CALYPSO: Auditable Sharing of Private Data over Blockchains

Eleftherios Kokoris-Kogias*, Enis Ceyhun Alp*, Sandra Deepthy Siby*, Nicolas Gailly*,
Linus Gasser*, Philipp Jovanovic*, Ewa Syta†, Bryan Ford*

*École polytechnique fédérale de Lausanne, Switzerland
†Trinity College, USA

***Abstract*—Securely sharing confidential data over a distributed ledger with a fully decentralized and efficient access-control mechanism is a non-trivial challenge to solve. Current blockchain systems either do not support such a functionality or fall back to semi-centralized solutions that provide storage and access control for sensitive data off-chain. In this work we present CALYPSO, the first fully decentralized, auditable access-control framework for secure blockchain-based data sharing which builds upon two abstractions. First, *on-chain secrets* enable collective management of (verifiably shared) secrets under a Byzantine adversary where an access-control blockchain enforces user-specific access rules and a secret-management cothority administrates encrypted data. Second, *skipchain-based identity and access management* enables efficient administration of dynamic, sovereign identities and access policies and, in particular, permits clients to maintain long-term relationships with respect to evolving user identities thanks to the trust-delegating forward links of skipchains. The evaluation of our CALYPSO implementation shows that the latency for processing read and write requests scales linearly with the number of secret-management trustees and is in the range of 0.2 to 8 seconds for 16 to 128 trustees. Lastly, three specific deployments of CALYPSO illustrate its feasibility and applicability to data-sharing problems faced by real-world organizations.**

I. INTRODUCTION

The massive aggregation of user-generated data by centralized service providers combined with the lack of mechanisms for users to audit and control the usage of their data presents a continuous threat to the privacy of billions [55]. This threat can turn into concrete perils for democracy itself as illustrated by the Facebook scandal [33], where Cambridge Analytica allegedly exploited personally identifiable information from millions of Facebook users to influence voter opinion in the 2016 US presidential election. New data privacy legislation, such as the European Union General Data Protection Regulation (GDPR) [10], is an important step towards addressing data abuses, however, it needs to be complemented by modern technological solutions that put users back into control.

Decentralized trust technologies have gained significant traction due to the success of cryptocurrencies such as Bitcoin [39] and Ethereum [56]. While they present a promising path forward to address the above data sharing challenges, they are not without issues. Blockchain-based applications [34], [42], [51], [57] often assume a shared access to sensitive data between independent and mutually distrustful parties but unfortunately fail to manage private data securely over the blockchain. These applications either ignore the problem altogether [24], [58], fall back to naive solutions [15] of simply publishing encrypted data on Bitcoin, or utilize semi-centralized approaches [3], [20], [59], where access information and hashes of the data are put on-chain but the secret data itself is not only stored but also managed off-chain in centralized or distributed storage [50] systems, creating single points of compromise or failure.

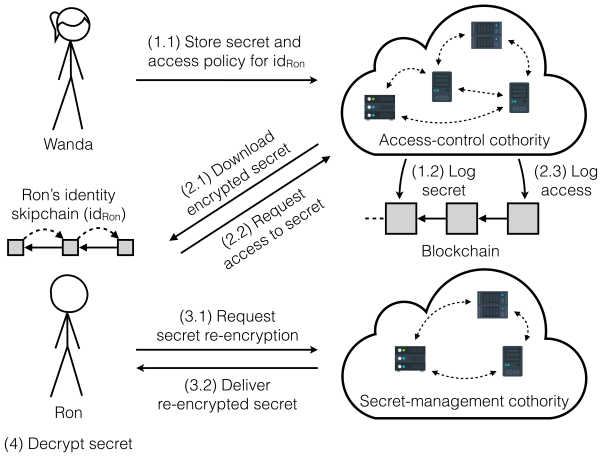

Fig. 1. Private, auditable data sharing in CALYPSO: (1) Wanda encrypts data under the secret-management cothority's key, specifying Ron as the authorized reader in the policy, and sends it to the access-control cothority which verifies and logs the information. (2) Ron downloads the encrypted secret from the blockchain and then requests access to it from the access-control cothority which logs the query if valid. (3) Ron asks the secret-management cothority for the secret shares of the key needed to decrypt the secret by proving that the previous authorization was successful. (4) Ron decrypts the secret.

From these observations, we identified the following key challenges that a secure blockchain-based data sharing system with access-control support should address: (1) Enable data owners to share and maintain control over their data and, in particular, enable them to update access-control rules, *e.g.*, to add or revoke access rights, retroactively. (2) Provide auditability of all data accesses, *e.g.*, to be able to hold misbehaving users accountable. (3) Allow users to update their identities without losing access to already shared data. (4) Ensure atomic execution of data access requests as well as identities and access rights updates, *e.g.*, to prevent participants from exploiting race conditions to gain unauthorized access to confidential data. (5) Support efficient data sharing with individual users and groups of users. (6) Prevent any single point of compromise or failure in the system.

In this paper we introduce CALYPSO, a new secure blockchain-based data sharing framework with access-control support that addresses the above challenges. Figure 1 provides an overview of CALYPSO which consists of two main components called *on-chain secrets* (OCS) and *skipchain-based identity and access (control) management* (SIAM).

On-chain secrets combine threshold cryptography [45], [47], [49] and blockchain technology [28], [56] to enable clients to securely share their encrypted data and the corresponding access-control policies with *collective authorities* (cothorities) responsible for enforcing access-control policies and delivering data to authorized readers. When a reader requests access to an on-chain secret, he provides a cryptographic authorization proof. CALYPSO atomically logs both request and proof to guarantee auditability and then delivers the information necessary for the reader to access the secret. We present two specific implementations of on-chain secrets, *one-time secrets* and *long-term secrets*, which have different trade-offs with respect to computational and storage overheads as well as the functionality they provide. Finally, we present two extensions to on-chain secrets that ensure user anonymity and provide post-quantum security, respectively.

Skipchain-based identity and access management supports both personal and federated identities and fine-grained resource access policies. It enables users to verifiably update their identities (public keys) and to efficiently share secrets even with large groups of people under custom rules without the need to create individual transactions. SIAM uses skipchains [40] to securely and efficiently enable dynamic identity and policy updates and revocation. Lastly, SIAM protects against race conditions by serializing identity and policy updates with data access requests over the blockchain.

To evaluate CALYPSO, we implemented a prototype in Go and ran experiments on commodity servers. We implemented both versions of on-chain secrets and show that they have a moderate overhead of 0.2 to 8 seconds for cothorities of 16 and 128 trustees, respectively, overall scaling linearly in the number of trustees in the cothorities. We also implemented SIAM and show that the overheads of dynamic identities and access policies are negligible compared to the static versions. Finally, we also discuss several deployments of CALYPSO that address the data sharing needs of real-world organizations.

In summary, this paper makes the following contributions.

- We introduce CALYPSO, a decentralized framework for auditable access control of protected resources over a distributed ledger that maintains confidentiality of the shared data and enables participants to update access rules for their resources even after they have been released.
- We present on-chain secrets and its two concrete implementations, one-time and long-term secrets, that enable fully transparent and efficient management of secret data over a blockchain via threshold cryptography.
- We introduce skipchain-based identity and access management for third-party verifiable, dynamic and sovereign identities and updateable access policies.
- We demonstrate the feasibility of using CALYPSO to address the data sharing needs of actual organizations by presenting three concrete deployments, auditable invoice issuing, clearance-enforcing document sharing, and patient-centric medical data sharing. We also provide an open-source implementation and evaluation results.

II. BACKGROUND

A. Blockchains and Skipchains

A blockchain is a distributed, append-only and tamper-evident log that is composed of blocks which are joined together via cryptographic hashes. Blockchains are used in many decentralized applications [7], [39]. CALYPSO can be deployed on top of any blockchain that supports programmability (*e.g.*, smart contracts [2], [54], [56]) which enables us to perform custom validation of transactions before logging.

In Section V, we introduce *identity skipchains* which are based on the skipchains data structure [40] that is similar to a simple blockchain but doubly-linked. Skipchains (Figure 3) track configuration changes of a well-known decentralized authority (a cothority [53]) by using every block as a representation of all public keys of the cothority necessary to authenticate the next block. When the cothority wants to alter its configuration, it creates a new block that includes the new set of public keys and signs it with the old set of public keys delegating the authority to the new set. This signature is a *forward link* [27] which clients follow through the skipchain to get up-to-date with the current authoritative group.

B. Threshold Cryptosystems

A (t, n)-secret sharing scheme [4], [47] enables a dealer to share a secret s among n trustees such that any subset of t trustees can reconstruct s, whereas smaller subsets cannot. Hence, the sharing scheme can withstand up to $t-1$ malicious participants. The downside of simple secret sharing schemes is that they assume an honest dealer. Verifiable secret sharing (VSS) [12] solves the problem by enabling the trustees to verify that the shares distributed by the dealer are consistent. VSS has a wide range of applications such as threshold signing and threshold encryption, described below. Finally, publicly verifiable secret sharing (PVSS) [45] is a variation of VSS that enables any external third-party to verify distributed shares.

Once we are able to securely share and hold a collective secret, we can construct more complex systems out of it. A distributed key generation (DKG) [16] protocol allows to create a collective public-private key pair without any dependence on a trusted dealer. Each trustee runs their own secret sharing protocol in parallel to contribute their own secret towards the key pair. The trustees validate each run, reach a consensus on validly shared secrets, and finally combine all shares to generate a private-public key pair $(\mathrm{sk}, g^{\mathrm{sk}})$ such that the public key $\mathrm{pk} = g^{\mathrm{sk}}$ is known to everyone whereas the private key sk is not known to any single trustee and can only be used when a threshold of them collaborates.

After the key pair is generated, anyone can encrypt data under the collective public key, for example, by using a threshold version of ElGamal [9]. For CALYPSO we use this threshold ElGamal cryptosystem in combination with non-interactive zero knowledge (NIZK) proofs [36], [49] (see Appendix B) to protect against chosen ciphertext and malleability attacks.

III. Calypso Overview

In this section we present an overview of Calypso. We start with a strawman solution called SimpleShare to motivate the challenges that any secure, decentralized data sharing system with access control like Calypso should address. We then outline the system goals that we derive from our observations about SimpleShare, describe system and threat models, and finally present a high-level overview on the secret sharing process in Calypso as outlined in Figure 1.

A. Strawman Data Sharing Solution

SimpleShare consists of a tamper-resistant public log, such as Bitcoin's blockchain, and a centralized identity provider, such as a PGP key server. As a motivation, consider a situation where Wanda wants to asynchronously share some secret data with Ron, *i.e.*, Wanda and Ron do not have to be online concurrently for the exchange and to guarantee that the data is available for Ron to retrieve at any point.

In SimpleShare, Wanda first needs to get Ron's public key from Ron's identity provider (a PGP server), then encrypt the secret under Ron's key, and finally post the ciphertext on the public blockchain to ensure availability. Later on, Ron can download the ciphertext from the blockchain and decrypt the data using his private key.

SimpleShare already provides functionality similar to what we envision for Calypso but has several drawbacks. (1) Once the encrypted data gets published, Wanda loses control over its access policy, *e.g.*, she cannot revoke access later since the ciphertext is encrypted under Ron's public key and publicly available. (2) Wanda does not know if Ron ever accessed the data, *i.e.*, there is no accountability of data accesses and hence Ron has plausible deniability should the data be misused. (3) If Wanda wants to share the same data with multiple people, she would need to encrypt it under each individual public key and publish all resulting ciphertexts. (4) If Ron wants to change his public key, *e.g.*, if his private key is compromised, he would lose access to the previously posted ciphertexts unless Wanda re-publishes the data using Ron's new key. (5) The identity provider is a single point of compromise or failure.

To address these issues, we introduce two components to SimpleShare thereby transforming it into Calypso.

