CALYPSO: Private Data Management for Decentralized Ledgers (Extended Version)

Eleftherios Kokoris Kogias
EPFL
lefteris2k@gmail.com

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 ledger technologies provide high *availability* and *integrity*, making them a key enabler for practical and secure computation of distributed workloads among mutually distrustful parties. However, many practical applications also require *confidentiality*, the third pillar of the CIA triad. In this work, we enhance permissioned and permissionless blockchains with the ability to manage confidential data without forfeiting availability or decentralization. More specifically, CALYPSO sets out to achieve two orthogonal goals that challenge modern distributed ledgers: (a) enable blockchains to auditably manage secrets and (b) protect distributed computations against arbitrage attacks when their results depend on the ordering and secrecy of inputs.

To this end, CALYPSO proposes on-chain secrets, a novel abstraction that enforces atomic deposition of an auditable trace whenever users access confidential data. Furthermore, CALYPSO provides user-controlled consent management that ensures revocation atomicity and accountable anonymity. Finally, to enable the permissionless deployment of CALYPSO, 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 the latency of processing transactions increases linearly to the added security (in number of trustees) and is in the range of 0.2 to 8 seconds for 16 to 128 trustees.

PVLDB Reference Format:
Ben Trovato, G. K. M. Tobin, Lars Thørväld, Lawrence P. Leipuner, Sean Fogarty, Charles Palmer, John Smith, Julius P. Kumquat, and Ahmet Sacan. CALYPSO: Private Data Management for Decentralized Ledgers. *PVLDB*, 12(xxx): xxxx-yyyy, 2021.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

xf

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Blockchain technology enables secure and public exchange of value online and has many interesting applications. Decentralized data-sharing can give rise to data markets controlled by users [62] and not solely by tech giants, such as Google or Facebook; it can enable sharing of (confidential) data between mutually distrustful parties, such as state institutions or even different countries; or it can bring the much needed transparency to lawful access requests [21]. Decentralized data life-cycle management can enable the implementation of secure key-recovery mechanisms for users or an information-publication version of a dead man's switch [19] that enables journalists to create contingency plans. Finally, if correctly implemented, decentralized data life-cycle management can guarantee fairness for lotteries [4], games (*e.g.*, poker [37]), and trading (*e.g.*, exchanges [14]).

In many of the above use cases it is crucial to exchange private data in a secure manner, which current decentralized data-sharing applications [40, 47] do not provide. They only provide partial solutions that either forfeit availability guarantees for the private data [24] or fall back on semi-centralized solutions for key management [6, 68]. Furthermore, decentralized applications that rely on the timing of data disclosure to enforce fairness are susceptible to front-running attacks where adversaries get early access to information enabling them to unfairly adapt their strategies. For example, the winner of the Fomo3D [58] lottery event enforced an early termination of the entire process by submitting a sequence of high-fee transactions, which significantly increased his winning probability. Thanks to that exploit, this participant obtained an overall prize of 10.5k Ether corresponding to USD $2.2 M at the time.

In this paper 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 CALYPSO needs to address three key challenges. First, CALYPSO has to 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 this accountability should not reveal the relationship between the users and instead provide anonymity to the data consumers. Second, CALYPSO has to enable data owners to maintain control over the data they share, and data consumers to be able to access even when their identities

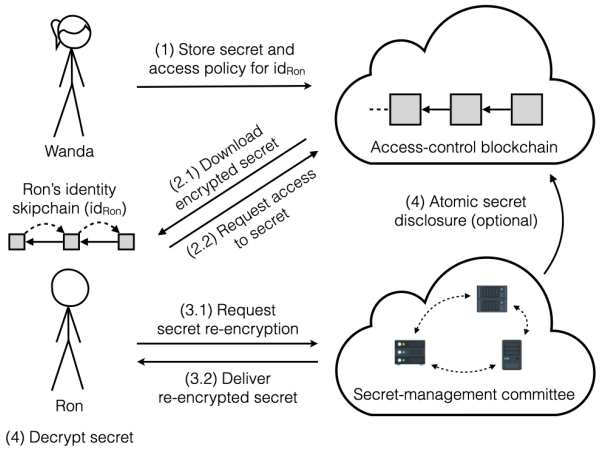

Figure 1: Auditable data sharing in CALYPSO: (1) Wanda encrypts data under the secret-management committee's public key, specifying the intended reader (*e.g.*, Ron) and the access policy, and then sends it to the access-control blockchain which verifies and logs it. (2) Ron downloads the encrypted secret from the blockchain and then requests access to it by contacting the access-control blockchain which logs the query if valid, effectively authorizing Ron's access to the secret. (3) Ron asks the secret-management committee for a re-encryption of the downloaded secret by proving that the previous authorization by the access-control blockchain was successful. (4) Ron decrypts the re-encrypted secret. If a specific application requires fairness, the data can be atomically disclosed on-chain.

(public keys) are updated. In particular, CALYPSO should allow for flexible updates to access-control rules and user identities, *e.g.*, to add or revoke access rights or keys. Third, CALYPSO needs to enable permissionless functionality in order to be deployable along open ecosystems such as existing blockchains. Figure 1 provides an overview of a typical data-sharing application using CALYPSO that builds on top of a novel abstraction called *on-chain secrets* (OCS) and provides dynamic access-control and identity management.

On-chain secrets addresses the first challenge by balancing the availability and confidentiality guarantees any data management system should provide with the decentralization requirements of blockchain technology [34, 67]. On-chain secrets combines threshold cryptography [57, 60, 61] with a blockchain that enforces access control and atomically discloses data to authorized parties. To further enable dynamic access-control and identity management CALYPSO combines on-chain secrets with skipchains [33, 46]. This results in the first decentralized role-based access-control [55] system that enables user-controlled consent management. Finally, CALYPSO addresses the third challenge by building incentives around the authorized decryption of data and enabling users to securely select the number and the identities they want to trust based on personal preference.

We implemented a prototype of CALYPSO in Go and evaluated it on commodity servers. Our experiments show that both versions of on-chain secrets that we propose scale 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 CALYPSO in the context of secure document-sharing and a zero-collateral decentralized lottery. In our experiments, we used both synthetic and real-world workloads and we compared 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 only a negligible amount of overhead to the centralized solution in comparison to a semi-centralized approach. For the zero-collateral lottery, we show that our CALYPSO-based solution requires only 1 round to finish outperforming existing solutions requiring $\log n$ rounds (n denotes the number of participants).

In summary, this paper makes the following contributions.

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

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

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 and 9). The code and a demo can be found at https://github.com/calypso-demo/.

2. MOTIVATION AND BACKGROUND

In this section, we first motivate CALYPSO by describing how it can enable security and fairness in data management and atomic data publication applications. We then summarize the main building blocks that 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 documents (*e.g.*, legal wills or estate plans) when certain conditions are met. This functionality also enables digital life insurances for whistleblowers where files are published automatically unless the custodian receives a digitally signed "heartbeat" message from the insured person on a regular basis [19, 52]. Moving to fully decentralized custodians bears new challenges in terms of how to specify and implement data publication and secure consent-management in this deployment model and how to integrate these components with each other.

