^{\alpha} \cdot H_1(m)^{\mathsf{sk}_2}$. A full formal definition is given in [52].

C. Identity-Based Non-Interactive Key Exchange

Boneh-Waters ID-NIKE. We give a self-contained definition of the Boneh-Water ID-NIKE [17] adapted for our asymmetric pairing setting.

Definition 2 (Boneh-Waters ID-NIKE [17])**.** *The Boneh-Waters identity-based key exchange consists of three efficiently computable algorithms* Setup, Extract, *and* SharedKey *as follows:*

```
$\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{A}}(\lambda)$
    $b \xleftarrow{\$} \{0,1\}$
    $Q_e \leftarrow \emptyset, Q_k \leftarrow \emptyset$
    $\mathsf{msk} \leftarrow \mathsf{Setup}(\lambda)$
    $O \leftarrow \{O\mathsf{Extract}, O\mathsf{Reveal}\}$
    $(\mathsf{id}_*, \mathsf{id}'_*) \leftarrow \mathcal{A}^O$
    $\gamma \leftarrow \mathsf{Test}(\mathsf{id}_*, \mathsf{id}'_*)$
    $\widehat{b} \leftarrow \mathcal{A}^O(\gamma)$
    if $(\widehat{b} = b) \wedge (\mathsf{id}_* \notin Q_e) \wedge$
       $(\mathsf{id}'_* \notin Q_e) \wedge ((\mathsf{id}_*, \mathsf{id}'_*) \notin Q_k)$
        return 1
    return 0

$O\mathsf{Extract}(\mathsf{id})$
    $sk_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{msk}, \mathsf{id})$
    $Q_e \leftarrow Q_e \cup \{\mathsf{id}\}$
    return $sk_{\mathsf{id}}$

$O\mathsf{Reveal}(\mathsf{id}, \mathsf{id}')$
    $sk_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{msk}, \mathsf{id})$
    $k_{\mathsf{id},\mathsf{id}'} \leftarrow \mathsf{SharedKey}(sk_{\mathsf{id}}, \mathsf{id}')$
    $Q_k \leftarrow Q_k \cup \{(\mathsf{id}, \mathsf{id}'), (\mathsf{id}', \mathsf{id})\}$
    return $k_{\mathsf{id},\mathsf{id}'}$

$\mathsf{Test}(\mathsf{id}_*, \mathsf{id}'_*)$
    if $b = 0$
        $sk_{\mathsf{id}_*} \leftarrow \mathsf{Extract}(\mathsf{msk}, \mathsf{id}_*)$
        $\gamma_0 \leftarrow \mathsf{SharedKey}(sk_{\mathsf{id}_*}, \mathsf{id}'_*)$
    if $b = 1$
        $\gamma_1 \xleftarrow{\$} \mathbb{G}_T$
    return $\gamma_b$
```

Fig. 2. IND-SK security game for ID-NIKEs

- Setup(λ)*: Choose a random* $\mathsf{msk} \xleftarrow{\$} \mathbb{Z}_q$ *and output* msk.
- Extract(msk, id)*: compute* $d_l = H_1(\mathsf{id})^{\mathsf{msk}}$ *and* $d_r = H_2(\mathsf{id})^{\mathsf{msk}}$*. Output* $sk_{\mathsf{id}} = (d_l, d_r)$*.*
- SharedKey$(sk_{\mathsf{id}}, \mathsf{id}')$*: We assume that identifiers are lexicographically ordered. Parse* sk_{id} *as* (d_l, d_r) *and output* $k_{\mathsf{id},\mathsf{id}'}$*:*

$$k_{\mathsf{id},\mathsf{id}'} = \begin{cases} e(d_l, H_2(\mathsf{id}')), & \textit{if } \mathsf{id} < \mathsf{id}' \\ e(H_1(\mathsf{id}'), d_r), & \textit{if } \mathsf{id} > \mathsf{id}' \end{cases}$$

Note that $\mathsf{SharedKey}(sk_{\mathsf{id}}, \mathsf{id}') = \mathsf{SharedKey}(sk'_{\mathsf{id}}, \mathsf{id})$ *for all* $\mathsf{id} \neq \mathsf{id}'$ *and* pp *generated by* Setup*.*

The security notion for such schemes is that of "indistinguishability of shared keys" (IND-SK) and is formalized by Paterson and Srinivasan [76]. In the IND-SK game, an adversary is tasked with distinguishing between the shared key for a pair of identities $(\mathsf{id}_*, \mathsf{id}'_*)$ and a random element from the key space, in this case, $\mathbb{G}_T$. The adversary may request identity keys and shared keys from its oracles. The security game is formalized in Figure 2.

We say that an ID-NIKE scheme Σ is IND-SK secure if for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Boneh and Waters [17] show that the ID-NIKE of Definition 2 is secure under the decision bilinear Diffie-Hellman (DBDH) assumption in the random oracle model.

D. Authenticated Encryption with Associated Data (AEAD)

Authenticated Encryption with Associated Data (AEAD) is a symmetric key primitive that encrypts and authenticates a message. Senders may choose to associate context data to the ciphertext in a cryptographically binding way. An AEAD scheme is defined by the following algorithms:

- $\mathsf{AEAD.Enc}_k(m, d) \rightarrow (c, \mathsf{tag})$. Given a key k, message m, and associated data d, encrypt m to produce the ciphertext c. Authenticate the associated data and ciphertext to produce a tag tag. Output (c, tag).
- $\mathsf{AEAD.Dec}_k(c, \mathsf{tag}) \rightarrow m'$. Given a key k, ciphertext c, and associated data tag, verify the authenticity of the associated data and ciphertext. If the verification rejects, output $m' \leftarrow \perp$. Otherwise decrypt c and output $m' \leftarrow m$.

IV. The Arke Contact Discovery Protocol

The Arke contact discovery protocol combines an ID-NIKE scheme with an unlinkable handshake. The ID-NIKE allows users to establish shared secrets amongst each other knowing only their (potentially low-entropy) identifiers. Using this shared secret, they can run the unlinkable handshake to exchange arbitrary messages through an untrusted key-value store. We describe a private and trust-minimized variant of the Boneh-Waters ID-NIKE (Section IV-A), and an unlinkable handshake protocol (Section IV-B), and show how to combine both to build a contact discovery protocol (Section IV-C).

A. Threshold Oblivious ID-NIKE

The ID-NIKE defined by Boneh and Waters [17] relies on a trusted third party to issue private keys to users. We modify their protocol to meet our privacy desiderata by *(i)* allowing users to verify the private keys they are issued, *(ii)* separating the key issuance operation into a registration and an extraction phase and, *(iii)* distributing the master secret key. We achieve modifications *(i)* and *(iii)* by applying techniques outlined by Boneh and Franklin [15]; we achieve modification *(ii)* by improving upon the result of Sui *et al.* [86]. We refer to the resulting protocol as a threshold and oblivious ID-NIKE.

Verifiable key issuance. One way to hold the trusted third party accountable is to allow other parties in the system to verify the issuance of private keys. To this effect, we modify the Setup algorithm to output a master public key mpk and introduce the VerifyPK and VerifyExtract algorithms:

- $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{msk}, \mathsf{mpk})$: choose a random $\mathsf{msk} \xleftarrow{\$} \mathbb{Z}_q$ and compute the corresponding public key $\mathsf{mpk} = (g_1^{\mathsf{msk}}, g_2^{\mathsf{msk}})$. Output msk and mpk.
- $\mathsf{VerifyPK}(\mathsf{pk}) \rightarrow \{0, 1\}$: parse pk as $(\mathsf{pk}_l, \mathsf{pk}_r)$. If $e(\mathsf{pk}_l, g_2) = e(g_1, \mathsf{pk}_r)$, output 1 (accept). Otherwise output 0 (reject).
- $\mathsf{VerifyExtract}(\mathsf{mpk}, \mathsf{id}, \theta) \rightarrow \{0, 1\}$: parse mpk as $(\mathsf{mpk}_l, \mathsf{mpk}_r)$ and θ as $(\theta_l, \theta_r) \in \mathbb{G}_1 \times \mathbb{G}_2$. If $e(\theta_l, g_2) = e(H_1(\mathsf{id}), \mathsf{mpk}_r)$ and $e(g_1, \theta_r) = e(\mathsf{mpk}_l, H_2(\mathsf{id}))$, output 1 (accept). Otherwise, output 0 (reject).

The VerifyPK algorithm enforces that the terms in the pk tuple are equal to the generators g_1 and g_2 taken to the *same* power. Indeed, consider $\mathsf{pk} = ({g_1}^x, {g_2}^y)$ for $x, y \in \mathbb{Z}_q$. We use the non-degenerate and the bilinear properties to show the following equivalence:

$$\begin{aligned} 1 \leftarrow \mathsf{VerifyPK}(\mathsf{pk}) &\iff e({g_1}^x, g_2) = e(g_1, {g_2}^y) \\ &\iff e(g_1, g_2)^x = e(g_1, g_2)^y \\ &\iff x = y \end{aligned} \quad (1)$$

As shown by Boneh and Franklin [15], albeit in a different setting, VerifyExtract accepts if and only if the input θ is equal to the expected private key. Given the public key $\mathsf{pk} = ({g_1}^x, {g_2}^y) = ({g_1}^x, {g_2}^x)$ from above, we can write:

$$\begin{aligned} &1 \leftarrow \mathsf{VerifyExtract}(\mathsf{pk}, \mathsf{id}, \theta) \\ &\iff \begin{cases} e(\theta_l, g_2) = e(H_1(\mathsf{id}), {g_2}^x) \\ e(g_1, \theta_r) = e({g_1}^x, H_2(\mathsf{id})) \end{cases} \\ &\iff (\theta_l, \theta_r) = (H_1(\mathsf{id})^x, H_2(\mathsf{id})^x) = \mathsf{Extract}(x, \mathsf{id}) \end{aligned} \quad (2)$$

Oblivious key issuance. In the Boneh-Waters ID-NIKE, users must reveal their identifier to a trusted third party to obtain their secret key. We follow the approach of Sui *et al.* [86] and separate this trusted party into two entities: a registration authority and a key-issuing authority. We allow the registration authority to learn identifiers but not to compute their private keys. Its role is to attest that a user A owns the identifier id_A. On the other hand, the key-issuing authority is able to produce private keys but does not learn which identities have requested keys.

To this effect, we introduce a setup algorithm for the registration authority, Setup_R, and replace the Extract algorithm by five efficiently computable algorithms Register, Blind, VerifyID, BlindExtract and Unblind:

- Setup_R: Produces private and public parameters for a registration authority.
- Register: Upon valid authentication, a registration authority produces a signature attesting that user A owns the identifier id_A.
- Blind: Produce a masked version of an identifier and its corresponding registration signature. The blinded identifier and signature are accompanied by optional proof of their validity.
- VerifyID: Verify that a valid registration signature was issued for a blinded identifier.
- BlindExtract: Given a blinded identifier, produce the corresponding blinded secret key.
- Unblind: Recover an identifier's secret key from a blinded secret key.

We give a concrete construction of an oblivious ID-NIKE in Appendix A. Our construction can be seen as an improvement over that of Sui *et al.* [86]. Indeed, while their approach succeeds in blinding the extracted secret key, it fails to provide anonymity from the key-issuing authority. Furthermore, their one-time password mechanism requires that the key-issuing authority maintain a list of registered users.

Distributed key issuance. In the oblivious setting described above, the key-issuing authority is still all-powerful in that it is able to extract the private key of any identifier. To minimize the trust placed in the key-issuing authority, we distribute it into n entities that each hold a share of the master secret key (a widely popular approach, suggested in [15] amongst many other works). Using a (t, n)-threshold DKG, we ensure that the ID-NIKE remains IND-SK secure when no more than t parties are malicious. As discussed in Section III-B, we do not require the strong security properties of Gennaro *et al.* [47], and instead rely on the weaker requirement of a key-expressable DKG [52]. Doing so allows us to instantiate our scheme using the efficient Pedersen-DKG [77].

We distribute the key-issuing authority by replacing the Setup_E algorithm with a key-expressable DKG [52]. The extraction algorithm is the same as $\mathsf{BlindExtract}$ but is renamed to $\mathsf{BlindPartialExtract}$ to emphasize the fact that it outputs blinded *partial* secret keys. Similarly, the verification of a partial private key is identical to $\mathsf{VerifyExtract}$ but is renamed to $\mathsf{VerifyPartialExtract}$. Finally, we introduce the $\mathsf{Combine}$ algorithm to reconstruct a secret key from a set of $t + 1$ key shares.

Definition 3 (Threshold and Oblivious ID-NIKE). *Let* Π_{ID} *be a knowledge sound zk-SNARK for the relation* $\mathcal{R}_{\mathsf{ID}}$. *We define the* (t, n)*-threshold variant of the oblivious Boneh-Waters ID-NIKE as follows:*

- $\mathbf{SetupDKG}_{\boldsymbol{E}}(\boldsymbol{\lambda}, \boldsymbol{t}, \boldsymbol{n}) \rightarrow (\mathbf{msk}_1, \ldots, \mathbf{msk}_{\boldsymbol{n}}, \mathbf{pp})$. *All* n *participants* $P_1, \ldots, P_n$ *jointly execute a key-expressable DKG to compute Shamir secret shares* $\mathsf{msk}_1, \ldots, \mathsf{msk}_n$ *of an (unknown) master secret key* msk. *They jointly output a* $\mathsf{transcript}$, *a set of partial public keys* $\{\mathsf{mpk}_i = (g_1^{\mathsf{msk}_i}, g_2^{\mathsf{msk}_i})\}_{i=1}^{n}$ *and master public key* $\mathsf{mpk} = (g_1^{\mathsf{msk}}, g_2^{\mathsf{msk}})$. *Output* msk_i *to* P_i *and* $\mathsf{pp} \leftarrow (\mathsf{transcript}, \mathsf{mpk})$.

- $\mathbf{Setup}_{\boldsymbol{R}}(\boldsymbol{\lambda}, \mathbf{pp}) \rightarrow (\mathbf{rsk}, \mathbf{pp})$. *Choose a random registration secret key* $\mathsf{rsk} \xleftarrow{\$} \mathbb{Z}_q$ *and compute the registration public key* $\mathsf{rpk} = (g_1^{\mathsf{rsk}}, g_2^{\mathsf{rsk}})$. *Output* rsk *and* $\mathsf{pp} \leftarrow \mathsf{pp}||\mathsf{rpk}$.

- $\mathbf{VerifyPK}(\mathbf{pk}) \rightarrow \{\mathbf{0}, \mathbf{1}\}$. *Parse* pk *as* $(\mathsf{pk}_l, \mathsf{pk}_r)$. *If* $e(\mathsf{pk}_l, g_2) = e(g_1, \mathsf{pk}_r)$, *output* 1 *(accept). Otherwise output* 0 *(reject).*

- $\mathbf{Register}(\mathbf{rsk}, \mathbf{id}) \rightarrow \boldsymbol{\tau}_{\mathbf{id}}$. *Compute* $\tau_l = H_1(\mathsf{id})^{\mathsf{rsk}}$ *and* $\tau_r = H_2(\mathsf{id})^{\mathsf{rsk}}$. *Output the registration signature* $\tau_{\mathsf{id}} = (\tau_l, \tau_r)$.

- $\mathbf{Blind}(\mathbf{pp}, \mathbf{id}, \boldsymbol{\tau}_{\mathbf{id}}) \rightarrow (\boldsymbol{\alpha}, \widehat{\mathbf{id}}, \widehat{\boldsymbol{\tau}_{\mathbf{id}}}, \boldsymbol{\pi})$. *Sample a random blinding factor* $\alpha \xleftarrow{\$} \mathbb{Z}_q$. *Compute*

$$\begin{aligned} \widehat{\mathsf{id}} &= (H_1(\mathsf{id})^{\alpha}, H_2(\mathsf{id})^{\alpha}) \\ \pi &= \Pi_{\mathsf{ID}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{ZK}}, \widehat{\mathsf{id}}, (\mathsf{id}, \alpha)) \\ \widehat{\tau_{\mathsf{id}}} &= {\tau_{\mathsf{id}}}^{\alpha} \end{aligned} \tag{3}$$

  Output the blinding factor α, *blind identifier* $\widehat{\mathsf{id}}$, *blind registration signature* $\widehat{\tau_{\mathsf{id}}}$ *and the blinding proof* π.

- $\mathbf{VerifyID}(\mathbf{pp}, \widehat{\mathbf{id}}, \widehat{\boldsymbol{\tau}_{\mathbf{id}}}, \boldsymbol{\pi}) \rightarrow \{\mathbf{0}, \mathbf{1}\}$. *Parse* rpk *as* (pk_l, pk_r), $\widehat{\mathsf{id}}$ *as* $(\widehat{\mathsf{id}}_l, \widehat{\mathsf{id}}_r)$, *and* $\widehat{\tau_{\mathsf{id}}}$ *as* $(\widehat{\tau}_l, \widehat{\tau}_r)$. *Check that the following equations hold:*

$$\begin{aligned} e(\widehat{\tau}_l, g_2) &\overset{?}{=} e\left(\widehat{\mathsf{id}}_l, pk_r\right) \\ e(g_1, \widehat{\tau}_r) &\overset{?}{=} e\left(pk_l, \widehat{\mathsf{id}}_r\right) \end{aligned} \tag{4}$$

$$\Pi_{\mathsf{ID}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{ZK}}, \mathsf{ID}, \pi_{\mathsf{ID}}) \overset{?}{=} 1 \quad \textit{(accept)}$$

  If all equations verify successfully output 1, *otherwise output* 0.

- $\mathbf{BlindPartialExtract}(\mathbf{msk}_{\boldsymbol{i}}, \widehat{\mathbf{id}}) \rightarrow \widehat{\boldsymbol{sk}_{\mathbf{id},\boldsymbol{i}}}$. *Compute and output the blind secret key share* $\widehat{sk_{\mathsf{id},i}} = \widehat{\mathsf{id}}^{\mathsf{msk}_i}$.

- $\mathbf{Unblind}(\widehat{\boldsymbol{sk}_{\mathbf{id},\boldsymbol{i}}}, \boldsymbol{\alpha}) \rightarrow \boldsymbol{sk}_{\mathbf{id},\boldsymbol{i}}$. *Compute and output the partial key* $sk_{\mathsf{id},i} = \widehat{sk_{\mathsf{id},i}}^{\frac{1}{\alpha}}$.

