CALYPSO: Private Data Management for Decentralized Ledgers

Eleftherios Kokoris-Kogias
IST Austria & Novi Research
ekokoris@ist.ac.at

Enis Ceyhun Alp
EPFL
enis.alp@epfl.ch

Linus Gasser
EPFL
linus.gasser@epfl.ch

Philipp Jovanovic
UCL
p.jovanovic@ucl.ac.uk

Ewa Syta
Trinity College
ewa.syta@trincoll.edu

Bryan Ford
EPFL
bryan.ford@epfl.ch

Abstract

Distributed ledgers provide high *availability* and *integrity*, making them a key enabler for practical and secure computation of distributed workloads among mutually distrustful parties. Many practical applications also require strong *confidentiality*, however. This work enhances permissioned and permissionless blockchains with the ability to manage confidential data without forfeiting availability or decentralization. The proposed Calypso architecture addresses two orthogonal challenges confronting modern distributed ledgers: (a) enabling the auditable management of secrets and (b) protecting distributed computations against arbitrage attacks when their results depend on the ordering and secrecy of inputs.

Calypso introduces on-chain secrets, a novel abstraction that enforces atomic deposition of an auditable trace whenever users access confidential data. Calypso provides user-controlled consent management that ensures revocation atomicity and accountable anonymity. To enable permissionless deployment, we introduce an incentive scheme and provide users with the option to select their preferred trustees. We evaluated our Calypso prototype with a confidential document-sharing application and a decentralized lottery. Our benchmarks show that transaction-processing latency increases linearly in terms of security (number of trustees) and is in the range of 0.2 to 8 seconds for 16 to 128 trustees.

1 Introduction

Blockchain or distributed ledger technology (DLT) enables the secure, public exchange of digital information and assets without trusted intermediaries, a foundation useful to many applications. Decentralized data sharing enables data markets that promise to put end-users back in control over how their data is shared and used, instead of surrendering data governance to a few tech giants [53, 71]. Decentralized data sharing can facilitate cooperation by mutually-distrustful parties, such as competing businesses or different countries. Decentralized data sharing can increase transparency of data processing, which is particularly important for sensitive cases, such as lawful data access requests [26]. Decentralized data life-cycle management allows users to implement and use secure key-recovery mechanisms, *e.g.*, enabling journalists to create the digital analogue of a dead man's switch to protect themselves and their sources [23]. These technologies can also be used to guarantee fairness in lotteries [4], games [42], and trading [17].

In the above scenarios, the security of data exchange and management is crucial, particularly when the confidentiality of private user data is at stake. However, security is not a given in current decentralized data-sharing applications [45, 54]. Existing approaches either forfeit availability guarantees for private data [29] or fall back on semi-centralized solutions for key management [6, 79], thereby subjecting data privacy to single points of failure or compromise.

Furthermore, decentralized applications that rely on the timing of data disclosure to ensure fairness are often susceptible to front-running attacks, in which adversaries that have early access to proposed but not-yet-committed transaction information adapt their strategies unfairly [24]. Front-running makes decentralized exchanges exploitable [17, 72] or resort to centralized order-book matching as in the 0x Project [16]. Front-running makes decentralized lotteries require collateral [4] or many interactive rounds [48]. Automated front-running attacks can even hijack careful attempts to recover from smart contract bugs [60]. Current defense mechanisms are ad-hoc and difficult to deploy [18].

We introduce Calypso, a new secure data-management framework that addresses the challenge of providing fair and verifiable access to confidential information without relying on a trusted party. To achieve this goal our system needs to address three key challenges. First, Calypso must provide accountability for all accesses to confidential data, to ensure that data is not improperly disclosed and to enforce proper recording of data accesses. Ideally, such an accountability mechanism should not reveal the relationship between users, instead providing anonymity to data consumers. Second, Calypso must ensure that data owners retain control over the data they share and allow data consumers to access data even when their digital identities (public keys) change. In particular, Calypso should allow for flexible updates of access-control rules and user identities, *e.g.*, to add or revoke access rights or keys. Third, Calypso needs to support permissionless functionality in order to be deployable alongside existing open blockchain ecosystems. Figure 1 illustrates a typical Calypso-based data-sharing application, itself building on a novel *on-chain secrets (OCS)* abstraction that provides dynamic access control and identity management.

On-chain secrets addresses the first challenge by combining the availability and confidentiality expected of data-management systems with the decentralization expected of blockchains [39, 78]. On-chain secrets combines threshold cryptography [65, 68, 70] to *hold* secrets, with a blockchain to *manage* those secrets. The blockchain enforces access control policies, indelibly records access authorizations, and ensures that disclosures are atomic with respect to all state changes potentially affecting authorization. To enable dynamic access-control and identity management, Calypso combines on-chain secrets with skipchains [38, 52], which allow one blockchain to follow and track another efficiently. This yields the first decentralized role-based access-control system [63], enabling user-controlled consent management. Finally, Calypso supports

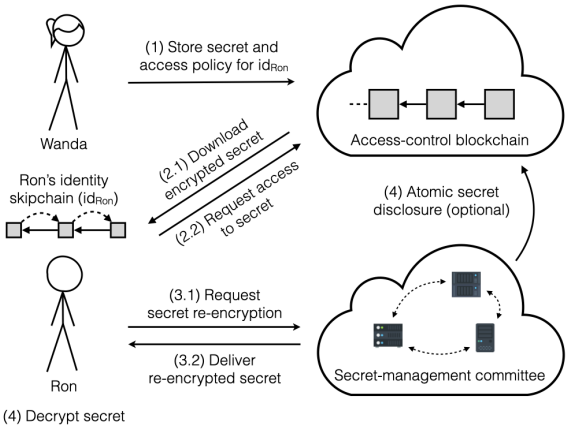

Figure 1: Auditable data sharing in Calypso

permissionless operation by building incentives around authorized data decryption, and allows users to select groups of secret-holding trustees based on personal preference and trust.

We implemented a prototype of Calypso in Go and evaluated it on commodity servers. Our experiments show that on-chain secrets scales linearly in the number of trustees (level of decentralization), exhibiting a moderate overhead of 2 to 17 seconds for 16 to 128 trustees. Furthermore, we evaluated two Calypso-based applications: secure document sharing and zero-collateral decentralized lottery. Our experiments use both synthetic and real-world workloads, and we compare Calypso to cloud-only and semi-centralized (cloud plus blockchain) solutions. For the document-sharing application, Calypso takes 10 to 20 (10 to 150) seconds to execute a write (read) request for low (4 trustees) to high (256 trustees) fault tolerance. For a realistic permissioned deployment, *e.g.*, with 4 to 16 trustees, Calypso adds negligible overhead compared with a semi-centralized approach. For the zero-collateral lottery, we show that our Calypso-based solution requires only a single round to finish, which simplifies and outperforms current state-of-the-art solutions requiring $\log n$ rounds to support n participants.

To summarize, this paper's contributions are as follows.

1) We introduce Calypso, a decentralized framework for auditable management of private data that provides threshold security guarantees for all three CIA properties (Section 3), ensures fairness, enables updates to access-control rules without compromising security, and protects reader privacy.

2) We present on-chain secrets (OCS) for transparent decentralized data management, and propose two variants for permissioned and permissionless deployments (Section 4 and 5).

3) We demonstrate the feasibility of using Calypso to address the data-sharing needs of real-world organizations by presenting three concrete use cases: auditable data sharing, data life-cycle management, and atomic data publication (Section 7). To evaluate our system and conduct these feasibility studies, we implemented Calypso, which was independently audited and is open-source (Section 8). The code and a demo may be found at https://github.com/calypso-demo/.

2 Motivation and Background

This section first motivates Calypso by describing how it enables data-management security as well as atomic data publication. We then summarize the main building blocks we employ.

2.1 Motivating Examples

2.1.1 Auditable Data Management: Centralized custodian systems that provide policy-based data publication mechanisms unlock a variety of useful data life-cycle management applications, such as automatic publication of confidential documents when certain conditions are met: *e.g.*, legal wills or estate plans. Another example is a digital "life insurance" policy for whistleblowers that publishes files automatically unless the custodian regularly receives a digitally-signed "heartbeat" message from the whistleblower [23, 59]. Moving to fully-decentralized custodians presents new challenges, however, in terms of how to specify and implement data publication and secure consent-management without fully trusting any centralized party, and how to integrate the publication and consent functions.

To enable secure decentralized data management, Calypso uses threshold cryptography and distributed ledger technology to protect the integrity and confidentiality of shared data, and to ensure accountability for data accesses by generating a third-party verifiable audit trail for data accesses. Calypso employs an expressive policy-control mechanism that enables atomic modification of access rights. Data owners can revoke the access of those who have not yet exercised their data access rights, and can know whether and which data was accessed before revocation. Decentralized application designers can use Calypso to achieve functionality such as monetizing data access, or providing proofs to aid investigations of data leaks or breaches. A representative application of this class is the document sharing application that we present in Section 7.1.

2.1.2 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 attacker can readily exploit early access to information to gain unfair advantage over honest participants through *front-running* [17, 66, 72]. Such attacks can affect decentralized applications such as lotteries [4], poker games [42], or exchanges [17]. In Calypso, inputs from the participants (*e.g.*, lottery randomness, trading bids, game moves) remain confidential up to a barrier point that is expressed by specific rules in a policy. After the barrier point, information is published and the computed result is atomically disclosed to every interested party: *e.g.*, which trades were successful at which prices, or the lottery winner. Consequently, Calypso resolves the tension between decentralization, fairness, and availability and provides a secure decentralized foundation for fair data disclosure. A representative application of this class is the zero-collateral lottery that we present in Section 7.3.

2.2 Blockchains and Skipchains

A blockchain is a distributed, append-only, tamper-evident log composed of blocks linked together via cryptographic hashes. Many decentralized applications build on blockchains [3, 22, 51]. Calypso does not attempt to innovate on this front and instead aims to be deployable alongside any blockchain or state-machine replication

system [13][1] that supports programmability and custom validation through some form of smart contracts [2, 74, 78].

Skipchains [52] track configuration changes of a decentralized authority or *cothority* [73] by using each block as a representation of all the public keys necessary to authenticate the next block. When a cothority's configuration evolves, it creates a new block with 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* [38] that clients follow to get up-to-date with the current authoritative group. In Section 4.3, we define identity and policy skipchains to track changes to the participants' identities and to data management policies governing their secrets, efficiently. Our construction is a simple extension of skipchains to support federated groups and enable expressive consent management.

2.3 Threshold Cryptosystems

A (t, n)-secret sharing scheme [9, 68] enables a dealer to share a secret s among n trustees such that any subset of t trustees can coordinate to reconstruct s, whereas any smaller subset learns *no* information about s. (t, n)-secret sharing can thus offer resilience against up to $t-1$ malicious participants, or $n-t$ offline participants, or both. Simple secret sharing schemes assume an honest dealer, however, leaving a single point of compromise. Verifiable secret sharing or VSS [27] solves this issue by enabling the trustees to check the shares they received from the dealer for consistency. Publicly verifiable secret sharing or PVSS [65] further allows external third parties, and not just recipients of shares, to verify all shares.

