Verifiable Management of Private Data under Byzantine Failures

Eleftherios Kokoris-Kogias
EPFL

Enis Ceyhun Alp
EPFL

Sandra Deepthy Siby
EPFL

Nicolas Gailly
Pegasys R&D, Consensys

Linus Gasser
EPFL

Philipp Jovanovic
EPFL

Ewa Syta
Trinity College

Bryan Ford
EPFL

Abstract

In this paper, we present Calypso, an auditable data-management framework that lends itself to decentralize the sharing and life-cycle management of private data as well as to enforce fair, atomic disclosure of data protecting against front-running attacks. To achieve these goals, Calypso deploys on-chain secrets, a novel abstraction that enforces the deposition of an auditable trace whenever users access confidential data. On-chain secrets combine verifiable secret sharing and blockchains to ensure accountability, fairness, and liveness of the data-sharing process without centralized intermediaries. By employing skipchains for identities and access-control rules, Calypso provides dynamically changing access control while enabling users to keep control over their identities.

Our benchmarks show that the latency of processing transactions scales linearly with the number of involved nodes (trustees) and is in the range of 0.2 to 8 seconds for 16 to 128 trustees. We also evaluated Calypso in two contexts using simulated and real workloads. First, a document sharing application, where, depending on the load, the latency overhead ranges from 0.2× to 5× compared to a semi-centralized system. Second, a zero-collateral lottery that, unlike the current state-of-the-art solutions, always terminates in one round independently of the number of participants.

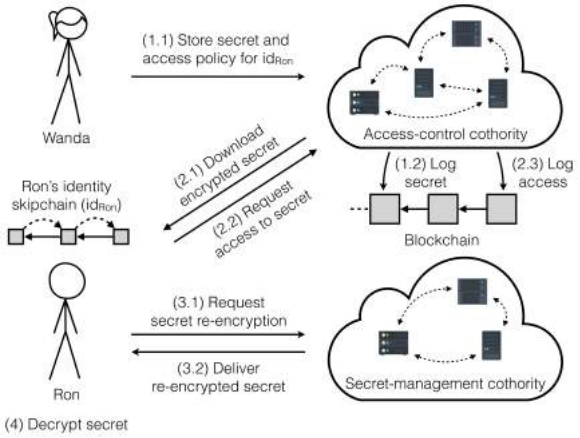

Figure 1: Auditable data sharing in Calypso: (1) Wanda encrypts data under the secret-management cothority's key, specifying the intended reader (*e.g.*, Ron) and the access policy, and then sends it to the access-control cothority which verifies and logs it. (2) Ron downloads the encrypted secret from the blockchain and then requests access to it by contacting the access-control cothority which logs the query if valid, effectively authorizing Ron's access to the secret. (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 by access-control cothority was successful. (4) Ron decrypts the secret. If a specific application requires fairness, the data can be atomically disclosed on-chain.

1 Introduction

New data privacy legislation, such as the European Union General Data Protection Regulation (GDPR) [17] or the EFF's call for information fiduciary rules for businesses [63], has reignited interest in secure management of private data. The ability to effectively share and manage data is one of the cornerstones of the digital revolution that turned many applications and processes to be data-driven and data-dependent. However, the commonly used centralized data management approach has repeatedly proven to be problematic in terms of security and fairness due to single points of failure or compromise. For example, centralized data-sharing solutions are exposed to the risk of bribery or coercion from parties requesting inspection of certain records for which they have no access permissions [16, 42]. Another risk is that these systems might not implement all phases of the data management cycle and, *e.g.*, "forget" to delete user data [22]. Finally, well-located servers may have an unfair advantage in accessing information faster than others, necessitating additional and often burdensome regulations, *e.g.*, to guarantee fairness for trading markets [37].

Replacing single points of failure with decentralized alternatives is an obvious approach to achieve more transparent and fair data sharing and management. Decentralized data-sharing can give rise to data markets controlled by users [69] and not solely by tech giants such as Google or Facebook; enable sharing of confidential data between mutually distrustful parties, such as state institutions or even different countries; or bring the much needed transparency to lawful access requests [19]. Decentralized data life-cycle management can enable effective and guaranteed data retention (*e.g.*, legal or corporate data retention policies or enforcement of the "right to be forgotten"), or an implementation of an information-publication version of a dead man's switch [15] that enables journalists to create a contingency plan to have entrusted data disclosed under specific circumstances. Finally, if correctly implemented, decentralized data life-cycle management can also result in fair lotteries [6], games (*e.g.*, poker [39]), and trading (*e.g.*, exchanges [11]).

Unfortunately, current decentralized data-sharing applications [44, 55] fail to manage private data securely unless they forfeit the full life-cycle management [23] and publish encrypted data on Bitcoin;

or rely on semi-centralized solutions [7, 77]. Furthermore, decentralized applications that rely on the timing of data disclosure to enforce fairness are susceptible to front-running attacks where the adversary gets early access to information and unfairly adapts their strategies. For example, the winner of the Fomo3D [64] event (gaining a prize of 10.5k Ether (USD $2.2 M at the time)) enforced an early termination of the lottery by submitting a sequence of high-fee transactions, which significantly increased his winning probability. Due to the lack of fairness guarantees, decentralized exchanges remain vulnerable [11, 71] or resort to centralized order-book matching (*e.g.*, 0x Project [10]), and decentralized lotteries require collateral [6] or run in a non-constant number of rounds [50].

In this paper we introduce Calypso, a new secure data-management framework that addresses the challenge of providing fair and verifiable access to private information without relying on a trusted party. To achieve this goal Calypso faces three key challenges. First, Calypso has to provide accountability for all accesses to confidential data to ensure that secrets are not improperly disclosed and to enforce proper recording of data accesses. Second, Calypso has to prevent front-running attacks and guarantee fair access to information. Third, Calypso has to enable data owners to maintain control over the data they share, and data consumers to maintain access even when their identities (public keys) are updated. In particular, Calypso should allow for flexible updates to access-control rules and user identities, *e.g.*, to add or revoke access rights or public keys. Figure 1 provides an overview of a typical data-sharing application using Calypso that builds on top of a novel abstraction called *on-chain secrets* (OCS) and provides dynamic access-control and identity management.

On-chain secrets addresses the first two challenges by combining threshold cryptography [62, 66, 67] and blockchain technology [35, 74] to enable users to share their encrypted data with *collective authorities* (cothorities) that are responsible for enforcing access-control and atomically disclosing data to authorized parties. Furthermore, Calypso combines on-chain secrets with skipchains [34, 53] in order to enable dynamic access-control and identity management. We present two concrete instantiations of on-chain secrets, namely *one-time secrets* and *long-term secrets*, that have different trade-offs in terms of functionality and computational and storage overheads.

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 scale linearly in the number of trustees exhibiting a moderate overhead of 0.2 to 8 seconds for cothorities with 16 to 128 trustees. Furthermore, in addition to evaluations on simulated data, we tested two deployments of Calypso using real data traces. First, we deployed a document-sharing application and tested it under different loads. Calypso takes 10-20 (10-150) seconds to execute a write (read) request, and has a 0.2× to 5× latency overhead compared to a semi-centralized solution that stores data in the cloud. Furthermore, we show that Calypso-based zero-collateral lotteries significantly outperform the state-of-the-art as they require 1 and $\log n$ rounds to finish, respectively, with n denoting the number of participants.

In summary, this paper makes the following contributions.

- We introduce Calypso, a decentralized framework for auditable management of private data while maintaining fairness and confidentiality (see Section 3). Calypso enables dynamic updates to access-control rules without compromising security.
- We present on-chain secrets and its two implementations, one-time and long-term secrets, that enable transparent and efficient management of data without requiring a trusted third party (see Section 4).
- We demonstrate the feasibility of using Calypso to address the data sharing needs of actual organizations by presenting three classes of realistic, decentralized deployments: auditable data sharing, data life-cycle management, and atomic data publication (see Section 5). To evaluate our system and conduct these feasibility studies, we created an implementation of Calypso which was independently audited and will be released as open-source (see Sections 6 and 7).