- $\mathbf{VerifyPartialExtract}(\mathbf{mpk}_{\boldsymbol{i}}, \mathbf{id}, \boldsymbol{\theta})$. *Parse* mpk_i *as* $(\mathsf{mpk}_{i,l}, \mathsf{mpk}_{i,r}) \in \mathbb{G}_1 \times \mathbb{G}_2$ *and* θ *as* $(\theta_l, \theta_r) \in \mathbb{G}_1 \times \mathbb{G}_2$. *If* $e(\theta_l, g_2) = e\left(H_1(\mathsf{id}), \mathsf{mpk}_{i,r}\right)$ *and* $e(g_1, \theta_r) = e\left(\mathsf{mpk}_{i,l}, H_2(\mathsf{id})\right)$, *output* 1 *(accept). Otherwise, output* 0 *(reject).*

- $\mathbf{Combine}(\{\boldsymbol{sk}_{\mathbf{id},\boldsymbol{i}}\}_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{t}+\mathbf{1}}) \rightarrow \boldsymbol{sk}_{\mathbf{id}}$. *Using a set of* $t+1$ *valid partial keys, compute* d_l *and* d_r *using Lagrange interpolation "in the exponent". Let* L_i *denote the Lagrange coefficient for the* i*-th share in the given set,* $d_l = \prod_{i=1}^{t+1} {d_{l,i}}^{L_i}$ *and* $d_r = \prod_{i=1}^{t+1} {d_{r,i}}^{L_i}$. [4] *Output the user key* $sk_{\mathsf{id}} = (d_l, d_r)$.

- $\mathbf{SharedKey}(\boldsymbol{sk}_{\mathbf{id}}, \mathbf{id}') \rightarrow \boldsymbol{k}_{\mathbf{id},\mathbf{id}'}$. *We assume that identifiers are lexicographically ordered. Parse* sk_{id} *as* (d_l, d_r) *and output* $k_{\mathsf{id},\mathsf{id}'}$*:*

$$k_{\mathsf{id},\mathsf{id}'} = \begin{cases} e(d_l, H_2(\mathsf{id}')), & \textit{if } \mathsf{id} < \mathsf{id}' \\ e(H_1(\mathsf{id}'), d_r), & \textit{if } \mathsf{id} > \mathsf{id}' \end{cases}$$

For all $\mathsf{id} \neq \mathsf{id}'$ *and* pp *generated by* $\mathsf{SetupDKG}_E$, *it holds that* $\mathsf{SharedKey}(\mathsf{pp}, sk_{\mathsf{id}}, \mathsf{id}') = \mathsf{SharedKey}(\mathsf{pp}, sk'_{\mathsf{id}}, \mathsf{id})$.

IND-SK security. We show that the threshold and oblivious ID-NIKE described here is IND-SK secure under the DBDH assumption in the random oracle model if Π_{ID} is a knowledge sound SNARK for $\mathcal{R}_{\mathsf{ID}}$.

Theorem 1. *The threshold and oblivious ID-NIKE of Definition 3 is IND-SK under the DBDH assumption when modeling the functions* H_1, H_2 *as random oracles, and if* Π_{ID} *is a knowledge sound SNARK for* $\mathcal{R}_{\mathsf{ID}}$.

Proof intuition. We provide intuition for the proof of Theorem 1; a full proof is presented in Appendix A. The proof follows from three lemmas:

- Lemma 1 shows that the (centralized) oblivious variant of the Boneh-Waters ID-NIKE is IND-SK secure under the same assumptions as the Boneh-Waters ID-NIKE if Π_{ID} is a knowledge sound SNARK for $\mathcal{R}_{\mathsf{ID}}$.
- Lemma 2 shows that the oblivious variant of the Boneh-Waters ID-NIKE is rekeyable with respect to the master secret key msk.

[4] As required, $d_l = \prod_{i=1}^{t+1} {d_{l,i}}^{L_i} = H_1(\mathsf{id})^{\sum_{i=1}^{t+1} \mathsf{msk}_{E,i} L_i} = H_1(\mathsf{id})^{\mathsf{msk}_E}$ and analogously for d_r.

- Lemma 3 shows that key-expressable DKGs are security preserving for rekeyable oblivious ID-NIKEs.

Combining the three lemmas, we show that one can replace the Setup_E algorithm of the oblivious variant of the Boneh-Waters ID-NIKE with a key-expressable DKG to obtain an IND-SK secure threshold and oblivious ID-NIKE.

We prove Lemma 1 by showing a reduction from the classic IND-SK security game to the oblivious IND-SK game. In a nutshell, the adversary performing the reduction takes on the role of the registration authority. It samples a registration key and can naturally answer the inner adversary's Register queries. To answer BlindExtract oracle queries, the reduction must first "unblind" the queried identifier. This is done by running the extractor for Π_{ID}. We show that this reduction strategy has an overwhelming success probability if Π_{ID} is a knowledge sound SNARK for $\mathcal{R}_{\mathsf{ID}}$.

We prove Lemma 2 in the same way Gurkan *et al.* [52] show the rekeyability of BLS signatures. Indeed, private keys are very similar in their structure to BLS signatures.

Finally, we prove Lemma 3 by showing a reduction from the IND-SK security of threshold and oblivious ID-NIKEs to that of oblivious ID-NIKEs. The reduction takes advantage of the key-expressability of the DKG to "convert" private keys and shared keys from the centralized setting to equivalent keys in the distributed setting.

Anonymity from the key-issuing authorities. Identifiers are kept hidden from the key-issuing authorities if Π_{ID} is a zero-knowledge SNARK for $\mathcal{R}_{\mathsf{ID}}$. We prove this claim by showing the existence of an algorithm SimulateID that *does not know an identifier* yet produces tuples $(\widehat{\mathsf{id}}_{\mathsf{sim}}, \widehat{\tau}_{\mathsf{sim}}, \pi_{\mathsf{sim}})$ which are statistically indistinguishable from tuples $(\widehat{\mathsf{id}}, \widehat{\tau_{\mathsf{id}}}, \pi)$ produced by an honest prover running Blind [35].

- $\mathbf{SimulateID(crs, td)} \rightarrow (\widehat{\mathbf{id}}_{\mathbf{sim}}, \widehat{\boldsymbol{\tau}}_{\mathbf{sim}}, \boldsymbol{\pi}_{\mathbf{sim}})$. Sample $\widehat{\tau}_{\mathsf{sim}} \xleftarrow{\$} \mathbb{G}_1 \times \mathbb{G}_2$ and compute:

$$\widehat{\mathsf{id}}_{\mathsf{sim}} = \widehat{\tau}_{\mathsf{sim}} \circ \mathsf{rpk}^{-1}$$
$$\pi_{\mathsf{sim}} = \Pi_{\mathsf{ID}}.\mathsf{Simulate}(\mathsf{crs}, \mathsf{td}, \widehat{\mathsf{id}}_{\mathsf{sim}})$$

By construction, the tuple $(\widehat{\mathsf{id}}_{\mathsf{sim}}, \widehat{\tau}_{\mathsf{sim}}, \pi_{\mathsf{sim}})$ satisfies the checks of VerifyID. Furthermore, since the blinding factors are sampled uniformly from $\mathbb{Z}_q$, then $(\widehat{\mathsf{id}}_{\mathsf{sim}}, \widehat{\tau}_{\mathsf{sim}})$ follow the same probability distribution as $(\widehat{\mathsf{id}}, \widehat{\tau_{\mathsf{id}}})$. Finally, by the zero-knowledge property of Π_{ID}, it holds that π_{sim} is statistically indistinguishable from π.

B. Unlinkable Handshake

Performing an identity-based key exchange only addresses half of the contact discovery problem. Users must also exchange an initial message (or flag) in a privacy-preserving way without prior knowledge of each other's network addresses. We present an unlinkable handshake protocol over a public, untrusted message board. We use the message board as a key-value store. In this section, we treat the store as a black box; Section V shows how to efficiently instantiate such storage with minimal trust assumptions and no single point of failure.

Overview. Using their shared ID-NIKE key, Alice and Bob each locally derive a "write tag", a "read tag" and an AEAD encryption key. They use the AEAD encryption key to encrypt their messages and post the resulting ciphertexts on the message board at a unique location derived from their "write tag". We allow all users and network observers to read from the store. However, only users that know read tags destined for them and the corresponding encryption key will be able to recover messages.

Definition 4. *Let $\mathbb{G}$ be an abelian group of prime order p with canonical generator g. Let* DLOG *be a non-interactive instantiation of the Schnorr proof of discrete logarithm compiled using the Fiat-Shamir heuristic. We use a variant of the proof where an extra "context" nonce is added to the transcript. This nonce will be used to bind a proof to a specific session between the message board and a user, thus preventing replay attacks. We denote $\pi_x^{(r)}$ as a proof of knowledge of the secret exponent x during session r.*

Let AEAD *be an IND-CCA secure authenticated encryption with associated data scheme. We denote $\mathcal{K}$ the set of accepted keys for this scheme and $\mathcal{C}$ the set of ciphertexts.*

The handshake is parametrized by two functions, a key derivation function $\mathsf{KDF} : \{0,1\}^* \rightarrow \mathcal{K}$ *and a* tag derivation function $\mathsf{TDF} : \{0,1\}^* \times \{0,1\} \rightarrow \mathbb{Z}_p$. *Assuming that every pair of users A and B have derived a shared secret s_{AB}, the unlinkable handshake is defined as:*

- $\mathbf{Write}^{(r)}(\boldsymbol{s_{AB}}, \mathbf{id}_{\boldsymbol{A}}, \mathbf{id}_{\boldsymbol{B}}, \boldsymbol{m}) \rightarrow (\mathbf{loc}_{\boldsymbol{w}}, \boldsymbol{\pi}_{\boldsymbol{w}}^{(\boldsymbol{r})}, \boldsymbol{c})$. *Compute a symmetric key $k = \mathsf{KDF}(s_{AB})$ and tag t_w such that:*

$$t_w = \begin{cases} \mathsf{TDF}(s_{AB}, 0), & \textit{if } \mathsf{id}_A < \mathsf{id}_B \\ \mathsf{TDF}(s_{AB}, 1), & \textit{if } \mathsf{id}_A > \mathsf{id}_B \end{cases}$$

Compute $\mathsf{loc}_w = g^{t_w}$. Using the derived key and tag, compute the ciphertext $c = \mathsf{AEAD.Enc}_k(g^{t_w}, m)$. Finally, for the current session r, compute the proof $\pi_{t_w}^{(r)} = \mathsf{DLOG.Prove}((g, loc_w), t_w, r)$. Output $(\mathsf{loc}_w, \pi_w^{(r)}, c)$.

- $\mathbf{VerifyWrite}^{(r)}(\mathbf{loc}_{\boldsymbol{w}}, \boldsymbol{\pi}_{\boldsymbol{w}}^{(\boldsymbol{r})}) \rightarrow \{\mathbf{0}, \mathbf{1}\}$. *Compute and output $b = \mathsf{DLOG.Verify}(\mathsf{loc}_w, \pi_w^{(r)}, r)$.*

- $\mathbf{Read}(\boldsymbol{s_{AB}}, \mathbf{id}_{\boldsymbol{A}}, \mathbf{id}_{\boldsymbol{B}}) \rightarrow \boldsymbol{m}$. *Compute a symmetric key $k = \mathsf{KDF}(s_{AB})$ and tag t_r such that:*

$$t_r = \begin{cases} \mathsf{TDF}(s_{AB}, 1), & \textit{if } \mathsf{id}_A < \mathsf{id}_B \\ \mathsf{TDF}(s_{AB}, 0), & \textit{if } \mathsf{id}_A > \mathsf{id}_B \end{cases}$$

Compute $\mathsf{loc}_r = g^{t_r}$. Retrieve the value c' associated with location loc_r in the store. Compute $m = \mathsf{AEAD.Dec}_k(c', \mathsf{loc}_r)$.

Importantly, A and B can derive the same AEAD symmetric key. Furthermore, A's read tag matches the definition of B's write tag (and conversely).

The handshake is said to be complete when a pair of users have both performed the Write *and* Read *operations. Let t_A, c_A and t_B, c_B be the write tags and ciphertexts derived by A and B respectively, we define the transcript of a completed handshake as:*

$$\mathsf{tr} \leftarrow (r, r', g^{t_A}, g^{t_B}, \pi_{t_A}^{(r)}, \pi_{t_B}^{(r')}, c_A, c_B)$$

Confidentiality. The handshake described above can be shown to preserve the confidentiality of the underlying messages. Indeed if KDF is a secure pseudorandom function, then the derived symmetric key k_{AB} is indistinguishable from random. This in turn allows us to uphold the IND-CCA property of the AEAD scheme.

Unlinkability. To meet our privacy goals, we need to ensure that observing a transcript does not leak information about the identities of the users that generated it. This property should still hold even if the adversary controls all other identities identities and is successful in completing handshakes with each of the target users. Furthermore, we assume that each identity has a *fixed* message that it tries to communicate.

We capture this security notion by defining an *unlinkability* game (see Figure 3). An adversary $\mathcal{A}$ is tasked with distinguishing between a transcript for the pair of identities $\mathsf{id}_*, \mathsf{id}'_*$ and a random transcript. The adversary is allowed to query any shared secret or valid transcripts, and may even complete valid handshakes with both of the target identities.

We say that a handshake HS is unlinkable if for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{Unlinkability}}_{\mathsf{HS},\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Theorem 2. *The handshake presented in Definition 4 is unlinkable if shared secrets between users are established using an IND-SK secure ID-NIKE.*

Proof (Theorem 2): Let Σ denote a secure ID-NIKE. Assume for the sake of contradiction that there exists an adversary $\mathcal{A}$ for which $\Pr\left[\mathsf{Exp}^{\mathsf{Unlinkability}}_{\mathsf{HS},\mathcal{A}}(\lambda) = 1\right] > \frac{1}{2} + \mathsf{negl}(\lambda)$.

We construct an adversary $\mathcal{B}$ that runs $\mathcal{A}$ as a sub-routine against the IND-SK game (Figure 2). Let T_M be a table mapping identifiers to messages. T_M is initialized as the empty table. $\mathcal{B}$ simulates any call to the function M (line 3 of $\mathcal{O}\mathsf{Transcript}$ and line 6 of Test) by running the following $\mathsf{SimMessage}$ routine: if $\mathsf{id} \in T_M$, return $T_M[\mathsf{id}]$; else, $m \xleftarrow{\$} \mathcal{M}$, write $T_M[\mathsf{id}] \leftarrow m$ and return m. $\mathcal{B}$ simulates $\mathcal{A}$'s oracles as follows:

- $\mathcal{O}\mathsf{Secret}$: replace line 1 of the $\mathcal{O}\mathsf{Secret}$ procedure by a call to $\mathcal{O}\mathsf{Reveal}$.
- $\mathcal{O}\mathsf{Transcript}$: replace line 1 of the $\mathcal{O}\mathsf{Transcript}$ procedure by a call to $\mathcal{O}\mathsf{Reveal}$. Replace line 3 with a call to $\mathsf{SimMessage}$.
- Test: $\mathcal{B}$ returns the same identity pair $\mathsf{id}_*, \mathsf{id}'_*$ that $\mathcal{A}$ outputs (line 4 of the game's code) and receives the value γ. Call $\mathsf{SimMessage}$ for each of the provided identities. Perform the loop of lines 7 and 8 of the test procedure replacing s by γ.

Notice that after all of $\mathcal{A}$'s queries, it holds that the exclusion sets of both games are equal. Indeed every update to Q generated the same update to Q_k and no queries were made to $\mathcal{B}$'s $\mathcal{O}\mathsf{Extract}$ oracle. Therefore, $Q_e = \emptyset$. Furthermore, by definition of $\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}(\lambda)$, s and γ follow the same distribution. Therefore:

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}(\lambda) = 1\right] = \Pr\left[\mathsf{Exp}^{\mathsf{Unlinkability}}_{\mathsf{HS},\mathcal{A}}(\lambda) = 1\right]$$

We have shown that $\mathcal{B}$ gains a non-negligible advantage in the IND-SK game against the secure ID-NIKE Σ, therefore reaching a contradiction. Thus, for a secure ID-NIKE scheme Σ there exists no PPT adversary $\mathcal{A}$ such that $\Pr\left[\mathsf{Exp}^{\mathsf{Unlinkability}}_{\mathsf{HS},\mathcal{A}}(\lambda) = 1\right] > \frac{1}{2} + \mathsf{negl}(\lambda)$. Therefore, HS is an unlinkable handshake. ■

Bilateral handshake. An important property of our handshake is that it is *bilateral*: each user may choose to participate or withhold from performing the handshake with a given user. In that sense, the adversary in the unlinkability game is stronger than most real-world adversaries. Indeed in the unlinkability game, the adversary may coerce any user into performing the handshake with her. In practice, the bilateral property of the handshake protects our system from "crawling attacks" as studied by Hagen *et al.* [53].

Overwrite protection. If TDF is a collision-resistant hash function (CRHF) then write and read tags may only be derived by users that know the relevant shared seed (except for a very unlikely collision). This in turn implies that only users that know a shared seed are able to produce a valid ZKPoK for the relevant tag. Thus verifying the ZKPoK in the Write protocol enforces *access control* for a given write location.

Bounded storage. Unfortunately, this access control is not enough to prevent a malicious user from filling up the message board with fake messages. This adversary can pick random tag values and produce valid proofs for those. The mitigation strategy depends on the nature of the store authorities.

Protecting our custom-built store authorities (Section V) against those attacks requires the introduction of a privacy-preserving rate-limiting mechanism. Users are allowed a fixed number of store writes per epoch; any further attempts to write should either require a re-authentication from the user or be prevented and optionally incur some form of punishment. Such a mechanism can be implemented using PrivacyPass [43]: users (or their client-side software) periodically authenticate to their registration authority and request PrivacyPass tokens which they can later redeem at each store write. PrivacyPass has the advantage of using lightweight cryptography and is in the process of being standardized by the IETF. On the other hand, it does not allow to identify cheaters as would be the case with more cryptography-intensive approaches [24], [79].

If the store authorities coincide with the maintainers of an existing blockchain (Section V-B), the native token required to pay for the blockchains' gas cost effectively acts as a rate-limiting mechanism. As a result, Arke does not need to introduce any new access control mechanism.