In distributed key generation or DKG [30, 36, 41], a set of n trustees create a collective private-public key pair $(\mathrm{sk}, \mathrm{pk})$ without a trusted dealer. Each trustee first acts as a VSS dealer, then the trustees combine their shared secrets. Each trustee i obtains a share sk_i of a joint secret key sk and a joint public key pk. Unlike regular VSS, no individual trustee learns anything about sk, and at least t trustees would have to collude to recover sk. After the DKG setup, trustees can use threshold protocols [76], including threshold encryption [70] or threshold signing [10, 69], in which at least t trustees collaborate while tolerating up to $n - t$ availability failures. This work uses threshold encryption to enable clients to place sensitive data into the custody of the trustees by encrypting the data using their joint public key pk.

3 Overview of CALYPSO

This section first uses two strawman solutions to illustrate the challenges a secure decentralized data-management system faces and how interconnected and fragile such a system's properties are, especially in a Byzantine environment. Based on these lessons we then summarize the system's goals and design, shown in Figure 1.

3.1 Strawman Data Management Protocols

Consider an application where Wanda is the operator of a paid service that provides asynchronous access to information about stock orders, and Ron is a customer. This application requires *auditability*, the main property we have set out to achieve. This means that Wanda should be able to audit the fact that Ron accessed the information and claim payment. At the same time, Ron should receive the stock information once he has paid the service fee even if Wanda is dishonest, *i.e.*, wants to steal the money and reveal nothing. Next, we present two strawman protocols and show that they provide auditability but not the other desirable properties.

3.1.1 Strawman I: The Trusted Custodian The first strawman assumes a simple trusted custodian. Wanda sends her information to the custodian, specifying that Ron can read the information if he pays the required fee. Wanda then publicly announces this on Bitcoin, which serves as both a public bulletin board and a payment processor. Ron sees the on-chain announcement, pays the fee in Bitcoin, then shows the transaction to the custodian, who in turn releases the information encrypted under Ron's public key.

Strawman I provides *auditability*, but places complete trust in the custodian. If the custodian crashes, Ron has no guarantee of getting the information. To avoid this risk, we need *decentralization* to tolerate any single point of failure. A malicious custodian could also give the information to Ron without payment, and without informing Wanda or leaving any record of this disclosure. We therefore require *confidentiality* with no single point of compromise, the third property of our system. Finally, even if the custodian does not compromise confidentiality, he releases the information on a first-come-first-serve basis. As a result, customers with better connectivity can make payments faster and thereby mount front-running attacks, on the stock market in this scenario. To protect against such attacks, we require *fair* access to published data.

3.1.2 Strawman II: The Secret Sharer The obvious way to decentralize data is via replication: *e.g.*, by having n custodians. Replication alone worsens the confidentiality of the system however, since we now have n custodians who can potentially leak confidential information. Instead, Strawman II symmetrically encrypts the information and publishes the encryption on-chain. The encryption does not have to be stored on-chain, but doing so guarantees high availability of the data. For confidentiality, Wanda uses (t, n)-threshold Shamir secret sharing [68] to split the encryption key between the n custodians. As a result, if fewer than t custodians are dishonest, both decentralization and confidentiality may be preserved.

Although Strawman II seems to solve two of our issues, it has critical weaknesses. One challenge is that Wanda can misbehave during the secret-sharing step, sending invalid secret shares to the custodians and preventing them from ever recovering any secret. In response, we force Wanda to post consistent shares on-chain using PVSS [65]. Now, however, an unauthorized adversary (*e.g.*, Eve) might mount a replay attack. Eve can copy the consistent shares available on-chain into a seemingly-independent new transaction, with Eve as the authorized reader and the payment recipient, and trick the custodians into decrypting it without authorization. Strawman II illustrates how fragile auditability is, and that simple solutions can easily fail if not carefully designed.

3.1.3 Additional Properties We now mention two more desirable properties. First, Ron and Wanda might not want their business relationship to be publicly known. We could use a blockchain providing sender anonymity, such as Zcash [64], for payment while hiding Ron's identity, but we would also like *receiver anonymity* to protect Wanda's identity. Second, if Ron needs to change his public key, he would lose access to all data authorized under the

[1] Directly inheriting their BFT properties.

old key. We ideally want Ron to have a *dynamic* self-sovereign identity [50] that he can evolve independently and retain access to previously-shared data. To provide all properties, we introduce three components and transform Strawman II into Calypso:[2]

(1) To enable auditability of data accesses and ensure atomic data delivery, we introduce *on-chain secrets* (OCS). We provide two constructions for OCS – long-term secrets (LTS) and one-time secrets (OTS) – in Section 4 and 5, which are suitable for permissioned and permissionless deployment, respectively.
(2) To enable receiver anonymity, we introduce in Section 4.2 the *on-chain blinded key exchange*, which enables Wanda to help Ron blind his identity, without forfeiting her right to hold him accountable.
(3) To enable decentralized, dynamic, self-sovereign identities and access policies, we extend skipchains and integrate them with Calypso in Section 4.3.

3.2 System Goals

Calypso sets out to address the following primary goals.

- **Auditability:** All data accesses are third-party verifiable and recorded in a tamper-resistant log.
- **Decentralization:** There are no single points of availability failure or security compromise in the system.
- **Confidentiality:** Secrets stored on-chain can be decrypted only by authorized clients after leaving an access record.
- **Fair access:** Clients are guaranteed authorized access to a secret after posting a valid access request on-chain. If a barrier point exists, authorized clients atomically get simultaneous access after the barrier, protecting against front-running.
- **Receiver anonymity:** An on-chain proof-of-access log does not identify the user unless an audit is requested.
- **Dynamic self-sovereign identities:** Users and organizations fully control the public keys representing their identities, and can update them verifiably and atomically.

3.3 System Model

There are four main participant roles in Calypso's architecture: *writers* who place secrets on-chain, *readers* who retrieve secrets, an *access-control blockchain* that is responsible for logging write and read transactions on-chain and enforcing access control for secrets, and a *secret-management committee* that is responsible for managing and delivering secrets. The access-control blockchain and secret-management committee can be deployed on the same set of servers or the access-control blockchain can be an independent blockchain (*e.g.*, Ethereum [78]) not managed by the system administrators. In the rest of the paper, we keep these roles separate for architectural clarity. We will use the names Wanda and Ron to refer to a (generic) writer and reader, respectively.

The access-control blockchain requires Byzantine fault-tolerant consensus [39, 40, 43, 51]. There are many ways to implement an access-control blockchain, *e.g.*, as a set of permissioned servers using BFT consensus or as an access-control smart contract on top of a permissionless cryptocurrency. The secret-management committee membership is fixed; it may be set up on a per-secret basis or in a more persistent setting, as discussed in Section 5.1. The secret-management trustees maintain their private keys and may need to maintain additional secret state, such as private-key shares. They do not run consensus for every transaction.

We denote the 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 the adversary is computationally bounded, secure cryptographic hash functions exist, and the decisional Diffie-Hellman assumption holds. We assume all participants verify the signatures of the messages they receive and process only those correctly signed. Calypso's goal is to enforce access accountability, but not application-specific data correctness, which is out of scope.

The secret-management committee consists of n trustees, of which f can fail or behave maliciously. We require $n \geq 2f + 1$ and set the secret-recovery threshold to $t = f + 1$. For the access-control blockchain we assume the relevant blockchain's trust assumptions hold: *e.g.*, that at most f of $3f + 1$ validators fail or misbehave in BFT-based consensus[39], or that less than 50% of the mining power colludes in a proof-of-work system like Bitcoin [51]).

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 committee, as readers can release a protocol transcript and prove the misbehavior of writers. Conversely, readers might try to get access to a secret and later claim 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, but after which anyone authorized can, ensuring fair access.

3.5 Architecture Overview

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 the secret and sends it in a write transaction $\mathrm{tx_w}$ to the access-control blockchain. The access-control blockchain verifies and logs $\mathrm{tx_w}$, making the encrypted secret available for Ron, the authorized reader. To access a secret, Ron retrieves the secret's ciphertext from the blockchain and sends to the access-control blockchain a read transaction $\mathrm{tx_r}$, containing an authorization from Ron's identity skipchain with respect to the current access-control policy.

If Ron is authorized to access the secret, the access-control blockchain logs $\mathrm{tx_r}$. Subsequently, Ron contacts the secret-management committee 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, trustees deliver the secret shares of the key needed to decrypt Wanda's secret as shared in $\mathrm{tx_w}$. In Section 4, we show the deployed system (Section 7) that adopts a permissioned model where trustees are externally accountable.

[2] Missing protocols and proofs are in the Appendix A

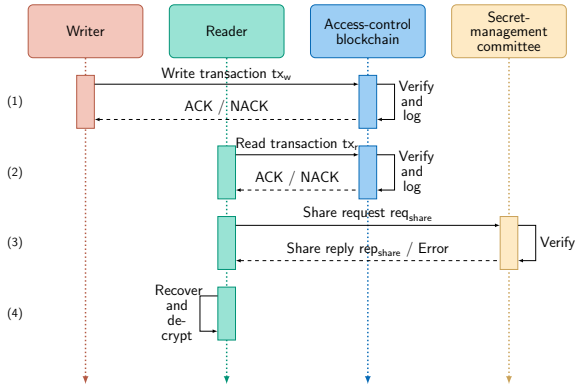

Figure 2: On-chain secrets protocol steps: (1) Write transaction, (2) Read transaction, (3) Share retrieval, (4) Secret reconstruction.

In Section 5, we extend Calypso to work in a permissionless model where clients choose the trustees on an ad-hoc basis and employ correct incentives using the blockchain as a payment layer.

4 Permissioned Deployment

In this section, we introduce Calypso's components and show how they achieve our goals. First, we introduce *long-terms secrets*, which provides auditable access-control and fair data-access in a permissioned setting utilizing well-defined sets of trustees for all clients. Second, we describe how Wanda can help Ron obfuscate his identity but still be able to deanonymize him in case of misbehavior. Finally, we describe *skipchain-based identity and access management*, which adds dynamic access control for Wanda and self-sovereign identity management for the public part of Ron's identity.

In both approaches, the writer can either encrypt the data directly or use a symmetric key and offload the storage of the symmetrically-encrypted data to a decentralized storage service such as IPFS [7]. If the encrypted secret is not on chain, the reader should ensure to obtain it (*e.g.*, by asking the writer to send it to him directly)[3] prior to requesting access in order to preserve decentralization.

4.1 Long-Term Secrets

To illustrate the challenges that long-term secrets address, we revisit the design of Strawman II, which uses PVSS to share secrets verifiably. While PVSS prevents Wanda from distributing bad shares, it still requires her to involve the trustees in every transaction she creates, and causes the size of each transaction to grow linearly in the number of trustees (one share per trustee).

To resolve these challenges, long-term secrets leverages the fact that the group of trustees is well-defined and persists over a period of time. Trustees generate a *shared* private-public key pair with a one-time DKG setup. Afterwards, Wanda uses threshold ElGamal encryption [19] to protect a symmetric key that is used to encrypt her secret. We make this transaction non-interactive with a zero-knowledge proof of correct encryption [14]. Overall this reduces the size of a write transaction from $O(n)$ to $O(1)$.

