Droplet: Decentralized Authorization and Access Control for Encrypted Data Streams

Hossein Shafagh and Lukas Burkhalter, *ETH Zurich;* Sylvia Ratnasamy, *UC Berkeley;* Anwar Hithnawi, *ETH Zurich & UC Berkeley*

https://www.usenix.org/conference/usenixsecurity20/presentation/shafagh

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Open access to the Proceedings of the 29th USENIX Security Symposium is sponsored by USENIX.

Droplet: Decentralized Authorization and Access Control for Encrypted Data Streams

Hossein Shafagh
ETH Zurich

Lukas Burkhalter
ETH Zurich

Sylvia Ratnasamy
UC Berkeley

Anwar Hithnawi
UC Berkeley & ETH Zurich

Abstract

This paper presents *Droplet*, a decentralized data access control service. Droplet enables data owners to securely and selectively share their encrypted data while guaranteeing data confidentiality in the presence of unauthorized parties and compromised data servers. Droplet's contribution lies in coupling two key ideas: *(i)* a cryptographically-enforced access control construction for encrypted data streams which enables users to define fine-grained stream-specific access policies, and *(ii)* a decentralized authorization service that serves user-defined access policies. In this paper, we present Droplet's design, the reference implementation of Droplet, and the experimental results of three case-study applications deployed with Droplet: Fitbit activity tracker, Ava health tracker, and ECOviz smart meter dashboard, demonstrating Droplet's applicability for secure sharing of IoT streams.

1 Introduction

The growing adoption of IoT has led to an ever-increasing number of applications that collect sensitive user data. This growth has come with mounting concerns over data privacy. To date, the norm has been that user data is collected and governed by application providers, e.g., Fitbit/Strava. The problem with this status quo is that, because data lives in narrow and disjoint silos, it severely limits a user's ability to control access to her data, extract additional value from it, or move data across applications. This problem has led many – from both the technical and non-technical communities – to call for a new *user-centric* model for IoT services, in which the storage of user data is decoupled from the application logic, and control over access to this data is in the hands of *end-users* rather than service providers [30, 70, 106, 109, 110].

However, if we are to realize this paradigm, we need system designs that tackle data privacy as a first-class citizen, while ensuring users ability to securely, selectively, and flexibly grant *data access* to third-party services[1]. Realizing such flexible yet secure access control is key if we are to extract insightful value from user data, e.g., drive large-scale analytics from IoT data.

Such access control must ideally provide the following properties: *(i)* strong data confidentiality and integrity, with cryptographic guarantees, accompanied with efficient cryptographic operations. This is particularly essential in the context of resource-constrained IoT devices and the high volumes of data they generate. *(ii)* fine-grained access control; specify who can access what temporal segment of a data stream. *(iii)* no trusted intermediaries; systems today rely heavily on trusted intermediaries, e.g., for delegated access, rendering them trust bottlenecks. In addition to the above, any solution must satisfy standard access control requirements, e.g., support for revocation and auditability.

No existing solution simultaneously provides all of the above properties. The de-facto standard deployments today [10, 33, 54, 75, 98] rely on trusted services (e.g., access control lists [96], Active Directory [37], OAuth [75]) and assume that the entity which enforces access control – e.g., Fitbit or a storage provider – is within the data owner's trusted domain and consequently can see the data in the clear. However, this approach does not meet our goals of user-centric control; in fact, as many have argued [32, 73, 94, 101, 106, 113], this approach fails to provide even basic data privacy since the provider sees data in cleartext and consequently can share or sell data without user consent [40, 104].

The alternative to the above approach is to rely on *end-to-end* encryption [47, 86, 88, 93, 94, 106, 113]; where data is encrypted at the user device and stored encrypted at the storage provider; encryption/decryption is only executed at authorized parties and services, without disclosing any encryption keys to intermediaries. This, however, introduces the challenge of selective sharing of encrypted data, i.e., supporting flexible access control policies. Solutions adopted today for sharing encrypted data [59, 65, 102] fall short in *expres-*

[1] Note that users can delegate control to a third-party provider just like today - this is permissible, just not the de-facto model.

siveness (i.e., allowing fine-grained access policies), *flexibility* (i.e., updates to access permissions), and *usability* (i.e., key management and revocation). For instance, a common approach is encrypting data under each data consumer's public key; this approach suffers from hard-coded policies [73, 112], and does not scale for high-volume and high-velocity data streams. Moreover, in many cases, this solution is not viable, since data consumers are not necessarily known in advance, as is the case in the IoT's publish-subscribe model [62].

The main question and the focus of this paper is: *how to realize a decentralized access control in a user-centric architecture?* A solution to access control has two parts: *(i)* data protection (e.g., encrypting data such that a principal can only access the authorized data segment), and *(ii)* authorization (e.g., verifying the identity of a principal and authenticity of access permissions).