1) To enable auditability of data accesses and ensure atomic data delivery, we introduce *on-chain secrets* (OCS) in Section IV.
2) To enable decentralized, dynamic user-sovereign identities and access policies, we introduce *skipchain-based identity and access management* (SIAM) in Section V.

B. System Goals

Calypso has the following primary goals.

1) **Confidentiality of Secrets:** Secrets stored on-chain can only be decrypted by authorized clients.
2) **Auditability:** All access transactions are third-party verifiable and recorded in a tamper-resistant log (blockchain).
3) **Atomic Data Delivery:** A client is guaranteed to receive a secret they are authorized for if any only if they posted an access request for that secret on the blockchain.
4) **Dynamic Sovereign Identities:** Users or organizations fully control their identities (public keys) and can update them in a third-party verifiable way.
5) **Decentralization:** No single point of compromise or failure.

C. System Model

There are four main entities in Calypso: *writers* who put secrets on-chain, *readers* who retrieve secrets, an *access-control collective authority* who is responsible for logging write and read transactions on-chain enforcing access control for secrets, and a *secret-management collective authority* who is responsible for managing and delivering secrets. In the rest of the paper we use Wanda and Ron to refer to a (generic) writer and reader, respectively. A collective authority or cothority is an abstract decentralized entity that is responsible for some authoritative action. For example, the set of Bitcoin miners can be considered a cothority that is maintaining the consistency of Bitcoin's state. We call the nodes of a cothority *trustees* and remark that these nodes do not need to be agreed upon and individually publicly known.

The access-control cothority requires a Byzantine fault-tolerant consensus [28], [29], [31], [39]. There are various ways to implement an access-control cothority, *e.g.*, as a set of permissioned servers that maintain a blockchain through BFT consensus or as an access-control enforcing smart contract on top of a permissionless cryptocurrency such as Ethereum. In our implementation, see Section VIII, we use BFT-based consensus due to its performance benefits [28].

The secret-management cothority may be set up on a per-secret basis or in a more persistent manner, the differences of which are discussed in Section IV. The secret-management trustees maintain their private keys and may need to maintain additional secret state, such as private key shares. They do not run consensus for every transaction.

We denote private and public key pairs of Wanda and Ron by $(\mathrm{sk}_W, \mathrm{pk}_W)$ and $(\mathrm{sk}_R, \mathrm{pk}_R)$. Analogously, we write $(\mathrm{sk}_i, \mathrm{pk}_i)$ to refer to the key pair of trustee i. To denote a list of elements we use angle brackets, *e.g.*, we write $\langle \mathrm{pk}_i \rangle$ to refer to a list of public keys $\mathrm{pk}_1, \ldots, \mathrm{pk}_n$. We assume that there is a registration mechanism through which writers have to register their public keys pk_W with the blockchain before they can start any secret-sharing processes. We denote an access-control label by policy, with $\mathrm{policy} = \mathrm{pk}_R$ being the simplest case where Ron is the only intended reader of Wanda's secret.

D. Threat Model

We make the usual cryptographic assumptions, namely that the adversary is computationally bounded, that cryptographically secure hash functions exist, and that there is a cyclic group $\mathbb{G}$ (with generator g) in which the decisional Diffie-Hellman assumption holds. We assume that participants,

including cothority trustees, verify the signatures of the messages they receive and process those that are correctly signed.

In the respective cothorities, we denote the total number of trustees by n and those that are malicious by f. Depending on the consensus mechanism that is used for the blockchain underlying the access-control cothority, we either require an honest majority $n = 2f+1$ for Nakamoto-style consensus [39] or $n = 3f + 1$ for classic BFT consensus [28]. In the secret-management cothority, we require $n = 2f + 1$ and set the threshold to recover a secret to $t = f + 1$. To ensure a proper ratio of honest and dishonest trustees, CALYPSO can use unbiasable publicly verifiable randomness [52] to select trustees from a larger pool of available nodes.

We emphasize the importance of ensuring consensus on any transaction included in the blockchain before the trustees hand out the corresponding inclusion proofs. This is especially important for Nakamoto-style consensus where forks might still occur within the last few blocks.

We assume that readers and writers do not trust each other. We assume that writers encrypt the correct data and share the correct symmetric key with the secret-management cothority, as readers can release a protocol transcript and prove the misbehavior of writers. Conversely, readers might try to get access to a secret and claim later that they have never received it. Additionally, writers might try to frame readers by claiming that they shared a secret although they have never done so.

We guarantee data confidentiality up to the point where an authorized reader gains access. To maintain confidentiality after this point, writers may rely on additional privacy-preserving technologies such as differential privacy [5] or homomorphic encryption [11]. Combining these techniques with CALYPSO is beyond the scope of this paper.

E. Architecture Overview

On a high level CALYPSO's enables Wanda, the writer, to share a secret with Ron, the reader, under a specific access-control policy. Figure 1 shows the general steps of sharing and retrieving secrets. When Wanda wishes to put a secret on-chain, she prepares an access-control policy, encrypts the secret and then sends a write transaction ($\mathrm{tx_w}$) to the access-control cothority. The access-control cothority verifies and then logs $\mathrm{tx_w}$ making the secret available for retrieval by the authorized reader Ron. To request access to a secret, Ron downloads the secret from the blockchain and then sends a read transaction ($\mathrm{tx_r}$) to the access-control cothority. If Ron is authorized to access the requested secret, the access-control cothority logs $\mathrm{tx_r}$. Subsequently, Ron contacts the secret-management cothority to recover the secret. The secret-management trustees verify Ron's request against the blockchain and then deliver the secret shares of the key needed to decrypt Wanda's secret as shared in $\mathrm{tx_w}$.

IV. ON-CHAIN SECRETS

In this section we introduce two on-chain secrets protocols, *one-time secrets* and *long-terms secrets*. Figure 2 provides an overview of on-chain secrets. The assumptions listed in the previous section apply to both protocols. We also present two protocol extensions: an on-chain blinded key exchange that allows to conceal the identities of the readers as well as a post-quantum secure version of on-chain secrets.

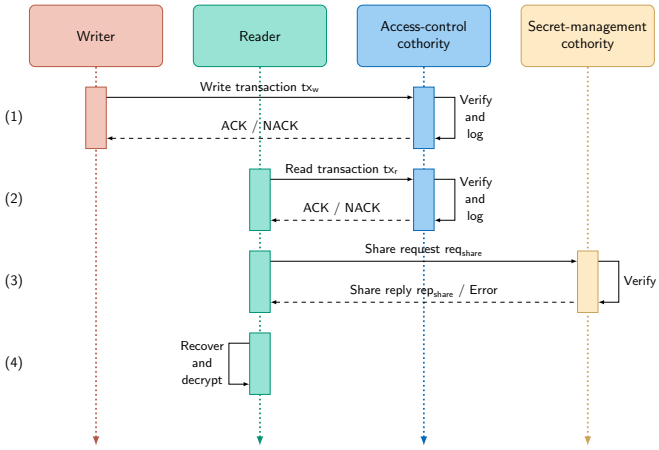

Fig. 2. On-chain secrets protocol steps: (1) Write transaction, (2) Read transaction, (3) Share retrieval, (4) Secret reconstruction.

One-time secrets uses PVSS and employs a per-secret secret-management cothority. One-time secrets' simplicity enables each $\mathrm{tx_w}$ to define a fresh, ad hoc group of secret-management trustees without any setup or coordination. This simplicity, however, comes at a price. The $\mathrm{tx_w}$ size and the encryption/decryption overhead are linear in the size of the secret-management cothority because all encrypted shares are included in the transaction and the number of secret shares is equal to the size of the secret-management cothority.

Long-term secrets, the second approach to implementing on-chain secrets, requires the secret-management cothority to perform a coordinated setup phase to generate a collective key (DKG) and to maintain a minimal state of their DKG secret shares. As a result, however, long-term secrets offers a constant encryption overhead and a flexible secret-management cothority membership through re-sharing the shared key or re-encrypting existing secrets to a new shared key.

The *on-chain private key exchange protocol* can be applied to both on-chain secrets protocols and hide the reader's identity by blinding the reader's public key in the write and read transactions. Lastly, the *post-quantum on-chain secrets* describe the modifications needed to achieve post-quantum security.

A. One-Time Secrets

One-time secrets is based on PVSS [45]. Wanda, the writer, first prepares a secret she wants to share along with a policy that lists the public key of the intended reader. She then generates a symmetric encryption key by running PVSS for the secret-management cothority members, encrypts the secret with the key she shared and then stores the resulting ciphertext either on-chain or off-chain. Finally, she sends a write transaction $\mathrm{tx_w}$ containing the information necessary for the verification and retrieval of her secret to the access-control cothority to have it logged. Ron, the reader, creates and sends

to the access-control cothority a read transaction $\mathrm{tx_r}$ for a specific secret. The trustees check $\mathrm{tx_r}$ against the secret's access policy and if Ron is authorized to access the secret, they log the transaction creating a proof of access. Ron sends this proof together with the encrypted secret shares from $\mathrm{tx_w}$ to each secret-management (PVSS) trustee and gets the secret key shares. Once Ron has received a threshold of valid shares, he recovers the symmetric key and decrypts the original data.

Write transaction protocol: Wanda, the writer and each trustee of the access-control cothority perform the following protocol to log the write transaction $\mathrm{tx_w}$ on the blockchain. Wanda initiates the protocol as follows.

1) Compute $h = \mathrm{H}(\mathrm{policy})$ to map the access-control policy to a group element h to be used as the base point for the PVSS polynomial commitments.
2) Choose a secret sharing polynomial $s(x) = \sum_{j=0}^{t-1} a_j x^j$ of degree $t-1$. The secret to be shared is $s = g^{s(0)}$.
3) For each secret-management trustee i, compute the encrypted share $\widehat{s_i} = \mathrm{pk}_i^{s(i)}$ of the secret s and create the corresponding NIZK proof $\pi_{\widehat{s_i}}$ that each share is correctly encrypted (see Appendix A). Create the polynomial commitments $b_j = h^{a_j}$, for $0 \leq j \leq t-1$.
4) Set $k = \mathrm{H}(s)$ as the symmetric key, encrypt the secret message m to be shared as $c = \mathrm{enc}_k(m)$, and compute $H_c = \mathrm{H}(c)$. Set $\mathrm{policy} = \mathrm{pk}_R$ to designate Ron as the intended reader of the secret message m.
5) Finally, prepare and sign the write transaction

$$\mathrm{tx_w} = [\langle \widehat{s_i} \rangle, \langle b_j \rangle, \langle \pi_{\widehat{s_i}} \rangle, H_c, \langle \mathrm{pk}_i \rangle, \mathrm{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$$

and send it to the access-control cothority.

The access-control cothority then logs the write transaction on the blockchain as follows.

1) Derive the PVSS base point $h = \mathrm{H}(\mathrm{policy})$.
2) Verify each encrypted share $\widehat{s_i}$ against $\pi_{\widehat{s_i}}$ using $\langle b_j \rangle$ and h (see Appendix A). This step guarantees that Wanda correctly shared the encryption key.
3) If all shares are valid, log $\mathrm{tx_w}$ in block b_w.

Read transaction protocol: After the write transaction has been recorded, Ron needs to log the read transaction $\mathrm{tx_r}$ through the access-control cothority before he can request the secret. To do so, Ron performs the following steps.

1) Retrieve the ciphertext c and b_w, which stores $\mathrm{tx_w}$, from the access-control cothority.
2) Check that $\mathrm{H}(c)$ is equal to H_c in $\mathrm{tx_w}$ to ensure that the ciphertext c of Wanda's secret has not been altered.
3) Compute $H_\mathrm{w} = \mathrm{H}(\mathrm{tx_w})$ as the unique identifier for the secret that Ron requests access to and determine the proof $\pi_{\mathrm{tx_w}}$ showing that $\mathrm{tx_w}$ has been logged on-chain.
4) Prepare and sign the transaction

$$\mathrm{tx_r} = [H_\mathrm{w}, \pi_{\mathrm{tx_w}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$$

and send it to the access-control cothority.