[3]He can ask the writer to send it to him directly, *e.g.*, if IPFS is unresponsive.

Next, Wanda must bind the secret shares to Ron's identity to prevent unauthorized reads by Eve. We do so by binding a policy to a ciphertext of the zero-knowledge proofs [46, 70].

Finally, we reduce Ron's overheads further by enabling him to delegate the verification and reconstruction of the $O(n)$ key shares to some trustee who Ron trusts only for liveness. This trustee does not obtain access to the secret itself, and Ron can always detect any misbehavior, and choose another trustee or finish the process himself. The full protocol is in the Appendix A.

4.1.1 Evolution of Secret-Management Committee: The secret-management committee is expected to persist over a longer period of time while maintaining security and availability. During its lifetime a number of issues can arise. First, trustees can join and leave the committee, causing churn. Second, even if the secret-management committee members do not change, the private shares of the servers should be refreshed regularly (*e.g.*, every month) to provide backward secrecy. Third, the collective key pair should be rotated periodically (*e.g.*, once every year) for additional protection in case the threat model is violated, *e.g.*, trustees do not delete their old key shares after a refresh or an adversary obtains more than t shares.

We address the first two issues by periodically running a *re-sharing* protocol that updates the private key shares but keeps the shared public key unchanged, which is crucial to avoid the need to re-encrypt all data under a new public key [77]. This step does not affect the availability of the system, which is particularly important when it comes to churn. More concretely, the re-sharing protocol is not, and does not need to be, triggered by members joining, as long-term secrets operates in a permissioned setting that requires approval of a new member. Hence, these joins are not "ad-hoc" or unexpected and the protocol can tolerate a threshold of members being temporarily or permanently unavailable.

Lastly, when the secret-management committee wants to rotate the threshold key pair completely, Calypso needs to collectively re-encrypt each individual secret under the new shared public key. To achieve this, we use translation certificates [35] that allow re-encryption of secrets without the involvement of their writers and without revealing the underlying secrets to any individual trustee.

4.2 On-chain Blinded Key Exchange

In our protocols so far, Wanda includes Ron's public key in a secret's policy to mark him as the authorized reader. Once Wanda's write transaction is logged, everyone knows that she has shared a secret with Ron. Correspondingly, once Ron's read transaction is logged, everyone knows that he has obtained the secret. While this property is desirable for some deployment scenarios we envision, certain applications may benefit from concealing the reader's identity.

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

(1) *Public Key Blinding.* Wanda generates a random blinding factor b and uses it to calculate a blinded version of Ron's public key $\text{pk}_{\widetilde{R}}$ from pk_R.
(2) *Write Transaction.* Wanda creates tx_w with the following modifications. Wanda encrypts b under pk_R to enable Ron to calculate the blinded version of his public key. Then, she uses $\text{pk}_{\widetilde{R}}$ instead of pk_R in the policy. After tx_w is logged, she notifies Ron on a separate, secure channel about tx_w and sends him $H(\text{pk}_R)$ as a fingerprint, which Ron uses to verify that Wanda used the correct (unblinded) encryption key.
(3) *Read Transaction.* To read Wanda's secret, Ron first retrieves b using his secret key sk_R. Then, he computes $\text{sk}_{\widetilde{R}} = b\,\text{sk}_R$ and uses this blinded private key to sign his tx_r anonymously.
(4) *Auditing.* If Wanda wants to prove that Ron generated the tx_r, she can release b. Then, anyone can unblind Ron's public key $\text{pk}_R = \text{pk}_{\widetilde{R}}^{-b}$, verify the signature on the transaction and confirm that only Ron could have correctly signed the transaction, as he is the only one who could calculate $\text{sk}_{\widetilde{R}}$.

4.3 Identity and Access Management

The above protocols do not yet offer dynamic access control or dynamic self-sovereign identities, only static public keys and access policies. In practice, participants may need to change or add public keys, to revoke a compromised private key, or to grant access to a new device. Similarly, it should be possible to change access polices in order to grant, update, or revoke resource access, and to define access-control rules for both individual identities and groups of users. Finally, an access-control system with the above properties should prevent freeze attacks [62] and race conditions.

To support dynamic self-sovereign identities, we build on role-based access control or RBAC [63]. RBAC policies evolve dynamically depending on the role of users, but traditionally rely on a central manager to assign roles. To address this challenge, the *skipchain-based identity and access management* (SIAM) subsystem of Calypso provides the following properties: (1) it supports identities for both individual users and groups; (2) it enables users to announce updates to resource access keys and policies; and (3) it prevents time-of-check-to-time-of-use (TOCTTOU) races by making data accesses atomic with respect to access policy updates.

We achieve the first two goals of SIAM by using skipchains [52] to encode individual identities and role memberships. As a result, our system is the first decentralized instantiation of RBAC. We use three types of skipchains in Calypso, as shown in Figure 3. *Personal identity skipchains* store the public keys that individual users control [38]. A user can have a number of public keys that may be used for accessing resources from different devices, for example. *Federated identity skipchains* specify identities and public keys of a collective identity that encompasses users with the same role (*e.g.*, part of the same group), such as employees of a company, members of a research lab, etc. They are recursive to provide scaling and ease of use. *Resource policy skipchains* track access rights of identities, personal or federated, to resources and enable dynamic access-control based on the role of each user. In addition to listing federated identities and their public keys, policy skipchains include access-control rules to enforce fine-grained update conditions.

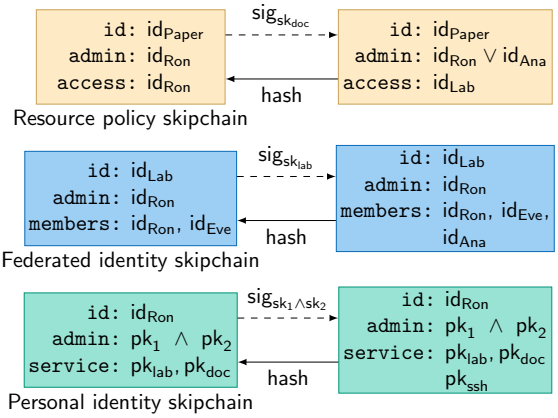

Figure 3: First, Ron updates his personal skipchain $\text{id}_\textbf{Ron}$ to include $\text{pk}_\textbf{ssh}$. He then uses $\text{sk}_\textbf{lab}$ to extend the federated skipchain $\text{id}_\textbf{lab}$ to add $\text{id}_\textbf{Ana}$ as a member. Finally, he adds $\text{id}_\textbf{Ana}$ as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

With SIAM, Ron can 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 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 = $\emptyset$.

Finally, SIAM can be made compatible with blinded key exchange (Section 4.2) as follows. After Ron receives a notification from Wanda about a new transaction, he checks via the received fingerprint $H(\text{pk}_R)$ that Wanda used his latest key. If not, Ron replies with an out-of-date error asking Wanda to re-do the encryption with his latest key, which she can look up via id_R.

Ensuring Atomicity: A final challenge for adapting RBAC in our setting is to guarantee atomicity of events, such as changing an identity (*e.g.*, to exclude someone and to later grant additional access rights). For example, Wanda, an administrator of the sales group, decides that Ron should leave the group and no longer be able to perform actions on behalf of the group. To do so, she removes Ron's identity skipchain from the federated skipchain of the sales group. Later on, however, Wanda may grant all employees access to the new corporate strategy plan. In a naive asynchronous access-control system, where it can take time for policy changes to propagate and for old cached credentials to expire (*e.g.*, as in OAuth2 [32]), there is a significant time window where Ron could access the shared document because he can still convince others that he belongs to the sales group until all his cached credentials have expired.

Calypso's design uses the blockchain to timestamp the latest skipchain versions, and skipchain changes are serialized together on-chain with the tx_w and tx_r that records and authorizes any data access. In the above example where Ron's access revocation is committed before Wanda's access grant, Ron is unable to give the secret-management committee a correct, timestamped tx_r proving access at any time, and thus cannot read the document.

5 Permissionless Deployment

This section introduces one-time secrets, an on-chain secrets protocol more suitable for a permissionless environment. One-time secrets does not assume the existence of a predefined set of trustees. Instead, it allows clients to choose the particular set of servers that will hold their secret and the secret recovery threshold. This flexibility comes at a cost, as transaction size is linear in the number of trustees, but the protocol remains non-interactive and the trustees can be stateless. Lastly, we show how to incentivize the trustees using the underlying cryptocurrency blockchain for payments.

5.1 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 id_R skipchain of the intended reader(s). She then runs PVSS [65] for her personal choice of secret-management committee members and uses the secret that was generated during PVSS as the symmetric key. To prevent against the replay attacks discussed in Strawman II, we bind the secret shares to the policy by deriving the base point of the PVSS consistency proofs from the policy. This may be done securely using Elligator maps [8]. See the Appendix D for details and an analysis of recommended group size based on Wanda's perception of how many adversarial nodes might collude.

Finally, Wanda sends tx_w to the access-control blockchain to log the secret and its access policy. Reading is similar to long-term secrets except for reconstruction, which is done by Ron.

Advantages and Shortcomings: One-time secrets 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, ad-hoc secret-management committee for each secret, without requiring the servers to maintain any protocol state.

One-time secrets has a few disadvantages, however. First, it incurs high PVSS setup and share reconstruction costs: Wanda needs to evaluate the secret-sharing polynomial at n points, and create n encrypted shares and NIZK proofs, along with t polynomial commitments. SCRAPE [12] can reduce the cost of verifying PVSS shares, but the cost remains linear in n. Second, the transaction size increases linearly with the secret-management committee size, as the secret-management trustees are stateless. This means that the tx_w must contain the encrypted shares, NIZK proofs and the polynomial commitments. Lastly, one-time secrets shares are bound to the initial set of trustees, so the secret-management committee cannot be changed without re-encrypting the secret.

5.2 Incentives

When deploying Calypso in a permissionless network, it is natural to question the trustees' motivation to participate. We envision a system where the trustees are service providers who establish their reputation to be selected by the users. For this reason, they lock collateral in order to participate. In the Appendix D we analyze the incentives assuming the trustees have locked collateral for one transaction. We expect, however, that the trustees provide more liquidity and Wanda is asking for collateral proportional to the value of her data when she creates a write transaction.

In this setting, we explore the best strategy for Ron and Wanda assuming that the trustees will act rationally. We assume that Wanda's data have some intrinsic value v, which Ron is willing to pay. Wanda will decide on a fraction $a < 1$, which is the fraction of v that the trustees will receive in exchange for protecting the secret. We have two challenges to solve: First, trustees might receive payment and do nothing. Second, trustees might accept a bribe and give Ron the data without waiting for a transaction on-chain.