C. Contact Discovery

Let $\mathcal{R}_{\mathsf{domID}}$ be a variant of $\mathcal{R}_{\mathsf{ID}}$ where part of the hash functions' input is public:

$$\mathcal{R}_{\mathsf{domID}} := \left\{ \begin{array}{l} \left((\widehat{\mathsf{id}}, \mathsf{dom}), (\mathsf{id}, \alpha)\right) : \\ \widehat{\mathsf{id}} = (H_1(\mathsf{id}||\mathsf{dom})^\alpha, H_2(\mathsf{id}||\mathsf{dom})^\alpha) \end{array} \right\}$$

Let ID-NIKE designate the threshold and oblivious ID-NIKE of Definition 3 where Π_{ID} is replaced with a proof Π_{domID} for $\mathcal{R}_{\mathsf{domID}}$, and HS designate the unlinkable handshake of Definition 4.

```
$\mathsf{Exp}^{\mathsf{Unlinkability}}_{\mathsf{HS},\mathcal{A}}(\lambda)$
  $b \xleftarrow{\$} \{0,1\}$
  $Q \leftarrow \emptyset$
  $O \leftarrow \{O\mathsf{Transcript}, O\mathsf{Secret}\}$
  $(\mathsf{id}_*, \mathsf{id}'_*) \leftarrow \mathcal{A}^O$
  $\mathsf{tr}_* \leftarrow \mathsf{Test}(\mathsf{id}_*, \mathsf{id}'_*)$
  $\widehat{b} \leftarrow \mathcal{A}^O(\mathsf{tr}_*)$
  if $(\widehat{b} = b) \wedge ((\mathsf{id}_*, \mathsf{id}'_*) \notin Q)$
    return $1$
  return $0$
```

```
$O\mathsf{Secret}(\mathsf{id}, \mathsf{id}')$
  $s \leftarrow S(\mathsf{id}, \mathsf{id}')$
  $Q \leftarrow Q \cup \{(\mathsf{id}, \mathsf{id}'), (\mathsf{id}', \mathsf{id})\}$
  return $s$

$O\mathsf{Transcript}(\mathsf{id}_1, \mathsf{id}_2)$
  $s \leftarrow S(\mathsf{id}_1, \mathsf{id}_2)$
  for $i = 1..2$ do
    $m_i \leftarrow M(\mathsf{id}_i)$
    $(\mathsf{loc}_i, \pi_i, c_i) \leftarrow \mathsf{Write}^{(r_i)}(s, \mathsf{id}_1, \mathsf{id}_2, m_i)$
  $Q \leftarrow Q \cup \{(\mathsf{id}_1, \mathsf{id}_2), (\mathsf{id}_2, \mathsf{id}_1)\}$
  return $(r_1, r_2, \mathsf{loc}_1, \mathsf{loc}_2, \pi_1, \pi_2, c_1, c_2)$
```

```
$\mathsf{Test}(\mathsf{id}_1, \mathsf{id}_2)$
  if $b = 0$
    $s \leftarrow S(\mathsf{id}_1, \mathsf{id}_2)$
    $m_1 \leftarrow M(\mathsf{id}_1), m_2 \leftarrow M(\mathsf{id}_2)$
  if $b = 1$
    $s \xleftarrow{\$} \mathcal{S}$
    $m_1 \xleftarrow{\$} \mathcal{M}, m_2 \xleftarrow{\$} \mathcal{M}$
  for $i = 1..2$ do
    $(\mathsf{loc}_i, \pi_i, c_i) \leftarrow \mathsf{Write}^{(r_i)}(s, \mathsf{id}_1, \mathsf{id}_2, m_i)$
  return $(r_1, r_2, \mathsf{loc}_1, \mathsf{loc}_2, \pi_1, \pi_2, c_1, c_2)$
```

Fig. 3. Unlinkability game. Here $\mathcal{M}$ and $\mathcal{S}$ respectively denote the set of messages and shared secrets. Similarly, $M : \mathcal{I} \rightarrow \mathcal{M}$ and $S : \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{S}$ denote the implicit maps from identities to messages and shared secrets. We assume that $S(a, b) = S(b, a)$.

We define the contact discovery protocol for a registration authority RA, key-issuing committee $(\mathsf{KA}_1, \ldots, \mathsf{KA}_n)$, user $\mathcal{U}$ and bulletin board BB as follows:

1) $\mathcal{U} \leftrightarrow \mathsf{RA}$: $\mathcal{U}$ and RA engage in an authentication protocol (defined by RA) to prove that the identifier $\mathsf{id}_\mathcal{U}$ belongs to $\mathcal{U}$. Upon successful completion, RA sends $\tau_\mathcal{U} = \text{ID-NIKE}.\mathsf{Register}(\mathsf{rsk}_{\mathsf{RA}}, \mathsf{id}_\mathcal{U}||\mathsf{dom})$.
2) $\mathcal{U} \leftrightarrow \mathsf{KA}_i$, for up to $2t{+}1$ key-issuing authorities (and a minimum of $t + 1$ in the ideal case): $\mathcal{U}$ computes
$$(\alpha, \widehat{\mathsf{id}_\mathcal{U}}, \widehat{\tau_\mathcal{U}}, \pi) = \text{ID-NIKE}.\mathsf{Blind}(\mathsf{pp}, (\mathsf{id}_\mathcal{U}||\mathsf{dom}), \tau_\mathcal{U})$$
and sends the blind key-issuance request $(\widehat{\mathsf{id}_\mathcal{U}}, \widehat{\tau_\mathcal{U}}, \pi)$. If $\text{ID-NIKE}.\mathsf{VerifyID}(\mathsf{pp}, (\widehat{\mathsf{id}_\mathcal{U}}, \mathsf{dom}), \widehat{\tau_\mathcal{U}}, \pi) = 1$, KA_i sends $\text{ID-NIKE}.\mathsf{BlindPartialExtract}(\mathsf{msk}_i, \widehat{\mathsf{id}_\mathcal{U}})$.
3) $\mathcal{U}$, one-time local operation: let $\widehat{sk_i}$ and α_i denote the i-th blind share and the i-th blinding factor, $\mathcal{U}$ computes:
$$sk_i = \text{ID-NIKE}.\mathsf{Unblind}(\widehat{sk_i}, \alpha_i)$$
$$sk = \text{ID-NIKE}.\mathsf{Combine}(\{sk_i\}_{i=1}^{t+1})$$
4) $\mathcal{U}$, locally, for each contact identifier id_C: compute a share secret $s_{\mathcal{U},C}$ As
$$s_{\mathcal{U},C} = \text{ID-NIKE}.\mathsf{SharedKey}(sk, \mathsf{id}_C)$$
5) $\mathcal{U} \leftrightarrow \mathsf{BB}$, store write for each contact id_C: $\mathcal{U}$ sends a write request
$$(\mathsf{loc}_w, \pi_w^{(r)}, c) = \mathsf{HS}.\mathsf{Write}^{(r)}(s_{\mathcal{U},C}, \mathsf{id}_\mathcal{U}, \mathsf{id}_C, m)$$
If $\mathsf{HS}.\mathsf{VerifyWrite}^{(r)}(\mathsf{loc}_w, \pi_w^{(r)}) = 1$, BB writes c in the location loc_w.
6) $\mathcal{U} \leftrightarrow \mathsf{BB}$, store read for each contact id_C: $\mathcal{U}$ and BB perform $\mathsf{HS}.\mathsf{Read}$.

For clarity, this definition omits checking the correctness of the key shares (performed by the user), that the public key of the registration authority maps to its recognized domain (performed by the store authorities), and the validity of the rate-limiting tokens (performed by the store authorities, see Section IV-B).

Discovery epochs. Taking advantage of the roughly synchronized clocks (see Assumption 5), we can define discovery epochs of fixed duration (e.g., one week or one month). At the end of each epoch, store entries can be wiped. This allows the store to drop any values that are left behind after a complete handshake. On the other hand, handshakes that were only partially completed during such an epoch are aborted and will require users to once again perform the discovery phase.

RAs and KAs in practice. Using the domain separation discussed above, multiple registration authorities can co-exist under the same committee of key-issuing authorities and even use the same identifiers. Identifiers may be phone numbers, email addresses, social media handles, ENS domains, etc. Registration can be performed by first parties, e.g., Twitter attests to the ownership of a given handle, or third-party, e.g., a service offers to authenticate phone numbers or email addresses via one-time challenges or using private and trustless web authentication methods [96], [94], [31], [29], [65]. Finally, key issuance may be performed by a committee of signers. This committee can be set up for contact discovery only or may take advantage of existing networks deployed in the wild such as Lit Protocol [67].

Forward secrecy. Although we have shown that messages on the store are securely encrypted, the Arke protocol does not provide confidentiality if the system is compromised. Indeed, the AEAD symmetric key is deterministically computed from the shared secret derived using an ID-NIKE. As shown by Paterson and Srinivasan [76], ID-NIKEs do not provide forward secrecy. Therefore, an adversary that succeeds in either *(i)* compromising $t + 1$ key-issuing authorities or more, *(ii)* compromising an identity's secret key or *(iii)* compromised a shared secret between two identities, will be able to recover messages from the store. To mitigate such risks, we recommend that users only include "public" information in their initial message, and use it to establish an out-of-bound communication channel. Such a message could contain public keys to establish an end-to-end encrypted channel over the Signal protocol or an Ethereum wallet address to receive payments.

Committee updates. In certain situations it may become necessary to reconfigure the composition of Arke's key-issuing committee. These include scenarios in which new members want to join the committee to further increase its resilience against compromise or in which existing members need to be removed from the committee, e.g., because their nodes have been offline for too long. Simply re-running the DKG-based setup in such situations is counter-productive, however, since it would produce a new key pair and force all existing clients to rerun the setup to switch to the new key pair resulting in large overheads for authorities and clients alike. To avoid that, Arke can use resharing techniques similar to those presented by Wong et al. [92] and as used in practice by drand [80]. These allow resharing an existing DKG-key to a new set of nodes by refreshing the individual key shares of each node without changing the actual shared key pair. That way the configuration of Arke's key-issuing committee can be changed without affecting clients in any way. It furthermore provides Arke with a mechanism to recover from node compromise assuming less than a threshold of nodes were corrupted at any given moment and honest nodes delete their old key shares after resharing is finished.

V. The Arke Key-Value Store

We present two types of distributed stores that fulfill the required properties set in Section II-C. Section V-A presents a custom store designed to be run by large messaging companies such as WhatsApp, Signal, and Telegram across multiple data centers. Section V-B illustrates how to leverage existing (production-ready) blockchains as Arke store without requiring any modification to their protocol.

A. Custom Arke Store

This store provides extremely low latency by forgoing consensus and instead leveraging simpler and more efficient broadcast-based primitives (based on Byzantine Consistent Broadcast [23]). This store is designed to sustain a Byzantine adversary (to withstand partially corrupt store operators) but Appendix B shows a straightforward conversion into a crash fault-tolerant store. Appendix B additionally details the protocol messages and data structures run by the store's nodes, provides complete algorithms, explains how to clean up storage, and how to scale the system by maximizing parallel processing of transactions and leveraging more hardware to increase its capacity. Appendix C formally proves the validity, consistency, and termination of this store protocol.

Figure 4 presents an overview of the protocol allowing user A to respectively write and read the key-value pairs $(\mathsf{loc}_{AB}, c_{AB})$ and $(\mathsf{loc}_{BA}, c_{BA})$ from the store.

Writing the store. Steps ❶-❸ of Figure 4 illustrate the high-level interactions between user A and the storage authorities to allow the user to write the distributed store. User A uses its writing tag t_{AB} (Section IV-B) as a private signing key to create and sign a *write transaction*. This transaction mutates (or creates) the key-value pair $(\mathsf{loc}_{BA}, c_{BA}) = (g_1^{t_{BA}}, c_{BA})$ of the Arke store (❶). The user transaction is then sent to each Arke storage authority (❷). The authorities check it for validity and lock the store entry to mutate (❸). The write operation is completed as soon as $2f+1$ authorities successfully terminate this step. Algorithm 1 of Appendix B-B describes in details how authorities process incoming write transactions.

Synchronization. Steps ❹-❼ of Figure 4 illustrate the store synchronization step. At this stage, user signature keys are not needed anymore, and the synchronization process may be performed by any user client or third-party synchronizer process. Storage authorities always provide idempotent replies to protocol messages: it is safe to send multiple times the same message to an authority. After processing a write transaction, each authority returns a *vote* to the user or synchronizer process (❹). The user collects the votes from a quorum of $2f+1$ authorities to form a *certificate* (❺). The certificate is then sent back to all validators (❻). The authorities check the certificate and upon success mutate the specified store entry and release the locks to allow future updates (❼). Algorithm 2 of Appendix B-B describes this step in details. The write and synchronization mechanisms can be seen as the 'Signed Echo Broadcast' implementation of a Byzantine consistent broadcast on the label $(\mathsf{loc}_{BA}, \mathsf{Version})$ [23].

Reading the store. Steps ❽-❿ of Figure 4 illustrate the minimal interactions between user A and the storage authorities to allow the user to read the distributed store. The user creates a *read transaction* to read the value c_{BA} associated with a specified store entry $\mathsf{loc}_{BA} = g_1^{t_{BA}}$ (❽). Each authority replies with a *read reply* containing the latest value they hold for that store entry or None if the entry is not in their store (❾). Finally, user A processes the replies performs the synchronization protocol described above (in case it did not terminate), and deduces the latest value associated with the queried key (❿). Algorithm 3 of Appendix B-B describes in details how readers process incoming read replies.

B. Existing Blockchains as Arke Store

Section V-A illustrates a minimal Arke store; we now show how Arke can natively leverage most types of existing blockchains as a store. User A wishing to write the key-value pairs $(\mathsf{loc}_{AB}, c_{AB})$ to the store first format the key $\mathsf{loc}_{AB} = g_1^{t_{AB}}$ into a blockchain address $addr_{AB}$. Virtually all existing blockchains format public keys into addresses by hashing $addr_{AB} = H(g_1^{t_{AB}} || const)$, where $const$ is a public and blockchain-specific constant. The next paragraphs illustrate how to implement an Arke store over different types of blockchains.

Payment-only platforms. An Arke store can be any distributed payment platform where the transaction format allows user-defined metadata. For instance, Arke can easily use Bitcoin [72] as a store. A user A wishing to write the store makes a Bitcoin transaction sending an arbitrary number of coins to the address $addr_{AB}$ (deterministically derived from loc_{AB} as mentioned above) and additionally, writes the `OP_RETURN` opcode. This opcode allows users to specify up to 80 arbitrary bytes within the transaction (by setting `OP_RETURN_MAX_BYTES` to 80); user A writes the byte representation of c_{AB}. User A reads the blockchains by locally generating $addr_{BA}$; it can then use any light client capable of parsing `OP_RETURN`, such as *Chain* [30], to retrieve the content of $addr_{BA}$ and parse c_{BA}. Alternatively, Arke can leverage other platforms not allowing to augment transactions with arbitrary metadata by encoding c_{AB} in the less significant digits of the transfer amount.

Smart contract platform. An Arke store can also consist of any traditional smart contract platforms [93], [3], [36], [11], [78], [4], [46], [40], [62], [26] or rollup [5], [74]. A dedicated smart contract maintains a key-value map of the

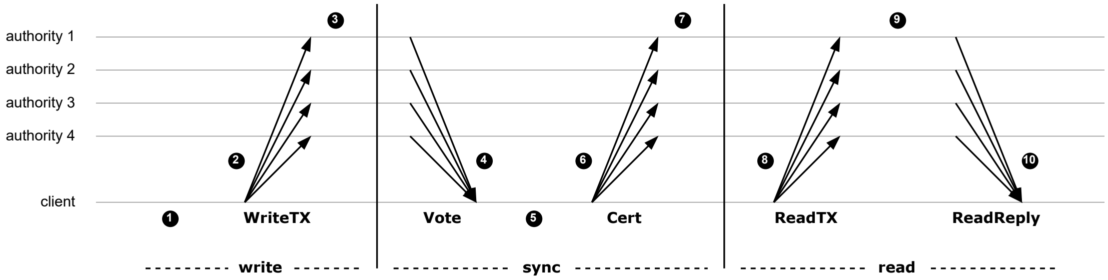

Fig. 4. Example of Arke write (❶-❸), sync (❹-❼), and read (❽-❿) protocol with 4 authorities.

pairs $(\mathsf{loc}_{AB}, c_{AB})$ that users can easily read and write. To implement good state hygiene, both user A and B can delete an entry of the key-value map by proving knowledge of the secret key associated with loc_{AB} (which they can locally derive).

Leverage consensus-less operations. Recent blockchains such as Sui [71] and Linera [66] allow users to program some types of transactions to entirely forgo consensus. For instance, Sui [71] is a smart-contract platform that forgoes consensus for single-writer operations and only relies on consensus for multi-writer operations, combining the two modes securely. As a result, any operation that can be expressed as a single-writer operation can leverage its consensus-less path and benefit from sub-second latency and lower gas fees. Arke can natively benefit from this feature. User A writes the store by creating a *owned object* [12] containing c_{AB} as the only field; it then transfers ownership of that object to the address $addr_{AB}$. User A reads the blockchain by locally deriving $addr_{BA}$ and querying all objects owned by that address. Appendix D implements an Arke store on Sui using exclusively owned objects in less than 10 LOC.

VI. Implementation and Evaluation

We implement all main Arke operations in Rust based on arkworks [6]. We additionally implement and evaluate our custom Arke store described in Section V-A. We open-source all our implementations[5] and measurement data to enable reproducible results[6]. In the following sections, we use `m5d.8xlarge` instances whenever experimenting on Amazon Web Services (AWS). These instances provide 10 Gbps of bandwidth, 32 virtual CPUs (16 physical cores) on a 2.5 GHz, Intel Xeon Platinum 8175, 128 GB memory, and run Linux Ubuntu server 22.04. We select this type of instance because it provides decent performance and is in the price range of 'commodity servers'.

A. Setup Phase

Table I shows the performance of all operations of the Arke setup protocol described in Section IV on a single CPU core. We perform our benchmarks on both a `m5d.8xlarge` Amazon Web Services (AWS) instance and a Macbook Pro equipped with an M1 processor. The function *Assemble private key* is evaluated for a committee of 10 authorities. We compute the average time over 50 runs.

Function	AWS	MBP
(RA) User registration	66.12 ms	4.13 ms
(User) Private key request	23,402.37 ms	2,259.66 ms
(KA) Issue blind partial key	358.05 ms	20.78 ms
(User) Assemble private key	584.91 ms	41.85 ms

TABLE I. Microbenchmark of the Arke setup functions on a `m5d.8xlarge` AWS instance and a Macbook Pro equipped with an M1 CPU. Each data point represents the average time (over 50 runs) in milliseconds required to evaluate the function. The function *Assemble private key* is evaluated for a committee of 10.

The table shows that user registration (performed by the registration authority) is cheap, taking respectively about 66 and 4 ms on our AWS instance and our M1 Macbook Pro. Generating private key requests is the most expensive operation; it takes about 23 seconds on our AWS instance and 2 seconds on an M1 Macbook Pro; this operation is however performed by the user (and only once) and thus does not take resources away from the key authorities. Issuing blind partial keys over a key request (performed by the key authority) is also cheap; it takes about 350 ms on our AWS instance and 20 ms on our M1 Macbook Pro, mostly spent verifying the user's key request. Assembling a quorum of blind partial keys into a full private key (performed by the user) takes about 600 ms on our AWS instance and 41 ms on our M1 Macbook Pro. We implement this operation pessimistically requiring the user to verify each blind partial key before aggregation.