The access-control cothority then logs the read transaction on the blockchain as follows.

1) Retrieve $\mathrm{tx_w}$ using H_w and use pk_R, as recorded in policy, to verify the signature on $\mathrm{tx_r}$.
2) If the signature is valid and Ron is authorized to access the secret, log $\mathrm{tx_r}$ in block b_r.

Share retrieval protocol: After the read transaction has been logged, Ron can recover the secret message m by running first the share retrieval protocol with the secret-management cothority to obtain shares of the encryption key used to secure m. To do so, Ron initiates the protocol as follows.

1) Create and sign a secret-sharing request

$$\mathrm{req_{share}} = [\mathrm{tx_w}, \mathrm{tx_r}, \pi_{\mathrm{tx_r}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$$

where $\pi_{\mathrm{tx_r}}$ proves that $\mathrm{tx_r}$ has been logged on-chain.
2) Send $\mathrm{req_{share}}$ to each secret-management trustee to obtain the decrypted shares.

Each trustee i of the secret-management cothority responds to Ron's request as follows.

1) Use pk_R in $\mathrm{tx_w}$ to verify the signature of $\mathrm{req_{share}}$ and $\pi_{\mathrm{tx_r}}$ to check that $\mathrm{tx_r}$ has been logged on-chain.
2) Compute the decrypted share $s_i = (\widehat{s_i})^{\mathrm{sk}_i^{-1}}$, create a NIZK proof π_{s_i} that the share was decrypted correctly (see Appendix A), and derive $c_i = \mathrm{enc}_{\mathrm{pk}_R}(s_i)$ to ensure that only Ron can access it.
3) Create and sign the secret-sharing reply

$$\mathrm{rep_{share}} = [c_i, \pi_{s_i}]_{\mathrm{sig}_{\mathrm{sk}_i}}$$

and send it back to Ron.

Secret reconstruction protocol: To recover the secret key k and decrypt the secret m, Ron performs the following steps.

1) Decrypt each $s_i = \mathrm{dec}_{\mathrm{pk}_R}(c_i)$ and verify it against π_{s_i}.
2) If there are at least t valid shares, use Lagrange interpolation to recover s.
3) Recover the encryption key as $k = \mathrm{H}(s)$ and use it to decrypt the ciphertext c to obtain the message $m = \mathrm{dec}_k(c)$.

Achieving system goals: The one-time secrets protocol achieves all goals except for dynamic sovereign identities.

Confidentiality of Secrets. The secret message m is encrypted under a symmetric key k which is securely secret-shared using PVSS among the secret-management trustees such that $t = f + 1$ shares are required to reconstruct it. The access-control trustees verify and log on the blockchain the encrypted secret shares which, based on the properties of PVSS, do not leak any information about k. After the secret-management trustees receive a valid request $\mathrm{req_{share}}$, they respond with their secret shares encrypted under the public key listed in policy from the respective write transaction $\mathrm{tx_w}$. Further, a dishonest reader cannot obtain access to someone else's secret through a new write transaction that uses a policy that lists him as the reader but copies secret shares from another transaction in hopes of having them decrypted by the secret-management cothority. This is because each transaction is bound to a specific policy which is used to derive the base point for the PVSS NIZK consistency proofs. Without the knowledge of the decrypted secret shares (and the key k), the malicious reader cannot generate correct proofs and all transactions without valid proofs are rejected. This means that only the intended reader obtains a threshold of secret shares necessary to recover k and then access m.

Auditability. Under the assumption that the access-control cothority provides Byzantine consensus guarantees, all properly created read and write transactions are logged by the access-control cothority on the blockchain. Once a transaction is logged, anyone can verify this fact and obtain a third-party verifiable transaction inclusion proof.

Atomic Data Delivery. Once a read transaction $\mathrm{tx_r}$ is logged by the access-control cothority, the reader can run the share retrieval protocol with the secret-management cothority. Under the assumption that $n = 2f + 1$, the reader receives at least $t = f + 1$ shares of the symmetric encryption key k from the honest trustees. This guarantees that the reader has enough shares to reconstruct k and access the secret message m using the secret reconstruction protocol.

Dynamic Sovereign Identities. While all participants maintain their own private keys and hence their identities, the identities used in write transactions cannot be updated without re-encrypting the secrets and posting new write transactions.

Decentralization. The protocols do not assume a trusted third party and they tolerate up to $t - 1$ failures.

Protocol advantages and shortcomings: The one-time secrets protocol uses existing and proven to be secure building blocks and its design is simple to implement and analyze. Further, it does not require a setup phase among the secret-management members, *e.g.*, to generate a collective private-public key pair. It also enables the use of a different secret-management cothority for each secret, without requiring the servers to maintain any protocol state.

However, one-time secrets has a few shortcomings too. First, it incurs high PVSS setup and share reconstruction costs as Wanda needs to evaluate the secret sharing polynomial at n points, create n encrypted shares and NIZK proofs, along with t polynomial commitments. Similarly, Ron has to verify up to n decrypted shares against the NIZK proofs and to reconstruct the secret on his device. Second, the transaction size increases linearly with the secret-management cothority size. Because the secret-management trustees do not store any per-secret protocol state making them nearly stateless, the write transaction $\mathrm{tx_w}$ must contain the encrypted shares, NIZK proofs and the polynomial commitments. Lastly, one-time secrets does not enable re-encryption of the shares to another set of trustees, preventing the possibility of moving shares from one set of secret-management trustees to another.

B. Long-Term Secrets

Long-term secrets address the above limitations through a dedicated secret-management cothority that persists over a long period of time and that maintains a collective private-public key pair used to secure access to the secrets.

After a one-time distributed key generation (DKG) phase (see Section II for details) performed by the secret-management cothority, Wanda, the writer, prepares her secret message, encrypts it with a symmetric key and then encrypts that key with the shared public key of the secret-management cothority. As a result, the overhead of encrypting secrets is constant as each write transaction contains a single ciphertext instead of individual shares. Ron, the reader, recovers the symmetric key by obtaining a threshold of securely blinded shares of the collective private key and reconstructing the symmetric key himself or with the help of a trustee he selects. Furthermore, the configuration of the secret-management cothority can change by re-sharing the shared key or re-encrypting all the secrets to a new secret-management cothority.

Protocol setup: Before any transactions can be created and processed, the secret-management cothority needs to run a DKG protocol to generate a shared private-public key pair. There exist a number of DKG protocols that are synchronous [16] or asynchronous [25]. Given the rarity of the setup phase, we assume a pessimistic synchrony assumption for the DKG (*e.g.*, every message is posted to the access-control cothority blockchain) and implement the DKG by Gennaro et al. [16] because of its simplicity and the fact that it permits a higher threshold of corruptions.

The output of the setup phase is a collective public key $\mathrm{pk_{smc}} = g^{\mathrm{sk_{smc}}}$, where $\mathrm{sk_{smc}}$ is the unknown collective private key. Each server i holds a share of the secret key denoted as sk_i and all servers know the public counterpart $\mathrm{pk}_i = g^{\mathrm{sk}_i}$. The secret key can be reconstructed by combining a threshold $t = f + 1$ of the individual shares. We assume that $\mathrm{pk_{smc}}$ is registered on the blockchain of the access-control cothority.

Write transaction protocol: Wanda and each trustee of the access-control cothority perform the following protocol to log the write transaction $\mathrm{tx_w}$ on the blockchain. Wanda initiates the protocol through the following steps.

1) Retrieve the collective public key $\mathrm{pk_{smc}}$ of the secret-management cothority.
2) Choose a symmetric key k and encrypt the secret message m to be shared as $c_m = \mathrm{enc}_k(m)$ and compute $H_{c_m} = \mathrm{H}(c_m)$. Set $\mathrm{policy} = \mathrm{pk}_R$ to designate Ron as the intended reader of the secret message m.
3) Encrypt k towards $\mathrm{pk_{smc}}$ using a threshold variant of the ElGamal encryption scheme [49]. To do so, embed k as a point $k' \in \mathbb{G}$, pick a value r uniformly at random, compute $c_k = (\mathrm{pk}_{\mathrm{smc}}^r k', g^r)$ and create the NIZK proof π_{c_k} to guarantee that the ciphertext is correctly formed, CCA-secure and non-malleable (see Appendix B).
4) Finally, prepare and sign the write transaction

$$\mathrm{tx_w} = [c_k, \pi_{c_k}, H_{c_m}, \mathrm{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$$

and send it to the access-control cothority.

The access-control cothority then logs the write transaction.

1) Verify the correctness of the ciphertext c_k using the NIZK proof π_{c_k}.
2) If the check succeeds, log $\mathrm{tx_w}$ in block b_w.

Read transaction protocol: After $\mathrm{tx_w}$ has been recorded, Ron needs to log a read transaction $\mathrm{tx_r}$ through the access-control cothority before he can request the decryption key shares. To do so, Ron performs the following steps.

1) Retrieve the ciphertext c_m and the block b_w, which stores $\mathrm{tx_w}$, from the access-control cothority.
2) Check that $\mathrm{H}(c_m)$ is equal to H_{c_m} in $\mathrm{tx_w}$ to ensure that the ciphertext c_m of Wanda's secret has not been altered.

3) Compute $H_{\mathrm{w}} = \mathrm{H}(\mathrm{tx}_{\mathrm{w}})$ as the unique identifier for the secret that Ron requests access to and determine the proof $\pi_{\mathrm{tx}_{\mathrm{w}}}$ showing that tx_{w} has been logged on-chain.
4) Prepare and sign the read transaction

$$\mathrm{tx}_{\mathrm{r}} = [H_{\mathrm{w}}, \pi_{\mathrm{tx}_{\mathrm{w}}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$$

and send it to the access-control cothority.

The access-control cothority then logs tx_{r} as follows.

1) Retrieve tx_{w} using H_{w} and use pk_R, as recorded in policy, to verify the signature on tx_{r}.
2) If the signature is valid and Ron is authorized to access the secret, log tx_{r} in block b_r.

Share retrieval protocol: After the read transaction has been logged, Ron can recover the secret data by running the share retrieval protocol with the secret-management cothority. To do so Ron initiates the protocol as follows.

1) Create and sign a secret-sharing request

$$\mathrm{req}_{\mathrm{share}} = [\mathrm{tx}_{\mathrm{w}}, \mathrm{tx}_{\mathrm{r}}, \pi_{\mathrm{tx}_{\mathrm{r}}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$$

where $\pi_{\mathrm{tx}_{\mathrm{r}}}$ proves that tx_{r} has been logged on-chain.
2) Send $\mathrm{req}_{\mathrm{share}}$ to each secret-management trustee to request the blinded shares.

Each trustee i of the secret-management cothority responds to Ron's request as follows.

1) Get g^r and pk_R from tx_{w} and prepare a blinded share $u_i = (g^r\,\mathrm{pk}_R)^{\mathrm{sk}_i}$ with a NIZK correctness proof π_{u_i}.
2) Create and sign the secret-sharing reply

$$\mathrm{rep}_{\mathrm{share}} = [u_i, \pi_{u_i}]_{\mathrm{sig}_{\mathrm{sk}_i}}$$

and send it back to Ron.

Secret reconstruction protocol: To retrieve the decryption key k and recover the secret m, Ron performs as follows.