To prevent a public-goods game [5], we must ensure: (a) only the first t trustees that provide decryption shares get paid (each one gets av/t) and (b) trustees that reply with an invalid share lose more than their expected payment. The solution to the invalid-share attack is also the solution to the second challenge, collateral. To disincentivize bribes, the trustees need to lock collateral. The total collateral locked by a threshold ($f + 1$) of trustees should be higher than v. Hence, we assume that every trustee locks v/f collateral. Ron may claim this collateral by proving that the trustee misbehaved (*e.g.*, produced an invalid share), and he gets the fraction a of it. The rest goes to Wanda. For the protocol to work the trustees send the encrypted shares on-chain claiming their payment, and on verification the smart contracts accepts them. Only the first t trustees get paid, and only after a dispute window Δ during which Ron can claim their collateral on proving misbehavior.

6 Achieving the System Goals

We now summarize how Calypso achieves the goals in Section 3.2.

Auditability: *All data accesses are third-party verifiable and recorded in a tamper-resistant log.*

Assuming the access-control blockchain implements Byzantine consensus, all correct read and write transactions are logged by the access-control blockchain. Once a transaction is logged, anyone can obtain a third-party-verifiable transaction-inclusion proof.

Decentralization: *There are no single points of availability failure or security compromise in the system.*

By design, the protocols do not assume a trusted third party. The secret-management committee tolerates up to $t - 1$ misbehaving trustees and up to $n - t$ offline trustees.

Confidentiality: *Secrets stored on-chain can be decrypted only by authorized clients after leaving an access record.*

With long-term secrets, the secret message m is encrypted under a symmetric key k that is then encrypted under a threshold public key of the secret-management committee. The ciphertext is bound to a specific policy through NIZK proofs [70] so it cannot be reposted in a new write transaction with a malicious reader listed in its policy. The access-control trustees log the write transaction tx_w that includes the encrypted key, which, because of the encryption scheme, does not leak any information about k. After the secret-management trustees receive a valid request $\mathrm{req}_\mathrm{share}$, they respond with the blinded shares of the shared private key encrypted under the public key in the policy of the respective tx_w. Due to 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. Thus, only the intended reader gets the secret shares.

With one-time secrets, the secret message m is encrypted under a symmetric key k, which is secret-shared using PVSS among the secret-management trustees such that t shares are required to reconstruct it. The access-control trustees verify and log on the blockchain the encrypted secret shares which, due to the properties

of PVSS, do not leak any information about k. After the secret-management trustees receive a valid request $\text{req}_{\text{share}}$, they respond with their secret shares encrypted under the public key listed in the policy from the respective tx_{w}. A dishonest reader cannot obtain access to someone else's secret through a ciphertext-replay attack, because each transaction is bound to a specific policy used to derive the base point for the PVSS NIZK consistency proofs. Without knowing the decrypted secret shares and the key k, the malicious reader cannot generate correct proofs, and all transactions without valid proofs are rejected. Only the intended reader thus obtains a threshold of secret shares necessary to recover k and access m.

Fair access: *Clients are guaranteed authorized access to a secret after posting a valid access request on-chain. If a barrier point exists, authorized clients atomically get simultaneous access after the barrier, protecting against front-running.*

Before a read transaction tx_{r} is logged by the access-control blockchain, PVSS and encryption protects the secret. Once a read transaction tx_{r} is logged and the barrier point has passed, any reader can run the share-retrieval protocol, obtaining t shares of the encryption key k from honest trustees to reconstruct k and access the message m. Since this access is possible only after the barrier, any transactions that any node subsequently proposes using this information can affect blockchain state only after the barrier.

Receiver Anonymity: *An on-chain proof-of-access log does not identify the user unless an audit is requested.*

The on-chain blinded-key exchange protocol exposes a composite public key as Ron's identity, which is secure under the DDH assumption. The on-chain encryption provides Ron the blinding factor, making him the only one who can reconstruct the composite private key. If Wanda reveals this blinding factor, it is easy to verify the DDH triplet exposed, and hence deanonymize Ron.

Dynamic self-sovereign identities: *Users and organizations fully control the public keys representing their identities, and can update them verifiably and atomically.*

Ron is always in control of his identity skipchain and can evolve it as he sees fit. Due to the authenticated forward and backward links Ron is able to prove paths from the genesis block used as policy to his current keys, and hence convince the secret-management to decrypt. Due to the on-chain time-stamping, even if some of Ron's stale keys are compromised, the adversary cannot use them to forge an alternate access path. The time-stamping smart contract will detect that the adversary's proposed access does not build on the latest version of Ron's identity and reject it.

7 Case Studies Using CALYPSO

We describe two real-world deployments of Calypso that resulted from collaborations with companies that needed a flexible, secure, and decentralized solution to manage data. We also describe an experimental zero-collateral, constant-round decentralized lottery.

7.1 Clearance-enforcing Document Sharing

To show the power of Calypso for auditable data sharing, we deployed a decentralized, clearance-enforcing document-sharing system enabling two organizations, A and B, to share a document D, such that a policy of confidentiality can be enforced on D. We realized this system with a contractor of the Ministry of Defense of a European country using a permissioned BFT blockchain and long-term secrets. This application is evaluated in Section 8.2.

Problem Definition. Organization A wants to share with organization B a document D whose entirety or certain parts are classified and should be accessible only by people with proper clearance. Clearance is granted to (or revoked from) employees individually as needed or automatically when they join (or leave) a 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 blockchain and a secret-management committee including trustees controlled by both organizations. To prevent any organization from having a majority of trustees, the service provider manages 1/3 of the trustees. Each organization then establishes federated identity skipchains with all the identities that have clearance, id_A and id_B, respectively, including 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.

Organization A creates a document D and labels each paragraph as confidential or unclassified. A then chooses a confidential symmetric key, and derives from it an unclassified subkey using a key derivation function. A encrypts the document, shares the ciphertext with B and shares the symmetric keys using Calypso with policy = id_B. Any employee of B whose public key is included in the set of classified employees as defined in the latest skipblock of id_B can retrieve the classified symmetric key via a read transaction and decrypt the full document. Unclassified readers can retrieve only the lower-clearance key. Calypso logs the tx_{r}, creates a proof of access and delivers the key. Both organizations can update their identity skipchains to ensure that at any given moment only authorized employees have access. As a result, both organizations can share information and maintain a secure audit log without having to trust each other or the service provider fully.

7.2 Patient-centric Medical Data Sharing

Calypso lends itself well to applications that require secure data sharing for research purposes. We are currently 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 evaluate this application as its performance properties are similar to the one above. To guarantee the confidentiality and decentralization of the system there should be at least three independent institutions each contributing a proportional number of trustees to the secret-management committee.

Problem Definition. Researchers face difficulties in gathering medical data, as patients increasingly refuse to approve access to their data for research amidst rapidly-growing privacy concerns [33]. Patients dislike consenting once and completely losing control over their data, and are more likely to consent to sharing their data with specific institutions [37]. 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 case

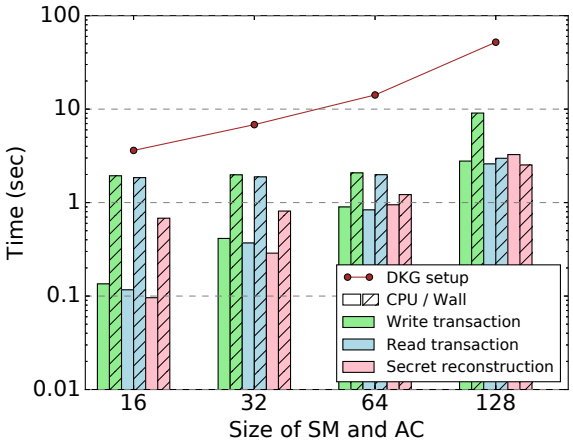

Figure 4: Latency of long-term secrets protocol for varying sizes of secret-management committee and access-control blockchain.

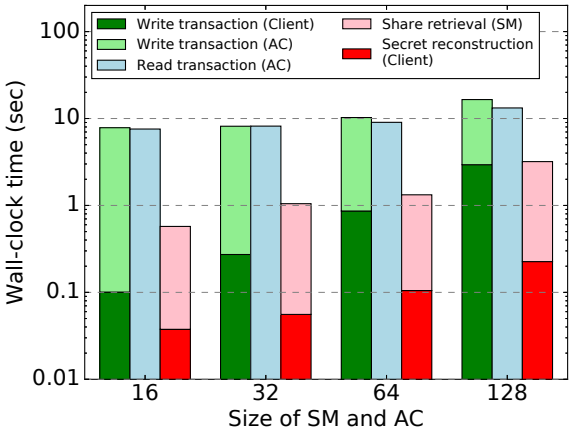

Figure 5: Latency of one-time secrets protocol for varying sizes of secret-management committee and access-control blockchain.

a patient is unable to grant access (*e.g.*, unconscious), the medical doctor can request an emergency exception allowed in the policy, and access the data while leaving an auditable proof of this action.

Solution with Calypso. We designed a preliminary architecture for a data-sharing application enabling a patient P to share her data with multiple potential readers. This deployment is different from the one above in that the data generator (hospital) and the data owner (P) are different. For this reason, we use a resource policy skipchain id_P representing P's data usage preferences. Policy skipchains can dynamically evolve by adding and removing authorized readers, and can include rich access-control rules.

Calypso enables P to initialize id_P when she first registers with the medical system. Initially, id_P is empty, indicating that P's data cannot be shared. If a new research organization or another hospital requests to access 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 consisting 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, with Calypso, P remains in control of her data and can unilaterally update or revoke access, solving the data availability versus consent-management challenge.

7.3 Decentralized Lottery

Prior proposals for decentralized lotteries either need collateral as in Ethereum's Randao [57], or run in a non-constant number of rounds [48]. Calypso enables a simpler design, as the lottery executes in one round and needs no collateral, because the participants cannot predict the randomness or abort.

Problem Definition. We assume there are n participants who want to run a decentralized zero-collateral lottery. A smart contract manages the lottery by collecting bids and deciding on the winner via public randomness. We evaluate this application in Section 8.3. We assume that a threshold of the secret-management committee is honest; further incentive analysis and slashing is necessary if the secret-management committee is rational.

Solution with Calypso. Each participant creates a tx_w with their secret contribution to the randomness and shares it using long-term secrets. After a predefined number of blocks (the barrier point), the input phase of closes. Any user can then generate a tx_r upon which the smart contract retrieves all committed inputs and posts the reconstructed values and their proofs. Finally, the smart contract computes the XOR of all (random) inputs and uses it to select the winner. Using the same idea we can see the power of Calypso on simplifying collaborative decentralized games (*e.g.*, poker).

8 Evaluation

We implemented all components of Calypso, namely long-term secrets, one-time secrets and SIAM, in Go [31]. For cryptographic operations we used Kyber [44], an advanced cryptographic library for Go. In particular, we used its implementation of the Edwards25519 elliptic curve providing 128-bit security. For the consensus mechanism required for the access-control blockchain, we used an implementation of ByzCoin [39], a scalable Byzantine consensus protocol. All our implementations are available as open source on GitHub and have gone through an independent security audit.