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. Furthermore, CALYPSO employs an expressive policy-mechanism that enables atomic modification of access

rights so that data owners can revoke the access rights of those who have not yet exercised their publicly verifiable rights to access the data. As a result, designers of decentralized applications can use CALYPSO to achieve additional functionalities, such as monetizing data accesses 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 adversary can easily gain unfair advantage over honest participants through front-running [14, 58, 63] (early, unfair access to information) if decentralized applications, such as lotteries [4], poker games [37], or exchanges [14], are not designed with such attacks in mind. In CALYPSO, inputs provided by the participants (*e.g.*, lottery randomness, trading bids, game moves) remain confidential up to a barrier point that is expressed by defining specific rules in a policy. After the barrier point, all the information is published and the computed result is atomically disclosed to every interested party (*e.g.*, which trades were successful, winner of the lottery). Consequently, CALYPSO resolves the tension between decentralization, fairness, and availability and provides a secure foundation for decentralized applications. A representative application of this class is the zero-collateral lottery that we present in Section 7.3.

2.2 Blockchains and Skipchains

Blockchain is a distributed, append-only and tamper-evident log that is composed of blocks that are connected to each other via cryptographic hashes and is used in many decentralized applications [18, 45, 3]. CALYPSO does not innovate on this front and can be deployed along any blockchain (or state-machine replication system [11])[1] that supports programmability (*i.e.*, smart contracts [2, 64, 67]), thereby enabling custom validation.

Skipchains [46] track configuration changes of a decentralized authority (cothority) by using each block as a representation of all public keys of the cothority that are necessary to authenticate the next block. When a cothority wants to alter its configuration, it creates a new block that includes the new set of public keys and signs it with the old set of public keys delegating trust to the new set. This signature is a *forward link* [33] that clients follow to get up-to-date with the current authoritative group. In Section 4.3, we define identity and policy skipchains. Our construction is a simple extension of the skipchains in order to support federated groups and enable expressive consent-management (Appendix E).

2.3 Threshold Cryptosystems

A (t, n)-secret sharing scheme [9, 60] enables a dealer to share a secret s among n trustees such that any subset of t trustees can reconstruct s, whereas smaller subsets cannot. Hence, the sharing scheme can withstand up to $t-1$ malicious participants. The downside of simple secret-sharing schemes is that they assume an honest dealer, an issue that verifiable secret sharing (VSS) [22] solves by enabling the trustees to verify that the distributed shares are consistent. VSS is used for threshold signing and threshold encryption. Publicly verifiable secret sharing (PVSS) [57] is a variation of VSS that enables external third-parties to verify the shares.

Once we are able to securely share and hold a collective secret, we can construct more complex systems out of it. A distributed key generation (DKG) [25, 31, 36] protocol allows to create a collective key pair without a trusted dealer. The DKG produces a private-public key pair $(\text{sk}, g^{\text{sk}})$ such that the public key $\text{pk} = g^{\text{sk}}$ is known to everyone whereas the private key sk is not known to any single trustee and can only be used when a threshold of trustees collaborates. Afterwards, anyone can encrypt data under this public key.

3. CALYPSO OVERVIEW

This section provides an overview of CALYPSO. We start with two strawman solutions to illustrate the challenges that a secure decentralized data-management system should address and show how interconnected and fragile the properties of such a system are, especially in a Byzantine environment. Based on our observations, we then derive the system goals and present the system design (as shown in Figure 1).

3.1 Strawman Protocols

As a motivation, consider an application on top of our system where Wanda is the operator of a paid service that provides asynchronous access to information about stock orders, and Ron is a customer. This applications requires *auditability*, which is a 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: Trusted Custodian

The first strawman is a simple trusted custodian. Wanda sends her information to the custodian specifying that Ron can read if he pays the fee. Then, Wanda 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 and then shows the transaction to the custodian who in turn releases the information encrypted under Ron's public key.

Strawman I provides *auditability*, however it puts too much trust on the custodian. First, if the custodian crashes, Ron has no guarantee of getting the information. To avoid this, the second property that our system has to have is *decentralization*. Second, if the custodian is malicious, he can, undetectably by Wanda, give the information to Ron without any proof of payment. To protect from this attack, we require the third property of our system to be *confidentiality*. Finally, even if the custodian does not compromise the 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 arbitrage attacks, on the stock market in this scenario. To protect against unfairness, the fourth property we require is *fair access to data*.

3.1.2 Strawman II: The Secret Sharer

[1] Directly inheriting their Byzantine Fault Tolerant properties.

The straightforward solution to achieve decentralization of data is replication: for instance, we can have n custodians. However, this worsens the confidentiality of the system since we now have n custodians who can potentially leak the information. Instead, Strawman II symmetrically encrypts the information and publishes the encryption on-chain. Note that the encryption does not have to be stored on-chain, however doing so guarantees high availability of data. To provide confidentiality, Wanda uses the (t, n)-threshold Shamir secret sharing scheme [60] to split the encryption key between the n custodians. As a result if at least t custodians are honest, both decentralization and confidentiality are preserved.

Although Strawman II seems to solve two of our issues, it is actually worse than Strawman I. The challenge is that Wanda can now fake her secret-sharing step, claiming that she did a (t, n) secret sharing while she actually sends garbage to the custodians, thereby making it impossible to ever recover the secret. To defend against this, we force Wanda to post consistent shares on-chain using PVSS [57]. However, now an unauthorized adversary (*e.g.*, Eve) can mount a replay attack. Eve can copy the consistent shares available on-chain in a seemingly independent new transaction (with Eve as the authorized reader and the payment recipient) and trick the custodians to decrypt without authorization. This shows how fragile *auditability* is. Strawman II shows that simple solutions can easily fail if they are not carefully designed.

3.1.3 Additional Properties

Before we introduce our goals, we mention two additional properties that are desirable from Ron's point of view. First, Ron might not want to make publicly known that he is accessing the stock market. In such as case, we can use a privacy preserving blockchain such as Zcash [56] for payment, but we also need *receiver anonymity*. Second, if Ron ever needs to change his public key, he would lose access to all data that are bound to authorization from the old key. We ideally want Ron to have a *dynamic sovereign identity*, which he can evolve independently and retain access to previous shared data. To provide all properties, we introduce three components and transform the Strawman II into CALYPSO.

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

3.2 System Goals

CALYPSO has the following primary goals.

- **Auditability:** All access transactions are third-party verifiable and recorded in a tamper-resistant log.
- **Decentralization:** There are no single points of compromise or failure.
- **Confidentiality:** Secrets stored on-chain can only be decrypted by authorized clients.
- **Fair access:** Clients are guaranteed to get access to a secret they are authorized for only if they posted a valid access request on-chain. If a barrier point exists, authorized clients get concurrent access after it (protecting against front-running attacks).
- **Receiver anonymity:** The auditable proof-of-access logged on-chain does not identify the user unless an audit is requested.
- **Dynamic sovereign identities:** Users (or organizations) fully control their identities (public keys) and can update them in a third-party verifiable way.

3.3 System Model

There are four main entities in CALYPSO's architecture: *writers* who put 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 (such as Ethereum [67]) not managed by the system administrators. In the rest of the paper, we keep them separate for architectural clarity. We also use Wanda and Ron to refer to a (generic) writer and reader, respectively.