2 Motivating Applications and Background

In this section, we first motivate Calypso by describing how it can enable security and fairness in three different classes of applications: auditable data sharing, data life-cycle management, and atomic data publication. (See Section 5 for real-worlds deployments using Calypso). We then summarize the main cryptographic building blocks that we rely on in the rest of the paper.

2.1 Motivating Examples

Auditable Data Sharing. Current cloud-based data-sharing systems (*e.g.*, Dropbox or Google Drive) provide a convenient way to store and share data, however, their main focus is on integrity whereas ensuring data confidentiality and access accountability are often secondary, if provided at all. Further, clients must trust the individual companies that these properties are indeed achieved in practice as clients, for variety of reasons, are typically not able to verify these security guarantees themselves. Furthermore, many systems often provide only retroactive and network-dependent detection mechanisms for data integrity failures [43, 48]. Consequently, any secure data-sharing system should be decentralized to avoid the need to rely on trusted third parties, and it should provide integrity, confidentiality, and accountability and ensure that any violations can be detected proactively.

Current decentralized data-sharing applications [44, 55, 70, 75] that focus on shared access to private data often fail to manage these private data in a secure and accountable manner, especially when it comes to sharing data between independent and mutually-distrustful parties. They either ignore these issues altogether [31, 76], fall back on naive solutions [23], or use semi-centralized approaches [7, 27, 77], where access information and hashes of the data are put on-chain but the secret data is stored and managed off-chain, hence violating the accountability requirement.

To address the above challenges and enable secure decentralized data-sharing applications, Calypso uses threshold cryptography and distributed-ledger technology to protect the integrity and confidentiality of shared data and to ensure data-access accountability by generating a third-party verifiable audit trail for data accesses. Designers of decentralized applications can further use Calypso to

achieve additional functionalities such as monetizing data accesses or providing proofs to aid investigations of data leaks or breaches.

Data Life-Cycle Management. Custodian systems can provide policy-based data deletion and publication mechanisms unlocking a variety of useful data life-cycle management applications.

Users have currently little to no control over how their data is stored, maintained, and processed at cloud-based service providers like Google or Facebook, making it particularly challenging for EU citizens to enact their *right to be forgotten* as mandated by the EU GDPR legislature [17]. Policy-based data deletion mechanisms of custodian systems could provide a way out of this dilemma.

Provable data publication based on user-specified policies would further permit automatic publication of documents, such as legal wills or estate plans, when certain conditions are met. This functionality would also enable to implement digital life insurances for whistleblowers where files are published automatically unless the custodian receives a digitally signed "heartbeat" message from the insured person on a regular basis [15, 59].

However, designing such custodian systems is a challenging task as straightforward centralized solutions provide little to no protection against single points of failure or compromise, bribery or coercion. This results in a high amount of risk for users who might rely on such (centralized) custodians. Previous projects have attempted to design such systems but they all exhibit shortcomings, *e.g.*, in terms of failure resilience or guaranteed erasure of data [22].

Moving to fully decentralized custodian, however, bears new challenges in terms of how to specify and implement data deletion, publication, and policy mechanisms in this deployment model and how to integrate these concepts with each other in a secure way.

Calypso solves these challenges with on-chain secrets and an expressive policy-mechanism that enables the revocation of access rights for everyone, effectively preventing any access to the secrets stored on-chain. Furthermore, the policy mechanism can express not only time-based but also event-based public decryption (*e.g.*, reveal data in the absence of a heartbeat message).

Atomic Data Publication. Security and fairness requirements significantly change when an application is deployed in a Byzantine, decentralized environment as opposed to a traditional, centralized setting. For example, an adversary can easily gain an unfair advantage over honest participants through front-running [11, 64, 71] if decentralized applications, such as lotteries [6], poker games [39], or exchanges [11], are not designed with such attacks in mind. To protect users against front-running, these applications often fall back to trusted intermediaries giving up decentralization, or they implement naive commit-and-reveal schemes exposing themselves to liveness risks where adversaries can DoS the application or force it to restart by refusing to reveal their inputs. To provide failure resilience and protect against such late aborts, many applications introduce complex incentive mechanisms whose security guarantees are usually difficult to analyze [6, 39].

In Calypso, all inputs committed by the participants (*e.g.*, lottery randomness, trading bids, game moves) remain confidential up to a barrier point that is expressed through specific rules defined in a policy. All of the decommitted values are taken into account to compute and atomically disclose the results of the protocol to every interested party. Consequently, Calypso resolves the tension between decentralization, fairness, and availability, providing a secure foundation for decentralized applications.

2.2 Blockchains and Skipchains

Blockchain is a distributed, append-only and tamper-evident log that is composed of blocks that are connected to each other via cryptographic hashes and are used in many decentralized applications [5, 14, 52]. Calypso can be deployed on top of any blockchain that supports programmability (*i.e.*, smart contracts [4, 72, 74]) thereby enabling custom validation of transactions.

Skipchains [53] track configuration changes of a decentralized authority (cothority) by using each block as a representation of all public keys of the cothority that are necessary to authenticate the next block. When a 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 trust to the new set. This signature is a *forward link* [34] that clients follow to get up-to-date with the current authoritative group. In Section 4.5, we use identity and policy skipchains. Our construction is a simple extension of the skipchains in order to support federated groups and enable expressive access-control (see Appendix D).

2.3 Threshold Cryptosystems

A (t, n)-secret sharing scheme [8, 66] 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, an issue that verifiable secret sharing (VSS) [20] solves by enabling the trustees to verify that the distributed shares are consistent. VSS is used for threshold signing and threshold encryption. Publicly verifiable secret sharing (PVSS) [62] is a variation of VSS that enables external third-parties to verify the 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) [24, 32] protocol allows to create a collective key pair without a trusted dealer. The DKG produces 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 trustees collaborates. Afterwards, anyone can encrypt data under this public key. For Calypso, we use the threshold ElGamal cryptosystem with non-interactive zero knowledge (NIZK) proofs [46, 67] (see Appendix B) to protect against replay attacks.

3 Calypso Overview

This section provides an overview of Calypso. We start with a strawman solution to motivate the challenges that any secure, decentralized data-sharing and management system should address. From our observations we then derive the system goals and finally present the system design (see Figure 1).

3.1 Strawman Data Management Solution

We assume that the strawman consists of a tamper-resistant public log, such as the Bitcoin blockchain, and that participants register their identities on-chain, *e.g.*, as PGP keys. Now consider an application on top of the strawman system where Wanda is the operator of

a paid service providing asynchronous access to some information and Ron is a customer. Once Ron has paid the fee, Wanda can simply encrypt the data under Ron's key, post the ciphertext on-chain which Ron can then retrieve and decrypt at his discretion.

This strawman approach provides the intended functionality but has several drawbacks. (1) There is no data access auditability because Ron's payment record does not prove that he actually accessed the data, and such a proof might be needed if the provided information is misused, for example. (2) If Wanda ever wants to change the access rights, *e.g.*, because Ron cancelled his access subscription, she cannot do so because the ciphertext is on-chain and Ron controls the decryption key. (3) If Ron ever needs to change his public key, he would lose access to all data encrypted under that key, unless Wanda re-publishes that data using Ron's new key. (4) Since the exchange of payment and data is not atomic, having submitted a payment successfully does not guarantee that Ron access to the data. (5) If Wanda makes the data available on a first-come-first-serve basis, then customers with better connectivity are able to make payments faster and thereby mount front-running attacks.

To address these issues, we introduce two new components and transform the strawman into Calypso.

(1) To enable auditability of data accesses and ensure atomic data delivery, we introduce *on-chain secrets* (OCS) in Sections 4.1 and 4.3.
(2) To enable decentralized, dynamic, user-sovereign identities and access policies, we extend skipchains and integrate them with Calypso in Section 4.5.

3.2 System Goals

Calypso has the following primary goals.

- **Confidentiality.** Secrets stored on-chain can only be decrypted by authorized clients.
- **Auditability.** All access transactions are third-party verifiable and recorded in a tamper-resistant log.
- **Fair access.** Clients are guaranteed to get access on a secret they are authorized for if any only if they posted an access request on-chain. If a barrier point exists, authorized clients get concurrent access after it (protecting against front-running attacks).
- **Dynamic sovereign identities.** Users (or organizations) fully control their identities (public keys) and can update them in a third-party verifiable way.
- **Decentralization.** There are no single points of compromise or failure.