First, we evaluate and compare the performance of two on-chain secrets protocols using micro-benchmarks. Next, we evaluate the performance of Calypso using two real-world applications: clearance-enforcing document sharing (Section 7.1) and a decentralized lottery (Section 7.3), using both synthetic and real-world data traces. For the document-sharing application, we compare Calypso with both a fully-centralized and a semi-centralized solution. As for the decentralized lottery, we compare a Calypso-based lottery to a state-of-the-art zero-collateral lottery. The synthetic workloads are significantly heavier than those from the real data traces. For the experimental evaluation of SIAM, see the Appendix D.4. We ran all our experiments on four Mininet [49] servers, each equipped with 256 GB of memory and 24 cores running at 2.5 GHz. To simulate a realistic network, we configured Mininet

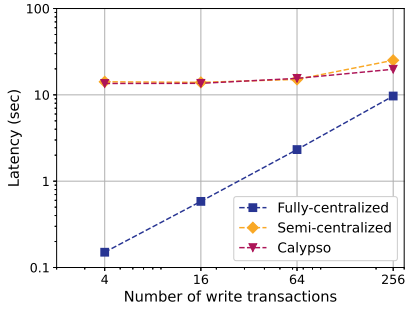

Figure 6: Write transaction latency for different loads in clearance-enforcing document sharing.

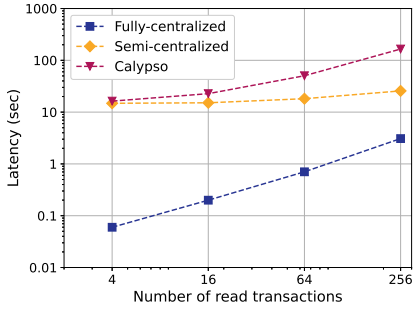

Figure 7: Read transaction latency for different loads in clearance-enforcing document sharing.

with a 100 ms latency between the nodes and a per-node bandwidth of 100 Mbps.

8.1 Mirco-benchmarks of On-chain secrets

The two main questions we wish to answer about on-chain secrets are whether read and write transaction latencies are acceptable in a realistic deployment, and whether the system can scale to hundreds of trustees to achieve strong decentralization. We compare Calypso against a centralized (single server) and a semi-centralized setup where secrets are stored off-chain and access policies are enforced by the access-control blockchain. We measure the total latency of both on-chain secrets protocols, separately analyzing the cost of the write, read, share-retrieval and share-reconstruction sub-protocols. We vary the number of trustees in the secret-management committee and access-control blockchain, where all trustees belong to both. A comparison of the transaction size for one-time secrets and long-term secrets is in the Appendix E.

8.1.1 Long-term Secrets Results for the permissioned setting appear in Figure 4. It shows the overall latency of the key setup (DKG), write, read, share retrieval and share reconstruction sub-protocols. Except for DKG setup, which is a one-time cost, all steps scale linearly in the size of the committee. Even for a committee of 128 servers, it takes less than 8 seconds to process a transaction. Furthermore, CPU time is significantly lower than wall-clock time due to the network overhead included in the wall-clock measurements. This experiment makes clear that the overhead of long-term secrets scales well with the added level of decentralization and can support workloads running on permissioned blockchains that tend to have similar latencies [2, 39] for the same level of decentralization.

8.1.2 One-time Secrets To answer our questions for the permissionless setting we look at Figure 5. We observe that creating tx_w takes almost one second on the client side 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 reconstructing the shared secret are faster than creating the tx_w– by an order of magnitude for large numbers of shares – because verification and reconstruction require fewer elliptic-curve cryptography operations than setting up the PVSS shares. Finally, the overhead of recovery on the secret-management committee is an order of magnitude higher than on the client side since the client sends a request to each trustee. Although these overheads look substantial, our microbenchmark demonstrates the feasibility of deploying one-time secrets in a permissionless setting with minimal overhead compared to the confirmation latency of minutes that existing open blockchains have [51, 78].

8.2 Clearance-Enforcing Document Sharing

We compare our clearance-enforcing document-sharing system with a fully-centralized access-control system, and with our implementation of a state-of-the-art semi-centralized approach [6, 21, 34, 67] that logs accesses and policies on-chain but entrusts the data to the cloud. We vary the simulated workload per block from 4 to 256 read and write transactions and report total time to execute all transactions. These experiments use a blocktime of 10 seconds.

Figure 6 shows that Calypso not only provides better security, but also has less latency overhead than the semi-centralized solution when executing write transactions. The difference becomes greater as the number of transactions increase: the semi-centralized solution is 20% slower than Calypso for 256 write transactions. The additional overhead of the semi-centralized solution is because in addition to logging the access-control policies on the blockchain, writers also have to separately store the secret in the cloud. On the other hand, if the users are comfortable outsourcing their data then Calypso is not suitable as it takes 2× to 100× more time to execute the write transactions compared to the fully-centralized solution.

Figure 7 shows the results of the same experiment for read transactions. The latency values have two components: storing the read transactions on the blockchain and decrypting the corresponding secrets. The semi-centralized solution takes 10× to 421× and Calypso takes 55× to 457× more time than the fully-centralized solution when executing the read transactions. These results show that Calypso incurs between 0.9× and 4.5× more latency overhead than the semi-centralized solution depending on the level of decentralization. The reason for Calypso's higher overhead is the secret reconstruction step that is executed by the secret-management committee. For smaller number of transactions Calypso and the semi-centralized solution have comparable latency values because they are dominated by the blocktime, which is almost the same for both systems. However, as the number of transactions increase, the secret reconstruction step starts dominating the total latency in Calypso and causes the larger overhead. More specifically, the

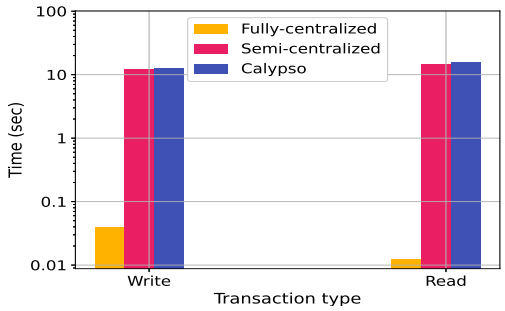

Figure 8: Average write and read transaction latencies replaying real-world data traces from clearance-enforcing document sharing.

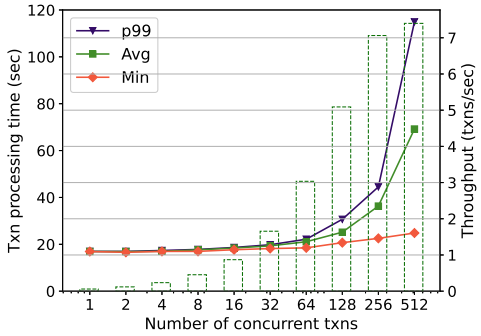

Figure 9: Performance for varying block size (level of concurrency)

secret reconstruction step of Calypso amounts to 11% (2 s) and 85% (125 s) of the total latency for 4 and 256 read transactions, respectively. For the semi-centralized solution the corresponding step of decrypting the secrets amounts to 0.4% (40 ms) and 19% (2.6 s) of the total latency for the same number of transactions. Allthough Calypso has moderate overhead compared to the semi-centralized solution, we believe the added security benefit is more important.

Next, we show the actual performance of the clearance-enforcing document sharing deployment of Calypso using real-world data traces from our governmental contractor partner mentioned in Section 7.1. 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 the fully-centralized 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 the latency is dominated by the blocktime due to the small number of transactions per block, meaning that for existing deployments Calypso's additional security comes at almost no cost.

Figure 9 finally offers insight into selecting an appropriate block size for Calypso. Transactions inside a block are executed concurrently, which, as we see in Figure 9, allows for higher throughput (bar graphs): an average throughput of 7.4 txns/sec with 512 concurrent transactions. This shows that Calypso can easily keep up with Bitcoin if deployed as an external secret-management service. However, the increased throughput comes at the cost of a higher overall latency for clients with a larger variance.

8.3 Decentralized Lottery

Finally, to show that Calypso can provide algorithmic speedup to certain applications, we compare our Calypso-based zero-collateral lottery with a corresponding tournament lottery by Miller et al. [48] on simulated and real workloads. Figure 10 shows that the Calypso-based lottery performs better both in terms of overall execution time and bandwidth usage. Specifically, our lottery runs in one round while the tournament runs in a logarithmic number of rounds due to its design consisting of multiple two-party lotteries.

Next, we evaluate both lotteries using transactions from an Ethereum-based lottery called Fire Lotto [75]. We consider transactions sent to the Fire Lotto smart contract over a period of 30 days, where each day is a different run of the lottery. We evaluate a naive implementation of the lottery where each decryption is run separately, and a batched version that implements a SELECT query to batch-decrypt all the tickets at once. Figure 11 shows the total time it takes to run the lotteries. Each data point in the graph corresponds to a single lottery run. As before, the 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 lottery. Furthermore, the optimized SELECT query takes advantage of the fact that the systems is doing the same operations for the same set of private keys and further reduces the latency around 50% compared to the naive implementation. Our results include only the latency of the reveal phase since the commit phase happens asynchronously over a full day.

9 Related and Future Work

In our deployments we demonstrated the power of Calypso, which enables mutually distrustful parties who want to collaborate within a blockchain ecosystem to auditably exchange data and payments, and be protected from front-running attacks. In one sentence, Calypso is the first truly decentralized system that provides the full CIA triad that modern businesses want from their data-management systems. As a result, new applications, such as accountable data sharing [26], time-locked vaults [59], and multi-party games [42] can now be deployed within blockchain ecosystems without the need to place full trust in a centralized manager.

Private data storage has been widely studied in databases [55], but adding decentralization is challenging. Vanish [28] guarantees that data self-destructs once it is no longer needed, to protect against accidental leakage. Calypso can implement similar functionality by adding timeouts to on-chain secrets, after which the symmetric encryption keys (or secret shares) are destroyed. The Calypso approach is more robust as it uses a blockchain ensuring Byzantine fault tolerance, instead of a DHT, which generally does not.

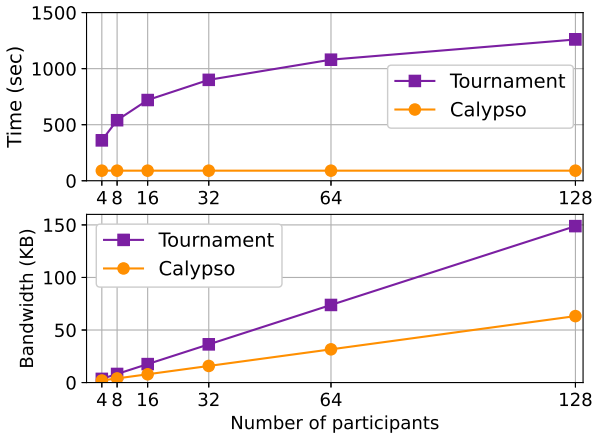

Figure 10: Lottery evaluation using simulated workloads.