The access-control blockchain requires a Byzantine fault-tolerant consensus [34, 35, 38, 45]. There are various ways to implement an access-control blockchain, *e.g.*, as a set of permissioned servers that maintains a blockchain using BFT consensus or as an access-control enforcing smart contract on top of a permissionless cryptocurrency. The secret-management committee membership is fixed; it may be set up on a per-secret basis or in a more persistent setting, the differences of which are discussed in Section 5.1.1. 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 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 $= \mathrm{pk}_R$ is the simplest case with Ron being the only reader.

3.4 Threat Model

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

We denote the number of trustees by n and the malicious by f. Depending on the consensus mechanism of the access-control blockchain, we either require an honest majority $n = 2f + 1$ for Nakamoto-style consensus [45] or $n = 3f + 1$ for classic BFT consensus [34][2]. In the secret-management

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

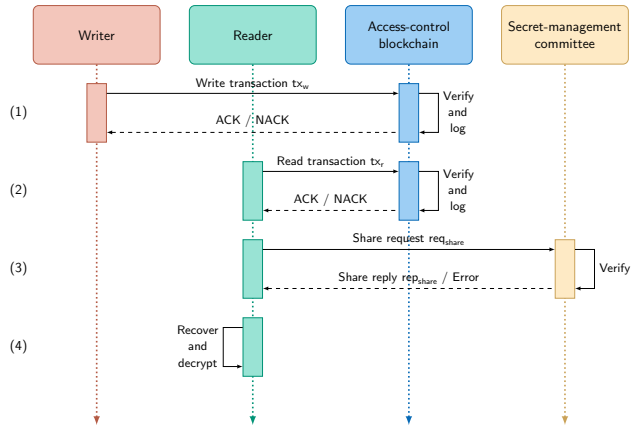

Figure 2: On-chain secrets protocol steps: (1) Write transaction, (2) Read transaction, (3) Share retrieval, (4) Secret reconstruction.

committee, we require $n = 2f + 1$ and set the threshold to recover a secret to $t = f + 1$.

We assume that readers and writers do not trust each other. We further assume that writers encrypt the correct data and share the correct symmetric key with the secret-management 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, thereby guaranteeing fairness.

3.5 Architecture Overview

On a high level CALYPSO enables Wanda, the writer, to share a secret with Ron, the reader, under a specific access-control policy. When Wanda wants to put a secret on-chain (see Figure 1), she encrypts the secret and sends a write transaction tx_w to the access-control blockchain. The access-control blockchain verifies and logs tx_w, making the secret available for Ron, the authorized reader. To access a secret, Ron downloads the secret from the blockchain and sends to the access-control blockchain a read transaction tx_r, which carries a valid authorization from Ron's identity skipchain with respect to the current policy.

If Ron is authorized to access the secret, the access-control blockchain logs 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 tx_w. In Section 4, we show the deployed system (Section 7) that adopts a permissioned model where trustees are externally accountable. 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 that they achieve all of our goals. Due to lack of space, we refer to Appendix A for further discussion of the security of CALYPSO. First, we introduce *long-terms secrets*, which provides auditable access-control and fair data-access in a permissioned setting with 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 the *skipchain-based identity and access management* that adds dynamic access-control for Wanda and self-sovereign identity management to the non-obfuscated part of Ron's identity. The writer can either encrypt the data directly or use a symmetric key and offload the storage of the symmetrically-encrypted data C_m to IPFS [7] or some other decentralized storage service. If C_m is not on chain, the reader should make sure he has the encrypted data[3] before he asks for access in order to preserve the decentralization.

4.1 Long-Term Secrets

In order to understand the challenges of long-term secrets we need to look into Strawman II. We can see that our first challenge is preventing Wanda from posting bad shares. To solve this, CALYPSO uses VSS [22], which forces Wanda to publicly announce a commitment to her shares against which all trustees can verify consistency. Using VSS however has two challenges. First, Wanda needs to interact with the trustees for every new transaction in order to get their agreement on the correctness of the transaction. Second, the size of the transaction grows linear to the number of trustees. In order to resolve these challenges we leverage the fact that the group of trustees is predefined and reduce the size of the write transactions from linear to the number of trustees (one share per trustee) to constant. In long-term secrets, trustees generate a shared public key during setup. Wanda can then use threshold ElGamal encryption [15] to hide the symmetric key. This transaction can be made non-interactive with a zero-knowledge proof of correct encryption [12]. Additionally, the transaction size only depends on the security parameter and not the number of trustees.

A third challenge is that Wanda needs to bind the secret shares with Ron's identity in order to prevent unauthorized reads by Eve. For this reason we use the zero-knowledge proofs of Lueks et al. [41, 61]. They are originally proposed to protect against chosen-ciphertext attacks, but the core construction of binding the policy with the ciphertext matches exactly our needs as well. One final performance challenge is that Ron needs to verify and reconstruct $O(n)$ shares. To alleviate this burden we let Ron delegate the costly operations of verifying and combining shares to a trustee who is assumed to be honest-but-curious and would not DoS Ron. Then we make sure that the trustee does not obtain direct access to the secret and Ron can always detect any misbehavior and ask another trustee or carry out the process himself. The full protocol can be found in the Appendix A.

4.1.1 Evolution of Secret-Management Committee

The secret-management committee is expected to persist over a long period of time while remaining secure and available. However, a number of issues can arise during its lifetime. First, trustees can join and leave, thereby causing churn. Second, even if the secret-management committee memberships remain static, the private shares of the servers should be

[3]He can ask the writer to send it to him directly if IPFS is unresponsive

refreshed regularly (*e.g.*, every month) to provide backward secrecy. Lastly, the shared private key of the secret-management committee should be rotated periodically (*e.g.*, once every year).

We address the first two problems by periodically re-sharing [66] the existing threshold public key when a server joins or leaves the secret-management committee, or when servers want to refresh their private key shares. Lastly, when the secret-management committee wants to rotate the threshold public/private key pair (pk_{smc}, sk_{smc}), CALYPSO needs to collectively re-encrypt each individual secret under the new shared public key. To achieve this, we generate and use translation certificates [30] such that the secrets can be re-encrypted without the involvement of their writers and without revealing the underlying secrets.

4.2 On-chain Blinded Key Exchange

In our protocols so far, Wanda includes the public key of Ron in a secret's policy to mark him as the authorized reader. Once Wanda's write transaction is logged, everyone knows that she has shared a secret with Ron and correspondingly, once his read transaction is logged, everyone knows that he has obtained the secret. While this property is desirable for some deployment scenarios we envision, certain application 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 and the signature under this private key is sufficient for the writer to prove that the intended reader created the read transaction. The protocol works as follows and achieves our goal of receiver anonymity.

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

4.3 Identity and Access Management

The CALYPSO protocols described so far do not provide dynamic access control or sovereign identities. They only support static identities (public keys) and access policies as they provide no mechanisms to update these objects. However, these assumptions are rather unrealistic, as participants might need to change or add new public keys to revoke a compromised private key or to extend access rights to a new device. Similarly, it should be possible to change access polices so that access to resources can be extended, updated or revoked; and to define access-control rules for individual identities and groups of users for greater flexibility. Finally, any access-control system that supports the above properties should prevent freeze attacks [54] and race conditions.