In this paper, we devise a new system architecture and a crypto-based data access construction to address the above problems. Droplet builds on three insights. The first is that *access control and authorization need to be co-designed* for end-to-end encrypted systems. The second insight is that time is the natural dimension of accessing data streams. Hence, we design our access control with *time as a prime access principle*. The third is that there is a need for decentralized authorization services that operate *without relying on trusted intermediaries*. This is a difficult requirement, which we address with replicated state machines. Such append-only distributed logs as underlying for example the certificate transparency [71] or blockchains, provide guarantees about the existence and status of a shared state in an environment, where no single trusted intermediary is in charge and control, providing a virtual global witness to prevent equivocation [105].

While blockchains provide an alternative trust model, their use comes with challenges. Currently deployed blockchains exhibit a high overhead and low bandwidth due to their consensus protocols. While read operations are fast, chain-writes are inherently slow. Hence, a key challenge is to bypass these limitations. We design Droplet such that blockchain operations are not on the critical path of reading and writing data; *we store the absolute minimum control metadata in the blockchain* and outsource data streams and metadata to off-chain storage, by leveraging indirections. This design minimizes the bandwidth requirements on the blockchain, and allows for lightweight clients, which only retrieve block headers and the accompanied compact Merkle proofs. Droplet's authorization service leverages an existing public blockchain to maintain a replicated access control state machine. This design allows any node to independently bootstrap the authorization state in a decentralized manner and check the access permissions (i.e., ensuring discoverability of access permissions without any out-of-band communication). Access permissions are cached at the storage node for their hosted content, allowing low latency lookups of access permissions.

To realize the crypto-based access control in Droplet; devices encrypt and sign their data locally. Data owners register ownership of data streams and define privacy-preserving access permissions through Droplet's authorization service. Only authorized principals are cryptographically capable of accessing (i.e., decrypting) authorized data segments. We design a novel key distribution and management construction to enable efficient key updates (i.e., succinct – key size is independent of the granted data access range) and fine-grained yet scalable sharing of both arbitrary temporal ranges and open-ended streams. Our design builds on key regression and hash trees via a layered encryption technique. In summary, Droplet ensures data owner's sovereignty and ownership over their data, such that they maintain the ultimate power to selectively and flexibly share their data.

With a prototype implementation[2] of Droplet, we quantify Droplet's overhead and compare its performance to the state-of-the-art systems. When deploying Droplet with Amazon's S3 as a storage layer, we experience a slowdown of only 3% in request throughput compared to the vanilla S3. Moreover, we show Droplet's potential as an authorization service for the serverless paradigm with an AWS Lambda-based prototype. We show Droplet's performance is within the range of the industry-standard protocol for authorization (OAuth2). We also deploy Droplet with a decentralized storage layer to give insights about its potential for the emerging decentralized storage services [65, 102]. With our example apps on top of Droplet, we show that real-world applications with unaltered user-experience (i.e., perceived delay) can be developed.

In summary, our contributions are:

- Droplet, a new decentralized authorization service that enables secure sharing of encrypted data and works without trusted intermediaries.
- a new crypto-enforced access control construction that provides flexible and fine-grained access control over encrypted data streams with succinct key states.
- a design that couples authorization with crypto-enforced access to mitigate the limitations of current authorization services (lack of cryptographic guarantees) and end-to-end encrypted data (static policies).
- an open-source prototype and evaluation of Droplet showing its feasibility, and competitive performance.

2 Droplet's Overview

Droplet's main objective is to empower users with full control (ownership) over their data while ensuring data confidentiality. More concretely, we want to facilitate flexible and fine-grained secure sharing of encrypted data without ever exposing the data in the clear to any intermediaries including the storage and authorization services. We define data ownership as having the right and control over data, wherein the owner can define/restrict access, restrict the scope of data utility (e.g., sharing aggregated/homomorphically-encrypted

[2] Droplet is available under https://dropletchain.github.io/

data), delegate these privileges, or give up ownership entirely without the need to rely on any trusted entities to facilitate this. A true realization of this definition requires work on two fronts: *(i)* privacy-preserving computation (i.e., differential privacy and secure computation) and *(ii)* secure and privacy-preserving access control of remotely stored data with strong confidentiality guarantees. In this work, we focus on the latter, specifically in the context of data streams.

2.1 Droplet in a Nutshell

At a high level, Droplet is a decentralized access control system that enables users to securely and selectively share their data streams with principals. Droplet's design marries a novel crypto-enforced access control construction tailored for time-series data and a decentralized authorization service. Our crypto-enforced access control construction enables users to express flexible stream access control policies (§3). The key idea behind our encryption-based access control is to serialize time series data into chunks where each chunk corresponds to a time segment and is encrypted with a unique encryption key. The challenge here becomes how to efficiently generate and manage a large number of unique encryption keys and allow expressing access polices with a minimum shared state that is then used to derive all decryption keys associated with the access policy. To address this specific challenge, we introduce a novel key management construction with a succinct key state, i.e., the key size does not grow with the temporal range of shared data (§3). Although crypto-based access control is powerful, it is not sufficient by itself, as it does not adequately handle authorization and revocation. To address this issue, we introduce a decentralized authorization service (§4) that interplays with our crypto-based access control construction.