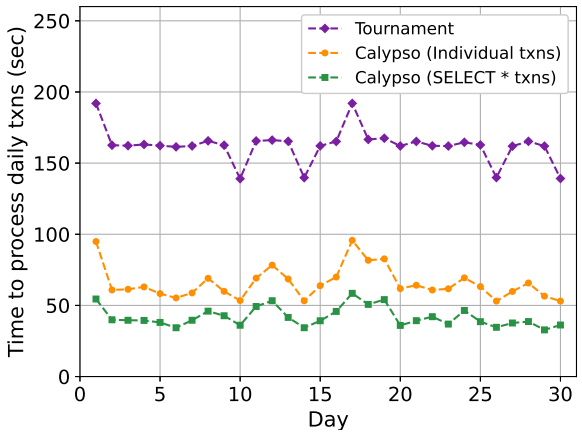

Figure 11: Lottery evaluation using Fire Lotto.

Nevertheless, Calypso is still limited to guaranteeing data confidentiality up to the point where an authorized reader gains access. To maintain confidentiality after this point, writers might rely on additional privacy-preserving technologies, such as differential privacy [20] or homomorphic encryption [25]. Differential privacy can also be used to help identify leaks in the case of multiple readers. Wanda can create multiple write transactions if she wants to pinpoint leaks and apply different noise or watermarks to each.

The closest work to ours is the decentralized data management platform Enigma [80], where users 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 centralized storage provider that can read and decrypt the data, or refuse to serve the data even if there is a valid on-chain proof. The Enigma storage provider is thus a single point of failure or compromise. Other projects [6, 21, 34, 67] that rely on centralized key-management and/or storage systems suffer from similar issues with atomicity and robustness to compromised service providers.

Instead of a cloud storage provider, Ekiden [15] trusts a secure enclave (*e.g.*, Intel SGX), which again becomes a single point of compromise. Calypso is the first system that truly supports the full CIA triad without having any single point of failure or compromise. This comes at moderate overhead, of course. Calypso has constant-sized write transactions only in the permissioned model. If a client in the permissionless model wants this feature, he needs to trust a predefined set of service providers and cannot choose any group arbitrarily. One possible extension is to combine multiple predefined sets of long-term secrets servers in a one-time secrets instance and generate one PVSS share per group. This would make the transaction linear to the number of groups instead of the number of trustees, hence reducing the total transaction size.

Finally, other privacy-focused blockchain systems [47, 64] do not sufficiently address the problem of sharing data. Although they allow committing confidential data on-chain, they rely on the initial data provider to reveal the data, which means that they do not have high availability. This is not an issue for these systems, as they focus on hiding the identity and amounts of monetary transactions, but the actual data might be inaccessible forever. The only thing available is zero-knowledge proofs that the system is consistent, hence they cannot be used to achieve our goals. Calypso's on-chain blinded key-exchange protocol enables Wanda to protect the identity of the intended reader of her secrets without forfeiting any of on-chain secrets' guarantees, but it requires knowing the reader's public key. If Ron wants to have both a dynamic identity and a hidden identity, he would still need to perform the exchange before the rotation and maintain the hidden key securely.

Despite its limitations, Calypso shows how to preserve the confidentiality of information and guarantee the fairness of disclosure, opening up new possibilities for investigation. For example, we are now closer to building decentralized marketplaces [71], or even using the already decentralized confidential data to build prediction models without seeing the data, but only the final result.

10 Conclusion

This paper has demonstrated how Calypso enables a blockchain to hold and manage secrets directly on-chain. Calypso achieves its goals through two key components. The first component, on-chain secrets, is deployed on top of a blockchain to enable transparent and efficient management of secret data via threshold cryptography. The second component, skipchain-based identity and access management, allows for dynamic identities and roles and user-managed access policies. We have implemented Calypso and shown that it can be efficiently deployed with blockchain systems to enhance their functionality. Lastly, we describe three deployments of Calypso to illustrate its applicability to real-world use cases.

Acknowledgments

We thank Nicolas Gailly, Vincent Graf, Jean-Pierre Hubaux, Wouter Lueks, Massimo Marelli, Carmela Troncoso, Juan-Ramón Troncoso-Pastoriza, Frédéric Pont, and Sandra Siby for their valuable feedback. This project was supported in part by the ETH domain under PHRT grant #2017−201, and by the AXA Research Fund, Byzgen, DFINITY, and the Swiss Data Science Center (SDSC).

References

[1] A. N. Amroudi, A. Zaghain, and M. Sajadieh. A Verifiable (k, n, m)-Threshold Multi-secret Sharing Scheme Based on NTRU Cryptosystem. *Wireless Personal*

Communications, 96(1):1393–1405, 2017.
[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 30:1–30:15, 2018.
[3] E. Androulaki, C. Cachin, A. De Caro, and E. Kokoris-Kogias. Channels: Horizontal Scaling and Confidentiality on Permissioned Blockchains. In *European Symposium on Research in Computer Security*, pages 111–131. Springer, 2018.
[4] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure Multiparty Computations on Bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458. IEEE, 2014.
[5] M. Archetti and I. Scheuring. Game theory of public goods in one-shot social dilemmas without assortment. *Journal of theoretical biology*, 299:9–20, 2012.
[6] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.
[7] J. Benet. IPFS – Content Addressed, Versioned, P2P File System. arXiv preprint arXiv:1407.3561, 2014.
[8] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *ACM CCS*, Nov. 2013.
[9] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, volume 48, pages 313–317, 1979.
[10] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *6th International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, Jan. 2003.
[11] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 553–570. IEEE, 2015.
[12] I. Cascudo and B. David. SCRAPE: Scalable Randomness Attested by Public Entities. In *15th International Conference on Applied Cryptography and Network Security (ACNS)*, July 2017.
[13] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Feb. 1999.
[14] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta. Demonstrating possession of a discrete logarithm without revealing it. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 200–212. Springer, 1986.
[15] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *arXiv preprint arXiv:1804.05141*, 2018.
[16] CoinDesk. Decentralized Exchanges Aren't Living Up to Their Name – And Data Proves It, July 2018.
[17] M. Czernik. On Blockchain Frontrunning, Feb. 2018.
[18] S. Czsun. Escaping the Dark Forest, Sept. 2020.
[19] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology (CRYPTO)*, Aug. 1989.
[20] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.
[21] A. Dubovitskaya, Z. Xu, S. Ryu, M. Schumacher, and F. Wang. Secure and Trustable Electronic Medical Records Sharing using Blockchain. *arXiv preprint arXiv:1709.06528*, 2017.
[22] V. Durham. Namecoin, 2011.
[23] J. Ellis. The Guardian introduces SecureDrop for document leaks. *Nieman Journalism Lab*, 2014.
[24] S. Eskandari, S. Moosavi, and J. Clark. Transparent Dishonesty: front-running attacks on Blockchain. In *3rd Workshop on Trusted Smart Contracts (WTSC)*, Feb. 2019.
[25] J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
[26] J. Feigenbaum. Multiple Objectives of Lawful-Surveillance Protocols (Transcript of Discussion). In *Cambridge International Workshop on Security Protocols*, pages 9–17. Springer, 2017.
[27] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 427–438. IEEE, 1987.
[28] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *USENIX Security Symposium*, pages 299–316, 2009.
[29] Genecoin. Make a Backup of Yourself Using Bitcoin, May 2018.
[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, volume 99, pages 295–310. Springer, 1999.
[31] The Go Programming Language, Feb. 2018.
[32] E. Hardt. The OAuth 2.0 Authorization Framework, Oct. 2012. RFC 6749.
[33] K. F. Hollis. To Share or Not to Share: Ethical Acquisition and Use of Medical Data. *AMIA Summits on Translational Science Proceedings*, 2016:420, 2016.
[34] L. Huang, G. Zhang, S. Yu, A. Fu, and J. Yearwood. SeShare: Secure cloud data sharing based on blockchain and public auditing. *Concurrency and Computation: Practice and Experience*, 2017.
[35] M. Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *Public key cryptography*, pages 632–632. Springer, 1999.
[36] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In *29th International Conference on Distributed Computing Systems (ICDCS)*, pages 119–128. IEEE, June 2009.
[37] K. K. Kim, P. Sankar, M. D. Wilson, and S. C. Haynes. Factors affecting willingness to share electronic health data among California consumers. *BMC medical ethics*, 18(1):25, 2017.
[38] E. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford. Managing Identities Using Blockchains and CoSi. Technical report, 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016), 2016.
[39] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium*, 2016.
[40] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *39th IEEE Symposium on Security and Privacy (SP)*, pages 19–34. IEEE, 2018.
[41] E. Kokoris-Kogias, A. Spiegelman, D. Malkhi, and I. Abraham. Bootstrapping Consensus Without Trusted Setup: Fully Asynchronous Distributed Key Generation. Cryptology ePrint Archive Report 2019/1015, Sept. 2019.
[42] R. Kumaresan, T. Moran, and I. Bentov. How to Use Bitcoin to Play Decentralized Poker. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 195–206. ACM, 2015.
[43] J. Kwon. TenderMint: Consensus without Mining, 2014.
[44] The Kyber Cryptography Library, 2010 – 2018.
[45] L. A. Linn and M. B. Koo. Blockchain for health data and its potential use in health it and health care related research. In *ONC/NIST Use of Blockchain for Healthcare and Research Workshop. Gaithersburg, Maryland, United States: ONC/NIST*, 2016.
[46] W. Lueks. *Security and Privacy via Cryptography: Having your cake and eating it too*. PhD thesis, [Sl: sn], 2017.
[47] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *34th IEEE Symposium on Security and Privacy (S&P)*, May 2013.
[48] A. Miller and I. Bentov. Zero-collateral lotteries in Bitcoin and Ethereum. In *Security and Privacy Workshops (EuroS&PW), 2017 IEEE European Symposium on*, pages 4–13. IEEE, 2017.
[49] Mininet – An Instant Virtual Network on your Laptop (or other PC), Feb. 2018.
[50] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel. A survey on essential components of a self-sovereign identity. *Computer Science Review*, 30:80–86, Nov. 2018.
[51] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
[52] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium*, pages 1271–1287, 2017.
[53] Ocean Protocol Foundation. Ocean Protocol: A Decentralized Substrate for AI Data & Services, May 2018.
[54] M. Pilkington. Blockchain Technology: Principles and Applications. *Research Handbook on Digital Transformation*, 2015.
[55] R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591, 2016.
[56] B. Rajabi and Z. Eslami. A Verifiable Threshold Secret Sharing Scheme Based On Lattices. *Information Sciences*, 2018.
[57] randao.org. Randao: Blockchain Based Verifiable Random Number Generator, 2018.
[58] A. Rapoport, A. M. Chammah, and C. J. Orwant. *Prisoner's dilemma: A study in conflict and cooperation*, volume 165. University of Michigan press, 1965.
[59] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, Mar. 1996.
[60] D. Robinson and G. Konstantopoulos. Ethereum is a Dark Forest. *Medium*, Aug. 2020.
[61] P. Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107, 2002.
[62] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine. Survivable Key Compromise in Software Update Systems. In *17th ACM Conference on Computer and Communications security (CCS)*, Oct. 2010.
[63] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
[64] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[65] B. Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *IACR International Cryptology Conference (CRYPTO)*, pages 784–784, Aug. 1999.
[66] SECBIT. How the winner got Fomo3D prize – A Detailed Explanation, Aug. 2018.
[67] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, pages 45–50. ACM, 2017.
[68] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.
[69] V. Shoup. Practical Threshold Signatures. In *Eurocrypt*, May 2000.
[70] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Advances in Cryptology — EUROCRYPT'98*, pages 1–16, 1998.
[71] H. Subramanian. Decentralized blockchain-based electronic marketplaces. *Communications of the ACM*, 61(1):78–84, 2017.
[72] M. H. Swende. Blockchain Frontrunning, Oct. 2017.
[73] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *37th IEEE Symposium on Security and Privacy*, May 2016.
[74] N. Szabo. Smart contracts. *Unpublished manuscript*, 1994.
[75] F. Team. Fire Lotto blockchain lottery, 2018.
[76] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. G. Gueta, and S. Devadas. Towards Scalable Threshold Cryptosystems. In *41st IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE Computer Society, may 2020.
[77] T. M. Wong, C. Wang, and J. M. Wing. Verifiable secret redistribution for archive systems. In *Security in Storage Workshop, 2002. Proceedings. First International IEEE*, pages 94–105. IEEE, 2002.
[78] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 2014.
[79] G. Zyskind, O. Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.
[80] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

A Full Protocol for Long-term secrets

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 $\mathrm{H}_2 : \mathbb{G}^3 \to \mathbb{Z}_q$.