3.3 System Model

There are four main entities in Calypso: *writers* who put secrets on-chain, *readers* who retrieve secrets, an *access-control collective authority* that is responsible for logging write and read transactions on-chain and enforcing access control for secrets, and a *secret-management collective authority* that 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. We call the nodes of a cothority *trustees*. For example, the set of Bitcoin miners can be considered a cothority that maintain the consistency of Bitcoin's state. The access-control cothority requires a Byzantine fault-tolerant consensus [35, 36, 40, 52]. There are various ways to implement an access-control cothority, *e.g.*, as a set of permissioned servers that maintains a blockchain using BFT consensus or as an access-control enforcing smart contract on top of a permissionless cryptocurrency such as Ethereum. The secret-management cothority membership is fixed; it may be set up on a per-secret basis or in a more persistent setting, the differences of which are discussed in Section 4. 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.

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 on the blockchain before they can start any secret-sharing processes. We denote an access-control label by policy, where policy = pk_R is the simplest case with Ron being the only reader.

3.4 Threat Model

We assume that the adversary is computationally bounded, secure cryptographic hash functions exist, and there is a cyclic group $\mathbb{G}$ (with generator g) in which the decisional Diffie-Hellman assumption holds. We assume that participants, including trustees, verify the signatures of the messages they receive and process only those that are correctly signed.

For 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 for the access-control cothority, we either require an honest majority $n = 2f + 1$ for Nakamoto-style consensus [52] or $n = 3f + 1$ for classic BFT consensus [35][1]. In the secret-management cothority, we require $n = 2f + 1$ and set the threshold to recover a secret to $t = f + 1$.

We assume that readers and writers do not trust each other. We further 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. Finally, the writer can define a *barrier point*, an event before which no one can access the secret guaranteeing fair access. 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 [12] or homomorphic encryption [18].

3.5 Architecture Overview

On a high level Calypso enables Wanda, the writer, to share a secret with Ron, the reader, under a specific access-control policy. When Wanda wants to put a secret on-chain (see Figure 1), she encrypts

[1] We assume the associated network model is strong enough to guarantee the security of the blockchain used.

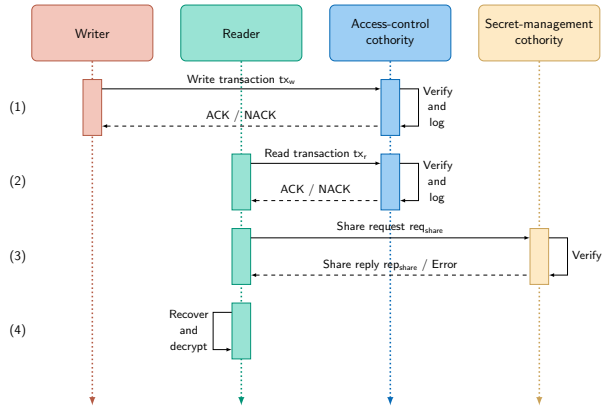

Figure 2: On-chain secrets protocol steps: (1) Write transaction, (2) Read transaction, (3) Share retrieval, (4) Secret reconstruction.

the secret and then sends a write transaction tx_w to the access-control cothority. The access-control cothority verifies and logs tx_w, making the secret available for retrieval by Ron, the authorized reader. To request access to a secret, Ron downloads the secret from the blockchain and sends to the access-control cothority a read transaction tx_r which carries a valid authorization from Ron's identity skipchain with respect to the current policy.

If Ron is authorized to access the requested secret, the access-control cothority logs tx_r. Subsequently, Ron contacts the secret-management cothority to recover the secret. The secret-management trustees verify Ron's request using the blockchain and check that the barrier point (if any) has occurred. Afterwards, the trustees deliver the secret shares of the key needed to decrypt Wanda's secret as shared in tx_w.

4 Calypso Design

In this section we introduce Calypso's components. First, we introduce two on-chain secrets protocols, *one-time secrets* and *long-terms secrets*, that provide auditable access control and fair data access. Both protocols can be enhanced for post-quantum security (see Appendix C). Second, we describe the *skipchain-based identity and access management* that adds dynamic access-control and self-sovereign identity management.

4.1 One-Time Secrets

In one-time secrets, 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, shares it via PVSS [62] for the secret-management cothority members, encrypts the secret with the key and stores the resulting ciphertext either on-chain or off-chain. It is important that Wanda binds the shared secret to the policy by deriving the base point of the PVSS consistency proofs from the policy. Finally, Wanda sends a write transaction tx_w to the access-control cothority to log the information for the verification and retrieval of her secret.

After some period of time, Ron, the reader, creates and sends to the access-control cothority a read transaction tx_r for Wanda's specific secret. The trustees check tx_r against the secret's policy and, if Ron is authorized to access the secret, they log the transaction creating a publicly-verifiable authorization proof. Ron sends this proof together with the encrypted secret shares from tx_w to each secret-management trustee and after tx_w is verified, he retrieves the secret key shares. Once Ron has received a threshold of valid shares, he can recover the symmetric key and decrypt the data. We describe the protocols next.

4.1.1 *One-Time Secrets Protocols*

Write Transaction Protocol. Wanda, the writer and each trustee of the access-control cothority perform the following protocol to log the write transaction tx_w on the blockchain. Wanda initiates the protocol as follows.

(1) Compute $h = \mathrm{H}(\text{policy})$ to map the access-control policy to a group element h to be used as the base point for the PVSS polynomial commitments. This prevents replay attacks as described later.
(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 $0 \le j \le t-1$.
(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 policy $= \mathrm{pk}_R$ to designate Ron as the intended reader of the secret message m.
(5) Finally, prepare and sign the write transaction

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

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 $h = \mathrm{H}(\text{policy})$.
(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 tx_w in block b_w.

Read Transaction Protocol. After the write transaction has been recorded, Ron needs to log the read transaction 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 block b_w, which stores tx_w, from the access-control cothority.
(2) Check that $\mathrm{H}(c)$ is equal to H_c in tx_w to ensure that the ciphertext c of Wanda's secret has not been altered.
(3) Compute $H_w = \mathrm{H}(tx_w)$ as the unique identifier for the secret that Ron requests access to and determine the proof π_{tx_w} showing that tx_w has been logged on-chain.
(4) Prepare and sign the transaction

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

and send it to the access-control cothority.

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

(1) Retrieve 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 message m by running 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
$$\text{req}_\text{share} = [\text{tx}_\text{w}, \text{tx}_\text{r}, \pi_{\text{tx}_\text{r}}]_{\text{sig}_{\text{sk}_R}}$$
where π_{tx_r} proves that tx_r has been logged on-chain.
(2) Send 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 tx_w to verify the signature of req_share and π_{tx_r} to check that tx_r has been logged on-chain.
(2) Compute the decrypted share $s_i = (\widehat{s_i})^{\text{sk}_i^{-1}}$, create a NIZK proof π_{s_i} that the share was decrypted correctly (see Appendix A), and derive $c_i = \text{enc}_{\text{pk}_R}(s_i)$ to ensure that only Ron can access it.
(3) Create and sign the secret-sharing reply
$$\text{rep}_\text{share} = [c_i, \pi_{s_i}]_{\text{sig}_{\text{sk}_i}}$$
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 $s_i = \text{dec}_{\text{pk}_R}(c_i)$ 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 $k = \text{H}(s)$ and use it to decrypt the ciphertext c to obtain the message m.

4.2 One-Time Secrets and System Goals

4.2.1 Achieving System Goals. The one-time secrets protocol achieves all goals except for dynamic sovereign identities.

Confidentiality. 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 req_share, they respond with their secret shares encrypted under the public key listed in the policy from the respective 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 tx_w in hopes of having them decrypted by the secret-management cothority (replay attack). 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 obtain a third-party verifiable transaction inclusion proof.

Fair Access. Once a read transaction tx_r is logged by the access-control cothority and the barrier point has passed, 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.