To achieve dynamic sovereign-identities, we look into role-based access control (RBAC) [55] policies. These policies can evolve dynamically depending on the role of users, however, they rely on a central manager to assign users to roles. In order to address this challenge, we use skipchains that enable sovereignty over identities and roles. Specifically, we introduce the *skipchain-based identity and access management* (SIAM) subsystem for CALYPSO that provides the following properties: (1) Supports identities for both individual users and groups. (2) Enables users to specify and announce updates to resource access keys and policies. (3) Enforces atomic data accesses and updates to access rights to prevent race conditions.

We achieve the first two goals of SIAM by using skipchains to encode the identities of individuals and the memberships of roles. As a result, our system is the first decentralized instantiation of an RBAC policy mechanism. More specifically, we deploy three types of skipchains in CALYPSO (as shown in Figure 3). *Personal identity skipchains* store the public keys that individual users control [33]. 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 in order to provide scaling and ease of use. *Resource policy skipchains* track access rights of identities, personal or federated, to certain resources and enable dynamic access-control 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.

When SIAM is used, Ron is able to evolve the id_R skipchain arbitrarily, *e.g.*, rotate existing access keys or add new devices, and still retain access to the encrypted resource. Similarly, Wanda can set up a resource policy skipchain id_P she is in charge of and include id_R as non-administrative members. Then, Wanda would use policy = id_P in tx_w seamlessly authorizing Ron to access the respective resource. Later Wanda can decide to revoke that resource for anyone, who has not yet accessed it, by setting policy = $\varnothing$.

Ensuring Atomicity. 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 later granting it more access rights. For example, Wanda, administrator of the sales group, decides that Ron should be fired because he is performing industrial espionage, hence she removes the identity skipchain of Ron from the federated skipchain of the sales group. Later, Wanda grants the rest of her employees access to the new corporate strategy plan. In a naive asynchronous access-control system where policy changes can take varying amounts of time to propagate and

take effect (*e.g.*, OAuth2 [27]), there is a significant time window in which Ron can still convince someone that he is part of the sales group, as he can prove membership to the controller of the sensitive object (*i.e.*, to a threshold of trustees).

A key idea in CALYPSO's design is using the blockchain to timestamp the latest versions of the skipchains. All skipchain changes are serialized together with the tx_w and tx_r on-chain. Hence the exclusion of Ron happens strictly after the granting of access. This means that Ron will be unable to provide a correctly timestamped proof to the secret-management committee and as a result cannot read the sensitive document. For more details on the integration of SIAM with on-chain secrets, please refer to Appendix F.

5. PERMISSIONLESS DEPLOYMENT

In this section, we introduce a different on-chain secrets protocol for CALYPSO that is 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 servers that will hold their secret and to choose the threshold. Having this flexibility comes at a cost as transaction size is linear in the number of trustees. However, we still manage to remain non-interactive so that the trustees can remain stateless. Lastly, we show how to incentivize the trustees using the underlying blockchain as a payment processor.

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 public key of the intended reader. She then runs PVSS [57] for a random or personal choice of secret-management committee members and uses the secret that was generated during PVSS as the symmetric key. To prevent against replay attacks, as 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. In Appendix D we show that this is secure if we use Elligator maps [8] when deriving the base point. Furthermore, we provide an analysis of the recommended group size based on Wanda's perception of how many adversarial nodes are present in the environment where she deploys one-time secrets.

Finally, Wanda sends a write transaction tx_w to the access-control blockchain to log the information for the verification and retrieval of her secret. The reading part is similar to long-term secrets except for the reconstruction, which has to be done by Ron. We describe the full protocol in Appendix B.

5.1.1 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.

However, one-time secrets has a few shortcomings compared to long-term secrets. First, it incurs a relatively high PVSS setup and share reconstruction cost as Wanda needs to evaluate the secret sharing polynomial at n points, and create n encrypted shares and NIZK proofs, along with t polynomial commitments. 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, preventing the possibility of updating the secret-management committee.

5.2 Incentives

When deploying CALYPSO in a permissionless network, it is natural to ask why the trustees will participate. We envision a system where the trustees are service providers that want to build trust with the users in order to be selected. For this reason, they lock some collateral. In Appendix D, we analyze the incentives assuming the trustees have locked collateral for one transaction, however we expect that the trustees provide more liquidity and Wanda is locking collateral proportional to the value of her data every time she creates a new write transaction.

In this setting, we want to see what is the best strategy for Ron and Wanda considering that the trustees will act rationally. We assume that Wanda's data have some intrinsic value v, which Ron is willing to pay. As a result Wanda will decide on a fraction $a < 1$ and av, which will be the amount of money the trustees will receive. We have two challenges to solve: First, trustees might receive payment and do nothing. Second, trustees might accept bribe and give Ron the data without waiting for a transaction on-chain.

In order to prevent a public-goods game [5], we need to ensure two things: only the first t trustees get paid (each one gets av/t) and trustees that reply with an invalid share need to lose more than their expected payment. The solution to the invalid-share attack is also the solution to the second challenge, collateral. In order to prevent bribes the trustees need to lock collateral. The total collateral locked by a sufficient threshold of trustees should be higher than v. Hence, we assume that every trustee locks v/f collateral. This collateral can be claimed by Ron by proving misbehavior of the trustee[4] and he gets $a < 1$ of it. The rest goes to Wanda. In order for the protocol to work the trustees send the encrypted shares on-chain claiming their payment and upon verification of the signature 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.

6. ACHIEVING SYSTEM GOALS

In this section we discuss how CALYPSO achieves its goals.

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

Under the assumption that the access-control blockchain provides Byzantine consensus guarantees, all properly created 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. For one-time secrets we further showed that the secret-management committee trustees will not deviate from the protocol and accept a bribe.

Decentralization: *There are no single points of compromise or failure.*

[4]The trustee produced an invalid signed share or it produced a valid share without although no read transaction exists on-chain

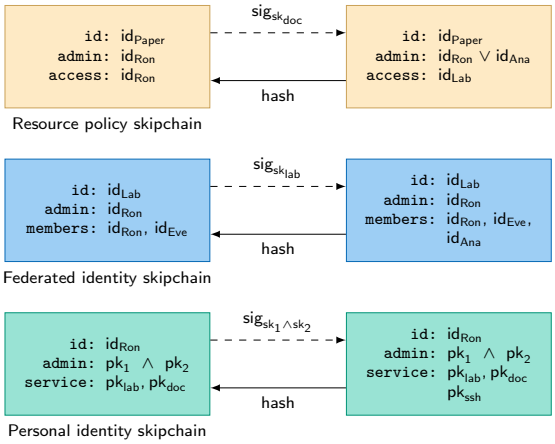

Figure 3: First, Ron updates his personal skipchain $\mathrm{id}_{\mathrm{Ron}}$ to include $\mathrm{pk}_{\mathrm{ssh}}$. He then uses $\mathrm{sk}_{\mathrm{lab}}$ to extend the federated skipchain $\mathrm{id}_{\mathrm{lab}}$ to add $\mathrm{id}_{\mathrm{Ana}}$ as a member. Finally, he adds $\mathrm{id}_{\mathrm{Ana}}$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

By design, the protocols do not assume a trusted third party and they tolerate up to $f = t - 1$ failures.