Setup Protocol. Initially, the secret-management committee needs to run a DKG protocol to generate a shared private-public key pair such that the private key is not known to any single party, but can be reconstructed by combining a threshold of key shares. There exist a number of DKG protocols that are synchronous [30] or asynchronous [41]. Given the rarity of the setup phase we run the DKG by Gennaro et al. [30] using the blockchain to emulate synchronous communication.

The output of the setup phase is a shared public key $\mathrm{pk}_{\mathrm{smc}} = g^{\mathrm{sk}_{\mathrm{smc}}}$, where $\mathrm{sk}_{\mathrm{smc}}$ 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 $\mathrm{pk}_i = g^{\mathrm{sk}_i}$. The secret key can be reconstructed by combining a threshold $t = f + 1$ of individual shares. We assume that $\mathrm{pk}_{\mathrm{smc}}$ is registered on-chain of the access-control blockchain, *e.g.*, in the genesis block.

Write Transaction Protocol

Wanda and the access-control blockchain perform the following protocol to log the tx_{w} on the blockchain. Wanda 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[4] $L \in \{0,1\}^l$. Wanda performs the following steps.

(1) Retrieve the threshold public key $\mathrm{pk}_{\mathrm{smc}}$ of the secret-management committee.
(2) Choose a symmetric key k and encrypt the secret message m using authenticated encryption [61] to be shared as $c_m = \mathrm{enc}_k(m)$ and compute $H_{c_m} = \mathrm{H}(c_m)$. Set policy = pk_R to designate Ron as the intended reader of the secret message m.
(3) Encrypt k towards $\mathrm{pk}_{\mathrm{smc}}$ using a threshold variant of the ElGamal encryption scheme. To do so, embed k as a point $k' \in \mathbb{G}$, pick a value r uniformly at random, compute $c_k = (\mathrm{pk}_{\mathrm{smc}}^r k', g^r)$ and create the NIZK proof π_{c_k} to guarantee that the ciphertext is correctly formed and resistant to replay attacks as follows.
(4) Choose at random $r, s \in \mathbb{Z}_q$. Compute:

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

(5) Finally, prepare and sign the write transaction: $\mathrm{tx}_{\mathrm{w}} = [c_k, \pi_{c_k}, H_{c_m}, \mathrm{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$, and send it to the access-control blockchain.

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

The access-control blockchain 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 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 blockchain.
(2) Check that $\mathrm{H}(c_m)$ 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 $H_{\mathrm{w}} = \mathrm{H}(\mathrm{tx}_{\mathrm{w}})$ as the unique identifier for the secret that Ron requests access to and determine the proof $\pi_{\mathrm{tx}_{\mathrm{w}}}$ showing that tx_{w} has been logged on-chain.
(4) Prepare and sign the tx_{r}: $\mathrm{tx}_{\mathrm{r}} = [H_{\mathrm{w}}, \pi_{\mathrm{tx}_{\mathrm{w}}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$, and send it to the access-control blockchain.

The access-control blockchain 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 committee. To do so Ron does as follows.

(1) Create and sign a secret-sharing request: $\mathrm{req}_{\mathrm{share}} = [\mathrm{tx}_{\mathrm{w}}, \mathrm{tx}_{\mathrm{r}}, \pi_{\mathrm{tx}_{\mathrm{r}}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$, where $\pi_{\mathrm{tx}_{\mathrm{r}}}$ proves that tx_{r} has been logged on-chain.
(2) Send $\mathrm{req}_{\mathrm{share}}$ to each secret-management trustee to request the blinded shares.

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

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

(1) Check if $e = \mathrm{H}_1(c, u, \bar{u}, w, \bar{w}, L)$ 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) Create and sign the secret-sharing reply: $\mathrm{rep}_{\mathrm{share}} = [u_i, e_i, f_i]_{\mathrm{sig}_{\mathrm{sk}_i}}$, and send it back to Ron.

Secret reconstruction

Ron 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 Ron

(1) Each secret-management server i prepares a blinded share $u_i = (g^r)^{\mathrm{sk}_i}$ along with its NIZK proof of correctness, computes $c_i = \mathrm{enc}_{\mathrm{pk}_R}(u_i)$, and sends (c_i, e_i, f_i) back to Ron.
(2) Run the decryption share check to make sure that the trustees are not misbehaving.
(3) 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 $\hat{h_i} = \frac{g^{f_i}}{h_i^{e_i}}$.
(4) 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.
(5) Ron recovers the encoded encryption key: $k' = (c_k)(\mathrm{pk}_{\mathrm{smc}}^r)^{-1} = (\mathrm{pk}_{\mathrm{smc}}^r k')(\mathrm{pk}_{\mathrm{smc}}^r)^{-1}$, retrieves the symmetric encryption key k from k', and finally decrypts the secret message $m = \mathrm{dec}_k(c_m)$.

Secret reconstruction at the trusted server

Ron authenticates himself using his public key g^{x_c}. One of the trustees is assigned to do the reconstruction for the client.

(1) Each secret-management server i ElGamal encrypts its secret key share $u_i = (g^r)^{\mathrm{sk}_i}$ using Ron's public key $\mathrm{pk}_R = g^{\mathrm{sk}_R}$ and its secret key sk_i instead of the usual random exponent. The encrypted share is $u_i' = g^{r\mathrm{sk}_i} g^{\mathrm{sk}_R \mathrm{sk}_i} = g^{(r+\mathrm{sk}_R)\mathrm{sk}_i} = g^{r'\mathrm{sk}_i}$ Then the trustee computes $\hat{h_i}$, as before and $\hat{u_i'} = u'^{\mathrm{sk}_i}$. 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) The trustee collects t valid shares, then uses Lagrange interpolation to reconstruct $g^{r'\mathrm{sk}_{\mathrm{smc}}} = g^{(r+\mathrm{sk}_R)\mathrm{sk}_{\mathrm{smc}}}$ which he sends to Ron. Note that the server never sees $g^{r\mathrm{sk}_{\mathrm{smc}}}$ and consequently cannot decrypt the secret message intended for Ron.
(3) Ron knows $\mathrm{pk}_{\mathrm{smc}} = g^{\mathrm{sk}_{\mathrm{smc}}}$ and sk_R, and can calculate $(g^{r\,\mathrm{sk}_R})^{-1}$. Then, he can recover $\mathrm{pk}_{\mathrm{smc}}^r$ as $(g^{r'\mathrm{sk}_{\mathrm{smc}}})(g^{r\,\mathrm{sk}_R})^{-1} = (g^{(r+\mathrm{sk}_R)\mathrm{sk}_{\mathrm{smc}}})(g^{r\,\mathrm{sk}_R})^{-1} = (g^{r\,\mathrm{sk}_{\mathrm{smc}}})(g^{r\,\mathrm{sk}_R})(g^{r\,\mathrm{sk}_R})^{-1} = g^{r\,\mathrm{sk}_{\mathrm{smc}}} = \mathrm{pk}_{\mathrm{smc}}^r$.

Finally, Ron recovers the symmetric key k and carries out the decryption as explained in the step above. If the authenticated decryption fails then Ron cannot distinguish between a bad server and Wanda's misbehavior. As a result he can either optimistically ask another server to do the interpolation or pessimistically do it himself and blame Wanda if decryption fails again. Another path would be for the server to contact the secret-management committee in order to generate a ZK-proof of correct re-encryption but we opted for the optimistic approach that has less overhead.

B One-Time Secrets Protocols

We follow the protocol in [65] 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:

Write Transaction Protocol.

Wanda, the writer and each trustee of the access-control blockchain perform the following protocol to log the write transaction tx_{w} on the blockchain. Wanda initiates the protocol as follows.

(1) Compute $h = \mathrm{H}(\mathrm{policy})$ to map [8] 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. 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 = pk_R to designate Ron as the intended reader of the secret message m.
(5) Finally, prepare and sign the write transaction: $\mathrm{tx}_{\mathrm{w}} = [\langle \widehat{s_i} \rangle, \langle b_j \rangle, \langle \pi_{\widehat{s_i}} \rangle, H_c, \langle \mathrm{pk}_i \rangle, \mathrm{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$, and send it to the access-control blockchain.

$\pi_{\widehat{s_i}}$ 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 $\log_h A_i = \log_{\mathrm{pk}_i} \widehat{s_i}$.

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

(1) Derive the PVSS base point $h = \mathrm{H}(\text{policy})$.
(2) Compute $A_i = \prod_{j=0}^{t-1} {c_j}^{i^j}$ using the polynomial commitments c_j, $0 \le j < t$.
(3) Compute $a'_{1i} = h^{r_i} A_i^{C_i}$ and $a'_{2i} = \mathrm{pk}_i^{r_i} \widehat{s_i}^{C_i}$
(4) Check that $\mathrm{H}(A_i, \widehat{s_i}, a'_{1i}, a'_{2i})$ matches the challenge C_i.
(5) If all shares are valid, log $\mathrm{tx_w}$ in block b_w.

Read Transaction Protocol.

After the write transaction has been recorded, Ron needs to log the read transaction $\mathrm{tx_r}$ through the access-control blockchain 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 $\mathrm{tx_w}$, from the access-control blockchain.
(2) Check that $\mathrm{H}(c)$ is equal to H_c in $\mathrm{tx_w}$ to ensure that the ciphertext c of Wanda's secret has not been altered.
(3) Compute $H_\mathrm{w} = \mathrm{H}(\mathrm{tx_w})$ as the unique identifier for the secret that Ron requests access to and determine the proof $\pi_{\mathrm{tx_w}}$ showing that $\mathrm{tx_w}$ has been logged on-chain.
(4) Prepare and sign the transaction: $\mathrm{tx_r} = [H_\mathrm{w}, \pi_{\mathrm{tx_w}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$, and send it to the access-control blockchain. The transaction can optionally bear a payment value v that the trustees receive upon replying.

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

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

Share Retrieval Protocol.