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

4.2.2 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 shared 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 a relatively 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. Second, the transaction size increases linearly with the secret-management cothority size, as the secret-management trustees do not store any per-secret protocol. This means that the tx_w must contain the encrypted shares, NIZK proofs and the polynomial commitments. Lastly, one-time secrets shares are bound the initial set of trustees, preventing the possibility of updating the secret-management cothority.

4.3 Long-Term Secrets

Long-term secrets addresses the limitations of one-time secrets through a dedicated secret-management cothority that persists over a long period of time and that maintains a shared private-public key pair used to secure access to the secrets. After a one-time distributed key generation (DKG) phase 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 threshold (DKG) key. As a result, the overhead of encrypting secrets is constant as each tx_w 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 shared private key and reconstructing the symmetric key himself or with the help of a trustee he selects.

4.3.1 Long-Term Secrets Protocols

Setup Protocol. Initially, 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 [24] or asynchronous [32]. Given the rarity of the setup phase we run the DKG by Gennaro et al. [24] using the blockchain as a bulletin board which emulates synchronous communication.

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

Write Transaction Protocol. Wanda and the access-control cothority perform the following protocol to log the tx_w on the blockchain. Wanda initiates the protocol through the following steps.

(1) Retrieve the threshold public key pk_{smc} of the secret-management cothority.
(2) Choose a symmetric key k and encrypt the secret message m to be shared as $c_m = \text{enc}_k(m)$ and compute $H_{c_m} = \text{H}(c_m)$. Set policy = pk_R to designate Ron as the intended reader of the secret message m.
(3) Encrypt k towards pk_{smc} using a threshold variant of the ElGamal encryption scheme [67]. To do so, embed k as a point $k' \in \mathbb{G}$, pick a value r uniformly at random, compute $c_k = (\text{pk}_{\text{smc}}^r k', g^r)$ and create the NIZK proof π_{c_k} to guarantee that the ciphertext is correctly formed and resistant to replay attacks (see Appendix B).
(4) Finally, prepare and sign the write transaction

$$\text{tx}_\text{w} = [c_k, \pi_{c_k}, H_{c_m}, \text{policy}]_{\text{sig}_{\text{sk}_W}}$$

and send it to the access-control cothority.

The access-control cothority then logs the tx_w.

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

Read Transaction Protocol. After tx_w has been recorded, Ron needs to log on-chain a tx_r 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 tx_w, from the access-control cothority.
(2) Check that $\text{H}(c_m)$ is equal to H_{c_m} in tx_w to ensure that the ciphertext c_m of Wanda's secret has not been altered.
(3) Compute $H_\text{w} = \text{H}(\text{tx}_\text{w})$ as the unique identifier for the secret that Ron requests access to and determine the proof π_{tx_w} showing that tx_w has been logged on-chain.
(4) Prepare and sign the read transaction

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

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. 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

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

where π_{tx_r} proves that tx_r has been logged on-chain.
(2) Send $\text{req}_{\text{share}}$ 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 $u_i = (g^r \text{ pk}_R)^{\text{sk}_i}$ with a NIZK correctness proof π_{u_i}.
(2) Create and sign the secret-sharing reply

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

and send it back to Ron.

Secret Reconstruction Protocol. Ron retrieves the key k and recovers the secret m as follows.

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

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

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

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

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

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

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

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 $\text{pk}_{\text{smc}}^{r'}$. Ron can detect if the trustee carries out the recovery incorrectly.

4.3.2 Evolution of the Secret-Management Cothority. The secret-management cothority is expected to persist over a long period of time while remaining secure and available. However, a number of issues can arise over its lifetime. First, trustees can join and leave resulting in churn. Second, even if the secret-management cothority membership remains static, the private shares of the servers should be refreshed regularly (*e.g.*, every month) to provide backward secrecy. Lastly, the shared private key of the secret-management cothority should be rotated periodically (*e.g.*, once every year).

We address the first two problems by periodically re-sharing [73] the existing threshold 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 threshold public/private key pair (pk_{smc}, sk_{smc}), Calypso needs to collectively re-encrypt each individual secret under the new shared public key. To achieve this, we generate and use translation certificates [30] such that the secrets can be re-encrypted without the involvement of their writers and without exposing the underlying secrets.

4.4 Long-Term Secrets and System Goals

4.4.1 Achieving System Goals. Long-term secrets achieves its goals similarly to one-time secrets with the following differences.

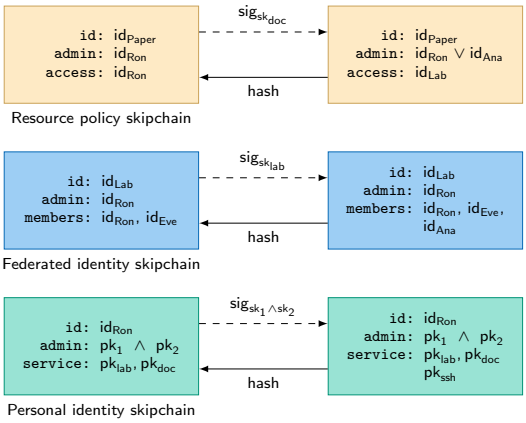

Figure 3: First, Ron updates his personal skipchain $\mathrm{id}_{\mathbf{Ron}}$ to include $\mathrm{pk}_{\mathbf{ssh}}$. He then uses $\mathrm{sk}_{\mathbf{lab}}$ to extend the federated skipchain $\mathrm{id}_{\mathbf{lab}}$ to add $\mathrm{id}_{\mathbf{Ana}}$ as a member. Finally, he adds $\mathrm{id}_{\mathbf{Ana}}$ as an admin and $\mathrm{id}_{\mathbf{lab}}$ as authorized readers to the policy skipchain $\mathrm{id}_{\mathbf{paper}}$ by using $\mathrm{sk}_{\mathbf{doc}}$.

Confidentiality. In long-term secrets, the secret message m is encrypted under a symmetric key k that is subsequently encrypted under a threshold 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 [67] 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 $\mathrm{tx_w}$ 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 $\mathrm{req_{share}}$, they respond with the blinded shares of the shared private key encrypted under the public key in the policy of the respective $\mathrm{tx_w}$. Based on the properties of the DKG protocol, the shared 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.

4.5 Skipchain-Based Identity and Access Management

The Calypso protocols described so far do not provide dynamic access control or sovereign identities. They only support static identities (public keys) and access policies as they provide no mechanisms to update these objects. However, these assumptions are rather unrealistic, as the participants might need to change or add new public keys to revoke a compromised private key or to extend access rights to a new device. 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. Finally, any access-control system that supports the above properties should prevent freeze attacks [60] and race conditions.

In order to achieve dynamic sovereign-identities, we introduce the *skipchain-based identity and access management* (SIAM) subsystem for Calypso that provides the following properties: (1) Supports identities for both individual users and groups. (2) Enables users to specify and announce updates to resource access keys and policies. (3) Enforces atomicity of accessing resources and updating resource access rights to prevent race conditions.

We achieve the first two goals of SIAM by deploying skipchains for groups and individuals. More specifically, we deploy three types of skipchains in Calypso (see Figure 3). *Personal identity skipchains* store the public keys that individual users control [34]. A user can maintain a number of public keys that may be used for access to resources by 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, etc. They are recursive in order to provide scaling and ease of use. *Resource policy skipchains* track access rights of identities, personal or federated, to certain resources and enable dynamic access control. In addition to listing federated identities and their public keys, policy skipchains include access-control rules to enforce fine-grained update conditions.

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. Similarly, Wanda can set up a resource policy skipchain id_P she is in charge of and include id_R as non-administrative members. Then, Wanda would use policy = id_P in $\mathrm{tx_w}$ seamlessly authorizing Ron to access the respective resource. Later Wanda can decide to revoke that resource for anyone, who has not yet accessed it, by setting policy = $\varnothing$.