Confidentiality: *Secrets stored on-chain can only be decrypted by authorized clients.*

In long-term secrets, the secret message m is encrypted under a symmetric key k that is subsequently encrypted under a threshold public key of the secret-management committee such that at least $t = f + 1$ trustees must cooperate to decrypt it. The ciphertext is bound to a specific policy through the use of NIZK proofs [61] so it cannot be reposted in a new write transaction with a malicious reader listed in its policy. The access-control trustees log the write transaction tx_{w} that includes the encrypted key, which, based on the properties of the encryption scheme, does not leak any information about k. After the secret-management trustees receive a valid request $\mathrm{req}_{\mathrm{share}}$, they respond with the blinded shares of the shared private key encrypted under the public key in the policy of the respective tx_{w}. Based on the properties of the DKG protocol, the shared private key is never known to any single entity and can only be used if t trustees cooperate. This means, only the intended reader gets the secret shares.

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

Fair access: *Clients are guaranteed to get access on a secret they are authorized for if any only if they posted an access request on-chain. If a barrier point exists, authorized clients get concurrent access after it (protecting against front-running attacks).*

Before a read transaction tx_{r} is logged by the access-control blockchain confidentiality protects all secrets. Once a read transaction tx_{r} is logged by the access-control blockchain and the barrier point has passed, the reader can run the share retrieval protocol with the secret-management committee. Under the assumption that $n = 2f + 1$, the reader receives at least $t = f + 1$ shares of the symmetric encryption key k from the honest trustees. This guarantees that the reader has enough shares to reconstruct k and access the secret message m.

Receiver Anonymity: *The auditable proof-of-access logged on-chain 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. Furthermore, the on-chain encryption provides Ron the blinding factor, which makes him the only one that can reconstruct the composite private key. If Wanda reveals that blinding factor then it is easy to verify the DDH triplet exposed, hence deanonymize Ron.

Dynamic sovereign identities: *Users (or organizations) fully control their identities (public keys) and can update them in a third-party verifiable way.*

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

7. CASE STUDIES USING CALYPSO

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

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 than enables two organizations, A and B, to share a document D, such that a policy of confidentiality can be enforced on D. We have realized this system with a contractor of the Ministry of Defense of a European country using a permissioned BFT blockchain and long-term secrets. The evaluation of this application is discussed in Section 9.2.

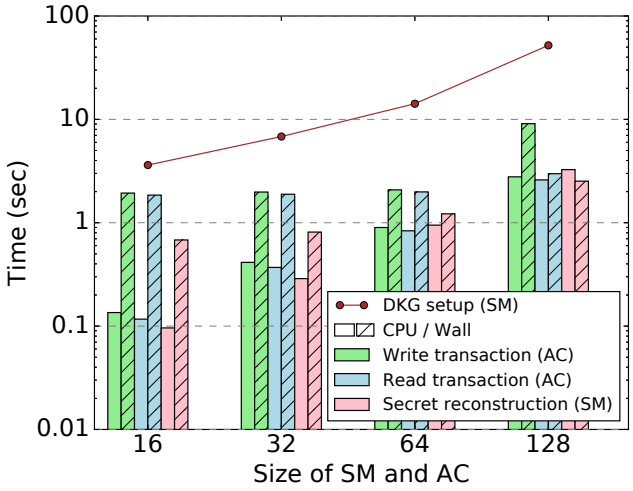

Figure 4: Latency of long-term secrets protocol for varying sizes of secret-management committee and access-control blockchain.

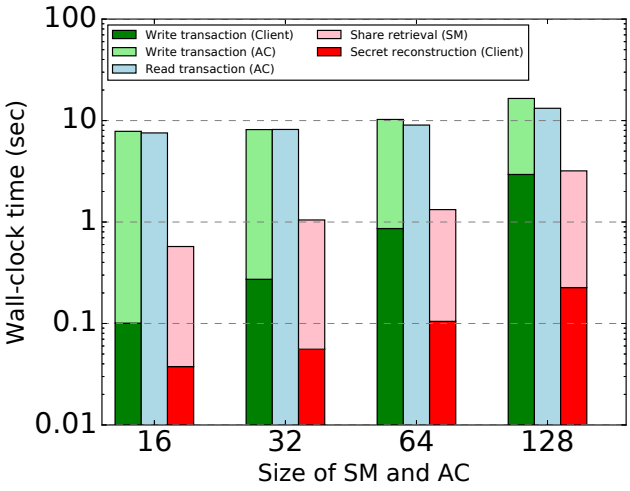

Figure 5: Latency of one-time secrets protocol for varying sizes of secret-management committee and access-control blockchain.

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

Solution with CALYPSO. First, A and B agree on a mutually-trusted blockchain system and define the secret-management committee whose trustees include servers controlled by both organizations. In order to prevent any of the organization from having a majority of trustees the service provider is also managing 1/3 of the trustees. Then, each organization establishes federated identity skipchains with all the identities that have clearance, id_A and id_B, respectively which include references to (a) federated skipchains for departments that have top-secret classification (*e.g.*, senior management), (b) federated skipchains for attributes that have top-secret classification (*e.g.*, ranked as captain) and (c) personal skipchains of employees.

Organization A creates a document D and labels each paragraph as confidential or unclassified. Then it derives a confidential symmetric key and an unclassified key, which is derived deterministically from the symmetric key using a secure hash function. Then A encrypts the document, shares the ciphertext with B and shares the symmetric keys using CALYPSO and and policy = id_B. Any employee of B whose public key is included in the set of classified employees as defined in the most current skipblock of id_B can retrieve the classified symmetric key by creating a read transaction and decrypt the full document, whereas unclassified employees can only retrieve 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 as needed to ensure that at any given moment only authorized employees have access. As a result, both organizations manage to share information and maintain the ability to access a secure audit log without having to trust each other and without having to fully rely on a service provider.

7.2 Patient-centric Medical Data Sharing

CALYPSO lends itself well for applications that require secure data-sharing for research purposes. We are in the process of working with hospitals and research institutions from a European country to build a patient-centric system to share medical data based on long-term secrets. We do not provide evaluation of this application as it is similar to the previous. In order to guarantee the confidentiality and decentralization of the system there should be at least three independent institutions maintaining a proportional number of trustees that run the secret-management committee.

Problem Definition. Researchers face difficulties in gathering medical data from hospitals as patients increasingly refuse to approve access to their data for research purposes amidst rapidly-growing privacy concerns [28]. Patients dislike consenting once and completely losing control over their data and are more likely to consent to sharing their data with specific institutions [32]. The goal of this collaboration is to enable patients to remain sovereign over their data; hospitals to verifiably obtain patients' consent for specific purposes; and researchers to obtain access to valuable patient data. In the case that a patient is unable to grant access (unconscious), the medical doctor can request an exception (specified in the policy) and access the data while leaving an auditable proof.

Solution with CALYPSO. We 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 previous one since 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 some of P's data, then P can update id_P by adding a federated identity of the research organization and specific rules. When new data is available for sharing, the hospital generates a new write transaction that consists of the encrypted and

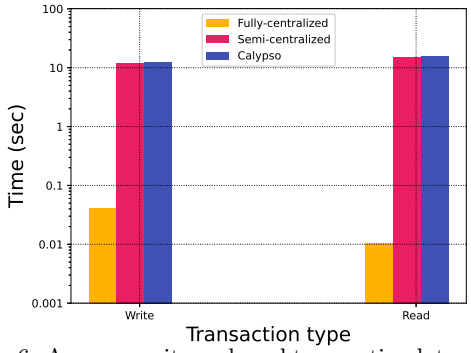

Figure 6: Average write and read transaction latencies replaying real-world data traces from clearance-enforcing document sharing.