B. The Arke Custom Store

We implement a networked multi-core Arke store authority as described in Section V-A. It uses `tokio` [1] for asynchronous networking and persists data structures using `Rocksdb` [2]. Our implementation uses TCP to achieve reliable point-to-point channels, necessary to correctly implement the distributed system abstractions.

We particularly aim to demonstrate the performance claims of Section II-C, reformulated as follows. **(C1)** Arke scales well with the committee size. **(C2)** Arke achieves low latency even under high load, in the WAN, and with large committee sizes. **(C3)** Arke achieves enough throughput to operate at planetary scale. **(C4)** Arke is robust when some parts of the system inevitably crash-fail. Note that evaluating BFT protocols in the presence of Byzantine faults is still an open question [9].

Experimental setup. We deploy a Arke testbed on AWS, using `m5d.8xlarge` instances across 10 different AWS regions: N. Virginia (us-east-1), Oregon (us-west-2), Canada (ca-central-1), Frankfurt (eu-central-1), Ireland (eu-west-1), London (eu-

[5] https://github.com/asonnino/arke

[6] https://github.com/asonnino/arke/tree/main/code/arke/results/results-main

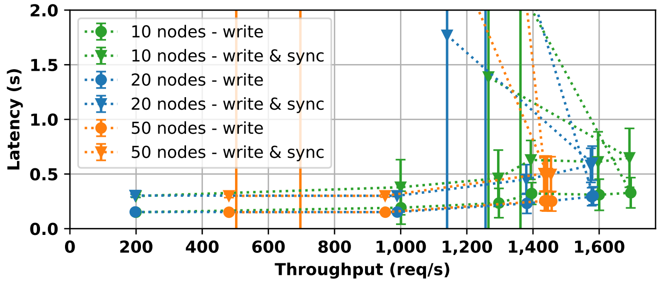

Fig. 5. Arke WAN latency-throughput with 10, 20, and 50 authorities (no faults); one shard per authority.

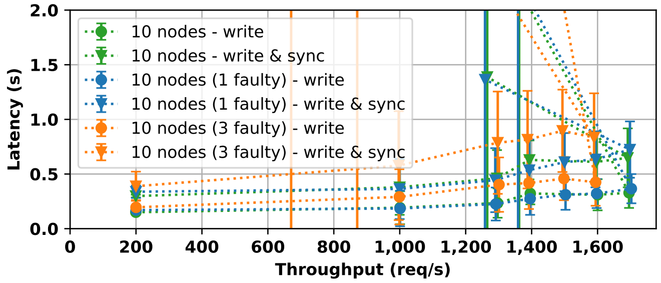

Fig. 6. Arke WAN latency-throughput with 10 validators (0, 1, and 3 faults); one shard per authority

west-2), Mumbai (ap-south-1), Singapore (ap-southeast-1), Tokyo (ap-northeast-1), and Sydney (ap-southeast-2). All data are persisted on the NVMe drives provided by the AWS instance (rather than the root partition).

In the following graphs, each data point in the latency graphs is the average of the latency of all operations of the run, and the error bars represent one standard deviation (error bars are sometimes too small to be visible on the graph). We instantiate one benchmark client colocate with each authority submitting client requests at a fixed rate for 3 minutes. We benchmark two operations; (i) *write* and (ii) *write* followed by *synchronize* (see Section V-A); we do not benchmark *read* as it is a simple database query common to many classic systems. When referring to *latency*, we mean the time elapsed from when the client submits the write request to when it assembles a certificate over the request (resp. when it is notified that a quorum of authorities is synchronized).

Benchmark in the common case. Figure 5 illustrates the latency and throughput of Arke for varying numbers of authorities. Every authority runs one shard (it thus runs on a single machine). We observe virtually no performance difference between runs with 10, 20, or even 50 authorities, thus validating our claim **(C1)**. Arke can process about 1,500 req/s with sub-second latency in all configurations. As expected the difference between simple *write* requests and *write* followed by *synchronize* is minimal. The latter displays a slightly higher latency due to the extra round-trip required to synchronize the authorities (about 100-200 ms) but throughput remains the same. This observation validates our claim **(C2)**. Based on the system usage estimates for the large-scale end-to-end encrypted messaging service WhatsApp (Section I), we would arrive at the requirement to process around 120 req/s. Thus Arke exceeds by over 10x the throughput required to operate at this scale which validates claim **(C3)**. Assuming Facebook Messenger, Signal, and Telegram have similar usage to WhatsApp, Arke can process the combined load of these services and thus operate at a planetary scale.

Benchmark under faults. Figure 6 shows the performance of Arke for a 10-authorities deployment when the system is experiencing (crash-)faults; after running without faults for one minute, 0, 1, and 3 authorities permanently crash. Every authority runs a single shard (each authority thus runs on a single machine). The figure shows that there is no noticeable throughput drop under crash faults. Arke can finalize around 1,500 req/s with a sub-second latency. The latency slightly increases with the number of faulty authorities (by at most 200 ms). Clients finalize operations as soon as the fastest quorum of authorities replies (see Section V-A); as authorities crash, clients are thus left with fewer authority replies from which to assemble certificates. This observation validates our claim **(C4)**. The performance of Arke shines compared to traditional consensus systems [10], [20], [21], [22], [27], [95] that are known to suffer 10x or 20x performance drop when experiencing leader failures [8], [21], [41], [54], [73], [81], [84].

VII. Related Work

We review related work under two different lenses. We first survey existing contact discovery schemes. Then, we review related cryptographic techniques to the ones used in Arke.

A. Contact Discovery

Arke implements a private contact discovery scheme by combining a key exchange with an unlinkable handshake. This architecture generalizes the constructions of Chaum *et al.* [32]. Their construction, known as UDM, implements both the key exchange and handshake by relying on honest-but-curious centralized parties. Furthermore, it requires to maintain a public mapping from (hashed) identifiers to public keys. Such a mapping requires storage that grows linearly in the number of system users. Finally, Chaum *et al.*[32] do not give proofs of the security and anonymity properties of their system.

Alternative architectures usually rely on Private Set Intersection (PSI) and Private Information Retrieval (PIR) schemes. PSI-based contact discovery protocols based on FHE [33], [34], [37] are unsuitable for large-scale contact discovery because the server's computation complexity increases linearly with the size of the database (for each client) during the online phase. A recent trend in mobile private contact discovery specifically designs protocols for large-scale contact discovery [44], [59], [61] typically by utilizing Bloom filters and Cuckoo filters to store the larger set. Kiss *et al.* [61] enhance PSI for the unbalanced setting and mobile clients by redistributing the necessary setup computation and communication costs, which depend linearly on the database size, to a precomputation phase. Building upon these promising outcomes, Kales *et al.* [59] build an unbalanced PSI protocol for mobile private contact discovery to optimize the performance and communication cost of two OPRF-based PSI protocols with malicious client security. PIR-PSI [44] combines two-server PIR and PSI for private contact discovery to achieve sublinear communication complexity in the database size. However, due to the lack of PIR-preprocessing, the servers perform online computation linearly in the database size for each query, making it unsuitable for large-scale deployments. Hetz *et al.* [55] integrate and further optimize the design of Kales *et al.* [59] by leveraging the two-server PIR protocol from Kogan *et al.* [63] which minimize the communication of unbalanced OPRF-based PSI Kales *et al.*. for mobile devices. They then enhance the performance of the balanced PSI protocol of Kolesnikov *et al.* [64] by utilizing PIR based on distributed point functions (DPFs) [18], [19] to reduce the input set sizes.

Despite these advancements, Signal still considers the complexity of PSI and PIR-based protocols to be too high for their purpose [69]. As a result, Signal currently runs a contact discovery service in an Intel Software Guard Extensions (SGX) enclave [39], [56], [57], [70] and hides the memory access patterns using Path ORAM [38], [85]. This approach scales well but the enclave is a single point of failure and attack, and relying on Intel SGX requires trust in Intel (as debated by many works [42], [58], [7], [82]). Arke instead relies on cryptographic techniques and a threshold assumption to solve private contact discovery with unprecedented scalability by keeping the complexity of the protocol constant regardless of the database size.

B. Cryptographic Techniques

Key escrow in ID-based cryptography. One of the main challenges in making Arke private and fault-tolerant is limiting the scope of the trusted third party in the Boneh-Waters ID-NIKE. This problem, known as the *key escrow* problem, is inherent to identity-based cryptography and is well-studied in the literature. Boneh and Franklin [15] introduce the first construction for an identity-based encryption scheme. In the same paper, they show that their construction can be thresholdized by replacing the TTP by a committee of non-colluding entities that each hold a Shamir secret share of a uniformly distributed secret key. This key distribution can be performed without a trusted party by running a DKG protocol that upholds the *correctness* and *secrecy* properties of Gennaro *et al.* [47]. We follow the same approach, and show that our system remains secure even when using more efficient but less secure DKG schemes such as the Pedersen-DKG [77].

Another (and orthogonal) approach to solving the key escrow problem is anonymous key issuance. This notion, formalized by Chow [35], reinforces the definition of blind key issuance [25], [50]. Sui *et al.* [86] propose a blind key-issuing protocol for the Boneh-Franklin IBE based on blind BLS signatures [16] and a password mechanism. This scheme however does not provide anonymity against the KA. Our construction can be seen as adapting that of Sui *et al.* [86] to the ID-NIKE setting and replacing the password mechanism with a zk-SNARK, thus achieving anonymity from the KA. Recently, Emura *et al.* [45] constructed an anonymous key issuance mechanism based on Boldyreva's [13] blind BLS signature alone. This scheme removes the need for the zk-SNARK and is therefore more efficient than the one presented in this work. However, it strengthens the role of the RA: it is now responsible for correctly verifying identities and for providing users with uniformly distributed randomness. Running such a scheme would further broaden the scope of *Assumption 1* (correct RA; see Section II-D).

There are many other techniques that mitigate the trust placed in the key-issuing authority. For example, Goyal [49] introduces the notion of *accountable* IBE. These are schemes in which the key-issuing authority is still all-powerful, however if it misbehaves, it runs the risk of being caught and punished. The design space for identity based cryptography is broad and an exhaustive exploration of these techniques is outside of the scope of this work.

IBE schemes. Identity-based encryption (IBE) schemes are close relatives to the ID-based key exchanges that we use. In fact, Paterson and Srinivasan [76] show how to convert any secure ID-NIKE into a secure IBE scheme. However, the difference in functionality is crucial in designing an efficient unlinkable handshake. In the IBE setting, any party may encrypt to someone's identity. The recipient is equipped with the decryption key, but cannot authenticate the sender from the ciphertext alone. On the other hand, combining an ID-NIKE with an AEAD scheme allows us to establish a symmetric channel, which implicitly authenticates the communicating party. Furthermore, the ID-NIKE functionality allows us to derive pseudorandom read and write tags for the unlinkable handshake. These tags are crucial in implementing a handshake in which the users are not required to perform trial decryption of all the stored messages.

Oblivious message retrieval. *Oblivious message retrieval* [68] (OMR) is very similar in spirit to our unlinkable handshake. Users have access to a public message board and are interested in knowing which messages are addressed to them, without having to read or trial decrypt the full board. Writing and reading relevant messages from the board should not reveal the sender or reader's identities. Note that, as opposed to our unlinkable handshake, this setting does not assume the existence of a shared secret between sender and recipient.

Liu and Tromer [68] achieve a practical OMR construction from fully-homomorphic encryption. Their scheme introduces an additional party, the *detector*. Given a detection key and an upper bound for the number of expected messages, the detector can perform "re-encryption" (decryption under FHE) to produce a digest indicating to the user which messages are relevant to their detection key. They estimate detector costs to be $1 per million messages scanned. Our approach forgoes this cost (and additional party) as users can identify relevant messages using only the ID-NIKE output.

VIII. Conclusion

Arke is the first Byzantine fault-tolerant privacy-preserving contact discovery system whose performance is independent of the total number of users in the system (i.e., the *database size*). Our experimental implementation shows that Arke can support 1,500 user requests per second in a large geo-replicated environment, thus largely surpassing the combined estimated needs of WhatsApp, Facebook Messenger, Signal, and Telegram. Furthermore, Arke can maintain this throughput while providing sub-second finality even when a third of the infrastructure is Byzantine. Arke is based on an unlinkable handshake mechanism built on an ID-NIKE protocol and on a custom broadcast-based distributed architecture forgoing the expense of consensus.

Acknowledgments

This work is partially supported by Mysten Labs and Geometry. We thank the cLabs team and the Celo community who inspired us to study privacy-preserving contact discovery several years ago. We thank Kostas Chalkias, Ben Riva, Joy, Francois Garillot, and Ge Gao for feedback on the early manuscript. Special thanks to Damir Shamanaev for suggesting the event-based implementation of the Arke smart contract-based store for Sui.

References

[1] https://github.com/tokio-rs/tokio, 2022.

[2] https://rocksdb.org/, 2022.

[3] Aptos Labs, "Committed to developing products and applications on the Aptos blockchain that redefine the web3 user experience," https://aptoslabs.com, 2023.

[4] ——, "Committed to developing products and applications on the Aptos blockchain that redefine the web3 user experience," https://aptoslabs.com, 2023.

[5] Arbitrum, "Secure Scaling for Ethereum," https://arbitrum.io, 2023.

[6] arkworks contributors, "`arkworks` zkSNARK ecosystem," 2022. [Online]. Available: https://arkworks.rs

[7] J. Aumasson and L. Merino, "SGX secure enclaves in practice security and crypto review," https://www.blackhat.com/docs/us-16/materials/us-16-Aumasson-SGX-Secure-Enclaves-In-Practice-Security-And-Crypto-Review.pdf, 2016.

[8] A. Baliga, I. Subhod, P. Kamat, and S. Chatterjee, "Performance evaluation of the quorum blockchain platform," *arXiv preprint arXiv:1809.03421*, 2018.

[9] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, and D. Malkhi, "Twins: Bft systems made robust," in *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[10] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," 2019.

[11] Binance, "BNB Smart Chain," https://www.bnbchain.org/en/smartChain, 2023.

[12] S. Blackshear, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, X. Li, M. Logan, A. Menon, T. Nowacki, A. Sonnino, B. Williams, and L. Zhang, "Sui Lutris: A Blockchain Combining Broadcast and Consensus," https://github.com/MystenLabs/sui/blob/main/doc/paper/sui-lutris.pdf, 2023.

[13] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme," in *6th International Workshop on Theory and Practice in Public Key Cryptography (PKC)*. Springer, 2003, pp. 31–46.

[14] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 435–464.

[15] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*. Springer, 2001, pp. 213–229.

[16] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[17] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in *International conference on the theory and application of cryptology and information security*. Springer, 2013, pp. 280–300.

[18] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Springer, 2015, pp. 337–367.

[19] ——, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1292–1303.

[20] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: an introduction," *R3 CEV, August*, vol. 1, p. 15, 2016.

[21] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, 2016.

[22] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on distributed cryptocurrencies and consensus ledgers*, vol. 310, 2016, p. 4.

[23] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[24] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich, "How to Win the Clone Wars: Efficient Periodic n-Times Anonymous Authentication," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*. ACM, 2006, pp. 201–210.

[25] J. Camenisch, G. Neven, and A. Shelat, "Simulatable adaptive oblivious transfer," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 573–590.

[26] Canto, "Canto," https://canto.io, 2023.

[27] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002. [Online]. Available: https://doi.org/10.1145/571637.571640

[28] L. Ceci, "Monthly global unique WhatsApp users 2020-2023," https://www.statista.com/statistics/1306022/whatsapp-global-unique-users/, 2023, statista. [Online; accessed 3-Nov-2023].

[29] S. Celi, A. Davidson, H. Haddadi, G. Pestana, and J. Rowell, "Distefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more," *Cryptology ePrint Archive*, 2023.

[30] Chain, "The bridge between your business and web3," https://chain.com, 2023.

[31] K. Y. Chan, H. Cui, and T. H. Yuen, "Dido: Data provenance from restricted TLS 1.3 websites," *Cryptology ePrint Archive*, 2023.

[32] D. Chaum, M. Yaksetig, A. T. Sherman, and J. De Ruiter, "UDM: Private user discovery with minimal information disclosure," *Cryptologia*, vol. 46, no. 4, pp. 347–379, Jul. 2022.

[33] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1223–1237.

[34] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[35] S. S. Chow, "Removing escrow from identity-based encryption: New security notions and key management techniques," in *International workshop on public key cryptography*. Springer, 2009, pp. 256–276.

[36] cLabs, "Celo," https://celo.org, 2022.

[37] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled psi from homomorphic encryption with reduced computation and communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1135–1150.

[38] G. Connell, "Technology Deep Dive: Building a Faster ORAM Layer for Enclaves," August 2022.

[39] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[40] Cronos Chain, "Cronos Chain," https://cronos.org, 2023.

[41] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[42] S. Davenport, "Sgx: the good, the bad and the downright ugly," https://www.virusbulletin.com/virusbulletin/2014/01/sgx-good-bad-and-downright-ugly, 2014.

[43] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy Pass: Bypassing Internet Challenges Anonymously," *Proceedings on Privacy Enhancing Technologies (PoPETS)*, vol. 2018, no. 3, pp. 164–180, 2018.

[44] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: scaling private contact discovery," *Cryptology ePrint Archive*, 2018.

[45] K. Emura, S. Katsumata, and Y. Watanabe, "Identity-based encryption with security against the kgc: A formal model and its instantiations," *Theoretical Computer Science*, vol. 900, pp. 97–119, 2022.

[46] Fantom, "Performance Matters," https://fantom.foundation, 2023.

[47] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer, 1999, pp. 295–310.

[48] ——, "Secure applications of pedersen's distributed key generation protocol," in *Topics in Cryptology—CT-RSA 2003: The Cryptographers' Track at the RSA Conference 2003 San Francisco, CA, USA, April 13–17, 2003 Proceedings*. Springer, 2003, pp. 373–390.

[49] V. Goyal, "Reducing trust in the pkg in identity based cryptosystems," in *Annual International Cryptology Conference*. Springer, 2007, pp. 430–447.

[50] M. Green and S. Hohenberger, "Blind identity-based encryption and simulatable oblivious transfer," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2007, pp. 265–282.

[51] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[52] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 147–176.

[53] C. Hagen, C. Weinert, C. Sendner, A. Dmitrienko, and T. Schneider, "All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers," Cryptology ePrint Archive, Paper 2020/1119, 2020. [Online]. Available: https://eprint.iacr.org/2020/1119

[54] R. Han, G. Shapiro, V. Gramoli, and X. Xu, "On the performance of distributed ledgers for internet of things," *Internet of Things*, p. 100087, 2019.