Ensuring Atomicity. One key idea on Calypso's design is using the blockchain to timestamp the latest versions of the skipchains. This guarantees atomicity of events such as changing an identity (*e.g.*, to exclude someone) and later granting it more access rights. For example, administrator Wanda of the sales group, decides that Ron should be fired because he is performing industrial espionage, hence she removed the identity skipchain of Ron from the federated skipchain of the sales group. Afterwards Wanda grants the rest of her employees access to the new corporate strategy plan. In a naive asynchronous access control system in which policy changes can take varying amounts of time to propagate and take effect (*e.g.*, OAuth2 [26]), there is significant accidental time window in which Ron can still convince someone that he is part of the sales group, as he might be able to still prove membership to the controller of the the sensitive object (*i.e.*, to a threshold of trustees).

In Calypso, all the changes of the skipchains are serialized together with the $\mathrm{tx_r}$ and $\mathrm{tx_w}$ on-chain. Hence the exclusion of Ron will be strictly after the granting of access. This means that Ron will be unable to provide a correctly timestamped proof to the secret-management cothority and as a result be unable to read the sensitive document. Due to lack of space we describe how SIAM is integrated with on-chain secrets in Appendix E.

4.6 Further Security Considerations

Our contributions are mainly pragmatic rather than theoretical as we employ only existing, well-studied cryptographic algorithms. While we have already discussed how Calypso achieves its security goals in the previous sections, we discuss now the influence of malicious parties on Calypso.

Malicious Readers and Writers. Calypso's functionality resembles a fair-exchange protocol [54] 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. Calypso protects 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 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 tx_w binds 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 tx_w and it is sent by an authorized reader as defined in the policy of tx_w. A logged tx_r serves as an access approval. The secret-management cothority releases the decryption shares to the authorized reader only after seeing a valid tx_r logging proof.

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 tx_r he needs to retrieve and verify the encrypted data against the hash posted in 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.

5 Case Studies Using Calypso

Below we describe two real-world deployments, one completed and one in-progress, of Calypso that resulted from collaborations with companies that needed a flexible, secure, and decentralized solution to manage data. We also describe a zero-collateral, constant-round decentralized lottery and compare it with existing solutions.

5.1 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 policy of confidentiality can be enforced on D. We have realized this system with a contractor of the Ministry of Defense of a European country using a permissioned BFT blockchain and long-term secrets. The evaluation of this application is discussed in Section 7.2.

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 proper clearance. Clearance is granted to (or revoked from) 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 requirements and securely tracking accesses to D for auditing.

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 federated identity skipchains with all the identities that have clearance, id_A and id_B, respectively which include references to (a) federated skipchains for departments that have top-secret classification (*e.g.*, senior management), (b) federated skipchains for attributes that have top-secret classification (*e.g.*, ranked as captain) and (c) personal skipchains of employees that need exceptional clearance.

Organization A creates a document D, labels each paragraph as confidential or unclassified and, encrypts it using a different symmetric key. A shares the ciphertext with B and generates tx_w, which contains the symmetric keys of the classified paragraphs and policy = 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. Calypso logs the tx_r, creates a proof of access and delivers the key. Both organizations can update their identity skipchains as needed to ensure that at any given moment only authorized employees can access.

5.2 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 hospitals and research institutions from a European country to build a patient-centric system to share medical data based on long-term secrets. We do not provide evaluation of this application as it is similar to the previous one.

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 [28]. Patients dislike consenting once and completely losing control over their data and are more likely to consent to sharing their data with specific institutions [33]. 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. In the case that a patient is unable to grant access (unconscious), the medical doctor can request an exception (specified in the policy) and access the data while leaving an auditable proof.

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. This deployment is different from the previously-described one in 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's wishes with respect to her data. Policy skipchains can dynamically evolve by adding and removing authorized readers, and can include rich access-control rules.

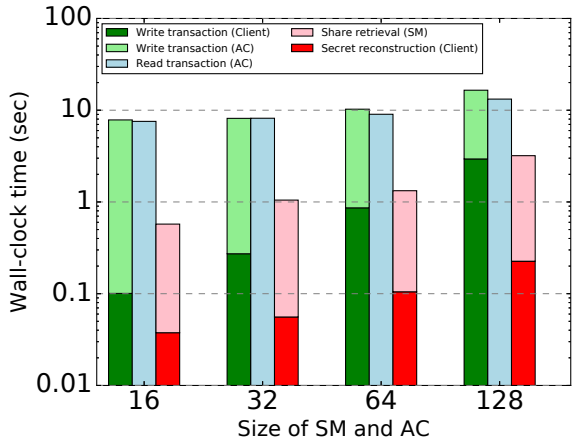

Figure 4: Latency of one-time secrets protocol for varying sizes of secret-management and access-control cothorities.

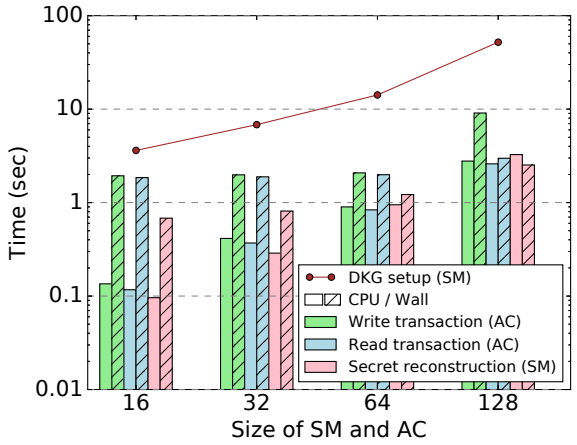

Figure 5: Latency of long-term secrets protocol for varying sizes of secret-management and access-control cothorities.

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 that consists of the 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. Hence, P remains in control of her data and can unilaterally update or revoke access.

5.3 Decentralized Lottery

Problem Definition. We assume there is a set of n participants who want to run a decentralized zero-collateral lottery selecting one winner. The lottery is managed by a smart contract that collects the bids and waits for the final randomness to decide on the winner. The evaluation of this application is in Section 7.3.

Solution with Calypso. Each participant creates a tx_w where the secret is their contribution to the randomness calculation and shares it using long-term secrets. After a predefined number of blocks (the barrier point) the input phase of the lottery closes. Afterwards the smart contract creates a tx_r to retrieve all inputs submitted before the barrier point and posts the reconstructed values and the corresponding proofs.

Once the final randomness has been computed as an XOR of all inputs, the smart contracts uses it to select the winner.

Comparison to Existing Solutions. Prior proposals for decentralized lotteries either need collateral (*e.g.*, Ethereum's Randao [57]) or run in a non-constant number of rounds [50]. Calypso enables a simpler decentralized lottery design, as the lottery executes in one round and needs no collateral because the participants cannot predict the final randomness or abort prematurely.

6 Implementation

We implemented all components of Calypso, on-chain secrets and SIAM, in Go [25]. For cryptographic operations we relied on Kyber [41], an advanced cryptographic 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 available implementation of ByzCoin [35], a scalable Byzantine consensus protocol. We implemented both on-chain secrets protocols, one-time and long-term secrets, run by the secret-management cothority. All our implementations are available as open source on GitHub and have gone through an independent security audit.

7 Evaluation

To evaluate Calypso, we use micro-benchmarks to compare on-chain secrets against a semi-centralized solution using simulated workloads. We further evaluate Calypso using simulated and real data traces in the context of clearance-enforcing document sharing (see Section 5.1) and a decentralized lottery (see Section 5.3). The synthetic workloads we generated were significantly heavier than those from the real data traces. For the experimental evaluation of SIAM, see Appendix F. We ran all our experiments on four Mininet [51] servers, each equipped with 256 GB of memory and 24 cores running at 2.5 GHz. To simulate a realistic network, we configured Mininet with a 100 ms point-to-point latency between the nodes and a per-node bandwidth of 100 Mbps.

We remark that we implemented and evaluated a centralized (a single server) solution and it trivially outperforms both, the state-of-the-art semi-centralized access control system and Calypso, which is fully decentralized. For this reason, we chose to focus our evaluation section on comparing Calypso to the aforementioned semi-centralized solution.