1) Wait to receive at least t valid shares $u_i = g^{(r+\mathrm{sk}_R)\mathrm{sk}_i} = g^{r'\mathrm{sk}_i}$ and then use Lagrange interpolation to recover the blinded decryption key

$$\mathrm{pk}_{\mathrm{smc}}^{r'} = \prod_{k=0}^{t} (g^{r'\mathrm{sk}_i})^{\lambda_i},$$

where λ_i is the i^{th} Lagrange element.
2) Unblind $\mathrm{pk}_{\mathrm{smc}}^{r'}$ to get the decryption key $\mathrm{pk}_{\mathrm{smc}}^{r}$ for c_k via

$$(\mathrm{pk}_{\mathrm{smc}}^{r'})(\mathrm{pk}_{\mathrm{smc}}^{\mathrm{sk}_R})^{-1} = (\mathrm{pk}_{\mathrm{smc}}^{r})(\mathrm{pk}_{\mathrm{smc}}^{\mathrm{sk}_R})(\mathrm{pk}_{\mathrm{smc}}^{\mathrm{sk}_R})^{-1}$$

3) Retrieve the encoded symmetric key k' from c_k via

$$(c_k)(\mathrm{pk}_{\mathrm{smc}}^{r})^{-1} = (\mathrm{pk}_{\mathrm{smc}}^{r}k')(\mathrm{pk}_{\mathrm{smc}}^{r})^{-1},$$

decode it to k, and finally recover $m = \mathrm{dec}_k(c_m)$.

Ron may delegate the costly verification and combination of shares to a trustee, *i.e.*, the first step of the above protocol. The trustee is assumed to be honest-but-curious and to not DoS Ron. The trustee cannot access the secret, as he does not know sk_R and hence cannot unblind $\mathrm{pk}_{\mathrm{smc}}^{r'}$. Ron can detect if the trustee carries out the recovery incorrectly.

Evolution of the secret-management cothority: The secret-management cothority is expected to persist over a long period of time and to remain secure and available. However, a number of issues can arise over its lifetime. First, servers can join and leave the cothority resulting in churn. Second, even if the secret-management cothority membership remains static, the private shares of the servers should be regularly (*e.g.*, every month) refreshed to thwart an attacker who can attempt to collect a threshold of shares over a period of time. Lastly, the collective private key of the secret-management cothority should be rotated periodically *e.g.*, once every year or two. Any change of the current collective private-public key pair would require re-encrypting all of the long-lived secrets, however, if done by simply choosing a new key pair.

We address the first two problems by periodically re-sharing [19] the existing collective public key when a server joins or leaves the secret-management cothority, or when servers wants to refresh their private key shares. Lastly, when the secret-management cothority wants to rotate the collective public/private key pair ($\mathrm{pk}_{\mathrm{smc}}$, $\mathrm{sk}_{\mathrm{smc}}$), CALYPSO needs to collectively re-encrypt each individual secret under the new collective public key. To achieve this, we can generate and use translation certificates [23] such that the secrets can be re-encrypted without the involvement of their writers and without exposing the underlying secrets to any of the servers.

Achieving system goals: Long-term secrets achieves its goals similarly to one-time secrets with these differences.

Confidentiality of Secrets. In long-term secrets, the secret message m is encrypted under a symmetric key k which is subsequently encrypted under a collective public key of the secret-management cothority such that at least $t = f + 1$ trustees must cooperate to decrypt it. The ciphertext is bound to a specific policy through the use of NIZK proofs [49] so it cannot be reposted in a new write transaction with a malicious reader listed in its policy. The access-control trustees log the write transaction tx_{w} that includes the encrypted key which, based on the properties of the encryption scheme, does not leak any information about k. After the secret-management trustees receive a valid request $\mathrm{req}_{\mathrm{share}}$, they respond with the blinded shares of the collective private key encrypted under the public key in policy from the respective tx_{w}. Based on the properties of the DKG protocol, the collective private key is never known to any single entity and can only be used if t trustees cooperate. This means, only the intended reader gets a threshold of secret shares necessary to recover k and access m.

C. On-chain Blinded Key Exchange

In both on-chain secrets protocols, Wanda includes the public key of Ron in a secret's policy to mark him as the authorized reader. Once Wanda's write transaction is logged, everyone knows that she has shared a secret with Ron and correspondingly, once his read transaction is logged, everyone knows that he has obtained the secret. While this property is desirable for many deployment scenarios we envision, certain application may benefit from concealing the reader's identity. See Section VII for a discussion of CALYPSO's deployment.

We introduce an *on-chain blinded key exchange* protocol, an extension that can be applied to both on-chain secrets protocols. This protocol allows the writer to conceal the intended reader's identity in the write transaction and to generate a blinded public key for the reader to use in his read transaction. The corresponding private key can only be calculated by the reader and the signature under this private key is sufficient for the writer to prove that the intended reader created the read transaction. The protocol works as follows.

1) *Public Key Blinding.* Wanda generates a random blinding factor b and uses it to calculate a blinded version of Ron's public key $\mathrm{pk}_{\widetilde{R}} = \mathrm{pk}_R^b = g^{b\,\mathrm{sk}_R}$.
2) *Write Transaction.* Wanda follows either the one-time secrets or long-term secrets protocol to create $\mathrm{tx_w}$ with the following modifications. Wanda encrypts b under pk_R to enable Ron to calculate the blinded version of his public key by picking a random number b' and encrypting b as $(c_{b_1}, c_{b_2}) = (g^{\mathrm{sk}_R\, b'} b, g^{b'})$. Then, she uses $\mathrm{pk}_{\widetilde{R}}$ instead of pk_R in the policy. Wanda includes $c_b = (c_{b_1}, c_{b_2})$ and policy in $\mathrm{tx_w}$. After $\mathrm{tx_w}$ is logged, she notifies Ron on a separate, secure channel that she posted $\mathrm{tx_w}$ such that he knows which transaction to retrieve.
3) *Read Transaction.* When Ron wants to read Wanda's secret, he first decrypts c_b using sk_R to retrieve $b = (c_{b_1})(c_{b_2}^{\mathrm{sk}_R})^{-1} = (g^{\mathrm{sk}_R\, b'} b)(g^{b' \mathrm{sk}_R})^{-1}$. Then, he can compute $\mathrm{sk}_{\widetilde{R}} = b\,\mathrm{sk}_R$ and use this blinded private key to anonymously sign his $\mathrm{tx_r}$.
4) *Auditing.* If Wanda wants to prove that Ron generated the $\mathrm{tx_r}$, she can release b. Then, anyone can unblind Ron's public key $\mathrm{pk}_R = \mathrm{pk}_{\widetilde{R}}^{-b}$, verify the signature on the transaction and convince themselves that only Ron could have validly signed the transaction as he is the only one who could calculate $\mathrm{sk}_{\widetilde{R}}$.

The on-chain blinded key exchange protocol enables Wanda to protect the identity of the intended reader of her secrets without forfeiting any of the on-chain secrets's guarantees, however, it requires the knowledge of the reader's public key. As a consequence, this protocol does not support dynamic identities discussed in Section V, as we cannot predict and blind an unknown, future public key. Nonetheless, this protocol provides a viable option for applications where relationship privacy is more important than dynamic identity evolution. An extension of this protocol that allows blinding of dynamic identities remains an open challenge to be addressed in future work.

D. Post-Quantum On-chain Secrets

The security of both on-chain secrets implementations relies on the hardness of the discrete logarithm (DL) problem. An efficient quantum algorithm [48] for solving the DL problem exists, however. One solution to provide post-quantum security in CALYPSO is to use post-quantum cryptography (*e.g.*, lattice-based cryptography [8]). Alternatively, we can implement on-chain secrets using the Shamir's secret sharing [47] scheme which is information theoretically secure. Unlike the publicly-verifiable scheme we previously used, Shamir's secret sharing does not prevent a malicious writer from distributing bad secret shares because the shares cannot be verified publicly.

To mitigate this problem, we add a step to provide accountability of the secret sharing phase by (1) requiring the writer to commit to the secret shares she wishes to distribute and (2) requesting that each secret-management trustee verifies and acknowledges that the secret share they hold is consistent with the writer's commitment. As a result, assuming $n = 3f + 1$ and secret sharing threshold $t = f + 1$, the reader can hold the writer accountable for producing a bad transaction should he fail to correctly decrypt the secret message.

Write transaction protocol: Wanda prepares her write transaction $\mathrm{tx_w}$ with the help of the secret-management and access-control cothorities, where each individual trustee carries out the respective steps. Wanda initiates the protocol by preparing a write transaction as follows.

1) Choose a secret sharing polynomial $s(x) = \sum_{j=0}^{t-1} a_j x^j$ of degree $t-1$. The secret to be shared is $s = s(0)$.
2) Use $k = \mathrm{H}(s)$ as the symmetric key to compute the ciphertext $c = \mathrm{enc}_k(m)$ for the secret message m and set $H_c = \mathrm{H}(c)$.
3) For each trustee i, generate a commitment $q_i = \mathrm{H}(v_i \parallel s(i))$, where v_i is a random salt value.
4) Specify the access policy and prepare and sign $\mathrm{tx_w}$.

$$\mathrm{tx_w} = [\langle q_i \rangle, H_c, \langle \mathrm{pk}_i \rangle, \mathrm{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$$

5) Send the share $s(i)$, salt v_i, and $\mathrm{tx_w}$ to each secret-management trustee using a post-quantum secure channel.

The secret-management cothority verifies $\mathrm{tx_w}$ as follows.

1) Verify the secret share by checking that $(s(i), v_i)$ corresponds to the commitment q_i. If yes, sign $\mathrm{tx_w}$ and send it back to Wanda as a confirmation that the share is valid.

The access-control cothority finally logs Wanda's $\mathrm{tx_w}$.

1) Wait to receive $\mathrm{tx_w}$ signed by Wanda and the secret-management trustees. Verify that at least $2f + 1$ trustees signed the transaction. If yes, log $\mathrm{tx_w}$.

Read transaction, share request, and reconstruction protocols: The other protocols remain unchanged except that the secret-management trustees are already in possession of their secret shares and the shares need not be included in $\mathrm{tx_r}$. Once Ron receives the shares from the trustees, he recovers the symmetric key k as before and decrypts c. If the decryption fails, then the information shared by Wanda (the key, the ciphertext, or both) was incorrect. Such an outcome would indicate that Wanda is malicious and did not correctly execute the $\mathrm{tx_w}$ protocol. In response, Ron can release the transcript of the $\mathrm{tx_r}$ protocol in order to hold Wanda accountable.

V. SKIPCHAIN IDENTITY AND ACCESS MANAGEMENT

The CALYPSO protocols described so far do not provide dynamic sovereign identities. They only support static identities (public keys) and access policies as they provide no mechanisms to update these values. These assumptions are rather unrealistic though, as the participants may need to change or add new public keys to revoke a compromised private key or to extend access rights to a new device, for example.

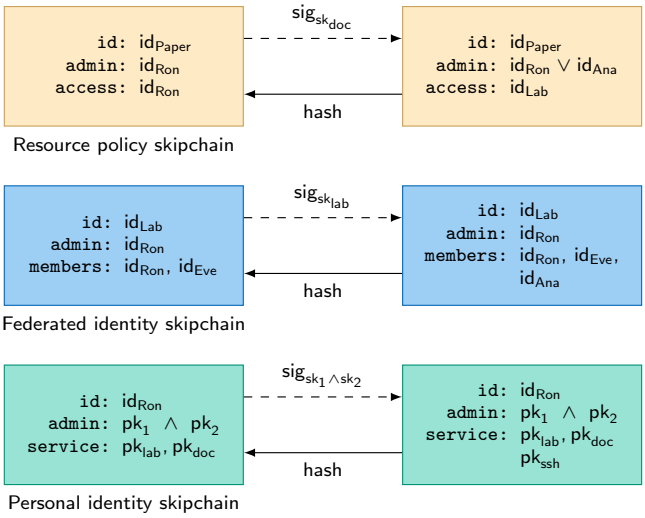

Fig. 3. Skipchain-based identity and access management (SIAM): First, Ron updates his personal identity skipchain $\mathrm{id}_{\mathsf{Ron}}$ to include $\mathrm{pk}_{\mathsf{ssh}}$. Afterwards, he uses $\mathrm{sk}_{\mathsf{lab}}$ to extend the federated identity skipchain $\mathrm{id}_{\mathsf{lab}}$ to add $\mathrm{id}_{\mathsf{Ana}}$ as a member. Finally, he adds $\mathrm{id}_{\mathsf{Ana}}$ as an admin and $\mathrm{id}_{\mathsf{lab}}$ as authorized readers to the resource policy skipchain $\mathrm{id}_{\mathsf{paper}}$ by using $\mathrm{sk}_{\mathsf{doc}}$.