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 (*e.g.*, Ethereum's Randao [50]) or run in a non-constant number of rounds [43]. CALYPSO enables a simpler design, as the lottery executes in one round and needs no collateral because the participants cannot predict the final randomness or abort prematurely.

Problem Definition. We assume there is a set of n participants who want to run a decentralized zero-collateral lottery selecting one winner. A smart contract manages the lottery by collecting bids and deciding on the winner via public randomness. We present the evaluation in Section 9.3. We assume that the secret-management committee acts as honest/malicious, further incentive analysis and slashing would be necessary if all the secret-management committee acted rationally.

Solution with CALYPSO. Each participant creates a tx_w with their secret contribution to the randomness calculation and shares it using long-term secrets. After a predefined number of blocks (the barrier point), the input phase of the lottery closes. 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. IMPLEMENTATION

We implemented all components of CALYPSO, namely long-term secrets, one-time secrets and SIAM, in Go [26]. For cryptographic operations we used Kyber [39], an advanced cryptographic library for Go. In particular, we used its implementation of the Edwards25519 elliptic curve that provides 128-bit security level. For the consensus mechanism required for the access-control blockchain, we used an implementation of ByzCoin [34], a scalable Byzantine consensus protocol. All our implementations are available as open source on GitHub and have gone through an independent security audit.

9. EVALUATION

First, we evaluate and compare the performance of two on-chain secrets protcols 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 Appendix F.1. We ran all our experiments on four Mininet [44] servers, each equipped with 256 GB of memory and 24 cores running at 2.5 GHz. To simulate a realistic network, we configured Mininet with a 100 ms point-to-point latency between the nodes and a per-node bandwidth of 100 Mbps.

9.1 Mirco-benchmarks

The two primary questions we want to answer for on-chain secrets are whether the latencies of read and write transactions are acceptable when deployed on top of blockchain systems and whether it can scale to hundreds of trustees to achieve a high degree of decentralization. We compare CALYPSO against a centralized (single server) and a semi-centralized setup (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 where we separately analyze 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 Appendix G.

9.1.1 *Long-term Secrets*

To answer our questions for the permissioned setting we look at Figure 4. It presents the overall latency costs of the key setup (DKG), write, read, share retrieval and share reconstruction sub-protocols. Except for the 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 the 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 who tend to have similar latencies [34, 2] for the same level of decentralization.

9.1.2 *One-time Secrets*

To answer our questions for the permissionless setting we look at Figure 5. We observe that the client-side creation of the tx_w takes almost one second for 64 trustees. This is expected as preparing the tx_w involves picking a polynomial, evaluating it at n points, and setting up the PVSS shares and commitments. Our experiments also show that verifying the NIZK decryption proofs and recovering the shared secret is substantially faster than creating the tx_w and differ by an order of magnitude for large numbers of

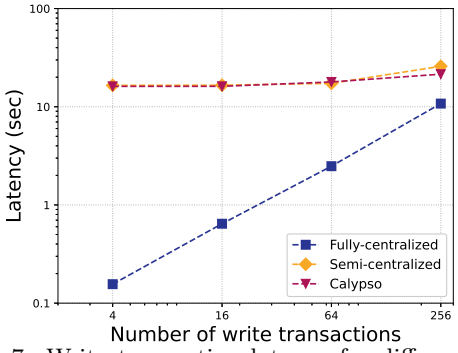

Figure 7: Write transaction latency for different loads in clearance-enforcing document sharing.

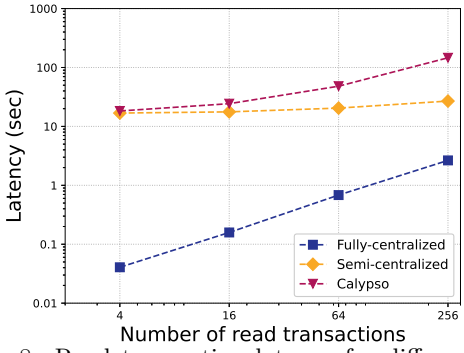

Figure 8: Read transaction latency for different loads in clearance-enforcing document sharing.

shares because the verification and reconstruction require less eliptic-curve cryptography operations than the setup of the PVSS shares. Finally, the overhead for the recovery on the secret-management committee is an order of magnitude higher than the client side since the client sends a request to each trustee. Although these overheads look substantial, the microbenchmark actually demonstrates the feasibility of deploying one-time secrets in a premissioneless setting with minmal overhead compared to the confirmation latency of minutes that existing open blockchain have [67].

9.2 Clearance-Enforcing Document Sharing

To show both the cost and the benefit of CALYPSO, we compare our clearance-enforcing document sharing deployment with both a fully-centralized access-control system and our implementation of a state-of-the-art semi-centralized access-control system (*e.g.*, [6, 17, 29, 59]), where accesses and policies are logged on-chain but the data is managed in the cloud. We vary the simulated workload per block from 4 to 256 read and write transactions and report the time it takes to execute all transactions. These experiments use a blocktime of 10 seconds.

Figure 7 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 more substantial 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 8 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 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 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 6 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.

9.3 Decentralized Lottery

Finally, to show that CALYPSO can provide algorithmic speedup to certain applications, we compare our CALYPSO-based zero-collateral lottery with the corresponding lottery by Miller et al. [43] (tournament) on simulated and real workloads. Figure 9 shows that CALYPSO-based lottery performs better both in terms of overall execution time and bandwidth usage. Specifically, our lottery runs in one round (it always takes two blocks to finish) while the tournament runs in a logarithmic number of rounds due to its design consisting of multiple two-party lotteries.

Next, we evaluate both lottery implementations using transactions from an Ethereum-based lottery called Fire Lotto [65]. 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. Figure 10 shows the total time it takes to run the lotteries. Each data point in the graph

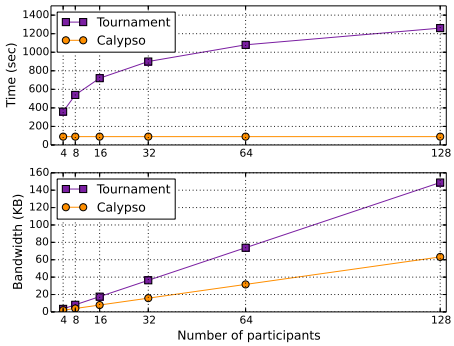

Figure 9: Lottery evaluation using simulated workloads.

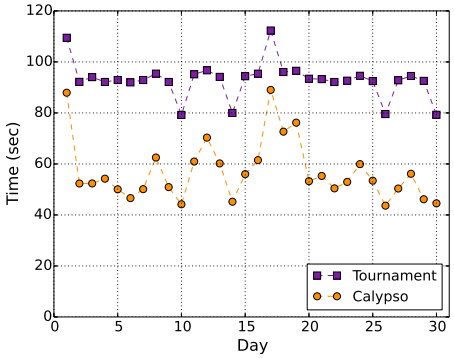

Figure 10: Lottery evaluation using Fire Lotto workloads.

corresponds to a single lottery run. As before, CALYPSO-based lottery performs better because it completes in one round whereas the tournament lottery requires a logarithmic number of interactions with the blockchain and consequently has a larger overhead. More specifically, while the blocktime of 15 seconds makes up $14 - 20\%$ of the total latency in CALYPSO, it contributes most of the per-round latency to the lottery. Our results only include the latency of the reveal phase since the commit phase happens asynchronously over a full day.