7.1 Mirco-benchmarks

The two primary questions we want to answer for on-chain secrets are whether the latency of read and write transactions is acceptable when deployed on top of blockchain systems and whether it can scale to hundreds of trustees to emulate a high degree of decentralization. We compare Calypso against a semi-centralized set-up (secrets stored off-chain and access policies enforced by the access-control cothority). We measure the overall latency of both on-chain secrets protocols, where we separately analyze the cost of the write, read, share retrieval and share reconstruction

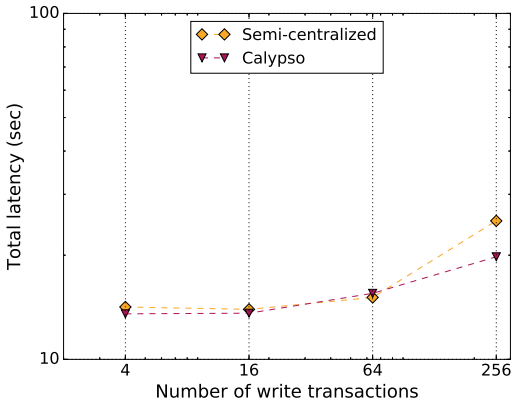

Figure 6: Write transaction latency for different loads in clearance-enforcing document sharing.

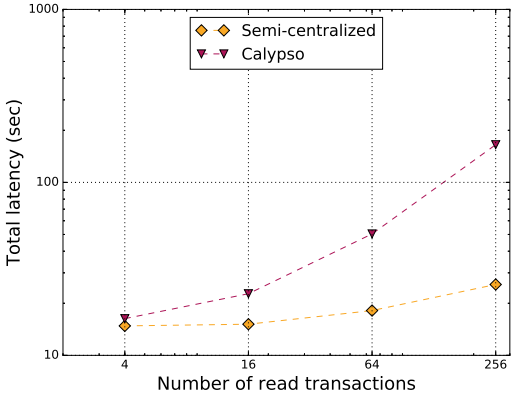

Figure 7: Read transaction latency for different loads in clearance-enforcing document sharing.

sub-protocols. We vary the number of trustees in the secret-management and access-control cothorities, where all trustees belong to both.

One-time Secrets. Figure 4 shows the latency results. We observe that the client-side creation of the tx_w is a costly operation, which takes almost one second for 64 trustees. This is expected as preparing the tx_w involves picking a polynomial, evaluating it at n points, and setting up the PVSS shares and commitments. Our experiments also show that verifying the NIZK decryption proofs and recovering the shared secret is substantially faster than creating the tx_w and differ by an order of magnitude for large numbers of shares because verifying the NIZK proofs and reconstructing the shared secret require less ECC operations than the setup of the PVSS shares. Finally, the overhead for the secret recovery on the secret-management cothority is an order of magnitude higher than on the client side since the client has to send a request to each trustee.

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 scale linearly in the size of the cothority. Even for a cothority of 128 servers, it takes less than 8 seconds to process a tx_w. The CPU-time is significantly lower than the wall-clock time due to the network overhead included in the wall-clock measurements. While the DKG setup is quite costly, it is a one-time event per epoch. The overhead of the share retrieval is linear in the size of secret-management cothority as it depends on sharing threshold t.

7.2 Clearance-Enforcing Document Sharing

We compare the clearance-enforcing document sharing deployment of Calypso with a state-of-the-art semi-centralized access-control system, where accesses and policies are logged on-chain but the data is managed in the cloud. We do not compare Calypso against fully-centralized solutions because they do not provide any decentralization, which is one of our main goals. However, we acknowledge that a fully-centralized solution would significantly outperform Calypso. We vary the simulated workload per block from 4 to 256 concurrent read and write transactions and report the time it takes to execute all transactions. These experiments use a blockchain with a blocktime of 7 seconds. Figure 6 shows

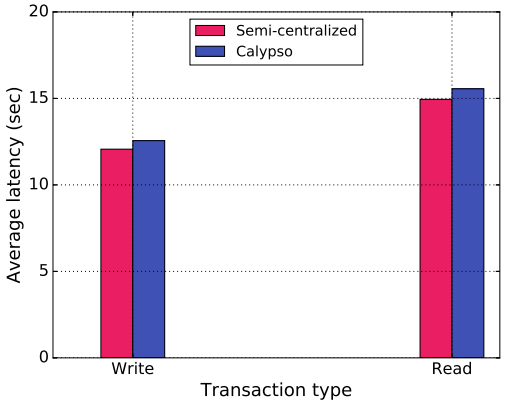

Figure 8: Average write and read transaction latencies replaying real-world data traces from clearance-enforcing document sharing.

that Calypso and the semi-centralized solution have comparable latency overhead for executing the write transactions. This is due to the fact that in both solutions, latency is dominated by the time it takes to store the write transactions on the blockchain. Figure 7 shows that Calypso takes 0.2× to 5× more time to execute the read transactions compared to the semi-centralized solution. The higher overhead of Calypso is due to the share reconstruction protocol.

Next, we evaluate the clearance-enforcing document sharing deployment of Calypso using real-world data traces from our governmental contractor partner mentioned in Section 5.1. Data traces are collected from the company's testbed over a period of 15 days. There are 1821 tx_w and 1470 tx_r, and the minimum, maximum and average number of transactions per block are 1, 7 and 2.62, respectively. We replayed the traces on Calypso and semi-centralized access-control system implementations. We use a blocktime of 10 seconds as it is in the original data traces. Figure 8 shows the average latency for the write and read transactions. The results show that Calypso and the semi-centralized system have comparable performance as latency is dominated by the blocktime.

We would like to remark that in both of our experiments, our blocktimes are significantly smaller than those of the current blockchains, such as Bitcoin. However, the goal of our experiments is

to show the additional performance overhead of moving from a semi-centralized system to Calypso, a fully-decentralized system. Since both systems incur the same blocktime cost, it does not affect our comparative analysis. We point out that using a blockchain with larger blocktimes would clearly increase the latency values for both Calypso and the semi-centralized implementation. However, in this case Calypso's overhead compared to the semi-centralized system becomes negligible, as the blocktime dominates the total latency.

7.3 Decentralized Lottery

We compared our Calypso-based zero-collateral lottery with the corresponding lottery by Miller et al. [50] (tournament) simulated and real workloads. Figure 9 shows that Calypso-based lottery performs better both in terms of the overall execution time and necessary bandwidth. Specifically, our lottery runs in one round (it always takes two blocks to finish the lottery) while the tournament runs in a logarithmic number of rounds due to its design consisting of multiple two-party lotteries.

Next, we evaluate both lottery implementations using the transactions from Fire Lotto [1], an Ethereum-based lottery, see Figure 10 for overall time comparisons. We considered transactions sent to the Fire Lotto smart contract over a period of 30 days and each data point in the graph corresponds to a single lottery run. As before, Calypso-based lottery performs better because it completes in one round whereas the tournament lottery requires a logarithmic number of interactions with the blockchain and consequently has a larger overhead. More specifically, while the blocktime of 15 seconds makes up $14 - 20\%$ of the total latency in Calypso, it contributes most of the per-round latency to the tournament lottery. We remark that our results only include the latency of the reveal phase since the commit phase happens asynchronously over a full day.

8 Related Work

The decentralized data management platform Enigma [77, 78] 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 that 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 [7, 13, 29, 65, 78] rely on centralized key-management and/or storage systems as well and hence suffer from similar issues with respect to atomicity and robustness against malicious service providers. Other privacy-focused blockchains [49, 61] do not address the issue of data sharing and access control but instead focus on hiding identity and transaction data via zero-knowledge proofs.

Existing decentralized identity-management systems, such as UIA [21] or SDSI/SPKI [58] 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 [26] 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 as in OAuth2 [45]. Finally, ClaimChain [38] is a decentralized PKI where users maintain repositories of claims about themselves and their contacts' public keys. However, it permits transfer of access-control tokens, which can result in unauthorized access to the claims. Finally, Blockstack [2] uses Bitcoin to provide naming and identity functionality, but it does not support private-data sharing with access control. Calypso can work along a Blockstack-like system if implemented on top of an expressive enough blockchain [74] and include Blockstack identities as part of SIAM.

A concurrent work to Calypso is Coconut [68] that uses threshold encryption to enable the issuance of privacy preserving credentials, which can also be an application of Calypso where the secret-management cothority manages the credentials of users and selectively reveals them when asked. Finally, CHURP [47] is also a concurrent work for efficient re-sharing of secrets that can also improve the performance of our long-term secrets implementation.