[55] L. Hetz, T. Schneider, and C. Weinert, "Scaling mobile private contact discovery to billions of users," *Cryptology ePrint Archive*, 2023.

[56] Intel, "Software guard extensions programming reference, ref. 329298-002us," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf,, 2014.

[57] ——, "Product change notification," https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf, 2015.

[58] A. Jackson, "Trust is in the keys of the beholder: Extending sgx autonomy and anonymity," Ph.D. dissertation, Interdisciplinary Center, Herzliya, 2017.

[59] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile Private Contact Discovery at Scale," in *Proceedings of the 28th USENIX Security Symposium*, 2019.

[60] M. Kelly, "You've been scraped," https://blog.mozilla.org/en/privacy-security/facebook-data-leak-explained/, April 2021.

[61] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *Cryptology ePrint Archive*, 2017.

[62] Klaytn, "A Sustainable and Verifiable Blockchain Built for All," https://klaytn.foundation, 2023.

[63] D. Kogan, H. Corrigan-Gibbs *et al.*, "Private blocklist lookups with checklist," 2021.

[64] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 818–829.

[65] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, "Janus: Fast privacy-preserving data provenance for TLS 1.3," *Cryptology ePrint Archive*, 2023.

[66] Linera, "Build on infrastructure with unprecedented scalability," https://linera.io, 2023.

[67] Lit Protocol, "How does lit protocol work," 2023. [Online]. Available: https://developer.litprotocol.com/resources/howItWorks

[68] Z. Liu and E. Tromer, "Oblivious message retrieval," in *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*. Springer, 2022, pp. 753–783.

[69] M. Marlinspike, "The Difficulty of Private Contact Discovery," January 2014.

[70] ——, "Technology Preview: Private Contact Discovery for Signal," September 2017.

[71] Mysten Labs, "Sui: Build without boundaries," https://sui.io, 2022.

[72] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

[73] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," *Security and Communication Networks*, vol. 2018, 2018.

[74] Optimism, "Ethereum, Scaled," https://www.optimism.io, 2023.

[75] PADO Labs, "The extension [documentation]," https://docs.padolabs.org/Products/Extension, 2023.

[76] K. G. Paterson and S. Srinivasan, "On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups," *Designs, Codes and Cryptography*, vol. 52, pp. 219–241, 2009.

[77] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*. Springer, 1991, pp. 522–526.

[78] Polygon, "Blockchain for Mass Adoption," https://polygon.technology, 2023.

[79] Privacy and Scaling Explorations Group, "Rate-limiting nullifiers documentation," 2023. [Online]. Available: https://rate-limiting-nullifier.github.io/rln-docs/

[80] Protocol Labs, "drand: Distributed randomness beacon," https://drand.love, 2023.

[81] R3, *Sizing and Performance*, 2018 (accessed January 17, 2020). [Online]. Available: https://docs.corda.r3.com/sizing-and-performance.html

[82] J. Rutkowska, "Thoughts on intel's upcoming software guard extensions," http://theinvisiblethings.blogspot.co.uk/2013/08/thoughts-on-intels-upcoming-software.html, 2016.

[83] Signal, "Signal: Speak freely," https://signal.org, 2022.

[84] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2705–2718.

[85] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: an extremely simple oblivious ram protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[86] A.-f. Sui, S. S. Chow, L. C. K. Hui, S.-M. Yiu, K.-P. Chow, W. W. Tsang, C. Chong, K. Pun, and H. Chan, "Separable and anonymous identity-based key issuing," in *11Th international conference on parallel and distributed systems (ICPADS'05)*, vol. 2. IEEE, 2005, pp. 275–279.

[87] Telegram, "Telegram: A new era of messaging," https://telegram.org, 2022.

[88] Telegram, "Telegram privacy policy," https://telegram.org/privacy, 2022.

[89] TLSNotary, "TLSNotary [source code]," https://github.com/tlsnotary/tlsn, 2023.

[90] WhatsApp Inc., "Terms of Service," https://www.whatsapp.com/legal#terms-of-service, 2022.

[91] WhatsApp LLC, "Whatsapp: Simple, secure, reliable messaging," https://www.whatsapp.com, 2022.

[92] T. Wong, C. Wang, and J. Wing, "Verifiable secret redistribution for archive systems," in *First International IEEE Security in Storage Workshop*, 2002.

[93] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[94] X. Xie, K. Yang, X. Wang, and Y. Yu, "Lightweight authentication of web data via garble-then-prove," *Cryptology ePrint Archive*, 2023.

[95] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[96] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for TLS," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1919–1938.

Appendix A
Security Proof: Theorem 1

Recall that Theorem 1 states that the threshold oblivious ID-NIKE of Definition 3 is IND-SK secure under the DBDH assumption if the hash functions H_1 and H_2 are modeled as random oracles and Π_{ID} is a knowledge sound SNARK for $\mathcal{R}_{\mathsf{ID}}$. The proofs follow from the three lemmas below:

Lemma 1. *The oblivious variant of the Boneh-Waters ID-NIKE (see Definition 5) is IND-SK secure, assuming that the Boneh-Waters ID-NIKE is IND-SK secure and* Π_{ID} *is a knowledge sound SNARK for* $\mathcal{R}_{\mathsf{ID}}$.

Lemma 2. *The oblivious variant of the Boneh-Waters ID-NIKE is rekeyable [52] with respect to the key issuer's master secret key. Furthermore, the* $\mathcal{O}\mathsf{Reveal}$ *oracle is rekeyable with respect to the master secret key.*

Lemma 3. *A key-expressable DKG [52] preserves IND-SK security for an oblivious ID-NIKE* Σ' *if:*

- Σ' *is rekeyable with respect to the master secret key.*
- $\mathsf{BlindExtract} = \mathsf{BlindPartialExtract}$
- *the* $\mathcal{O}\mathsf{Reveal}$ *oracle is rekeyable with respect to the master secret key.*

We prove each lemma individually in the following subsections.

A. Proof of Lemma 1

To prove Lemma 1, we make explicit the definition of our oblivious variant of the (centralized) Boneh-Water ID-NIKE as described in Section IV.

Definition 5 (Oblivious Boneh-Waters ID-NIKE)**.** *Let* Π_{ID} *be a knowledge sound SNARK (e.g., Groth16 [51]) for the relation* $\mathcal{R}_{\mathsf{ID}}$ *as defined in Section III-A. We assume that the public parameters for* Π_{ID} *are pre-computed and passed to all algorithms as part of the variable* pp. *The oblivious Boneh-Waters ID-NIKE is defined by the following eight efficient algorithms:*

- $\mathbf{Setup}_{E}(\lambda) \to (\mathbf{msk}, \mathbf{mpk})$. *Choose a random key-extraction secret key* $\mathsf{msk} \xleftarrow{\$} \mathbb{Z}_q$ *and compute the key-extraction public key* $\mathsf{mpk} = (g_1^{\mathsf{msk}}, g_2^{\mathsf{msk}})$. *Output* msk *and* mpk.
- $\mathbf{Setup}_{R}(\lambda) \to (\mathbf{rsk}, \mathbf{rpk})$. *Choose a random registration secret key* $\mathsf{rsk} \xleftarrow{\$} \mathbb{Z}_q$ *and compute the registration public key* $\mathsf{rpk} = (g_1^{\mathsf{rsk}}, g_2^{\mathsf{rsk}})$. *Output* rsk *and* rpk.
- $\mathbf{VerifyPK}(\mathbf{pk}) \to \{0, 1\}$. *Parse* pk *as* $(\mathsf{pk}_l, \mathsf{pk}_r)$. *If* $e(\mathsf{pk}_l, g_2) = e(g_1, \mathsf{pk}_r)$, *output* 1 *(accept). Otherwise output* 0 *(reject).*
- $\mathbf{Register}(\mathbf{rsk}, \mathbf{id}) \to (\boldsymbol{\tau}_{\mathbf{id}})$. *Compute* $\tau_l = H_1(\mathsf{id})^{\mathsf{rsk}}$ *and* $\tau_r = H_2(\mathsf{id})^{\mathsf{rsk}}$. *Output* $\tau_{\mathsf{id}} = (\tau_l, \tau_r)$.
- $\mathbf{Blind}(\mathbf{pp}, \mathbf{id}, \boldsymbol{\tau}_{\mathbf{id}}) \to (\boldsymbol{\alpha}, \widehat{\mathbf{id}}, \widehat{\boldsymbol{\tau}}_{\mathbf{id}}, \boldsymbol{\pi})$. *Sample a random blinding factor* $\alpha \xleftarrow{\$} \mathbb{Z}_q$. *Compute* $\widehat{\mathsf{id}} = (H_1(\mathsf{id})^{\alpha}, H_2(\mathsf{id})^{\alpha})$, $\pi = \Pi_{\mathsf{ID}}.\mathsf{Prove}(\mathsf{crs}, \mathsf{id}, \alpha, \widehat{\mathsf{id}}, H_1, H_2)$ *and* $\widehat{\tau_{\mathsf{id}}} = {\tau_{\mathsf{id}}}^{\alpha}$. *Output* $(\alpha, \widehat{\mathsf{id}}, \widehat{\tau_{\mathsf{id}}}, \pi)$.
- $\mathbf{VerifyID}(\mathbf{pp}, \widehat{\mathbf{id}}, \widehat{\boldsymbol{\tau}_{\mathbf{id}}}, \boldsymbol{\pi}) \to \{0, 1\}$. *Parse* rpk *as* (pk_l, pk_r), $\widehat{\mathsf{id}}$ *as* $(\widehat{\mathsf{id}}_l, \widehat{\mathsf{id}}_r)$, *and* $\widehat{\tau_{\mathsf{id}}}$ *as* $(\widehat{\tau}_l, \widehat{\tau}_r)$. *Check that the following equations hold:*
$$e(\widehat{\tau}_l, g_2) \stackrel{?}{=} e\left(\widehat{\mathsf{id}}_l, pk_r\right)$$
$$e(g_1, \widehat{\tau}_r) \stackrel{?}{=} e\left(pk_l, \widehat{\mathsf{id}}_r\right) \tag{5}$$
$$\Pi_{\mathsf{ID}}.\mathsf{Verify}(\mathsf{pp}_{\mathsf{ZK}}, \widehat{\mathsf{id}}, \pi) \stackrel{?}{=} 1 \quad \textit{(accept)}$$
If all equations verify successfully output 1, *otherwise output* 0.
- $\mathbf{BlindExtract}(\mathbf{msk}, \widehat{\mathbf{id}}) \to \widehat{\boldsymbol{sk}_{\mathbf{id}}}$. *Compute and output* $\widehat{sk_{\mathsf{id}}} = \widehat{\mathsf{id}}^{\mathsf{msk}}$.
- $\mathbf{Unblind}(\widehat{\boldsymbol{sk}_{\mathbf{id}}}, \boldsymbol{\alpha}) \to \boldsymbol{sk}_{\mathbf{id}}$. *Compute and output* $sk_{\mathsf{id}} = \widehat{sk_{\mathsf{id}}}^{\frac{1}{\alpha}}$.
- $\mathbf{VerifyExtract}(\mathbf{mpk}, \mathbf{id}, \boldsymbol{\theta})$. *Parse* mpk *as* $(\mathsf{mpk}_l, \mathsf{mpk}_r)$ *and* θ *as* $(\theta_l, \theta_r) \in \mathbb{G}_1 \times \mathbb{G}_2$. *If* $e(\theta_l, g_2) = e(H_1(\mathsf{id}), \mathsf{mpk}_r)$ *and* $e(g_1, \theta_r) = e(\mathsf{mpk}_l, H_2(\mathsf{id}))$, *output* 1 *(accept). Otherwise, output* 0 *(reject).*
- $\mathbf{SharedKey}(\mathbf{pp}, \boldsymbol{sk}_{\mathbf{id}}, \mathbf{id}') \to \boldsymbol{k}_{\mathbf{id},\mathbf{id}'}$. *As in the classic Boneh-Waters ID-NIKE, we assume that identifiers are lexicographically ordered. Parse* sk_{id} *as* (d_l, d_r) *and output* $k_{\mathsf{id},\mathsf{id}'}$*:*
$$k_{\mathsf{id},\mathsf{id}'} = \begin{cases} e(d_l, H_2(\mathsf{id}')), & \textit{if } \mathsf{id} < \mathsf{id}' \\ e(H_1(\mathsf{id}'), d_r), & \textit{if } \mathsf{id} > \mathsf{id}' \end{cases}$$

We also define an appropriate variant of the IND-SK game. As in the classic IND-SK game (recall Figure 2), the adversary $\mathcal{A}$ must determine whether some value γ is the shared key for a pair of target identities or a random element from $\mathbb{G}_T$. $\mathcal{A}$ may register any identities of her choice and use the registration token to obtain those identities' secret keys. $\mathcal{A}$ may also query the shared key for any identity pair of her choice. The game is formally described in Figure 7.

We say that an oblivious ID-NIKE scheme Σ' is IND-SK secure if for any probabilistic polynomial-time adversary $\mathcal{A}$:
$$\Pr\left[\mathsf{Exp}_{\Sigma',\mathcal{A}}^{\mathsf{ObliviousIND-SK}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Proof (Lemma 1): We prove Lemma 1 by contradiction. Suppose that there exists an adversary $\mathcal{A}$ such that:
$$\Pr\left[\mathsf{Exp}_{\Sigma',\mathcal{A}}^{\mathsf{ObliviousIND-SK}}(\lambda) = 1\right] > \frac{1}{2} + \mathsf{negl}(\lambda)$$
where Σ' designates the oblivious ID-NIKE of Definition 5. Let Σ designate the Boneh-Waters ID-NIKE. We will construct

```
$\mathsf{Exp}^{\mathsf{ObliviousIND-SK}}_{\Sigma',\mathcal{A}}(\lambda)$
1 : $b \xleftarrow{\$} \{0,1\}$
2 : $Q_r \leftarrow \emptyset, Q_k \leftarrow \emptyset$
3 : $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}_E(\lambda)$
4 : $(\mathsf{rsk}, \mathsf{rpk}) \leftarrow \mathsf{Setup}_R(\lambda)$
5 : $(\mathsf{crs}, \mathsf{td}) \leftarrow \Pi_{\mathsf{ID}}.\mathsf{Setup}(\lambda)$
6 : $\mathsf{pp} \leftarrow (\mathsf{mpk}, \mathsf{rpk}, \mathsf{crs})$
7 : $O \leftarrow \{O\mathsf{Register},$
        $O\mathsf{BExtract}, O\mathsf{Reveal}\}$
8 : $(\mathsf{id}_*, \mathsf{id}'_*) \leftarrow \mathcal{A}^O(\mathsf{pp})$
9 : $\gamma \leftarrow \mathsf{Test}(\mathsf{id}_*, \mathsf{id}'_*)$
10 : $\widehat{b} \leftarrow \mathcal{A}^O(\gamma)$
11 : if $(\widehat{b} = b) \wedge (\mathsf{id}_* \notin Q_r) \wedge$
       $(\mathsf{id}'_* \notin Q_r) \wedge ((\mathsf{id}_*, \mathsf{id}'_*) \notin Q_k)$
12 :   return 1
13 : return 0

$O\mathsf{Register}(\mathsf{id})$
1 : $\tau \leftarrow \mathsf{Register}(\mathsf{rsk}, \mathsf{id})$
2 : $Q_r \leftarrow Q_r \cup \{\mathsf{id}\}$
3 : return $\tau$

$O\mathsf{BExtract}(\widehat{\mathsf{id}}, \widehat{\tau}, \pi)$
1 : if $\mathsf{VerifyID}(\mathsf{pp}, \widehat{\mathsf{id}}, \widehat{\tau}, \pi) = 0$
2 :   return $\perp$
3 : else
4 :   $\widehat{sk_{\mathsf{id}}} \leftarrow \mathsf{BlindExtract}(\mathsf{msk}, \widehat{\mathsf{id}})$
5 :   return $\widehat{sk_{\mathsf{id}}}$
```

Fig. 7. Indistinguishability of shared keys (IND-SK) security game for oblivious ID-NIKEs. OReveal and Test are defined as in Figure 2.

an adversary $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine, and gains a non-negligible advantage in $\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}$.

Reduction overview. The reduction strategy is simple: $\mathcal{B}$ will take on the role of "registration authority" and emulate $\mathcal{A}$'s oracles. When $\mathcal{A}$ produces a test query $(\mathsf{id}_*, \mathsf{id}'_*)$, $\mathcal{B}$ forwards that query to her own Test routine. Similarly, when $\mathcal{A}$ produces a guess $\widehat{b}$, $\mathcal{B}$ forwards that guess as her own.

$\mathcal{A}$ and $\mathcal{B}$ are subject to the same Test routine. Therefore, comparing the win conditions for both experiments (line 8 of Figure 2 and line 11 of Figure 7) reveals that $\mathcal{B}$ wins in $\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}$ if $\mathcal{A}$ wins in $\mathsf{Exp}^{\mathsf{ObliviousIND-SK}}_{\Sigma',\mathcal{A}}$ and $Q_e \subseteq Q_r$; put more directly, $\mathcal{B}$ wins if $\mathcal{A}$ wins and $\mathcal{B}$'s OExtract queries are a subset of $\mathcal{A}$'s ORegister queries.

Running $\mathcal{A}$'s oracles. To run $\mathcal{A}$ as a subroutine, $\mathcal{B}$ must correctly emulate its oracles while maintaining $Q_e \subseteq Q_r$. By definition, the OReveal and Test procedures are identical in both the classical and oblivious IND-SK game. It also follows that the exclusion sets Q_k (the collection of OReveal queries) are identical for $\mathcal{A}$ and $\mathcal{B}$.

$\mathcal{B}$ can imitate ORegister by taking on the role of the registration authority. Indeed $\mathcal{B}$ runs Setup_R and replies to $\mathcal{A}$'s queries by running Register.

To emulate the OBExtract oracle, $\mathcal{B}$ must first *extract* the queried identifier and blinding factor from $\mathcal{A}$. She can then query her own OExtract oracle to obtain the secret key for the extracted identifier. More specifically, $\mathcal{B}$ runs the following procedure:

```
if $\mathsf{VerifyID}(\mathsf{pp}, \widehat{\mathsf{id}}, \widehat{\tau}, \pi) = 0$
  return $\perp$
else
  $(\mathsf{id}, \alpha) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{crs}, \mathsf{qt})$
  $\mathsf{sk}_{\mathsf{id}} \leftarrow O\mathsf{Extract}(\mathsf{id})$
  $\widehat{sk_{\mathsf{id}}} \leftarrow {sk_{\mathsf{id}}}^{\alpha}$
  return $\widehat{sk_{\mathsf{id}}}$
```

where qt is the transcript of all of $\mathcal{A}$'s oracle queries and their respective answers.