Similarly, it should be possible to change access polices so that access to resources can be extended, updated or revoked and to define access-control rules for individual identities and groups of users for greater flexibility and efficiency. Finally, any access-control system that supports the above properties should be efficient as well as secure to prevent freeze and replay attacks [40], and race conditions between applying changes to access rights and accessing the resources.

In order address these challenges and achieve dynamic sovereign identities, we introduce the *skipchain-based identity and access management* (SIAM) subsystem for CALYPSO that provides the following properties: (1) Enable users to specify and announce updates to resource access keys and policies. (2) Support identities for both individual users and groups of users. (3) Protect against replay and freeze attacks. (4) Enforce atomicity of accessing resources and updating resource access rights to prevent race conditions.

A. Architecture

In SIAM, we introduce three types of skipchains [40], structures similar to a blockchain but doubly-linked, see Section II for details. *Personal identity skipchains* store the public keys that individual users control and use to access resources. A user can maintain a number of public keys that correspond to his identity that are used for access to resources on different devices, for example. *Federated identity skipchains* specify identities and public keys of a collective identity that encompasses users that belong to some group, such as employees of a company, members of a research lab, a board of directors, etc. *Resource policy skipchains* track access rights of identities, personal or federated, to certain resources and enable dynamic access control. In addition to listing identities and their public keys, policy skipchains include access-control rules allowing to enforce fine-grained update conditions for write transactions. Section VIII describes a simple access-control list (ACL) we created for our implementation.

The main insight is that skipchains securely maintain a verifiable timeline of changes to the identities and policies they represent. This means that these identities or polices can evolve dynamically and independently of the specific applications that make use of them and that at any point the applications can verfiably obtain the most up-to-date version of each.

The SIAM skipchains are under the self-sovereign control of individual users or groups of users. To track administrative rights, each SIAM skipchain includes in its skipblocks the (public) admin keys of users authorized to make updates and the policy under which a skipchain can be updated. These update policies can be expressed as arbitrary boolean circuits, *e.g.*, requiring approvals from just a single administrator or a certain set of them. Since the admin keys are used only for skipchain updates, they should be stored in cold wallets, such as hardware security modules, for increased security.

To evolve a SIAM skipchain and consequently the identities or access policies it represents, its admins follow the skipchain's update policies and create a new skipblock that reflects the necessary changes, and then publicly announce it, *e.g.*, by pushing the update to the storage provider(s) maintaining a public interface to the SIAM skipchain. Users and services that follow SIAM skipchains can get notified automatically about those updates and accept them if they are proposed by authorized users and adhere to the update policies. Since the latest skipblock represents the current state of a skipchain, *i.e.*, identities of all currently authorized users or all current access rules, revocation is trivially supported as the admins simply push a new skipblock to the respective skipchain that omits the public key or access rule that needs to be revoked. Figure 3 provides an overview on SIAM.

B. Integration Into CALYPSO

To integrate SIAM with CALYPSO, the long-term secrets protocols described in Section IV-B are adapted as follows. Assume that Ron has logged the unique identifier id_R of his personal identity skipchain on the access-control blockchain. If Wanda wants to give Ron access to a resource, she simply sets $\mathrm{policy} = \mathrm{id}_R$ instead of $\mathrm{policy} = \mathrm{pk}_R$ in tx_{w}.

This means that instead of defining access rights in terms of Ron's static public pk_R, she does so in terms of Ron's skipchain and consequently, any public key(s) specified in the most current most current block of id_R. Then, the resource is encrypted under the shared public key of the secret-management cothority as before. To request access, Ron creates the read transaction

$$\mathrm{tx}_{\mathrm{r}} = [H_w, \pi_{\mathrm{tx}_{\mathrm{w}}}, \mathrm{pk}_{R'}]_{\mathrm{sig}_{\mathrm{sk}_{R'}}}$$

where $H_w = \mathrm{H}(\mathrm{tx}_{\mathrm{w}})$ is the unique identifier for the secret that Ron requests access to, $\pi_{\mathrm{tx}_{\mathrm{w}}}$ is the blockchain inclusion proof for tx_{w}, and $\mathrm{pk}_{R'}$ is one of Ron's public keys that

he wishes to use from the latest block of the id_R skipchain. After receiving $\mathrm{tx_r}$, the access-control cothority follows the id_R skipchain to retrieve the latest skipblock and verifies $\mathrm{pk}_{R'}$ against it. Then, the access-control cothority checks the signature on $\mathrm{tx_r}$ using $\mathrm{pk}_{R'}$ and, if valid, logs $\mathrm{tx_r}$. Once $\mathrm{tx_r}$ is logged, the rest of the protocol works as described in Section IV-B, where the secret-management cothority uses $\mathrm{pk}_{R'}$ for re-encryption to enable Ron to retrieve the resource.

C. Achieving SIAM Goals

When SIAM is used, Ron is able to evolve the id_R skipchain arbitrarily, *e.g.*, rotate existing access keys or add new devices, and still retain access to the encrypted resource without needing Wanda to update the initial write transaction. Analogously, Wanda can efficiently provide a group of users access to a resource by using a federated identity id_F that these users are a part of, instead of adding each user individually, by setting $\mathrm{policy} = \mathrm{id}_F$ in $\mathrm{tx_w}$. This approach outsources the resource access maintenance to the admins of the id_F skipchain as they are in charge of the federated identity's membership and can add and remove members at will. Alternatively, Wanda can set up a resource policy skipchain id_P she is in charge of and include id_F as non-administrative members along with any other rules she wants to have enforced. Then, Wanda would use $\mathrm{policy} = \mathrm{id}_P$ in $\mathrm{tx_w}$ authorizing id_F to access the respective resource under the specified rules.

In order for users to defend against freeze and replay attacks we require them to generate freshness proofs of their SIAM skipchains. To do this they submit in regular time periods (*e.g.*, every hour) the head of their skipchain for timestamping on the blockchain. This prevents freeze and replay attacks as an adversary that managed to subvert an old SIAM skipblock cannot convince a client to accept the adversarial actions as authoritative, given that the skipblock the adversary refers to is different from the fresh one appearing on the blockchain. This practice further ensures the atomicity of read, write, and (skipchain) update operations, as the moment a SIAM update happens the client should send the new SIAM skipblock for timestamping. This effectively serializes reads, writes, and updates and therefore prevents race conditions.

VI. Further Security Consideration

Our contributions are mainly pragmatic rather than theoretical as we employ only existing, well-studied cryptographic algorithms. We already discussed achieving CALYPSO's security goals in the previous sections. On-chain secrets protocols achieve all goals but dynamic sovereign identities which is addressed by SIAM. In this section, we discuss the effect of malicious parties on CALYPSO.

Malicious readers and writers: CALYPSO's functionality resembles a fair-exchange protocol [41] in which a malicious reader may try to access a secret without paying for it and a malicious writer may try to get paid without revealing the secret. In CALYPSO, we protect against such attacks by employing the access-control and secret-management cothorities as decentralized equivalents of trusted third parties that mediate interactions between readers and writers.

The access-control cothority logs a write transaction on the blockchain only after it successfully verifies the encrypted data against the corresponding consistency proof. This ensures that a malicious writer cannot post a transaction for a secret that cannot be recovered. Further, as each $\mathrm{tx_w}$ binds its contents to its policy, it protects against attacks where malicious writers naively extract contents of already posted transactions and submit them with a different policy listing themselves as the authorized readers. Similarly, before logging a read transaction, the access-control cothority verifies that it refers to a valid $\mathrm{tx_w}$ and it is sent by an authorized reader as defined in the policy of $\mathrm{tx_w}$. A logged read transaction serves as an access approval. The secret-management cothority releases the decryption shares to the authorized reader only after confirming the reader presents an auditable proof of $\mathrm{tx_r}$.

Malicious trustees: Our threat model permits a fraction of the access-control and secret-management cothority trustees to be dishonest. The thresholds ($t = f + 1$) used in on-chain secrets, however, prevent the malicious trustees from being able to pool their secret shares and access writers' secrets or to prevent an an authorized reader from accessing their secret by withholding the secret shares. Further, even if some individual malicious trustees refuse to accept requests from the clients or to participate in the protocols altogether, the remaining honest trustees are be able to carry out all protocols by themselves thereby ensuring service availability.

Malicious storage providers: Wanda may choose to store the actual encrypted data either on-chain or off-chain by choosing to outsource the storage to external providers. Because the data is encrypted, it can be shared with any number of possibly untrusted providers. Before Ron creates a $\mathrm{tx_r}$ he needs to retrieve and verify the encrypted data against the hash posted in $\mathrm{tx_w}$. If Ron cannot obtain the encrypted data from the provider, he can contact Wanda to expose the provider as dishonest and receive the encrypted data directly from Wanda or an alternative storage provider.

VII. Calypso in the Wild

Below we describe three real-world deployments, two completed and one in-progress, of CALYPSO that resulted from collaborations with companies that needed a flexible, secure, and decentralized solution to share data.

A. Auditable Online Invoice Issuing

Together with the main invoice regulator of a European country, we built an auditable online invoice issuing system. It uses HyperLedger Fabric v1.0 as the access-control blockchain together with long-term secrets. While the system uses static identities, they are blinded as needed using the protocol described in Section IV-C.

Problem definition: The system consists of a set of potentially mutually distrustful sellers and buyers as well as a regulator, who are all part of a dynamic ecosystem. To keep track of all business relationships without the need for

an intermediary the system relies on blockchain technology. A seller wishes to verifiably issue an invoice to a buyer while granting additional access to the regulator. The invoice contains confidential information that both parties want to protect. The goal was to allow the invoices to be logged and tracked, and to enable the regulator to access the details if an issue arises between the parties.

Solution with CALYPSO: This system required a straightforward deployment of CALYPSO. The sellers generate write transactions where the secret message is the invoice and the authorized readers are both the buyer and the regulator. When the buyer sees the respective write transaction, he issues a read transaction to access the invoice. If there is an issue with the invoice or no invoice has been issued for a certain amount of time, the buyer reports it to the regulator who can audit the transactions and the invoice. Analogously, the seller can request the regulator to audit if an issue arises on his side. Using CALYPSO's blinded identities in the write transactions hides the relationships between the sellers and buyers and consequently details such as trade frequencies and types of purchases, which is advantageous from a business perspective.

B. Clearance-enforcing Document Sharing

We have used CALYPSO to deploy a decentralized, clearance-enforcing document sharing system than enables two organizations, A and B, to share a document D, such that a confidential policy can be enforced on D. We have realized this system with a contractor of the Defense Ministry of a European country using ByzCoin [28] adapted for a permissioned setting as the blockchain and long-term secrets.

Problem definition: Organization A wants to share with organization B a document D whose entirety or certain parts are classified as confidential and should only be accessible by people with a proper clearance. The clearance is granted (or revoked) to employees individually as needed or automatically when they join (or leave) a specific department so the set of authorized employees continuously changes. The goal is to enable the mutually distrustful A and B to share D while dynamically enforcing the specific clearance requirement and securely tracking accesses to D for auditing purposes.