Acknowledgments

We would like to thank Vincent Graf, Jean-Pierre Hubaux, Wouter Lueks, Massimo Marelli, Carmela Troncoso, Juan-Ramón Troncoso-Pastoriza, and Frédéric Pont, for their comments and feedback. This project was supported in part by the grant #2017 − 201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain and by grants from the AXA Research Fund, Byzgen, DFINITY, and the Swiss Data Science Center (SDSC). Eleftherios Kokoris-Kogias is supported in part by the IBM PhD Fellowship. Philipp Jovanovic and Ewa Syta are supported in part by the Research Institute.

References

[1] 2018. Fire Lotto blockchain lottery.
[2] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. 2016. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 181–194.
[3] Ali Nakhaei Amroudi, Ali Zaghain, and Mahdi Sajadieh. 2017. A Verifiable (k, n, m)-Threshold Multi-secret Sharing Scheme Based on NTRU Cryptosystem. *Wireless Personal Communications* 96, 1 (2017), 1393–1405.
[4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. 30:1–30:15. https://doi.org/10.1145/3190508.3190538
[5] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. 2018. Channels: Horizontal Scaling and Confidentiality on Permissioned Blockchains. In *European Symposium on Research in Computer Security*. Springer, 111–131.
[6] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 443–458.
[7] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. 2016. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*. IEEE, 25–30.
[8] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, Vol. 48. 313–317.

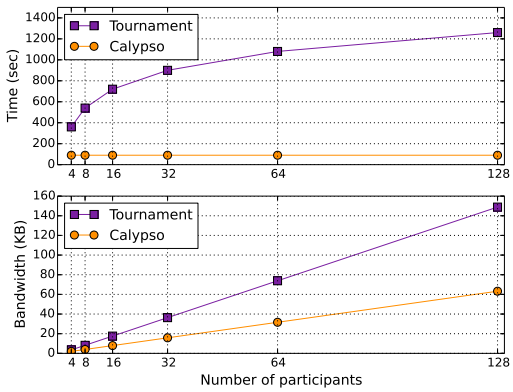

Figure 9: Lottery evaluation using simulated workloads.

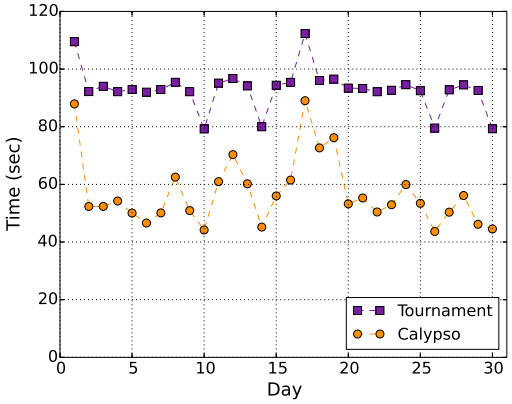

Figure 10: Lottery evaluation using Fire Lotto workloads.

[9] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 553–570.

[10] CoinDesk. 2018. Decentralized Exchanges ArenâĂŹt Living Up to Their Name - And Data Proves It.

[11] Matt Czernik. 2018. On Blockchain Frontrunning .

[12] Irit Dinur and Kobbi Nissim. 2003. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 202–210.

[13] Alevtina Dubovitskaya, Zhigang Xu, Samuel Ryu, Michael Schumacher, and Fusheng Wang. 2017. Secure and Trustable Electronic Medical Records Sharing using Blockchain. *arXiv preprint arXiv:1709.06528* (2017).

[14] Vincent Durham. 2011. Namecoin.

[15] Justin Ellis. 2014. The Guardian introduces SecureDrop for document leaks. *Nieman Journalism Lab* (2014).

[16] Tom Embury-Dennis and Simon Calder. 2018 (accessed on Oct 27, 2018). British Airways website theft: Customers urged to contact banks as airline launches investigation over stolen data.

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

[18] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012 (2012), 144.

[19] Joan Feigenbaum. 2017. Multiple Objectives of Lawful-Surveillance Protocols (Transcript of Discussion). In *Cambridge International Workshop on Security Protocols*. Springer, 9–17.

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

[21] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. 2006. Persistent Personal Names for Globally Connected Mobile Devices. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[22] Roxana Geambasu, Tadayoshi Kohno, Amit A Levy, and Henry M Levy. 2009. Vanish: Increasing Data Privacy with Self-Destructing Data.. In *USENIX Security Symposium*. 299–316.

[23] Genecoin. 2018. Make a Backup of Yourself Using Bitcoin.

[24] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, Vol. 99. Springer, 295–310.

[25] golang 2018. The Go Programming Language.

[26] Ed Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749.

[27] HIPAA Journal. 2017. The Benefits of Using Blockchain for Medical Records.

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

[29] Longxia Huang, Gongxuan Zhang, Shui Yu, Anmin Fu, and John Yearwood. [n.d.]. SeShare: Secure cloud data sharing based on blockchain and public auditing. *Concurrency and Computation: Practice and Experience* ([n. d.]).

[30] Markus Jakobsson. 1999. On quorum controlled asymmetric proxy re-encryption. In *Public key cryptography*. Springer, 632–632.

[31] IBM Blockchain Juan Delacruz. 2018. Blockchain is tackling the challenge of data sharing in government.

[32] Aniket Kate and Ian Goldberg. 2009. Distributed Key Generation for the Internet. In *29th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 119–128.

[33] Katherine K Kim, Pamela Sankar, Machelle D Wilson, and Sarah C Haynes. 2017. Factors affecting willingness to share electronic health data among California consumers. *BMC medical ethics* 18, 1 (2017), 25.

[34] Eleftherios Kokoris-Kogias, Linus Gasser, Ismail Khoffi, Philipp Jovanovic, Nicolas Gailly, and Bryan Ford. 2016. *Managing Identities Using Blockchains and CoSi*. Technical Report. 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016).

[35] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium*.

[36] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *39th IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–34.

[37] Charles R Korsmo. 2013. High-Frequency Trading: A Regulatory Strategy. *U. Rich. L. Rev.* 48 (2013), 523.

[38] Bogdan Kulynych, Wouter Lueks, Marios Isaakidis, George Danezis, and Carmela Troncoso. 2018. ClaimChain: Improving the Security and Privacy of In-band Key Distribution for Messaging. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. ACM, 86–103.

[39] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. 2015. How to use bitcoin to play decentralized poker. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 195–206.

[40] Jae Kwon. 2014. TenderMint: Consensus without Mining. (2014).

[41] kyber 2010 – 2018. The Kyber Cryptography Library.

[42] Timothy B. Lee. 2018 (accessed on Jul 27, 2018). FacebookâĂŹs Cambridge Analytica Scandal, Explained [Updated].

[43] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. 2004. Secure Untrusted Data Repository (SUNDR). In *6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[44] Laure A Linn and Martha B Koo. 2016. 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*.

[45] T. Lodderstedt, S. Dronia, and M. Scurtescu. 2013. OAuth 2.0 Token Revocation. RFC 7009.

[46] W Lueks. 2017. *Security and Privacy via Cryptography Having your cake and eating it too*. Ph.D. Dissertation. [Sl: sn].

[47] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. 2019. Dynamic-Committee Proactive Secret Sharing. (2019).

[48] David Mazieres and Dennis Shasha. 2002. Building secure file systems out of Byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. ACM, 108–117.

[49] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *34th IEEE Symposium on Security and Privacy (S&P)*.

[50] Andrew Miller and Iddo Bentov. 2017. Zero-collateral lotteries in Bitcoin and Ethereum. In *Security and Privacy Workshops (EuroS&PW), 2017 IEEE European Symposium on*. IEEE, 4–13.

[51] mininet 2018. Mininet – An Instant Virtual Network on your Laptop (or other PC).

[52] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.