Unfortunately, this process is not a perfect emulation of OBExtract. Indeed, the extractor $\mathcal{E}$ may fail to recover a valid witness (id, α). This would lead $\mathcal{B}$ to output a value that does not follow the expected distribution for blind keys. Furthermore, even if the extractor is successful, it may be the case that the extracted identity is not one of $\mathcal{A}$'s registered identities; thus breaking the invariant imposed by our reduction $Q_e \subseteq Q_r$. We capture both of these failure conditions in the EmulateOracle experiment defined in Figure 8.

Win probability in $\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}},\mathsf{P}}$. We show that for any arbitrary PPT algorithm P, the success probability in EmulateOracle is overwhelming if Π_{ID} is a knowledge sound SNARK. The success probability can be written as:

$$\Pr\left[\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}},\mathsf{P}}(\lambda) = 1\right] = \Pr\left[(\widehat{sk} = \widehat{sk_*}) \wedge (Q_e \subseteq Q_r)\right] \tag{6}$$

First, we show that if $\Pi_{\mathsf{ID}}.\mathcal{E}$ is successful in extracting a valid witness, then $\widehat{sk} = \widehat{sk_*}$. Assume $\left(\widehat{\mathsf{id}_*}, (\mathsf{id}, \alpha)\right) \in \mathcal{R}_{\mathsf{ID}}$, then:

$$\begin{aligned} sk^{\alpha} &= O\mathsf{Extract}(\mathsf{id})^{\alpha} \\ &= (H_1(\mathsf{id})^{\mathsf{msk}}, H_2(\mathsf{id})^{\mathsf{msk}})^{\alpha} \\ &= (H_1(\mathsf{id})^{\alpha}, H_2(\mathsf{id})^{\alpha})^{\mathsf{msk}} \\ &= \widehat{\mathsf{id}_*}^{\mathsf{msk}} \end{aligned}$$

Therefore, using EXT as shorthand notation for the event $\left(\widehat{\mathsf{id}_*}, (\mathsf{id}, \alpha)\right) \in \mathcal{R}_{\mathsf{ID}}$:

$$\Pr\left[\widehat{sk} = \widehat{sk_*}\right] \geq \Pr\left[\mathsf{EXT}\right] \tag{7}$$

Using the result from Equation (7) and applying Bayes' theorem to Equation (6), we express the EmulateOracle success probability as:

$$\Pr\left[\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}},\mathsf{P}}(\lambda) = 1\right] \geq \Pr\left[Q_e \subseteq Q_r \mid \mathsf{EXT}\right] \Pr\left[\mathsf{EXT}\right] \tag{8}$$

By definition, $\Pr\left[\mathsf{EXT}\right]$ denotes the probability that the extractor for Π_{ID} is successful in recovering a valid witness. Therefore, it holds that $\Pr\left[\mathsf{EXT}\right] > 1 - \mathsf{negl}(\lambda)$ if Π_{ID} is a knowledge sound SNARK.

We now evaluate the probability $\Pr\left[Q_e \subseteq Q_r \mid \mathsf{EXT}\right]$. Let $\mathsf{id} \in \mathcal{I}$, $\alpha \in \mathbb{Z}_q$ such that $\left(\widehat{\mathsf{id}_*}, (\mathsf{id}, \alpha)\right) \in \mathcal{R}_{\mathsf{ID}}$. Assume, for the sake of argument, that $\mathsf{id} \notin Q_r$. Parsing $\widehat{\tau_*}$ as $(\widehat{\tau_l}, \widehat{\tau_r})$ and rpk as (pk_l, pk_r), we know from lines 8 and 9 of Figure 8 that:

$$e\left(\widehat{\tau_l}, g_2\right) = e\left(H_1(\mathsf{id})^{\alpha}, pk_r\right) \tag{9}$$

Using the bilinear property of our pairing, we can rewrite Equation (9) as:

$$e\left(\widehat{\tau_l}^{\frac{1}{\alpha}}, g_2\right) = e\left(H_1(\mathsf{id}), pk_r\right) \tag{10}$$

Notice that *Equation* (10) is the verification equation for a BLS signature. Here $(\mathsf{id}, \widehat{\tau_l}^{\frac{1}{\alpha}})$ is a valid BLS message-signature pair for the secret key rsk. However, if $\mathsf{id} \notin Q_r$, then $(\mathsf{id}, \widehat{\tau_l}^{\frac{1}{\alpha}})$ is in fact a forgery. Since BLS signatures are

```
$\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}},\mathsf{P}}(\lambda, aux, O\mathsf{Extract})$
$Q_r \leftarrow \emptyset, Q_e \leftarrow \emptyset$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}_E(\lambda)$
$(\mathsf{rsk}, \mathsf{rpk}) \leftarrow \mathsf{Setup}_R(\lambda)$
$(\mathsf{crs}, \mathsf{td}) \leftarrow \Pi_{\mathsf{ID}}.\mathsf{Setup}(\lambda)$
$\mathsf{pp} \leftarrow (\mathsf{mpk}, \mathsf{rpk}, \mathsf{crs})$
$\left(\mathsf{id}_*, \alpha_*, (\widehat{\mathsf{id}_*}, \widehat{\tau}_*, \pi_*)\right) \leftarrow \mathsf{ForceValidRequest}_{\mathsf{P}}(\mathsf{rsk}, \mathsf{pp}, aux)$
$\widehat{sk_*} \leftarrow \mathsf{BlindExtract}(\mathsf{msk}, \widehat{\mathsf{id}_*})$
$(\mathsf{id}, \alpha) \leftarrow \Pi_{\mathsf{ID}}.\mathcal{E}_{\mathsf{P}}(\mathsf{crs}, aux)$
$sk \leftarrow O\mathsf{Extract}(\mathsf{id})$
$\widehat{sk} \leftarrow sk^{\alpha}$
if $(\widehat{sk} = \widehat{sk_*}) \wedge (Q_e \subseteq Q_r)$
    return $1$
return $0$
```

```
$\mathsf{ForceValidRequest}_{\mathsf{P}}(\mathsf{rsk}, \mathsf{pp}, aux)$
while $\left((\widehat{\mathsf{id}_*}, (\mathsf{id}_*, \alpha_*)) \notin \mathcal{R}_{\mathsf{ID}}\right) \vee \left(\mathsf{VerifyID}(\mathsf{pp}, \widehat{\mathsf{id}_*}, \widehat{\tau}_*, \pi_*) = 0\right)$ do :
    $\mathsf{id}_q \leftarrow \mathsf{P}(\mathsf{pp}, aux)$
    $aux \leftarrow aux || (\mathsf{id}_q, \mathsf{Register}(\mathsf{rsk}, \mathsf{id}_q))$
    $Q_r \leftarrow Q_r \cup \{\mathsf{id}_q\}$
    $\left(\mathsf{id}_*, \alpha_*, (\widehat{\mathsf{id}_*}, \widehat{\tau}_*, \pi_*)\right) \leftarrow \mathsf{P}(\mathsf{pp}, aux)$
return $\left(\mathsf{id}_*, \alpha_*, (\widehat{\mathsf{id}_*}, \widehat{\tau}_*, \pi_*)\right)$
```

Fig. 8. Blind identity extraction game. $O\mathsf{Extract}$ is defined as in Figure 2. P is an arbitrary PPT algorithm and aux denotes auxiliary inputs to P.

existentially unforgeable in the random oracle model assuming the CDH problem is hard, we can conclude that:

$$\Pr\left[Q_e \subseteq Q_r \mid \mathsf{EXT}\right] > 1 - \mathsf{negl}(\lambda)$$

Having established that the probabilities $\Pr\left[Q_e \subseteq Q_r \mid \mathsf{EXT}\right]$ and $\Pr\left[\mathsf{EXT}\right]$ are both overwhelming, we can rewrite Equation (8) as:

$$\Pr\left[\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}},\mathsf{P}}(\lambda) = 1\right] > 1 - \mathsf{negl}(\lambda)$$

thus proving that the success probability in $\mathsf{Exp}^{\mathsf{EmulateOracle}}_{\Pi_{\mathsf{ID}}}$ is overwhelming if Π_{ID} is a knowledge sound SNARK.

Successful reduction. As $\mathcal{A}$ is a probabilistic polynomial-time algorithm, it will produce at most a polynomial number of queries to $O\mathsf{Register}$. Therefore, the probability that $\mathcal{B}$ is successful in answering *all* off $\mathcal{A}$'s $O\mathsf{BExtract}$ queries is also overwhelming. In that case, $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s oracles. Thus we establish:

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}(\lambda) = 1 \mid \mathsf{Exp}^{\mathsf{ObliviousIND-SK}}_{\Sigma',\mathcal{A}}(\lambda) = 1\right] > 1 - \mathsf{negl}(\lambda)$$

Using the law of total probability and Bayes' theorem, it holds that:

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND-SK}}_{\Sigma,\mathcal{B}}(\lambda) = 1\right] > 1 - \mathsf{negl}(\lambda)$$

thus proving that our reduction is successful.

Therefore, we conclude that the oblivious variant of the Boneh-Waters ID-NIKE (Definition 5) is IND-SK secure, assuming that the Boneh-Waters ID-NIKE is IND-SK secure and Π_{ID} is knowledge sound.

■

B. Proof of Lemma 2

We prove Lemma 2 using a similar argument to the one given in Gurkan *et al.* [52] (Appendix D.2) for the rekeyability of BLS signatures. The goal is to show that all algorithms behave as expected when fed a linear combination of private keys (and the corresponding public key) instead of the expected uniformly distributed private key.

We briefly recall some notions introduced by the *rekeyability* definition of [52]. Given a function f_{msk} that relates two secret keys msk_A and msk_B, we say that an algorithm Π_i is *rekeyable with respect to the secret key* if there exists an efficient algorithm rekey_i such that:

$$\begin{aligned}\mathsf{rekey}_i(\alpha, &\mathsf{mpk}_A, \mathsf{msk}_B, x, \Pi_i(\mathsf{msk}_A, x; r)) \\ &= \Pi_i(f_{\mathsf{msk}}(\alpha, \mathsf{msk}_A, \mathsf{msk}_B), x; r)\end{aligned}$$

for all $x \in \mathsf{Domain}(\Pi_i)$ and randomness r.

Similarly, given a function f_{mpk} that relates the corresponding public keys, we say that algorithms (Π_i, Π_j) are *rekeyable with respect to the secret key* if *(1)* Π_i is rekeyable with respect to the secret key and, *(2)*:

$$\begin{aligned}&\Pi_j(\mathsf{mpk}_A, y) = \\ &\quad \Pi_j(f_{\mathsf{mpk}}(\alpha, \mathsf{mpk}_A, \mathsf{mpk}_B), \mathsf{rekey}_i(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, y))\end{aligned}$$

for all $y \in \mathsf{Image}(\Pi_i)$.

Proof (Lemma 2): We show that all algorithms in the ID-NIKE construction of Definition 5 that take the master secret key as input are *rekeyable with respect to the master secret key*. Furthermore, let $\mathsf{UnblindVerifyExtract}$ denote the sequential applications of $\mathsf{Unblind}$ and $\mathsf{VerifyExtract}$, we show that $(\mathsf{BlindExtract}, \mathsf{UnblindVerifyExtract})$ is rekeyable with respect to the secret key. We do so by giving explicit definitions for f_{msk}, f_{mpk}, rekey_{BE} the rekeying function for $\mathsf{BlindExtract}$ and rekey_{OR}, the rekeying function for the $O\mathsf{Reveal}$ oracle (as defined in Figure 2).

Let $(\mathsf{msk}_A, \mathsf{mpk}_A) \leftarrow \mathsf{Setup}_E(\lambda)$ and $(\mathsf{msk}_B, \mathsf{mpk}_B) \leftarrow \mathsf{Setup}_E(\lambda)$. Given some coefficient $\alpha \in \mathbb{N}$, we define the function f_{msk} relating master secret keys and f_{mpk} relating master public keys as:

$$\begin{aligned} f_{\mathsf{msk}}(\alpha, \mathsf{msk}_A, \mathsf{msk}_B) &= \alpha\mathsf{msk}_A + \mathsf{msk}_B \\ f_{\mathsf{mpk}}(\alpha, \mathsf{mpk}_A, \mathsf{mpk}_B) &= (\mathsf{mpk}_A)^{\alpha} \circ \mathsf{mpk}_B \end{aligned}$$

Notice that:

$$\begin{aligned} f_{\mathsf{mpk}}(\alpha, \mathsf{mpk}_A, \mathsf{mpk}_B) &= (\mathsf{mpk}_A)^{\alpha} \circ \mathsf{mpk}_B \\ &= (g_1, g_2)^{\alpha\mathsf{msk}_A + \mathsf{msk}_B} \\ &= (g_1, g_2)^{f_{\mathsf{msk}}(\alpha, \mathsf{msk}_A, \mathsf{msk}_B)} \end{aligned} \tag{11}$$

Rekeying $\mathsf{VerifyPK}$. Plugging the values from Equation (11) into the $\mathsf{VerifyPK}$ algorithm will accept if and only if the original public key mpk_A was indeed well-formed (see Equation (1)). Thus, $\mathsf{VerifyPK}$ is rekeyable with respect to the public key.

Rekeying $\mathsf{BlindExtract}$. Given a blinded identifier $\widehat{\mathsf{id}}$ and a blind key $\widehat{sk} \leftarrow \mathsf{BlindExtract}(\mathsf{msk}_A, \widehat{\mathsf{id}})$, we define rekey_{BE} as:

$$\mathsf{rekey}_{BE}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, \widehat{\mathsf{id}}, \widehat{sk}) = \widehat{sk}^{\alpha} \circ \widehat{\mathsf{id}}^{\mathsf{msk}_B}$$

As required:

$$\begin{aligned} \widehat{sk}^{\alpha} \circ \widehat{\mathsf{id}}^{\mathsf{msk}_B} &= \widehat{\mathsf{id}}^{\alpha\mathsf{msk}_A} \circ \widehat{\mathsf{id}}^{\mathsf{msk}_B} \\ &= \widehat{\mathsf{id}}^{\alpha\mathsf{msk}_A + \mathsf{msk}_B} \\ &= \mathsf{BlindExtract}(f_{\mathsf{msk}}(\alpha, \mathsf{msk}_A, \mathsf{msk}_B), \widehat{\mathsf{id}}) \end{aligned}$$

We can show that $(\mathsf{BlindExtract}, \mathsf{UnblindVerifyExtract})$ is rekeyable with respect to the secret key by observing the previously shown equalities:

$$f_{\mathsf{mpk}}(\alpha, \mathsf{mpk}_A, \mathsf{mpk}_B) = (g_1, g_2)^{\alpha\mathsf{msk}_A + \mathsf{msk}_B}$$
$$\mathsf{rekey}_{BE}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, \widehat{\mathsf{id}}, \widehat{sk}) = \widehat{\mathsf{id}}^{\alpha\mathsf{msk}_A + \mathsf{msk}_B}$$

As shown in Equation (2), the $\mathsf{VerifyExtract}$ algorithm always outputs 1 when the equalities above are respected.

Rekeying $\mathcal{O}\mathsf{Reveal}$. Given an identity pair $(\mathsf{id}, \mathsf{id}')$ and their shared key $k \leftarrow \mathcal{O}\mathsf{Reveal}_{\mathsf{msk}_A}(\mathsf{id}, \mathsf{id}')$, we define rekey_{OR} as:

$$\begin{aligned} &\mathsf{rekey}_{OR}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, (\mathsf{id}, \mathsf{id}'), k) \\ &\qquad = \begin{cases} k^{\alpha} \cdot e\left(H_1(\mathsf{id}), H_2(\mathsf{id}')\right)^{\mathsf{msk}_B}, & \text{if } \mathsf{id} < \mathsf{id}' \\ k^{\alpha} \cdot e\left(H_1(\mathsf{id}'), H_2(\mathsf{id})\right)^{\mathsf{msk}_B}, & \text{if } \mathsf{id} > \mathsf{id}' \end{cases} \end{aligned}$$

Assuming without loss of generality that $\mathsf{id} < \mathsf{id}'$, it holds that:

$$\begin{aligned} &k^{\alpha} \cdot e\left(H_1(\mathsf{id}), H_2(\mathsf{id}')\right)^{\mathsf{msk}_B} \\ &\quad = e\left(H_1(\mathsf{id}), H_2(\mathsf{id}')\right)^{\alpha\mathsf{msk}_A} \cdot e\left(H_1(\mathsf{id}), H_2(\mathsf{id}')\right)^{\mathsf{msk}_B} \\ &\quad = \mathcal{O}\mathsf{Reveal}_{f_{\mathsf{msk}}(\alpha, \mathsf{msk}_A, \mathsf{msk}_B)}(\mathsf{id}, \mathsf{id}') \end{aligned}$$

■

C. Proof of Lemma 3

Finally, we prove Lemma 3. To do so, we introduce the experiment $\mathsf{Exp}^{\mathsf{ThrOblIND-SK}}_{\Sigma,\mathcal{A}}$. This game is a DKG variant of Figure 7, constructed as prescribed by Gurkan *et al.* [52]. It is identical to $\mathsf{Exp}^{\mathsf{ObliviousIND-SK}}_{\Sigma,\mathcal{A}}$ with the initial Setup_E step (line 3) being replaced by a key-expressible DKG denoted by $\mathsf{SetupDKG}_E$ and defined as follows:

- **$\mathsf{SetupDKG}_E(\lambda, t, n) \rightarrow (\mathsf{msk}_1, \ldots, \mathsf{msk}_n, \mathsf{pp})$.** Participants $P_1, \ldots, P_n$ execute a key-expressible DKG to compute Shamir secret shares $\mathsf{msk}_1, \ldots, \mathsf{msk}_n$ of an (unknown) master secret key msk. They jointly output a transcript and master public key $\mathsf{mpk} = (g_1^{\mathsf{msk}}, g_2^{\mathsf{msk}})$. Output msk_i to P_i and $\mathsf{pp} \leftarrow (\mathsf{transcript}, \mathsf{mpk})$.

Proof (Lemma 3): Let Σ denote an oblivious ID-NIKE, and Σ' denote a key-expressible DKG variant of the same oblivious ID-NIKE. Let $\mathcal{A}$ be a PPT adversary in the experiment $\mathsf{Exp}^{\mathsf{ThrOblIND-SK}}_{\Sigma',\mathcal{A}}$ with key-extraction public key mpk. We construct an adversary $\mathcal{B}$ that retains the same advantage as $\mathcal{A}$ but against $\mathsf{Exp}^{\mathsf{ObliviousIND-SK}}_{\Sigma,\mathcal{B}}$ with public key mpk_A.