Solution with CALYPSO: First, A and B agree on a mutually trusted blockchain system to implement the access-control cothority whose trustees include servers controlled by both organizations. Then, each organization establishes *confidential* federated identity skipchains, id_A and id_B, respectively. The hashes of the skipchains' genesis blocks are logged on the access-control cothority blockchain and write and read transactions can be posted by employees authortized in id_A and id_B, respectively. Organization A creates a document D, labels each paragraph as confidential or unclassified and encrypts each paragraph using a different symmetric key. A shares the ciphertext with B and generates write transactions whose payload are the symmetric keys of the classified paragraphs and $\mathrm{policy} = \mathrm{id}_B$. Any employee of B whose public key is included in the set of classified employees as defined in the most current skipblock of id_B can retrieve the symmetric keys by creating read transactions. The access-control and secret-management cothorities log the $\mathrm{tx_r}$, create a proof of access and deliver the key. Both organizations can update their identity skipchains as needed to ensure that at any given moment only authorized employees can post transactions.

C. Patient-centric Medical Data Sharing

CALYPSO lends itself well for applications that require secure data sharing for research purposes. We are in the process of working with a few hospitals and research institutions from a European country to build a patient-centric system to share patient medical data. We expect to use a sharded blockchain (*e.g.*, OmniLedger [29]) along with long-term secrets.

Problem definition: Researchers face difficulties in gathering medical data from hospitals as patients increasingly refuse to approve access to their data for research purposes amidst rapidly growing privacy concerns [21]. Patients dislike consenting once and completely losing control over their data and are more likely to approve to share their data with specific institutions [26]. The goal of this collaboration is to enable patients to remain sovereign over their data, hospitals to verifiably obtain patients' consent for specific purposes, and researchers to obtain access to valuable patient data.

Solution with CALYPSO: We have designed a preliminary architecture for a data sharing application that enables a patient P to share her data with multiple potential readers over time. The main difference from the previously described deployments is the fact that the data generator (hospital) and the data owner (P) are different. For this reason, we use a resource policy skipchain id_P such that the hospital can represent P wishes with respect to her data. Policy skipchains can dynamically evolve, adding and removing authorized readers, and can include rich access-control rules.

CALYPSO enables P to initialize id_P when she first registers with the medical system, Initially, id_P is empty indicating that P's data cannot be shared. If a new research organization or another hospital requests to access some of P's data, then P can update id_P by adding a federated identity of the research organization and specific rules. When new data is available for sharing, the hospital generates a new write transaction which consists of encrypted and possibly obfuscated or anonymized medical data and id_P as policy. As before, users whose identities are included in id_P can post read transactions to obtain access. This way P remains in control of her data and can unilaterally update or revoke access at any point.

VIII. IMPLEMENTATION

We implemented all components of CALYPSO, on-chain secrets and SIAM, in Go [17]. For cryptographic operations we relied on Kyber [32], an advanced crypto library for Go. In particular, we used its implementation of the Edwards25519 elliptic curve that provides a 128-bit security level. For the consensus mechanism required for the access-control cothority, we used a publicly available implementation of ByzCoin [28], a scalable Byzantine consensus protocol. We implemented both on-chain secrets protocols, one-time and

long-term secrets, run by the secret-management cothority. For SIAM, we implemented signing and verifying using a JSON-based ACL as described below. All of our implementations are available under an open source license on GitHub.

We used a simple JSON-based access-control language to describe policies in CALYPSO, however, different deployments of CALYPSO might benefit from more expressive ACLs. A policy consists of a unique identifier, a version number, and a list of rules that regulate the access to secrets stored in $\mathrm{tx_w}$. A rule has the following three fields. An `action` field which refers to the activity that can be performed on the secret (*e.g.*, `READ` or `UPDATE`). A `subjects` field listing the identities (*e.g.*, $\mathrm{id_{Ron}}$) that are permitted to perform the `action`. Lastly, an `expression` field which is a string of the form `operator` : [`operands`], where the `operator` is a logical operation (`AND` and `OR` in our case) and `operands` are either `subjects` or other `expressions` that describe the conditions under which the rule can be satisfied. More concretely, a sample expression could be $\{\texttt{AND} : [\mathrm{id_{Lab}}, \mathrm{id_{Ron}}]\}$, which means that signatures of both $\mathrm{id_{Lab}}$ and $\mathrm{id_{Ron}}$ are required to satisfy that rule. To express more complex conditions we can nest expressions, for example $\{\texttt{OR} : [\{\texttt{AND} : [\mathrm{id}_1, \mathrm{id}_2]\}, \{\texttt{AND} : [\mathrm{id}_3, \mathrm{id}_4]\}]\}$ evaluates to $((\mathrm{id}_1\ \texttt{AND}\ \mathrm{id}_2)\ \texttt{OR}\ (\mathrm{id}_3\ \texttt{AND}\ \mathrm{id}_4))$. We describe single and multi-signature access requests against policies and outline how they are created and verified in Appendix C.

IX. EVALUATION

We evaluated and compared both on-chain secrets protocols as well as the overheads introduced through the use of the dynamic identities and policies of SIAM. The primary questions we wish to investigate for on-chain secrets are whether its latency overheads are acceptable when deployed on top of blockchain systems and whether it can scale to hundreds of validators as required to ensure a high degree of confidence in the security of the system. In all experiments we checked the time it takes to create read and write transactions given different sizes of secret-management and access-control cothorities. For SIAM, we evaluate the latency overhead of creating and verifying access requests against SIAM skipchains both for simple identities as well as for complex policies.

We ran all our experiments on 4 Mininet [38] servers, each equipped with 256 GB of memory and 24 cores running at 2.5 GHz. To mimic realistic network conditions, we configured Mininet with a 100 ms point-to-point latency between all nodes and a maximum bandwidth of 100 Mbps for each node.

A. On-chain Secrets

In our experiments, we measure the overall latency of both on-chain secrets protocols, as shown in Figure 2, where we investigate the cost of the write, read, share retrieval and share recontruction sub-protocols. In the experiments, we vary the number of trustees to determine the effects on the latency and we remark that in our implementation all trustees are part of both cothorities.

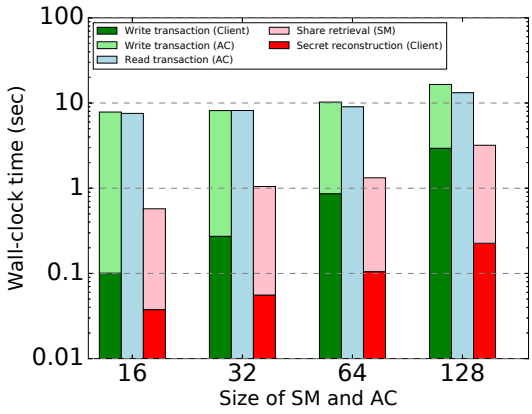

Fig. 4. Latency of one-time secrets protocol for varying sizes of the secret-management (SM) and access-control (AC) cothorities.

One-time secrets: Figure 4 shows the latency results for varying sizes of access-control and secret-management cothorities. First, we observe that the client-side creation of the $\mathrm{tx_w}$ is a costly operation which takes almost one second for 64 secret-management trustees. This is expected as preparing the $\mathrm{tx_w}$ involves picking a polynomial and evaluating it at n points, and setting up the PVSS shares and commitments, all of which involve expensive ECC operations. Second, we observe that the $\mathrm{tx_w}$ and $\mathrm{tx_r}$ processing times at the access-control cothority are comparable, but a write takes $\approx 250\,\mathrm{ms}$ longer on average than a read. This is due to the fact that the access-control trustees verify all NIZK encryption proofs. Our experiments also show that verifying the NIZK decryption proofs and recovering the shared secret are substantially faster than creating the $\mathrm{tx_w}$ and differ by an order of magnitude for large numbers of shares (*e.g.*, for 128 shares, $\approx 250\,\mathrm{ms}$ vs $\approx 3\,\mathrm{sec}$). This is due to the fact that verifying the NIZK proofs and reconstructing the shared secret require less ECC computations than the computationally expensive setup of the PVSS shares. Finally, we observe that the overhead for the secret-management cothority part of the secret recovery is an order of magnitude higher than the client side. This is expected as the client sends a request to each secret-management trustee and waits until a threshold of them replies.

Long-term secrets: Figure 5 presents the overall latency costs of the cothority setup (DKG), write, read, share retrieval and share reconstruction sub-protocols. Except for the DKG setup, all steps of the long-term secrets protocol scale linearly in the size of the cothority. Even for a large cothority of 128 servers, it takes less than 8 seconds to process a write transaction. The CPU-time is significantly lower than the wall-clock time due to the network (WAN) overhead that is included in the wall-clock measurements. While the DKG setup is quite costly, especially for a large number of servers, it is a one-time cost incurred only at the start of a new epoch. The overhead of the share retrieval is linear in the secret-management cothority as the number of shares t, which need to be validated and interpolated, increases linearly in the size

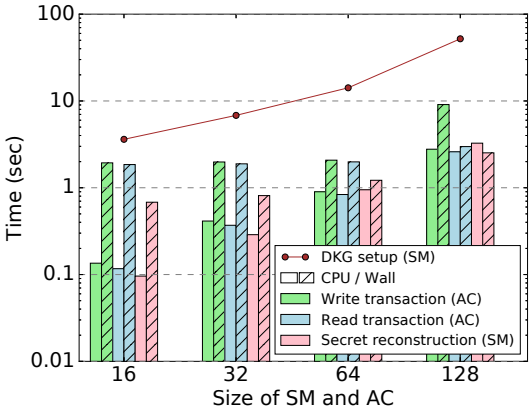

Fig. 5. Latency of long-term secrets protocol for varying sizes of secret-management (SM) and access-control (AC) cothorities.

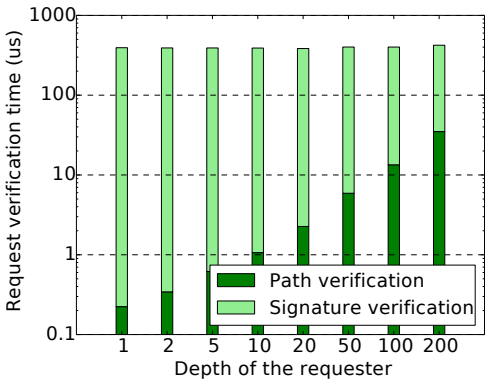

Fig. 6. Single-signature request verification.

of the secret-management cothority.

B. Skipchain-based Identity and Access Management

For SIAM, we benchmark the cost of validating the signature on a read transaction which is the most resource and time intensive operation. We distinguish single and multi-signature requests. The single signature case represents simple requests where one identity is requesting access while multi-signature requests occur for more complex access-control rules.

Single-signature request verification: For single-signature requests, the verification time is the sum of the signature verification and the time to validate the identity of the reader requesting access by checking it against the identity of the target reader as defined in the policy. The validation is done by finding the path from the target's skipchain to the requester's skipchain. We vary the *depth* of the requester, which refers to the distance between the two skipchains. Figure 6 shows the variation in request verification time depending on the requester's depth. We observe that most of the request verification time is required for signature verification which takes $\approx 385\,\mu s$ and accounts for $92.04 - 99.94\%$ of the total time.

Multi-signature request verification: As signature verification is the main overhead, we investigate the effect of verifying multi-signature requests. We create requests with a varying number of signers and investigate the number of request per second we can verify. Figure 7 shows the results for a requester skipchain's depth of 2 and 10. There is a significant reduction in the number of requests that can be verified when the number of signers increases whereas the depth of the requester is not significant.