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

(1) Create and sign a secret-sharing request: $\mathrm{req_{share}} = [\mathrm{tx_w}, \mathrm{tx_r}, \pi_{\mathrm{tx_r}}]_{\mathrm{sig}_{\mathrm{sk}_R}}$, where $\pi_{\mathrm{tx_r}}$ proves that $\mathrm{tx_r}$ has been logged on-chain.
(2) Send $\mathrm{req_{share}}$ to each secret-management trustee to obtain the decrypted shares.

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

(1) Use pk_R in $\mathrm{tx_w}$ to verify the signature of $\mathrm{req_{share}}$ and $\pi_{\mathrm{tx_r}}$ to check that $\mathrm{tx_r}$ has been logged on-chain.
(2) Compute the decrypted share $s_i = (\widehat{s_i})^{\mathrm{sk}_i^{-1}}$, create a NIZK proof π_{s_i} that the share was decrypted correctly. The proof shows the knowledge of the unique value that satisfies $\log_g \mathrm{pk}_i = \log_{s_i} \widehat{s_i}$.
(3) Derive $c_i = \mathrm{enc}_{\mathrm{pk}_R}(s_i)$ to ensure that only Ron can access it.
(4) Create and sign the secret-sharing reply: $\mathrm{rep_{share}} = [c_i, \pi_{s_i}]_{\mathrm{sig}_{\mathrm{sk}_i}}$, and send it back to Ron or publish on-chain claiming payment.

Secret Reconstruction Protocol.

To recover the secret key k and decrypt the secret m, Ron performs the following steps.

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

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 [68]. We need the following assumptions to provide confidentiality. First, we assume that Wanda has post-quantum confidential and authenticated point-to-point communication channels [11] 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 [1].

Write Transaction Protocol Wanda prepares her write transaction $\mathrm{tx_w}$ with the help of the secret-management committee and access-control blockchain, 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 $\mathrm{tx_w}$.

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

(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 committee 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 blockchain 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 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 Security Considerations and Incentive Structure

Our contributions are mainly pragmatic rather than theoretical as we employ mostly existing, well-studied cryptographic algorithms in a black box, modular fashion. For the replay attack adversary that was not considered in prior work we provide a sketch of the security proofs. Then, we show the incentive compatibility of our permissionless protocol and analyze the number of trustees Wanda should for sufficient security.

D.1 Replay attack

In both long-term secrets and one-time secrets Wanda posts on-chain the ciphertexts which the adversary (Eve) can easily access. The replay attack consists of Eve copying the ciphertext and creating a new transaction that includes the ciphertext (or a homomorphic modification of it), but changes the policy from Ron to Eve. As a result, Eve can now authorize decryption of the new transaction and get the decrypted shares from the secret-management committee.

D.1.1 Long-term secrets Security Argument In order to show that long-term secrets is secure under this attack we need to show that Eve is unable to generate a valid transaction after seeing the ciphertext.

Recall the ciphertext includes form $(\mathrm{pk}_{\mathrm{smc}}^r k', g^r, g^s, \bar{u} = \bar{g}^r, \bar{w} = \bar{g}^s L, e = \mathrm{H}_1(c, u, \bar{u}, w, \bar{w}, L), f = s{+}re)$. Eve wants to take $\mathrm{pk}_{\mathrm{smc}}^r k', g^r$ and generate a new valid transaction so that she can convince the secret-management committee to reveal $\mathrm{pk}_{\mathrm{smc}}^r$ to her, since she does not know r. This would not be a problem if Wanda was using simple threshold encryption. However in Calypso, Eve needs to generate $e' = \mathrm{H}_1(c, u, \bar{u}, w, \bar{w}, L')]$ (where L' is her public key instead of Ron's) and $f = s + re$ however she know neither s nor r. From the two she can trivially choose a new s' since it is not crucial for decryption, however she still does not know r and she cannot recover it form $u = g^r$ (DLOG is hard in $\mathbb{G}$).

Let's assume that Eve can somehow generate a valid f such that $g^{f'} = g^{s'} + g^{re'}$ and convince access-control blockchain to log the transaction as valid. Then we can use Eve's algorithm to solve the DDH problem as the triple $(g^r, g^{e'}, g^f - g^{s'})$ is a DDH triple and Eve generated it without knowing r which should be hard.

This means that under the ROM model the only valid e comes from including the original L in H_1, which ties the transaction to the policy.

D.1.2 One-time secrets Security Argument In order to bind one-time secrets with the policy L we use it to derive a base point from $H(L)$. Eve wants to change the policy to L', hence she would need to do the proofs using $H(L')$, but she does not know the secrets. Since we know that our zk-proof of knowing the secret shares are secure and Eve does not know the secret shares we get the security directly from PVSS.

However we changed one thing in PVSS that can break security if not handled properly. Instead of having a random base point we derive it from $H(L)$. As a result if Eve could compute an exponent a such that $H(L)^a = H(L')$ she could homomorphically apply the exponent to all proofs and make them work for her policy. We need to make sure when deriving the point from $H(L)$ that it is indistinguishable from random to prevent this attack. If we simply cast $H(L)$ into a scalar a and derive $H = \mathbb{G}^a$ then Eve could also find a as she knows L. Then she would derive a' from $H(L')$ and raise all the proofs to a'/a making them work for $H(L')$.

The security of one-time secrets comes from using Elligator maps [8] when deriving the base point from $H(L)$ which makes sure that the point is random. As a result for Eve to break one-time secrets she can to nothing better than guessing, which has negligible probability of succeeding.

D.2 Incentive Analysis

In this section we analyze the incentives that rational participants have when running Calypso in a permissionless mode. The trustees have three possible deviations: (a) do not release their share (b) claim they released their share but encrypt garbage and (c) release their share even if there is no valid read transaction. Ron has one possible deviation which is to bribe the trustees in order to release their shares without paying Wanda. Wanda has one possible deviation which is to bribe the trustees to not reply or to release garbage. Next we analyze these scenarios.

The easiest to analyze is Wanda's deviation. Let's say that she sends a bribe ϵ to the trustees. Given that she cannot stop them from sending a message the trustees best course of action is to accept the bribe and still make money from Ron, hence Wanda just loses money. Notice that this is not a fair-exchange problem since in fair exchange both parties have some secret. In this example the trustees have no secret to trade. They can send the message whenever they want and Wanda cannot stop them.

Next we analyze Ron's deviation. In order for a trustee to take a bribe Ron needs to send him av/t for the payment the trustee would normally make and v/f for the collateral that the trustee risks (again notice that the trustee has no way to stop Ron from slashing). As a result Ron needs to pay in total $tv/f + av$. From this he can claim back a percentage a of the collateral through slashing the t trustees for a total of atv/f. As a result his expected cost is $tv/f + av - atv/f = (f + 1)v/f + fav/f - a(f + 1)v/f = (fv + v + fav - afv - av)/f = (f + 1 - a)v/f$. But $a < 1$ hence $(f + 1 - a)v/f > (f + 1 - 1)v/f = v$. Hence a rational Ron will not bribe the trustees. This analysis trivially extends to Ron having shares from Byzantine parties that did not ask for a bribe. This hold

because slashing a Byzantine party makes av/f and legally paying an rational a party costs $av/f + 1$. Hence Ron will prefer to slash and pay than to bribe.

Finally, we analyze the deviations of the trustees. The third deviation is the trustees action when bribed. We already showed that Ron will never bribe the trustees $v/f + av/t$ or more because he loses money. On the other hand the trustee will never accept a bribe of less since av/t is his expected revenue for following the protocol and v/f his expected loss for deviating. Similarly the trustee has no incentive to put garbage in a transaction since he can only lose his collateral.

The last deviation we look into is for the trustee to act as a benign fault. Clearly here the collateral is not at risk, and if t rational parties agree to not reply they can hold Ron hostage. If we look the game from the perspective of a single rational trustee it is a prisoner's dilemma game [58]. If he follows the hostage protocol and the other f trustees do as well then he can hold Ron hostage and gain more than av/f, however, if a single party from the f releases his share (mounting a front-running attack to everyone in the hostage cluster) then the expected payoff for the rest of the hostage cluster drops to 0. As a result, given that no $f + 1$ trustees are managed by a single adversary the rational behavior is to follow the protocol.

D.3 Selecting one-time secrets Group Size

From the rationality analysis it is clear that the minimum one-time secrets size is 3 in order to prevent hostage situations. In this section, we analyze the recommended size for Wanda based on her perception of dishonest nodes in the group. The goal is to have at least $f + 1$ rational parties in her selection with high probability (failure probability 10^{-6}). We model this problem as a random sampling protocol. In order to compute the appropriate group size for different expected percentage of dishonest parties we use the binomial distribution:

$$P[X \leq c] = \sum_{k=0}^{c} \binom{w}{k} p^k (1-p)^{w-k} \quad (1)$$

Table 1 displays the results for the evaluation for various percentages of adversarial power p.

p (%)	1	5	10	20	30	45
Group Size	7	13	24	41	127	501

Table 1: Recommended one-time secrets group size

D.4 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 12 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.

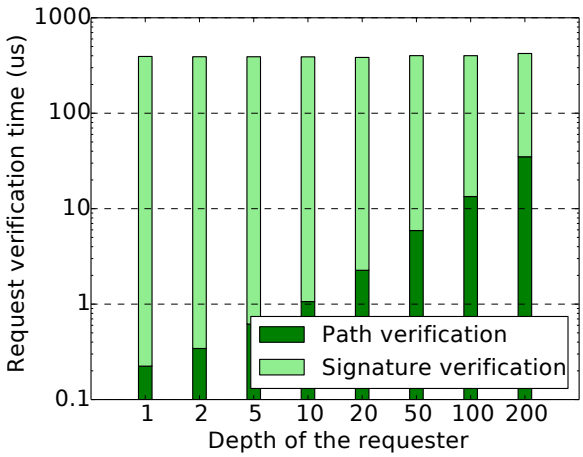

Figure 12: Single-signature request verification.

Table 2: tx_w size for varying secret-management committee sizes

	tx_w size (bytes)	
Number of trustees	One-time secrets	Long-term secrets
16	4'086	160
32	8'054	160
64	15'990	160
128	31'926	160
160	39'894	160
192	47'862	160
224	55'830	160
256	63'798	160

E Evaluation of Transaction Size in On-chain secrets

The size of transactions is smaller in long-term secrets than in one-time secrets because the data is encrypted under the secret-management's threshold public key which results in a constant overhead regardless of the committee's size. We can see that the costs of tx_r and tx_w are almost equal as they are dominated by adding a block to the access-control blockchain blockchain. Table 2 shows tx_w sizes in one-time secrets and long-term secrets for different secret-management committee configurations. In one-time secrets, a tx_w stores three pieces of PVSS-related information: encrypted

shares, polynomial commitments and NIZK encryption consistency proofs. As the size of this information is determined by the number of PVSS trustees, the size of the tx_w increases linearly with the size of the secret-management committee. In long-term secrets tx_w uses the shared key of the secret-management committee and does not need to include the encrypted shares. As a result, long-term secrets has constant write transaction size.