[53] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. 2017. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 1271–1287.
[54] Henning Pagnia and Felix C Gärtner. 1999. *On the impossibility of fair exchange without a trusted third party.* Technical Report. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany.
[55] Marc Pilkington. 2015. Blockchain Technology: Principles and Applications. *Research Handbook on Digital Transformation* (2015).
[56] Bahman Rajabi and Ziba Eslami. 2018. A Verifiable Threshold Secret Sharing Scheme Based On Lattices. *Information Sciences* (2018).
[57] randao.org. 2018. Randao: Blockchain Based Verifiable Random Number Generator.
[58] R.L. Rivest and B. Lampson. 1996. SDSI: A Simple Distributed Security Infrastructure.
[59] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).
[60] Justin Samuel, Nick Mathewson, Justin Cappos, and Roger Dingledine. 2010. Survivable Key Compromise in Software Update Systems. In *17th ACM Conference on Computer and Communications security (CCS)*.
[61] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 459–474.
[62] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *IACR International Cryptology Conference (CRYPTO)*. 784–784.
[63] Adam Schwartz and Cindy Cohn. 2018 (accessed October 27, 2018). âĂIJInformation FiduciariesâĂİ Must Protect Your Data Privacy.
[64] SECBIT. 2018. How the winner got Fomo3D prize - A Detailed Explanation.
[65] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. 2017. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*. ACM, 45–50.
[66] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613.
[67] Victor Shoup and Rosario Gennaro. 1998. Securing threshold cryptosystems against chosen ciphertext attack. *Advances in Cryptology — EUROCRYPT'98* (1998), 1–16.
[68] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, and George Danezis. 2018. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. *arXiv preprint arXiv:1802.07344* (2018).
[69] Hemang Subramanian. 2017. Decentralized blockchain-based electronic marketplaces. *Commun. ACM* 61, 1 (2017), 78–84.
[70] Melanie Swan. 2015. *Blockchain: Blueprint for a new economy.* O'Reilly Media, Inc.
[71] Martin Holst Swende. 2017. Blockchain Frontrunning .
[72] Nick Szabo. 1994. Smart contracts. *Unpublished manuscript* (1994).
[73] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. 2002. Verifiable secret redistribution for archive systems. In *Security in Storage Workshop, 2002. Proceedings. First International IEEE*. IEEE, 94–105.
[74] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper* (2014).
[75] David Yermack. 2017. Corporate governance and blockchains. *Review of Finance* 21, 1 (2017), 7–31.
[76] Ernst & Young. 2018. Blockchain in health.
[77] Guy Zyskind, Oz Nathan, et al. 2015. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 180–184.
[78] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471* (2015).

A Publicly Verifiable Secret Sharing

We follow the protocol in [62] 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 $N = \{1, \ldots, n\}$ 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 $i \in \{1, \ldots, n\}$, 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 $0 \le j < t$.
(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, $0 \le j < t$.
(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.

B Full Encryption/Decryption Protocol for Long-term secrets

We follow the extension of the TDH2 protocol of Shoup [67] described by Lueks [46]. 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 threshold public key $\mathrm{pk}_{\mathrm{smc}}$ such that it can be decrypted by anyone that is included in policy[2] $L \in \{0,1\}^l$. The client performs the following steps.

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

(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.

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.

C Post-Quantum One-Time Secrets

The one-time secrets implementation can be converted to a post-quantum secure version by using Shamir's secret sharing [66]. We need the following assumptions to provide confidentiality. First, we assume that Wanda has post-quantum confidential and authenticated point-to-point communication channels [9] with the trustees. Second, we assume that the cryptographic protocols (for access control, authentication and blockchain security) are upgraded gradually over time to achieve post-quantum security. To protect Calypso from confidentiality violations by quantum attackers, we need to ensure that the on-chain secrets generated now are post-quantum secure.

Unlike the publicly-verifiable scheme we previously used, Shamir's secret sharing does not prevent a malicious writer from distributing bad secret shares. To mitigate this problem, we 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 the consistency of their secret share against 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 a bad transaction should he fail to correctly decrypt the secret message.

We sketch the protocol for one-time secrets below. We remark that long-term secrets can also achieve post-quantum security through verifiable secret sharing that relies on lattices [56] or NTRU [3].

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:

(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 for encrypting the secret message m. $c = \mathrm{enc}_k(m)$ 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 secure channel.

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

- Check 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 $\mathrm{tx_w}$.

- 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 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

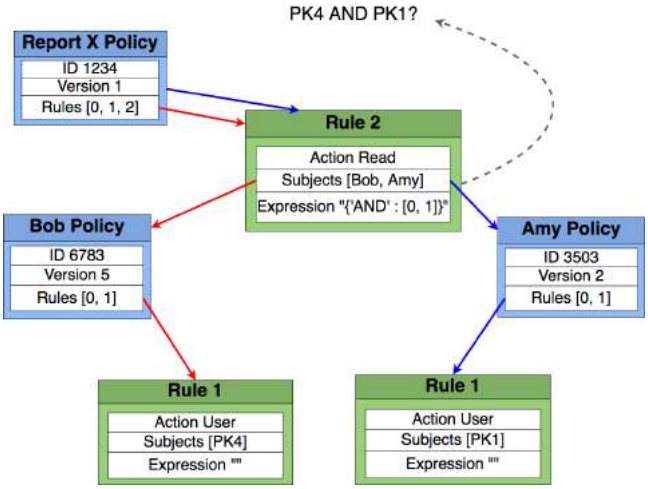

Figure 11: Verifier's path checking for multi-signature requests.

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 (*e.g.*, provided bad shares or used a higher-order polynomial). In response, Ron can release the transcript of the $\mathrm{tx_r}$ protocol in order to hold Wanda accountable.

D 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 ($\mathrm{sig_{req}}$) and the public key used (pk). On receiving an access request, the verifier checks that the $\mathrm{sig_{req}}$ 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 $(\mathrm{req}, [\mathrm{sig_{req}}])$ where $[\mathrm{sig_{req}}]$ is a list of signatures from the required-for-access parties. The verification process is similar to the single signature case.

Figure 11 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 âĂŸANDâĂŹ : [0,1] is checked against the signatures. If all checks pass, the request is considered to be verified.

JSON Access-Control Language A sample policy for a document, expressed in the JSON based language, is shown in Figure 12. 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]}"
         }
    ]
}
```

Figure 12: Sample Policy in JSON access-control language.

E Integration of SIAM and Calypso

To integrate SIAM with Calypso, the long-term secrets protocols described in Section 4.3 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 policy = id_R instead of policy = pk_R in $\mathrm{tx_w}$.

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

$$\mathrm{tx_r} = [H_w, \pi_{\mathrm{tx_w}}, \mathrm{pk}_{R'}]_{\mathrm{sig}_{\mathrm{sk}_{R'}}}$$

where $H_w = \mathrm{H}(\mathrm{tx_w})$ is the unique identifier for the secret that Ron requests access to, $\pi_{\mathrm{tx_w}}$ is the blockchain inclusion proof for $\mathrm{tx_w}$, 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

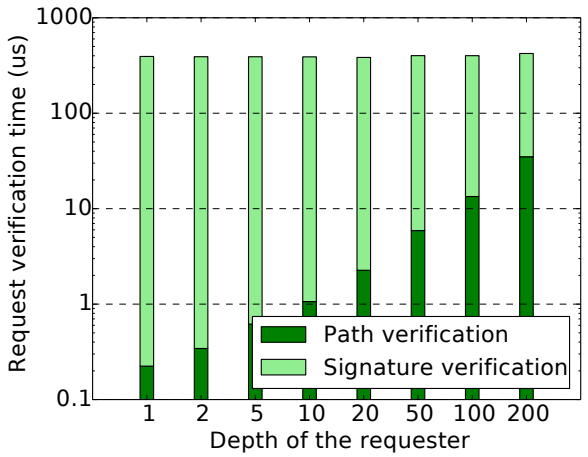

Figure 13: Single-signature request verification.

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 tx_r using $\mathrm{pk}_{R'}$ and, if valid, logs tx_r. Once tx_r is logged, the rest of the protocol works as described in Section 4.3, where the secret-management cothority uses $\mathrm{pk}_{R'}$ for re-encryption to enable Ron to retrieve the resource.

F SIAM Evaluation

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 complex access-control rules.

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 13 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 µs and accounts for 92.04 – 99.94% of the total time. We observe that even at a depth of 200, a relatively extreme scenario, path finding takes only about 35 µs.