10. 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 [21], time-locked vaults [52], and multi-party games [37] can now be deployed within blockchain ecosystems without the need for a centralized manager.

Private data storage has been widely studied in databases [48], however adding decentralization is challenging. One interesting work on secure data-sharing is Vanish [23], which guarantees that data, which is no longer needed or used, self-destructs to protect against accidental leakage. CALYPSO can implement a similar functionality by adding a time out on the write transaction after which the symmetric encryption keys (or the secret shares) are destroyed, but it is more robust as it uses a blockchain instead of a DHT.

Nevertheless, CALYPSO is still limited to guarantee 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 [16] or homomorphic encryption [20]. 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 a different noise to each.

The closest work to ours is the decentralized data management platform Enigma [69] that provides a comparable functionality to CALYPSO. Users own and control their data and a blockchain enforces access control by logging valid requests (as per the on-chain policy). However, Enigma stores the confidential data at a non-decentralized storage provider that can read and/or decrypt the data or refuse to serve the data even if there is a valid on-chain proof. The storage provider in Enigma is therefore a single point of compromise/failure. Other projects [6, 17, 29, 59] rely on centralized key-management and/or storage systems as well and hence suffer from similar issues with respect to atomicity and robustness against malicious service providers.

Ekiden [13], takes a different approach than Enigma. It removes the need for a cloud storage provider, however this comes at a cost of trusting a secure enclave (*e.g.*, Intel SGX), which is not decentralized as CALYPSO. CALYPSO is the first system that truly supports the full CIA triad without having any single point of failure or compromise. However, this comes at moderate overhead. CALYPSO can only provide constant-sized write transactions in the permissioned model. If a client in the permissionless model wants this feature, he needs to trust a predefined set of service providers as he cannot randomly choose anyone. One possible extension is to combine multiple predefined sets of long-term secrets servers in an one-time secrets instance and generating 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 [42, 56] do not sufficiently address the problem of sharing data. Although they allow commiting confidential data on-chain, they rely on the intial 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. The on-chain blinded key-exchange protocol is designed with a goal in mind. It enables Wanda to protect the identity of the intended reader of her secrets without forfeiting any of on-chain secrets' guarantees, however, it requires knowledge of reader's public key. If Ron wants to have 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 [62], or even using the already decentralized confidential data to build prediction models without seeing the data, but only the final result.

11. CONCLUSION

This paper shows how to enable a blockchain to hold secrets using CALYPSO. CALYPSO achieves its goals by introducing two separate components. The first component, on-chain secrets, is deployed on top of a blockchain to enable transparent and efficient management of secret data via threshold cryptography. The second component, skipchain-based identity and access management, allows for dynamic identities and 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 applications.

12. 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] 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.

[11] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Feb. 1999.

[12] 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.

[13] 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.

[14] M. Czernik. On Blockchain Frontrunning, Feb. 2018.

[15] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology (CRYPTO)*, Aug. 1989.

[16] 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.

[17] 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.

[18] V. Durham. Namecoin, 2011.

[19] J. Ellis. The Guardian introduces SecureDrop for document leaks. *Nieman Journalism Lab*, 2014.

[20] J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

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

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] J. Kwon. TenderMint: Consensus without Mining. 2014.

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

[40] 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.

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

[42] 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.

[43] 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.

[44] Mininet – An Instant Virtual Network on your Laptop (or other PC), Feb. 2018.

[45] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[46] 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.

[47] M. Pilkington. Blockchain Technology: Principles and Applications. *Research Handbook on Digital Transformation*, 2015.

[48] R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591, 2016.

[49] B. Rajabi and Z. Eslami. A Verifiable Threshold Secret Sharing Scheme Based On Lattices. *Information Sciences*, 2018.

[50] randao.org. Randao: Blockchain Based Verifiable Random Number Generator, 2018.

[51] 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.

[52] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, Mar. 1996.

[53] P. Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107, 2002.

[54] 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.

[55] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[56] 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.

[57] 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.

[58] SECBIT. How the winner got Fomo3D prize - A Detailed Explanation, Aug. 2018.

[59] 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.

[60] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[61] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Advances in Cryptology — EUROCRYPT'98*, pages 1–16, 1998.

[62] H. Subramanian. Decentralized blockchain-based electronic marketplaces. *Communications of the ACM*, 61(1):78–84, 2017.

[63] M. H. Swende. Blockchain Frontrunning, Oct. 2017.

[64] N. Szabo. Smart contracts. *Unpublished manuscript*, 1994.

[65] F. Team. Fire Lotto blockchain lottery, 2018.

[66] 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.

[67] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 2014.

[68] 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.

[69] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

APPENDIX

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 [25] or asynchronous [36]. Given the rarity of the setup phase we run the DKG by Gennaro et al. [25] 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[5] $L \in \{0,1\}^l$. 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 [53] to be shared as $c_m = \mathrm{enc}_k(m)$ and compute $H_{c_m} = \mathrm{H}(c_m)$. Set policy $= \mathrm{pk}_R$ 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:

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

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.

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.

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

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 distingusih 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 appraoch that has less overhead.

B. ONE-TIME SECRETS PROTOCOLS

We follow the protocol in [57] 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 $\mathrm{tx_w}$ on the blockchain. Wanda initiates the protocol as follows.

1. Compute $h = \mathrm{H}(\text{policy})$ 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 $= \mathrm{pk}_R$ to designate Ron as the intended reader of the secret message m.
5. Finally, prepare and sign the write transaction: $\mathrm{tx_w} = [\langle \widehat{s_i} \rangle, \langle b_j \rangle, \langle \pi_{\widehat{s_i}} \rangle, H_c, \langle \mathrm{pk}_i \rangle, \text{policy}]_{\mathrm{sig}_{\mathrm{sk}_W}}$, and send it to the access-control 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: $\text{tx}_\text{r} = [H_\text{w}, \pi_{\text{tx}_\text{w}}]_{\text{sig}_{\text{sk}_R}}$, 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 tx_w using H_w and use pk_R, as recorded in policy, to verify the signature on tx_r.
2. If the signature is valid and Ron is authorized to access the secret, log tx_r in block b_r.

Share Retrieval Protocol.

After the read transaction has been logged, Ron can recover the secret message m by running the share retrieval protocol with the secret-management 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: $\text{req}_\text{share} = [\text{tx}_\text{w}, \text{tx}_\text{r}, \pi_{\text{tx}_\text{r}}]_{\text{sig}_{\text{sk}_R}}$, where π_{tx_r} proves that tx_r has been logged on-chain.
2. Send req_share to each secret-management trustee to obtain the decrypted shares.