$\mathcal{B}$ receives the public key mpk_A from its challenger. Let n be the number of participants expected by $\mathcal{A}$ and I the set of indices that $\mathcal{A}$ corrupts. $\mathcal{B}$ runs $\mathtt{SimDKG}(\mathsf{Sim}, I, n)$, acting as Sim to interact with $\mathcal{A}$ and obtains the tuple $(\mathsf{transcript}, \mathsf{mpk}, \alpha, \mathsf{mpk}_B, \mathsf{msk}_B)$ as per the definition of a key-expressable DKG. Note that by definition $\mathsf{mpk} = f_{\mathsf{mpk}}(\alpha, \mathsf{mpk}_A, \mathsf{mpk}_B)$.

$\mathcal{B}$ can emulate $\mathcal{A}$'s oracles as follows:

- $\mathcal{O}\mathsf{BExtract}_{\mathsf{msk}}(\mathsf{pp}, \widehat{\mathsf{id}}, \widehat{\tau}, \pi)$ - $\mathcal{B}$ queries $\mathcal{O}\mathsf{BExtract}_{\mathsf{msk}_A}(\mathsf{pp}, \widehat{\mathsf{id}}, \widehat{\tau}, \pi)$ to obtain the value $\widehat{sk_{\mathsf{id}}}$. It computes
$$\widehat{sk'_{\mathsf{id}}} = \mathsf{rekey}_{BE}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, \widehat{\mathsf{id}}, \widehat{sk_{\mathsf{id}}})$$
and outputs $\widehat{sk'_{\mathsf{id}}}$.
- $\mathcal{O}\mathsf{Reveal}_{\mathsf{msk}}(\mathsf{id}, \mathsf{id}')$ - $\mathcal{B}$ queries $\mathcal{O}\mathsf{Reveal}_{\mathsf{msk}_1}(\mathsf{id}, \mathsf{id}')$ to obtain the value $k_{\mathsf{id},\mathsf{id}'}$. It computes
$$k'_{\mathsf{id},\mathsf{id}'} = \mathsf{rekey}_{OR}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, (\mathsf{id}, \mathsf{id}'), k_{\mathsf{id},\mathsf{id}'})$$
and outputs $k'_{\mathsf{id},\mathsf{id}'}$. Notice that $\mathcal{B}$ is able to rekey $k_{\mathsf{id},\mathsf{id}'}$ without knowledge of either of the user secret keys sk_{id} and $sk_{\mathsf{id}'}$.
- $\mathsf{Test}_b(\mathsf{id}, \mathsf{id}')$ - $\mathcal{B}$ queries $\mathsf{Test}_b(\mathsf{id}, \mathsf{id}')$ to obtain the value $k^{(b)}$. It computes
$$k^{(b)}_* = \mathsf{rekey}_{OR}(\alpha, \mathsf{mpk}_A, \mathsf{msk}_B, (\mathsf{id}, \mathsf{id}'), k^{(b)})$$
and outputs $k^{(b)}_*$.

When $\mathcal{A}$ returns a bit $\hat{b}$, $\mathcal{B}$ returns that same bit. $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s oracles and key expressability implies that it also perfectly simulates the DKG. $\mathcal{A}$ and $\mathcal{B}$ run in the same experiment and return the same bit, therefore their advantages are equal. ■

Appendix B
Detailed Arke Custom Store

This appendix complements Section V-A It details the protocol messages and data structures run by the store's nodes, provides complete algorithms, explains how to clean up storage, and how to scale the system by maximizing parallel processing of transactions and leveraging more hardware to increase its capacity.

A. Protocol Messages and Data Structures

Arke storage authorities and users run the read and write protocol described in Section V-A by exchanging the following messages:

- A *write transaction* (WRITETX) is a structure sent by user A to the storage authorities to update a specific store entry. The transaction is signed by user A using the tag t_{AB} as the secret key and contains the following fields:

  - The value c_{AB} to write on the store.
  - The location of the store $\mathsf{loc}_{AB} = g_1^{t_{AB}}$ where to write.
  - A version number ensures the freshness of the transaction.
  - The current epoch number.
  - A signature by t_{AB} over the transaction's fields.

  The transaction also supports a few self-explanatory access operations, such as *version*(WRITETX) to get its version number and functions to access the key-value pair to update.

- A *vote* (VOTE) on a write transaction contains the transaction itself as well as the identifier and signature of a store authority.
- A *certificate* (CERT) on a write transaction contains the transaction itself as well as the identifiers and signatures from at least a quorum of $2f + 1$ storage authorities. A certificate may not be unique, and the same logical certificate may be signed by a different quorum of storage authorities. However, two different valid certificates on the same transaction are treated as representing semantically the same certificate. The identifiers of signers are included in the certificate (i.e., accountable signatures [14]) to identify validators ready to process the certificate. Similarly to transactions, certificates support several self-explanatory access functions to get its version number and the key-value pair to update.
- A *read transaction* (READTX) is a structure specifying a store entry $\mathsf{loc}_{BA} = g_1^{t_{BA}}$ to read.
- A *read reply* (READREPLY) on a read transaction contains the transaction itself as well as the latest tuple (CERT, VOTE) known by a store authority. It also contains the identifier and signature of that authority.

Each store authority maintains two persistent tables abstracted as key-value maps, with the usual contains, get, and set operations.

- The *lock map* records the last valid update to a store entry embedded in the last valid certificate CERT seen by the authority. It also stores the last vote VOTE that the authority generated to further update the key. Alternatively, it may hold None if the store entry does not exist or the authority did not see the transaction before. The lock map is defined as follows:

  $\mathsf{LockDb}[\textit{key}(\text{WRITETX})] \rightarrow (\text{CERT}, \mathsf{LockVoteOption})$

B. Store Core Operations

We detail the operations performed by the authorities when receiving write transactions and certificates from users and describe how users process read replies from the authorities.

Process write transaction. Algorithm 1 shows how storage authorities process write transactions; that is, step ❸ of Figure 4 (see Section V-A). Upon receiving a write transaction WRITETX the storage authority calls PROCESSTX to perform several checks:

- **Check (1.1):** It ensures that the author of WRITETX is authorized to write in the specified store location. That is, check that WRITETX is correctly signed using the secret key corresponding to the public key $\mathsf{loc}_{AB} = g_1^{t_{AB}}$ included in the transaction as the public key.
- **Check (1.2):** It tries to acquire a (mutex) guard over the store entry *key*(WRITETX); otherwise, it returns an error and terminates the processing of WRITETX. Acquiring a guard ensures that no other task can concurrently perform the next step of the algorithm on the same key.
- **Check (1.3):** It ensures the transaction is for the current epoch Epoch. This check is crucial to maintain consistency across epochs as the LockDb store is partially reset upon epoch change (see Appendix B-C).
- **Check (1.4):** It ensures the version number of WRITETX is the next natural integer expected in the sequence (Line 14). If it is the first time the authority writes this store entry (i.e., LockDb[loc] is empty), the value PrevCert at Line 13 is a placeholder certificate without content and with version number zero; and LockVote = None.
- **Check (1.5):** It checks that LockDb[*key*(WRITETX)] is either None or set to *the same* transaction WRITETX, and atomically sets it to VOTE. In other words, no other transaction WRITETX′ ≠ WRITETX has been signed for the same version number. This is an important validity check to implement *byzantine consistent broadcast* [23] and ensure safety.

If all checks are successful then the authority returns a vote VOTE, i.e., a signature on the write transaction. Processing a transaction is idempotent upon success, and always returns a vote (VOTE) within the same epoch. Any party may collate a transaction and votes (VOTE) from a quorum of $2f + 1$ authorities of epoch Epoch, to form a certificate CERT. Many tasks can call ProcessTx concurrently (or in parallel). Arke only acquires mutexes[7] on the minimum amount of data: the store entry that the transaction is trying to update (Algorithm 1 Line 7).

Process write certificates. Algorithm 2 shows how storage authorities process write certificates; that is, step ❼ of Figure 4 (see Section V-A). Upon receiving a certificate CERT a Arke authority calls ProcessCert of Algorithm 2 to perform a number of checks:

- **Check (2.1):** It ensures the certificate is signed by a quorum of $2f+1$ authorities. Optionally, the authority may re-check that the writer is authorized to update the specified store entry (check (1.1)); if they aren't the certificate CERT is proof of catastrophic failure and that the BFT assumption broke.

[7] This mutex ensures that correct authorities never return two different votes over the same store entry update. The following scenario may happen if we omit the mutex Line 7. Two different transactions (WRITETX and WRITETX′) updating the same store entry (with the same version) may be submitted concurrently to the authority. Both transactions pass all checks until Line 20. The first transaction then assigns the lock Line 20 and the authority returns VOTE; the second transaction then overwrites the lock and the validator returns a conflicting VOTE′.

Algorithm 1 Process WRITETX

```
    // Executed upon receiving a write transaction from a user.
    // Many tasks can call this function concurrently.
 1: procedure PROCESSWRITETX(WRITETX)
 2:     // Check (1.1): Check transaction validity (Appendix B-B)
 3:     if !valid(WRITETX) then return Error
 4:
 5:     // Check (1.2): Try to acquire a mutex over key(WRITETX)
 6:     loc ← key(WRITETX)
 7:     guard = ACQUIREGUARD(loc)          ▷ Error if cannot guard
 8:
 9:     // Check (1.3): Ensure WRITETX is for the current epoch.
10:     if epoch(WRITETX) ≠ Epoch then return Error
11:
12:     // Check (1.4): Check WRITETX's version
13:     (PrevCert, LockVote) ← LockDb[loc]     ▷ None if no loc
14:     Version ← version(PrevCert) + 1         ▷ Expected version
15:     if Version ≠ version(WRITETX) then return Error
16:
17:     // Check (1.5): Only sign non-conflicting transactions
18:     VOTE ← sign(WRITETX)
19:     if LockVote == None then
20:         LockDb[loc] ← (PrevCert, VOTE)
21:     else if LockVote ≠ VOTE then
22:         return Error
23:
24:     // Return a vote on WRITETX
25:     return VOTE
```

- **Check (2.2):** It tries to acquire a guard over the store entry *key*(CERT); otherwise, it returns an error and terminates the processing of CERT. Acquiring a guard ensures that no other task can concurrently perform the next step of the algorithm on the same key, or call PROCESSWRITETX (Algorithm 1) with a new transaction over the same store entry *key*(CERT).
- **Check (2.3):** It ensures the certificate is for the current epoch Epoch. This check is crucial to maintain consistency across epochs as the LockDb store is partially reset upon epoch change (see Appendix B-C).
- **Check (2.4):** It ensures that CERT is newer than the latest certificate seen by the authority. This check ensures the state of the authority cannot be reverted by replaying older certificates.

If all check succeeds, the value associated with the store entry *key*(CERT) is updated to *value*(CERT) and the version number expected for the next update to *version*(CERT). These two operations are implicitly performed at Line 16: the latest value and version of *key*(CERT) are persisted as part of the certificate CERT. Further, the lock previously set to LockVote is now released in order to accept future updates of *key*(CERT).

Process read replies. Algorithm 3 shows how the reader processes read replies received from a quorum of storage authorities; that is, step ❿ of Figure 4 (see Section V-A). The reader collects at least $2f+1$ read replies [READREPLY]. Check (3.1) filters out

1) Any malformed or empty reply. Malformed replies do not contain valid authorities' signatures and empty replies contain (CERT, VOTE) = (None, None).
2) Any reply concerning protocol messages with epoch number e such that $e + E \leq$ Epoch. The parameter E is the maximum number of epochs for which the storage authorities keep a store entry, and Epoch is the current epoch of the reader.

Algorithm 2 Process CERT

```
    // Executed upon receiving a write certificate from a user.
    // Many tasks can call this function concurrently.
 1: procedure PROCESSWRITECERT(CERT)
 2:     // Check (2.1): Check certificate validity (Appendix B-B)
 3:     if !valid(CERT) then return Error
 4:
 5:     // Check (2.2): Try to acquire a mutex over key(WRITETX)
 6:     loc ← key(WRITETX)
 7:     guard = ACQUIREGUARD(loc)          ▷ Error if cannot guard
 8:
 9:     // Check (2.3): Ensure CERT is for the current epoch
10:     if epoch(CERT) ≠ Epoch then return Error
11:
12:     // Check (2.4): Check CERT's version
13:     (PrevCert, LockVote) ← LockDb[loc]     ▷ None if no loc
14:     Version ← version(PrevCert)             ▷ Expected version
15:     if Version < version(CERT) then
16:         LockDb[loc] ← (CERT, None)       ▷ Write value(CERT)
17:
18:     return Ack          ▷ Acknowledgement certificate processing
```

Algorithm 3 Process READREPLY

```
    // Executed upon receiving read replies from an authority.
 1: procedure PROCESSREADREPLY([READREPLY])
 2:     // Check (3.1): Filter out invalid replies (Appendix B-B).
 3:     [READREPLY] ← valid([READREPLY])
 4:
 5:     if ![READREPLY] then              ▷ If the reply set is empty
 6:         return None
 7:
 8:     (CERT, VOTE) ← HIGESTREPLY([READREPLY])
 9:     if CERT ≥ VOTE then
10:         DISSEMINATECERT(CERT)                     ▷ Optional
11:         return value(CERT)
12:     else
13:         WRITETX ← tx(VOTE)
14:         return FINISHSYNC(WRITETX)           ▷ Finish sync
```

After this check, if the set [READREPLY] is empty replies, the reader reads None (Line 6). Alternatively, the reader looks for the highest certificate and the highest valid vote (Line 8). These are simply the certificate and valid vote included in the set [READREPLY] with the highest version. A valid vote contains a WRITETX that passes Check (1.1) of Algorithm 1. Finally, the reader compares the highest certificate CERT with the highest vote VOTE. If the certificate has a higher version than the vote, the reader optionally disseminates the certificate to any authority who missed it (Line 10) and then reads *value*(CERT). Alternatively, the reader concludes that further authority synchronization is needed (Line 14). It then performs the synchronization steps ❹-❼ of Figure 4 described in Section V-A, or waits for another party to synchronize the authorities. The reader then re-tries the read operation.

C. Epoch Change

Epoch changes serve two main purposes, they allow unlocking any store entry partially written by faulty writers and they are used to clean up storage by deleting hold entries.

Transactions unlocking. A faulty writer may sign two conflicting transactions WRITETX and $\text{WRITETX}'$ with the same version number and both updating the same store entry $\mathsf{loc} = \mathit{key}(\text{WRITETX}) = \mathit{key}(\text{WRITETX}')$. It is then possible that a set of $f + 1$ correct authorities process WRITETX and lock $\mathsf{LockDb}[\mathsf{loc}] \leftarrow (\mathsf{PrevCert}, \text{VOTE})$ (Line 20 of Algorithm 1), and the other f correct authorities process $\text{WRITETX}'$ and lock $\mathsf{LockDb}[\mathsf{loc}] \leftarrow (\mathsf{PrevCert}, \text{VOTE}')$. As a result, there may never be a certificate neither over WRITETX nor over $\text{WRITETX}'$. The store entry loc is then effectively locked forever.

Arke allows unlocking loc at the end of every epoch by dropping all locks. That is, authorities forget all votes they issued during the epoch. Authorities set $\mathsf{LockDb}[\mathsf{loc}] \leftarrow (\mathsf{PrevCert}, \mathsf{None})$ for every entry in their store[8]. Intuitively, dropping all locks at epoch change is safe because the check (2.3) of Algorithm 2 ensures certificates are only valid for a single epoch (see Section C).

Storage cleanup. One of the main properties of Arke is its ability to clean up storage after long periods of inactivity. Correct authorities delete keys that have not been updated in the last E epochs. That is, they drop the store entries $\mathsf{LockDb}[\mathsf{loc}]$ for every entry loc associated with a certificate CERT where $\mathit{epoch}(\text{CERT}) + E < \mathsf{Epoch}$ (where Epoch is the current epoch). This operation is performed asynchronously and lazily at runtime to avoid the cost of iterating through the store upon epoch change. Upon loading the latest certificate from storage (Line 13 Algorithm 2), the store LockDb returns None if $\mathsf{PrevCert}$ should be deleted. Intuitively, this operation is safe (see Section C) because readers only consider a certificate CERT if $\mathit{epoch}(\text{CERT}) + E > \mathsf{Epoch}$ (check (3.1) of Algorithm 3), and it preserves liveness because readers and correct authorities are in the same epoch Epoch for a duration $\delta > 0$ (i.e., correct authorities have roughly synchronized clocks, see Section II-D).

D. Scaling the Arke Store

Arke scales and achieves high performance with two main strategies: (i) authorities can process multiple transactions and certificates in parallel, and (ii) they can take advantage of more hardware to further increase throughput.

Scaling on multiple cores. Algorithm 1 and Algorithm 2 are designed to take advantage of all the CPU cores available on the authority machine. This is achieved by taking a simple guard on the store entry to update (rather than on the entire state) and processing non-conflicting updates in parallel. Both functions PROCESSWRITETX (Algorithm 1) and PROCESSCERT (Algorithm 2) can be called by multiple tasks.

Scaling on multiple machines. storage authorities can scale and arbitrarily increase their throughput by using more hardware. That is, rather than limiting each authority to operate on a single server, they could operate on a rack or even an entire data center. Arke requires no state sharing between the machines of the authority and thus allows for a very efficient sharding at each authority by key. Each machine is responsible to handle write, sync, and read operations only on a predefined subset of the keys. The consistent broadcast channel implementing the write operation is executed on a per-entry basis. Therefore, the protocol does not require any state sharing between shards. Section VI illustrates how storage authorities take advantage of multiple machines to linearly increase their throughput.

E. Crash Faults Only

This store can be easily converted to only tolerate crash faults rather than more general Byzantine faults. Since the protocol is essentially leaderless, it does not require any leader-rotation sub-protocol (contrarily to typical Paxos and Raft-based protocols) and can be simply converted by removing signatures from each protocol message (Appendix B-A). The system can then operate with a committee of $2f + 1$ (rather than $3f + 1$) and tolerate up to f faults.

APPENDIX C
CUSTOM STORE PROOFS

We argue that Arke store presented in Section V-A and Section B satisfies the security properties defined in Section II-C under the assumptions defined in Section II-D.

A. Validity

The validity of Arke relies on assumption 2 (BFT) and assumption 3 (cryptography) defined in Section II-D. Arke can avoid relying on the BFT assumption for validity if we augment Algorithm 2 (Appendix B-B) to (re-)run Check (1.1) of Algorithm 1 upon processing certificates (Appendix B-B).

Authenticated writes. We start by showing that users can only update the Arke store at locations associated with their own username. That is, malicious users cannot interfere with the discovery protocol of other users.

Lemma 4. *No correct storage authority issues a vote* VOTE *over a transaction* WRITETX *writing the Arke store at a location* $\mathsf{loc}_{BC} = g_1^{t_{BC}}$ *if the transaction's author does not know* t_{BC}.