X. Related Work

The decentralized data management platform Enigma [59], [60] provides comparable functionality to CALYPSO. Users own and control their data and a blockchain enforces access control by logging valid requests (as per the on-chain policy). However, Enigma stores the confidential data at a non-decentralized storage provider who can read and/or decrypt the data or refuse to serve the data even if there is a valid on-chain proof. The storage provider in Enigma is therefore a single point of compromise/failure. Other projects [3], [6], [22], [46], [60] commonly rely on centralized key-management and/or storage systems as well and hence suffer from comparable issues with respect to atomicity and robustness against malicious service providers. Vanish [14] is another secure data-sharing system which ensures that no-longer-usable data self-destructs to protect against accidental leakage. CALYPSO can provide similar functionality by adding time-outs to write transactions after which (honest) trustees destroy their secret key shares making the secret inaccessible. Vanish, however, relies on DHTs and is thus not as robust as the blockchain-based CALYPSO. Other privacy-focused blockchains [37], [44] do not address the issue of data sharing and access control but instead focus on hiding identity and transaction data through zero-knowledge proofs. Existing decentralized identity management systems, such as UIA [13] or SDSI/SPKI [43] enable users to control their identities but they lack authenticated updates via trust-delegating forward links of skipchains which enable CALYPSO to support secure long-term relationships between user identities and secure access control over shared data. OAuth2 [18] is an access-control framework where an authorization server can issue access tokens to authenticated clients which the latter can use to retrieve the requested data from a resource server. CALYPSO can emulate OAuth2 without any single points of compromise/failure where the access-control blockchain and the secret-management cothority act as decentralized versions of the authorization and resource servers, respectively. Further, thanks to CALYPSO's serialization of access requests and SIAM updates, it is not vulnerable to attacks exploiting race conditions when revoking access rights or access keys like OAuth2 [35]. ClaimChain [30] is a decentralized PKI where users maintain repositories of claims about their own and contacts' public keys. However, it permits transfer of access-control tokens, which can result in unauthorized access to the claims. Finally, Blockstack [1] uses Bitcoin to provide naming and identity functionality, but it does not support private-data sharing with access control. CALYPSO can

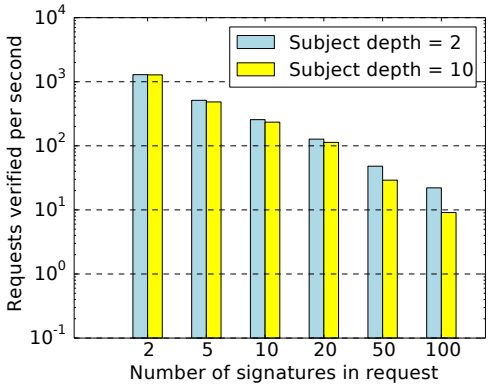

Fig. 7. Multi-signature request verification throughput.

work along a Blockstack-like system if implemented on top of an expressive enough blockchain [56] and include Blockstack identities as part of SIAM.

XI. Conclusion

We have presented Calypso, the first fully decentralized framework for auditable access control on protected resources over a distributed ledger that maintains confidentiality and control of the resources even after they have been shared. Calypso achieves its goals by introducing two separate components. The first component, on-chain secrets, is deployed on top of a blockchain to enable transparent and efficient management of secret data via threshold cryptography. The second component, skipchain-based identity and access management, allows for dynamic identities and resource access policies. We have implemented Calypso and shown that it can be efficiently deployed with blockchain systems to enhance their functionality. Lastly, we described three deployments of Calypso to illustrate its applicability to real-world applications.

References

[1] Ali, M., Nelson, J., Shea, R., and Freedman, M. J. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (Denver, CO, June 2016), USENIX Association, pp. 181–194.

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A. D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018* (2018), pp. 30:1–30:15.

[3] Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on* (2016), IEEE, pp. 25–30.

[4] Blakley, G. R. Safeguarding cryptographic keys. In *Proceedings of the national computer conference* (1979), vol. 48, pp. 313–317.

[5] Dinur, I., and Nissim, K. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2003), ACM, pp. 202–210.

[6] Dubovitskaya, A., Xu, Z., Ryu, S., Schumacher, M., and Wang, F. Secure and Trustable Electronic Medical Records Sharing using Blockchain. *arXiv preprint arXiv:1709.06528* (2017).

[7] Durham, V. Namecoin, 2011.

[8] El Bansarkhani, R., and Meziani, M. An efficient lattice-based secret sharing construction. In *IFIP International Workshop on Information Security Theory and Practice* (2012), Springer, pp. 160–168.

[9] Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31*, 4 (July 1985), 469–472.

[10] European Parliament and Council of the European Union. General Data Protection Regulation (GDPR). *Official Journal of the European Union (OJ) L119* (2016), 1–88.

[11] Fan, J., and Vercauteren, F. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive 2012* (2012), 144.

[12] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *Foundations of Computer Science, 1987., 28th Annual Symposium on* (1987), IEEE, pp. 427–438.

[13] Ford, B., Strauss, J., Lesniewski-Laas, C., Rhea, S., Kaashoek, F., and Morris, R. Persistent Personal Names for Globally Connected Mobile Devices. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Nov. 2006).

[14] Geambasu, R., Kohno, T., Levy, A. A., and Levy, H. M. Vanish: Increasing Data Privacy with Self-Destructing Data. In *USENIX Security Symposium* (2009), pp. 299–316.

[15] Genecoin. Make a Backup of Yourself Using Bitcoin, May 2018.

[16] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt* (1999), vol. 99, Springer, pp. 295–310.

[17] The Go Programming Language, Feb. 2018.

[18] Hardt, E. The OAuth 2.0 Authorization Framework, Oct. 2012. RFC 6749.

[19] Herzberg, A., Jarecki, S., Krawczyk, H., and Yung, M. Proactive secret sharing or: How to cope with perpetual leakage. *Advances in Cryptology — CRYPTO'95* (1995), 339–352.

[20] HIPAA Journal. The Benefits of Using Blockchain for Medical Records, Sept. 2017.

[21] Hollis, K. F. To Share or Not to Share: Ethical Acquisition and Use of Medical Data. *AMIA Summits on Translational Science Proceedings 2016* (2016), 420.

[22] Huang, L., Zhang, G., Yu, S., Fu, A., and Yearwood, J. SeShare: Secure cloud data sharing based on blockchain and public auditing. *Concurrency and Computation: Practice and Experience*.

[23] Jakobsson, M. On quorum controlled asymmetric proxy re-encryption. In *Public key cryptography* (1999), Springer, pp. 632–632.

[24] Juan Delacruz, I. B. Blockchain is tackling the challenge of data sharing in government, May 2018.

[25] Kate, A., and Goldberg, I. Distributed key generation for the internet. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on* (2009), IEEE, pp. 119–128.

[26] Kim, K. K., Sankar, P., Wilson, M. D., and Haynes, S. C. Factors affecting willingness to share electronic health data among California consumers. *BMC medical ethics 18*, 1 (2017), 25.

[27] Kokoris-Kogias, E., Gasser, L., Khoffi, I., Jovanovic, P., Gailly, N., and Ford, B. Managing Identities Using Blockchains and CoSi. Tech. rep., 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016), 2016.

[28] Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium* (2016).

[29] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *Security and Privacy (SP), 2018 IEEE Symposium on* (2018), Ieee, pp. 19–34.

[30] Kulynych, B., Isaakidis, M., Troncoso, C., and Danezis, G. ClaimChain: Decentralized Public Key Infrastructure. *CoRR abs/1707.06279* (2017).

[31] Kwon, J. TenderMint: Consensus without Mining.

[32] The Kyber Cryptography Library, 2010 – 2018.

[33] Lee, T. B. Facebooks Cambridge Analytica Scandal, Explained [Updated], 2018 (accessed July 27, 2018).

[34] Linn, L. A., and Koo, M. B. Blockchain for health data and its potential use in health it and health care related research. In *ONC/NIST Use of Blockchain for Healthcare and Research Workshop. Gaithersburg, Maryland, United States: ONC/NIST* (2016).

[35] Lodderstedt, T., Dronia, S., and Scurtescu, M. OAuth 2.0 token revocation, Aug. 2013. RFC 7009.

[36] Lueks, W. *Security and Privacy via Cryptography Having your cake and eating it too*. PhD thesis, [Sl: sn], 2017.

[37] MIERS, I., GARMAN, C., GREEN, M., AND RUBIN, A. D. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *34th IEEE Symposium on Security and Privacy (S&P)* (May 2013).
[38] Mininet – An Instant Virtual Network on your Laptop (or other PC), Feb. 2018.
[39] NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
[40] NIKITIN, K., KOKORIS-KOGIAS, E., JOVANOVIC, P., GAILLY, N., GASSER, L., KHOFFI, I., CAPPOS, J., AND FORD, B. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), USENIX Association, pp. 1271–1287.
[41] PAGNIA, H., AND GÄRTNER, F. C. On the impossibility of fair exchange without a trusted third party. Tech. rep., Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, 1999.
[42] PILKINGTON, M. Blockchain technology: principles and applications. *Browser Download This Paper* (2015).
[43] RIVEST, R., AND LAMPSON, B. SDSI: A simple distributed security infrastructure, Apr. 1996. http://theory.lcs.mit.edu/~cis/sdsi.html.
[44] SASSON, E. B., CHIESA, A., GARMAN, C., GREEN, M., MIERS, I., TROMER, E., AND VIRZA, M. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on* (2014), IEEE, pp. 459–474.
[45] SCHOENMAKERS, B. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *IACR International Cryptology Conference (CRYPTO)* (1999), pp. 784–784.
[46] SHAFAGH, H., BURKHALTER, L., HITHNAWI, A., AND DUQUENNOY, S. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop* (2017), ACM, pp. 45–50.
[47] SHAMIR, A. How to Share a Secret. *Communications of the ACM 22*, 11 (1979), 612–613.
[48] SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review 41*, 2 (1999), 303–332.
[49] SHOUP, V., AND GENNARO, R. Securing threshold cryptosystems against chosen ciphertext attack. *Advances in Cryptology — EUROCRYPT'98* (1998), 1–16.
[50] STOICA, I., MORRIS, R. T., KARGER, D. R., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM* (2001), pp. 149–160.
[51] SWAN, M. *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc., 2015.
[52] SYTA, E., JOVANOVIC, P., KOKORIS-KOGIAS, E., GAILLY, N., GASSER, L., KHOFFI, I., FISCHER, M. J., AND FORD, B. Scalable Bias-Resistant Distributed Randomness. In *38th IEEE Symposium on Security and Privacy* (May 2017).
[53] SYTA, E., TAMAS, I., VISHER, D., WOLINSKY, D. I., JOVANOVIC, P., GASSER, L., GAILLY, N., KHOFFI, I., AND FORD, B. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *37th IEEE Symposium on Security and Privacy* (May 2016).
[54] SZABO, N. Smart contracts. *Unpublished manuscript* (1994).
[55] VENKATADRI, G., ANDREOU, A., LIU, Y., MISLOVE, A., GUMMADI, K. P., LOISEAU, P., AND GOGA, O. Privacy Risks with Facebooks PII-based Targeting: Auditing a Data Brokers Advertising Interface. In *IEEE Symposium on Security and Privacy (SP)* (2018), pp. 221–239.
[56] WOOD, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper* (2014).
[57] YERMACK, D. Corporate governance and blockchains. *Review of Finance 21*, 1 (2017), 7–31.
[58] YOUNG, E. . Blockchain in health, May 2018.
[59] ZYSKIND, G., NATHAN, O., ET AL. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE* (2015), IEEE, pp. 180–184.
[60] ZYSKIND, G., NATHAN, O., AND PENTLAND, A. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471* (2015).

APPENDIX A
PUBLICLY VERIFIABLE SECRET SHARING

We follow the protocol in [45] where a dealer wants to distribute shares of a secret value among a set of trustees. Let $\mathbb{G}$ be a cyclic group of prime order q where the decisional Diffie-Hellman assumption holds. Let g and h denote two distinct generators of $\mathbb{G}$. We use $N = \{1, \dots, n\}$ to denote the set of trustees, where each trustee i has a private key sk_i and a corresponding public key $\mathrm{pk}_i = g^{\mathrm{sk}_i}$. The protocol runs as follows:

Dealing the shares: The dealer initiates the PVSS protocol as follows.