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

1. Use pk_R in tx_w to verify the signature of req_share and π_{tx_r} to check that tx_r has been logged on-chain.
2. Compute the decrypted share $s_i = (\widehat{s_i})^{\text{sk}_i^{-1}}$, create a NIZK proof π_{s_i} that the share was decrypted correctly. The proof shows the knowledge of the unique value that satisfies $\log_g \text{pk}_i = \log_{s_i} \widehat{s_i}$.
3. Derive $c_i = \text{enc}_{\text{pk}_R}(s_i)$ to ensure that only Ron can access it.
4. Create and sign the secret-sharing reply: $\text{rep}_\text{share} = [c_i, \pi_{s_i}]_{\text{sig}_{\text{sk}_i}}$, and send it back to Ron 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 $s_i = \text{dec}_{\text{pk}_R}(c_i)$ and verify it against π_{s_i}.
2. If there are at least t valid shares, use Lagrange interpolation to recover s.
3. Recover the encryption key as $k = \text{H}(s)$ and use it to decrypt the ciphertext c to obtain the message m.

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 [60]. We need the following assumptions to provide confidentiality. First, we assume that Wanda has post-quantum confidential and authenticated point-to-point communication channels [10] 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 [49] or NTRU [1].

Write Transaction Protocol

Wanda prepares her write transaction 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 $k = \text{H}(s)$ as the symmetric key for encrypting the secret message m. $c = \text{enc}_k(m)$ and set $H_c = \text{H}(c)$.
3. For each trustee i, generate a commitment $q_i = \text{H}(v_i \parallel s(i))$, where v_i is a random salt value.
4. Specify the access policy and prepare and sign tx_w.
$$\text{tx}_\text{w} = [\langle q_i \rangle, H_c, \langle \text{pk}_i \rangle, \text{policy}]_{\text{sig}_{\text{sk}_W}}$$
5. Send the share $s(i)$, salt v_i, and tx_w to each secret-management trustee using a secure channel.

The secret-management committee verifies tx_w as follows.

- Check that $(s(i), v_i)$ corresponds to the commitment q_i. If yes, sign tx_w and send it back to Wanda as a confirmation that the share is valid.

The access-control blockchain finally logs Wanda's tx_w.

- Wait to receive tx_w signed by Wanda and the secret-management trustees. Verify that at least $2f+1$ trustees signed the transaction. If yes, log 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 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 tx_w protocol (*e.g.*, provided bad shares or used a higher-order polynomial). In response, Ron can release the transcript of the 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 [51]. 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

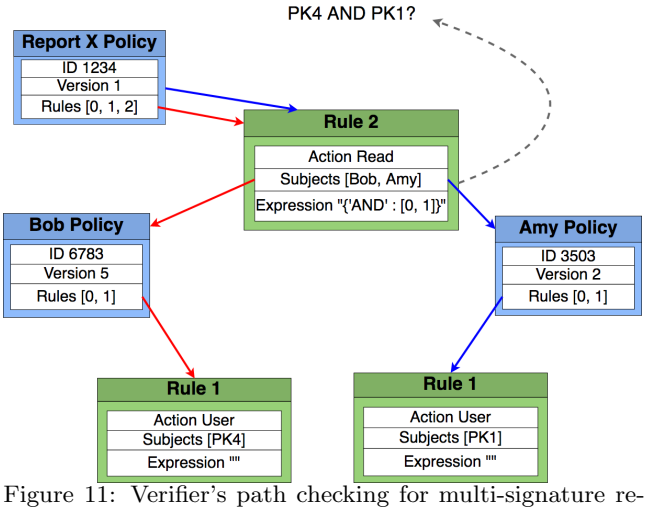

Figure 11: Verifier's path checking for multi-signature requests.

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

E. ACCESS REQUESTS AND VERIFICATION

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

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

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

Figure 11 shows an example of the path verification performed by the verifier. Report X has a policy with a Rule granting read access to Bob and Amy. There is an expression stating that both Bob's and Amy's signatures are required to obtain access. Hence, if Bob wants access, he sends a request $(\mathrm{req}, [\mathrm{sig}_{\mathrm{req},\mathrm{pk}_1}, \mathrm{sig}_{\mathrm{req},\mathrm{pk}_4}])$, where req $= [1234, 2, \text{“}ReportX''']$ The verifier checks the paths from the policy to Bob's pk_4 and Amy's to pk_1 are valid. Paths are shown in red and blue respectively. Then the expression AND : [0,1] is checked against the signatures. If all checks pass, the request is considered to be verified.

JSON Access-Control Language

A sample policy for a document, expressed in the JSON based language, is shown in Figure 12. The policy states that it has one Admin rule. The admins are S1 and S2 and they are allowed to make changes to the policy. The Expression field indicates that any changes to the policy require both S1 and S2's signatures.

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

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

F. INTEGRATION OF SIAM AND CALYPSO

To integrate SIAM with CALYPSO, the long-term secrets protocols described in Section 4.1 are adapted as follows. Assume that Ron has logged the unique identifier id_R of his personal identity skipchain on the access-control blockchain. If Wanda wants to give Ron access to a resource, she simply sets policy $= \mathrm{id}_R$ instead of policy $= \mathrm{pk}_R$ in tx_{w}.

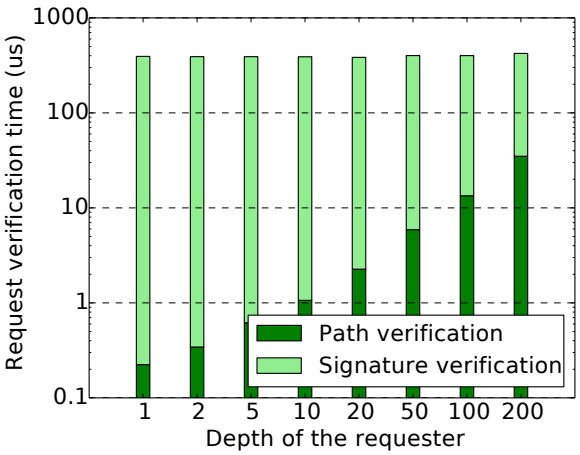

Figure 13: Single-signature request verification.

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

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

where $H_w = \text{H}(\text{tx}_\text{w})$ is the unique identifier for the secret that Ron requests access to, π_{tx_w} is the blockchain inclusion proof for tx_w, and $\text{pk}_{R'}$ is one of Ron's public keys that he wishes to use from the latest block of the id_R skipchain. After receiving tx_r, the access-control blockchain follows the id_R skipchain to retrieve the latest skipblock and verifies $\text{pk}_{R'}$ against it. Then, the access-control blockchain checks the signature on tx_r using $\text{pk}_{R'}$ and, if valid, logs tx_r. Once tx_r is logged, the rest of the protocol works as described in Section 4.1, where the secret-management committee uses $\text{pk}_{R'}$ for re-encryption to enable Ron to retrieve the resource.

F.1 SIAM Evaluation

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

For single-signature requests, the verification time is the sum of the signature verification and the time to validate the identity of the reader requesting access by checking it against the identity of the target reader as defined in the policy. The validation is done by finding the path from the target's skipchain to the requester's skipchain. We vary the *depth* of the requester, which refers to the distance between the two skipchains. Figure 13 shows the variation in request verification time depending on the requester's depth. We observe that most of the request verification time is required for signature verification which takes $\approx 385\,\mu\text{s}$ 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 $35\,\mu\text{s}$.

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

G. 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. 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.