Proof: Check (1.1) of Algorithm 1 requires the user to prove knowledge of t_{BC} (through a digital signature); otherwise WRITETX is ignored and the protocol returns an error. ■

Lemma 5. *No correct storage authority issues a vote* VOTE *over a transaction* WRITETX *generated by user A (known by username* id_A*) writing the Arke store at a location* loc_{BC} *derived from the usernames* id_B *(of user B) and* id_C *(of user C), with* $\mathsf{id}_A \neq \mathsf{id}_B \neq \mathsf{id}_C$.

Proof: Let's assume a correct authority issues a vote VOTE over WRITETX writing the Arke store at a location $\mathsf{loc}_{BC} = g_1^{t_{BC}}$. The privacy property of the Arke key-derivation protocol (Theorem 1) along with the collision-resistance of the hash-function H (assumption 3, see Section II-D) ensures only users B and C can obtain t_{BC}. As

[8] This operation may be performed lazily at runtime to avoid the cost of iterating through the store upon every epoch change.

a result, user A generated WRITETX without the knowledge of t_{BC} and a correct authority issued VOTE over WRITETX. This is however a direct contradiction of Lemma 4. ■

Theorem 3 (Authenticated Writes). *No user A (known by username id_A) can generate a transaction* WRITETX *that updates the store of correct storage authorities at a location loc_{BC} derived from the usernames id_B (of user B) and id_C (of user C), with $\mathsf{id}_A \neq \mathsf{id}_B \neq \mathsf{id}_C$.*

Proof: Let's assume a correct storage authority updates its storage at location loc_{BC} as specified by WRITETX. The Arke store is only updated by Algorithm 2 (Line 16) upon processing a valid certificate (Check (2.1)). User A thus obtains a valid certificate CERT over WRITETX. The BFT assumption (assumption 2, see Section II-D) ensures there are at most f Byzantine authorities; user A thus obtained at least $f+1$ votes over WRITETX from correct storage authorities. This is however a direct contradiction of Lemma 5 (ensuring that no correct authorities issue a vote over WRITETX). ■

Replay prevention. Theorem 3 ensures that no malicious user A can generate a transaction to update the Arke at locations unrelated to its username. We now show Arke withstands replays of old certificates (generated by correct users). This is particularly important as the storage authorities may drop part of their LockDb store upon cleanup (Appendix B-C).

Theorem 4 (Deliver-Once). *Once a correct storage authority processes a (valid) certificate* CERT*, it does not update its* LockDb *storage with a certificate* CERT$'$ *older than* CERT.

Proof: Let's assume a storage authority stores CERT$'$ in its LockDb store (Line 16 of Algorithm 2) after it processed CERT. Since CERT$'$ is older than CERT, it follows that either (i) *epoch*(CERT) $>$ *epoch*(CERT$'$), or (ii) *version*(CERT) $>$ *version*(CERT$'$). In case (i), Check (2.3) of Algorithm 2 ensures the authority stops processing CERT$'$ and returns an error. In case (ii), Check (2.4) of Algorithm 2 ensures the authority ignores CERT$'$ and does not update its LockDb storage. As a result, there are no scenarios where a correct storage authority updates its LockDb with CERT$'$ after processing CERT, hence a contradiction. ■

B. Consistency

We show the consistency properties of Arke described in Section II-C, namely *write consistency* and *read consistency*. These properties heavily rely on assumption 2 (BFT), assumption 3 (cryptography), and assumption 5 (roughly synchronized clocks) defined in Section II-D. The lemmas and theorems of this section implicitly assume that no adversary can forge a vote (assumption 2 (cryptography)).

Lemma 6 (BCB Consistency). *No two conflicting transactions, namely transactions sharing the same storage location* loc*, version* $\mathsf{Version}$*, and epoch* Epoch*, are certified.*

Proof: The proof of this lemma directly follows from the consistency property of Byzantine consistent broadcast (BCB) over the label $(\mathsf{loc}, \mathsf{Version}, \mathsf{Epoch})$ [23]. Let's assume two conflicting transactions WRITETX$_A$ and WRITETX$_B$ taking as input the same storage location loc with version $\mathsf{Version}$ are certified during the same epoch Epoch. Then $f+1$ correct storage authority performed (1.3), Check (1.4), and Check (1.5) of Algorithm 1 and produced VOTE$_A$ over WRITETX$_A$; and $f+1$ correct storage authority did the same and produced VOTE$_B$ ove WRITETX$_B$. Correct storage authorities reject transactions for older epochs (Check (1.3)) and with versions older than their latest certificate (Check (1.4)). Both WRITETX$_A$ and WRITETX$_B$ thus contain the current epoch and a version higher than the latest certificate known to the authority. Finally, a correct storage authority performs the check (1.5) and does not successfully process both (conflicting) WRITETX$_A$ and WRITETX$_B$; it instead returns an error at Line 22. As a result, a set of $f+1$ correct storage authority produced VOTE$_A$ but not VOTE$_B$, and a distinct set of $f+1$ correct storage authority produced VOTE$_B$ but not VOTE$_A$. Hence there should be $f+1+f+1 = 2f+2$ correct storage authority additionally to the f byzantine. However $N = 3f+1 < 3f+2$ hence a contradiction. ■

Lemma 6 operates over the label $(\mathsf{loc}, \mathsf{Version}, \mathsf{Epoch})$ rather than only $(\mathsf{loc}, \mathsf{Version})$ because check (1.5) of Algorithm 1 relies on the integrity of the votes stored in LockDb. These votes may however be dropped upon epoch change (Appendix B-C). There can thus exist multiple certificates with the same $(\mathsf{loc}, \mathsf{Version})$ but different epochs. This is not a problem because certificates carry their epoch number and are only valid for the current epoch (see Check (2.3) of Algorithm 2).

Write consistency. Write consistency intuitively ensures that correct storage authorities do not hold conflicting records.

Theorem 5 (Write Consistency). *No two correct storage authorities hold conflicting certificates in their* LockDb *store. That is, two certificates sharing the same storage location, version, and epoch.*

Proof: Let's assume the LockDb store of two correct storage authorities S and S' respectively hold conflicting the certificates CERT and CERT$'$. Check (2.1) ensures correct authorities only store valid certificates in their LockDb store. This implies that authority S received the valid certificate CERT and authority S' received the valid (conflicting) certificate CERT$'$. Lemma 6 however ensures CERT $=$ CERT$'$, hence a contradiction. ■

Read consistency. Read consistency intuitively ensures that two correct users attempting to read the same storage location do not read different values.

Lemma 7 (Safe Cleanup). *No correct user reads the value c if at least one correct storage authority deletes c (upon cleanup).*

Proof: Let's assume a correct user reads c and one correct storage authority deletes c. A correct authority S at epoch e_s deletes a value c wrote at epoch e_c when

$$e_s > E + e_c \tag{12}$$

(where $E > 0$ is a system parameter, see Appendix B-C). Check (3.1) ensures correct users at epoch e_u only read c if

$$e_u < E + e_c \tag{13}$$

Furthermore, assumption 5 (roughly synchronized clocks, see Section II-D) ensures that either

$$e_u = e_s,\ e_u = e_s + 1, \text{ or } e_u = e_s - 1 \tag{14}$$

Substituting Equation (14) into Equation (12), we (conservatively) find that authority S deletes c when

$$e_u > E + e_c - 1 \tag{15}$$

Combining Equation (13) and Equation (15), we find that a correct reader only reads c when S deletes it if the two following conditions are both met:

$$\begin{cases} e_u < E + e_c, \text{ and} \\ e_u > E + e_c - 1 \end{cases}$$

There exists however no e_u (and thus no e_v) for which both conditions hold, hence a contradiction. ■

Theorem 6 (Read Consistency). *No two correct users sending a read transaction* READTX *for the same store location* loc *read two different values c and c′.*

Proof: Let's assume two correct users read the different values c and c' for the same store location loc. Users only read values from (valid) certificates (Line 11 of Algorithm 3). As a result, one correct user read c while the other read c' This either implies that (i) there exist two correct and conflicting certificates over c and c' (which would be a contradiction of Lemma 6) or (ii) that one user reads $c' =$ None after a correct authority deletes c' (which would be a contradiction of Lemma 7). ■

C. Termination

We prove the termination (liveness) properties of Arke described in Section II-C, namely *write termination* and *read termination*. These properties heavily rely on assumption 2 (BFT), assumption 3 (cryptography), assumption 4 (network model), and assumption 5 (roughly synchronized clocks) of Section II-D. The termination properties only apply to *correct* transactions and certificates defined in Definition 6 and Definition 7, respectively.

Definition 6 (Correct Write Transaction). *A correct transaction* WRITETX *is valid (see Appendix B-B), contains the expected version, and does not non-equivocates (i.e., it is the only transaction over the triple* (loc, *Version*, *Epoch*)*).*

Definition 7 (Correct Certificate). *A correct certificate* CERT *is valid (see Appendix B-B) and contains the highest version number generated for the specific store entry it writes.*

Writer termination. Writer termination intuitively means that a correct writer can eventually update the storage authorities to make its key discoverable. The writer starts this process by submitting a transaction WRITETX manifesting its intent to make its key discoverable. Arke considers the key discoverable when $f + 1$ correct storage authorities hold a certificate over WRITETX.

The following lemmas assume the existence of a correct synchronizer. As discussed in Section V-A such synchronizer does not need the knowledge of any secret and can be implemented by the writer or by correct storage authorities (in which case its existence is implied by assumption 2 (BFT) of Section II-D).

Lemma 8 (WRITETX Availability). *If a correct user submits a transaction* WRITETX *to the storage authorities, a correct synchronizer eventually learns* WRITETX.

Proof: A correct user terminates the process of submitting WRITETX when a set $\{S\}$ of $2f+1$ storage authorities receive WRITETX (see Section II-B). The synchronizer queries all $(3f + 1)$ storage authorities and waits for the first $2f + 1$ replies. Since at most f of those authorities are Byzantine (assumption 2 (BFT), see Section II-D), the synchronizer is guaranteed to receive a set $\{S'\}$ of $2f+1$ replies. By quorum intersection, at least one correct authority is part of both $\{S\}$ and $\{S'\}$ and thus delivers WRITETX to the synchronizer. ■

Lemma 9. *During periods of synchrony, a correct synchronizer can obtain a certificate* CERT *over a correct transaction* WRITETX.

Proof: The proof of this lemma directly follows from the termination property of Byzantine consistent broadcast (BCB) [23]. The synchronizer first disseminates WRITETX to all $(3f + 1)$ storage authorities. Since WRITETX is valid, Check (1.1) succeeds. Check (1.2) always passes for the first copy of WRITETX received by the authority (at any given time). During periods of synchrony, assumption 4 (network) and assumption 5 (roughly synchronized clocks) ensure Check (1.3) succeeds; indeed correct authorities receive WRITETX during the same epoch Epoch of its generation and remain sufficiently long in epoch Epoch. Check (1.4) passes since WRITETX contains the next expected version number. Finally, correct transactions do not equivocate; thus WRITETX is the first and only transaction accessing a particular storage location, and always passes Check (1.5). Since all checks pass, the BFT assumption (assumption 2 (BFT)) ensures that at least $2f + 1$ authorities reply with a vote VOTE over WRITETX. The synchronizer then locally aggregates these votes into a certificate CERT. ■

Lemma 10. *During periods of synchrony, at least $f + 1$ correct storage authorities at epoch* Epoch *can hold a correct certificate* CERT *over a transaction* WRITETX *generated at epoch* Epoch *if a correct synchronizer holds* CERT.

Proof: The synchronizer repetitively disseminates CERT to all $(3f + 1)$ storage authorities until it receives acknowledgments from a set $\{S\}$ of $2f + 1$ authorities. Correct authorities always acknowledge the receipt of CERT. Indeed, Check (2.1) passes since CERT is valid, and Check (2.2) always passes for the first copy of CERT received by the authority (at any given time). During periods of synchrony, assumption 4 (network) ensures Check (1.3) succeeds; indeed the authorities receive CERT during epoch Epoch. Finally, Check (1.4) passes since CERT is correct and thus contains the highest version generated for its store entry. Since $\{S\}$ contains at most f Byzantine authorities (assumption 2, BFT), the remaining $f + 1$ storage authorities of $\{S\}$ are correct and thus hold CERT. ■

Theorem 7 (Writer Termination). *During periods of synchrony, if a correct writer submits a correct transaction* WRITETX *(generated at epoch* Epoch*), at least $f+1$ correct storage authorities eventually receive a certificate* CERT *over*

WriteTx.

Proof: During periods of synchrony, assumption 4 (network) ensures a correct synchronizer manages to perform the following steps within the same epoch Epoch; and assumption 5 (roughly synchronized clocks) ensures correct authorities remain sufficiently long in epoch Epoch. (i) A correct synchronizer obtains WriteTx after the correct writer submits it to the storage authorities (Lemma 8). (ii) The synchronizer obtains a certificate Cert over WriteTx (Lemma 9). (iii) The synchronizer disseminates Cert to the storage authorities; Lemma 9 ensures a least $f+1$ correct storage authorities hold Cert. ■

Theorem 7 mentions that writer termination is only guaranteed during periods of synchrony where the synchronizer manages to complete the synchronization protocol within the epoch of the transaction's generation. Assumption 4 (network) ensures that a period of synchrony eventually happens; a correct user generates and submits its transaction every epoch until then. This is not a practical limitation as Arke's epochs are long (e.g., 10 days) and the protocol is responsive [95] (i.e., it does not need to wait until the end of each epoch to make progress).

Reader termination. Reader termination guarantees that a user B can eventually discover the key of user A if (i) user A made its key discoverable to user B, and (ii) user B knows the username id_A of user A.

Lemma 11. *During periods of synchrony, if $f+1$ correct storage authorities hold a certificate* Cert *over the key values* (loc, c) *(with $c \neq \mathsf{None}$), a user knowing* loc *can eventually read c.*

Proof: The user continuously queries all $(3f+1)$ storage authorities at location loc until it receives $2f+1$ valid replies (that is, replies passing Check (3.1)). Under assumption 2 (BFT), quorum intersection ensures at least one of those replies originated from a correct storage authority holding Cert. The user then parses Cert to obtain c. During periods of synchrony (assumption 4, network), the steps above run before storage cleanup and thus $c \neq \mathsf{None}$. ■

Theorem 8 (Read Termination)**.** *During periods of synchrony, A correct user B can eventually discover the key pk_A of user A known by username id_A if (i) user A made pk_A discoverable to user B, and (ii) user B knows the username id_A.*

Proof: From condition (i) it follows that user A derived the shared key k and the writing tag t_{AB}, and submitted a transaction WriteTx to write the key-value

$$(\mathsf{loc}_{AB}, c_{AB}) = (g_1^{t_{AB}}, \mathsf{AEAD}_k(\mathsf{pk}_A))$$

to the storage authorities. Theorem 7 then ensures $f+1$ correct storage authorities hold a certificate Cert over WriteTx. Condition (ii) indicates that user B knows id_A; by definition of ID-NIKE (Section IV-B) user B can also derive the same shared key k and the writing tag t_{AB}; user B can thus compute $\mathsf{loc}_{AB} = g_1^{t_{AB}}$. Under assumption 4 (network), Lemma 11 ensures user B can use loc_{AB} to eventually retrieve Cert before storage cleanup. Finally, user B uses the shared k to decrypt $c_{AB} = \mathsf{AEAD}_k(\mathsf{pk}_A)$ (embedded into Cert) and recover pk_A. ■

Theorem 8 guarantees reader termination only during periods of synchrony. This assumption is necessary for the proofs since storage authorities clean up their storage after a fixed number of epochs. This assumption is however overly theoretical as store entries are only deleted after several months.

Key discovery termination. Theorem 9 argues that correct users eventually succeed in running the setup phase (Section IV) and obtain long-term credentials over a username they own.

Theorem 9 (Key Discovery Termination)**.** *A correct user A owning username id_A can eventually receive the long-term credentials $(H_1(\mathsf{id}_A)^s, H_2(\mathsf{id}_A)^s)$.*

Proof: This theorem is proven by construction on the setup protocol described in detail in Section IV. The user first proves ownership of id_A and receives an attestation from the KYC provider. The user then continuously sends this attestation to all $(3f+1)$ credentials authorities. Assumption 2 (BFT) ensures the user eventually receives $2f+1$ partial long-term credential $\{(H_1(\mathsf{id}_A)^{s_i}, H_2(\mathsf{id}_A)^{s_i})\}$, $i \in [0, \ldots, 2f+1]$ (algorithms defined in Definition 3). The user then aggregates those partial long-term credentials into a consolidated long-term credential $(H_1(\mathsf{id}_A)^s, H_2(\mathsf{id}_A)^s)$ using Lagrange interpolation (see algorithm *Combine* of Definition 3). ■

Appendix D
Sui Move Arke Store

This section complements Section V-B by presenting a Sui move contract implementing an Arke store using exclusively owned objects. As a result, this contract does not require consensus and can operate exclusively throughput the consensusless path of Sui.

```
module arke::arke {
    use sui::tx_context::{TxContext};
    use sui::object::{Self, UID};
    use sui::transfer;

    /// A discovery object holding a cipher.
    struct Discovery has key, store {
        id: UID,
        cipher: vector<u8>
    }

    /// Initialize a discovery object with a cipher and
         transfer it to a specific address.
    entry fun write(cipher: vector<u8>, addr: address,
         ctx: &mut TxContext) {
        let discovery = Discovery {
            id: object::new(ctx),
            cipher: cipher
        };
        transfer::transfer(discovery, addr);
    }

    /// Delete the discovery object when it is no longer
         needed.
    entry fun delete(discovery: Discovery) {
        let Discovery { id, cipher: _ } = discovery;
        object::delete(id);
    }
}
```

The contract starts by defining a `Discovery` object holding a cipher c_{AB}. It then exposes two functions, `write` and `delete`. User A writes the store by calling the `write` function parametrized with the cipher c_{AB} and an address

$addr_{AB}$ uniquely derived from the key loc_{AB} (see Section V-B); the function creates a discovery (owned) object holding c_{AB} and transfers its ownership to $addr_{AB}$. User B reads the blockchain by locally deriving $addr_{AB}$ and querying all objects owned by that address; the query will return the discovery object created by user A. For good hygiene, both users A and B can delete the object when no longer needed by calling the `delete` function.

Altnerative implementation. This contract can alternatively be implemented through events (no objects); every party emits an event that is read from the blockchain by the other party. This implementation is cheaper as it does not involve object mutation and does not require state cleanup. However, the client software will have to rely on full nodes to relate these events and manually verify them through specific message sequence numbers (to detect selective censorship) and integrity checks. In contrast, the object-based implementation depicted above is slightly more expensive but it is easier to implement and verify as it does not require any additional logic on the client side.