1) Choose a secret sharing polynomial $s(x) = \sum_{j=0}^{t-1} a_j x^j$ of degree $t-1$. The secret to be shared is $s = g^{s(0)}$.
2) For each trustee $i \in \{1, \dots, n\}$, compute the encrypted share $\widehat{s_i} = \mathrm{pk}_i^{s(i)}$ of the shared secret s and create the corresponding NIZK encryption consistency proof $\pi_{\widehat{s_i}}$. Create the polynomial commitments $b_j = h^{a_j}$, for $0 \leq j < t$.
3) Publish all $\widehat{s_i}$, $\pi_{\widehat{s_i}}$, and b_j.

$\pi_{\widehat{s_i}}$ proves that the corresponding encrypted share $\widehat{s_i}$ is consistent. More specifically, it is a proof of knowledge of the unique $s(i)$ that satisfies:

$$A_i = h^{s(i)}, \widehat{s_i} = \mathrm{pk}_i^{s(i)}$$

where $A_i = \prod_{j=0}^{t-1} {b_j}^{i^j}$. In order to generate $\pi_{\widehat{s_i}}$, the dealer picks at random $w_i \in \mathbb{Z}_q$ and computes:

$$a_{1i} = h^{w_i}, a_{2i} = \mathrm{pk}_i^{w_i},$$
$$C_i = \mathrm{H}(A_i, \widehat{s_i}, a_{1i}, a_{2i}), r_i = w_i - s(i)C_i$$

where H is a cryptographic hash function, C_i is the challenge, and r_i is the response. Each proof $\pi_{\widehat{s_i}}$ consists of C_i and r_i, and it shows that $\log_h A_i = \log_{\mathrm{pk}_i} \widehat{s_i}$.

Verification of the shares: Each trustee i verifies their encrypted share $\widehat{s_i}$ against the corresponding NIZK encryption consistency proof $\pi_{\widehat{s_i}}$ to ensure the validity of the encrypted share. To do so, each trustee performs the following steps.

1) Compute $A_i = \prod_{j=0}^{t-1} {c_j}^{i^j}$ using the polynomial commitments c_j, $0 \leq j < t$.
2) Compute $a'_{1i} = h^{r_i} A_i^{C_i}$ and $a'_{2i} = \mathrm{pk}_i^{r_i} \widehat{s_i}^{C_i}$
3) Check that $\mathrm{H}(A_i, \widehat{s_i}, a'_{1i}, a'_{2i})$ matches the challenge C_i.

Decryption of the shares: If their share is valid, each trustee i creates their decrypted share as follows.

1) Compute the decrypted share $s_i = (\widehat{s_i})^{\mathrm{sk}_i^{-1}}$ and the corresponding NIZK decryption consistency proof π_{s_i}, which proves that s_i is the correct decryption of $\widehat{s_i}$. The proof shows the knowledge of the unique value that satisfies $\log_g \mathrm{pk}_i = \log_{s_i} \widehat{s_i}$.
2) Publish s_i and π_{s_i}.

Reconstructing the shared secret: If there are at least t correctly decrypted shares, then the Lagrange interpolation can be used to recover the shared secret s.

APPENDIX B
FULL ENCRYPTION/DECRYPTION PROTOCOL FOR LONG-TERM SECRETS

We follow the extension of the TDH2 protocol of Shoup [49] described by Lueks [36]. Let $\mathbb{G}$ be a cyclic group

of prime order q with generators g and $\bar{g}$. We assume the existence of two hash functions: $\mathrm{H}_1 : \mathbb{G}^6 \times \{0,1\}^l \to \mathbb{G}$ and $\mathrm{H}_2 : \mathbb{G}^3 \to \mathbb{Z}_q$.

Encryption: A client encrypts a message under the collective public key $\mathrm{pk}_{\mathrm{smc}}$ such that it can be decrypted by anyone that is included in policy[1] $L \in \{0,1\}^l$. The client performs the following steps.

1) Choose a symmetric key k to symmetrically encrypt the message and then embed k as a point $k' \in \mathbb{G}$.
2) Choose at random $r, s \in \mathbb{Z}_q$. Compute:

$$c = \mathrm{pk}_{\mathrm{smc}}^r k', u = g^r, w = g^s, \bar{u} = \bar{g}^r, \bar{w} = \bar{g}^s,$$
$$e = \mathrm{H}_1\left(c, u, \bar{u}, w, \bar{w}, L\right), f = s + re.$$

The ciphertext is $(c, L, u, \bar{u}, e, f)$.

Decryption of the shares: Given a ciphertext $(c, L, u, \bar{u}, e, f)$ and a matching authorization to L, each trustee i performs the following steps.

1) Check if $e = \mathrm{H}_1\left(c, u, \bar{u}, w, \bar{w}, L\right)$ by computing $w = \frac{g^f}{u^e}$ and $\bar{w} = \frac{\bar{g}^f}{\bar{u}^e}$, which is a NIZK proof that $\log_g u = \log_{\bar{g}} \bar{u}$.
2) If the share is valid, choose $s_i \in \mathbb{Z}_q$ at random and compute:

$$u_i = u^{\mathrm{sk}_i}, \hat{u_i} = u^{s_i}, \hat{h_i} = g^{s_i},$$
$$e_i = \mathrm{H}_2\left(u_i, \hat{u_i}, \hat{h_i}\right), f_i = s_i + \mathrm{sk}_i e_i$$

3) Publish (i, u_i, e_i, f_i), where u_i is the corresponding share.

Note that if the policy L has changed, then e cannot be computed correctly. Given that an adversary will not know r, he cannot change the e to match his new policy.

Secret reconstruction: A client can reconstruct the secret and obtain the decryption key k both on the client side or at an untrusted server. We describe both schemes below.

***Secret reconstruction at the client*:**

1) Run the decryption share check to make sure that the trustees are not misbehaving.
2) If the check passes then verify that (u, u_i, h_i) is a DH triple by checking that $e_i = \mathrm{H}_2\left(u_i, \hat{u_i}, \hat{h_i}\right)$, where $\hat{u_i} = \frac{u^{f_i}}{{u_i}^{e_i}}$ and $\hat{h_i} = \frac{g^{f_i}}{{h_i}^{e_i}}$.
3) If there are at least t valid shares, (i, u_i), the recovery algorithm is doing Lagrange interpolation of the shares:

$$\mathrm{pk}_{\mathrm{smc}}^r = \prod_{k=0}^{t} {u_i}^{\lambda_i}$$

where λ_i is the i^{th} Lagrange element.
4) Compute the inverse of $\mathrm{pk}_{\mathrm{smc}}^r$ and find $k' = \frac{c}{\mathrm{pk}_{\mathrm{smc}}^r}$. From k' derive the decryption key k and recover the original message.

[1]This policy is the identifier (hash of genesis block) of an identity skipchain

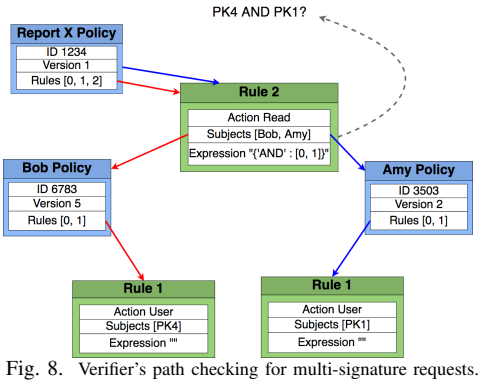

Fig. 8. Verifier's path checking for multi-signature requests.

***Secret reconstruction at the trusted server*:** The client authenticates themselves using their public key g^{x_c}. One of the trustees is assigned to do the reconstruction for the client.

1) Each trustee that created their decryption share as $g^{rx_i} = u_i$, ElGamal encrypts the share for the client using x_i as the blinding factor instead of a random r'. The new share becomes $g^{rx_i} g^{x_c x_i} = g^{(r+x_c)x_i} = g^{r'x_i} = u'^{x_i} = u'_i$. Then the trustee computes $\hat{h_i}$, as before and $\hat{u'_i} = u'^{s_i}$. Finally $e'_i = \mathrm{H}_2\left(u'_i, \hat{u'_i}, \hat{h_i}\right)$ and $f'_i = s_i + x_i e'_i$
2) Any trustee can pool the shares and reconstruct the secret with Lagrange interpolation as shown above. The end result is $g^{r'x} = g^{(r+x_c)x}$
3) The client gets $g^{(r+x_c)x}$ and as they know g^x and x_c, they can find $-x_c$ and compute $g^{x^{-x_c}} = g^{-xx_c}$. Finally they compute $g^{rx} = g^{(r+x_c-x_c)x}$ and decrypt as mentioned above.

APPENDIX C
ACCESS REQUESTS AND VERIFICATION

In this section, we outline how we create and verify access requests. A request consists of the policy and the rule invoked that permits the requester to perform the action requested. There is also a message field where extra information can be provided *e.g.*, a set of documents is governed by the same policy but the requester accesses one specific document.. A request req is of the form: $\mathrm{req} = [\mathrm{id}_{Policy}, \mathrm{index}_{Rule}, M]$, where id_{Policy} is the ID of the target policy outlining the access rules; index_{Rule} is the index of the rule invoked by the requester; and M is a message describing extra information.

To have accountability and verify that the requester is permitted to access, we use signatures. The requester signs the request and creates a signature consisting of the signed request $(\mathrm{sig}_{\mathrm{req}})$ and the public key used (pk). On receiving an access request, the verifier checks that the $\mathrm{sig}_{\mathrm{req}}$ is correct. The verifier then checks that there is a valid path from the target policy, id_{Policy}, to the requester's public key, pk. This could involve multiple levels of checks, if the requester's key is not present directly in the list of *subjects* but included transitively in some federated SIAM that is a *subject*. The verifier searches along all paths (looking at the last version timestamped by the access-control cothority) until the requester's key is found.

Sometimes, an access request requires multiple parties to sign. Conditions for multi-signature approval can be described using the *expression* field in the rules. An access request in this case would be of the form $(\mathrm{req}, [\mathrm{sig}_{\mathrm{req}}])$ where $[\mathrm{sig}_{\mathrm{req}}]$ is

a list of signatures from the required-for-access parties. The verification process is similar to the single signature case.

Figure 8 shows an example of the path verification performed by the verifier. Report X has a policy with a Rule granting read access to Bob and Amy. There is an expression stating that both Bob's and Amy's signatures are required to obtain access. Hence, if Bob wants access, he sends a request $(\mathrm{req}, [\mathrm{sig}_{\mathrm{req},\mathrm{pk}_1}, \mathrm{sig}_{\mathrm{req},\mathrm{pk}_4}])$, where $\mathrm{req} = [1234, 2, \text{“}ReportX'']$ The verifier checks the paths from the policy to Bob's pk_4 and Amy's to pk_1 are valid. Paths are shown in red and blue respectively. Then the expression AND : [0,1] is checked against the signatures. If all checks pass, the request is considered to be verified.

APPENDIX D
JSON ACCESS-CONTROL LANGUAGE

A sample policy for a document, expressed in the JSON based language, is shown in Figure 9. The policy states that it has one Admin rule. The admins are S1 and S2 and they are allowed to make changes to the policy. The Expression field indicates that any changes to the policy require both S1 and S2's signatures.

```
{
    "ID" :  2345
    "Version" : 1,
    "Rules" :
    [
        {
            "Action" : "Admin",
            "Subjects" : [S1, S2],
            "Expression" : "{'AND' : [S1, S2]}"
         }
    ]
}
```

Fig. 9. Sample Policy in JSON access-control language.