Consequently, data owners are not required to exchange any encryption keys directly with data consumers. Our decentralized authorization, in its essence, is similar to OAuth2. However, we realize the access control state machine on top of an existing blockchain (§4.2), and eliminate the need for trust intermediates on which OAuth2 realizations heavily depend. The access control state machine assembles the current global state (i.e., access permissions and data ownership) through embedded private state transitions.

2.2 Security Model

Threat model. *(i) Data storage:* we consider an adversary who is interested in learning about users' data. Our threat model covers malicious storage nodes, potential real-world security vulnerabilities leading to data leakages, and also external adversaries who gain access to data as a result of system compromise. *(ii) Access Permissions State:* an adversary may access and bootstrap the access control state machine, but it cannot alter or learn sensitive information about the access permissions (e.g., sharing relationships or keying material). For an adversary to alter the access permission states, it needs to break the security of the underlying blockchain. The standard

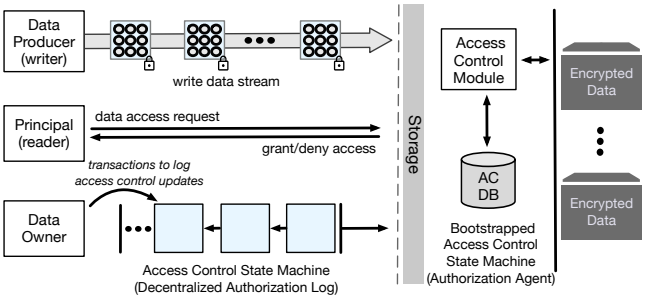


Figure 1: Abstract protocol flow. Data is E2E encrypted with encryption-based access control. The data owner stores access permission updates in the decentralized authorization log. The storage service validates access requests based on the access permissions from the access control state machine.

blockchain threat model assumes that an adversary cannot control a large percentage of nodes in the network, for the blockchain to be considered secure. The actual ratio depends on the deployed consensus protocol by the underlying blockchain. For instance, given n blockchain nodes and f adversary nodes, a ratio of $n = 2f + 1$ for Nakamoto-style consensus mechanisms [79] or $n = 3f + 1$ for PBFT consensus mechanisms [16] is required for the honest majority.

Guarantees. Droplet embodies a decentralized encryption-based access control mechanism that enables secure and selective access to stream data within the above-discussed threat model. Data is encrypted at the client-side, and keys are at no time disclosed to intermediaries, i.e., storage and authorization services, guaranteeing data confidentiality, integrity, and authenticity. Decryption keys are only shared with authorized parties via a blockchain-based indirection, ensuring *asynchronicity*, i.e., keys are established without requiring participants to be online at the same time. In case decryption keys are compromised, Droplet guarantees that only the user's data stream segment associated with the key is disclosed, and the compromised keys cannot be used to disclose past or future data beyond the temporal segment associated with the key. Data partitions are signed, allowing parties without decryption keys to verify data authenticity and integrity. Droplet enables checking the freshness of data, and it provides data immutability optionally via an authenticated data structure anchored in the blockchain, such that even the data owner can no longer modify past data. Droplet cryptographically prevents evicted users from accessing future data. Though evicted users may have already cached past data, they are, however, prevented from future access. Droplet encodes user-defined access permissions in the blockchain, eliminating trusted intermediaries and assuring collusion-resistance and auditability. Moreover, we employ privacy-preserving access permissions, preventing an observer from learning the sharing parties' identities. Droplet does not protect against denial-of-service attacks, nor does it hide access patterns. It could be extended with ORAM

techniques to hide access patterns [64, 99]. Cryptographic techniques alone are not sufficient to prevent a malicious storage provider from denial-of-service or deconstruction of data. Hence, adequate replication strategies on multiple providers are necessary to ensure the preservation and availability of data. In §C, we discuss the security guarantees in more detail.

Assumptions. In Droplet, we make the following assumptions. We assume the storage nodes to be available. This is a valid assumption since storage nodes can face financial (and potentially legal) consequences upon detection of misbehavior. Droplet guarantees data confidentiality even if malicious storage nodes hand over data illegitimately, as data is end-to-end encrypted. We assume the adversaries to be subject to the standard cryptographic hardness and the underlying blockchain to be secure, i.e., similar to previous work [3, 6, 19, 105], we assume transactions are append-only, ordered, and immutable after a confirmation period and the blockchain to be highly available. We assume users store their keys securely and that key recovery techniques are deployed (we discuss in §9 potential recovery techniques, such as Shamir's secret sharing). We assume data producers to report correct data and to perform data serialization and encryption correctly. We assume there is a financial agreement between the storage provider and data owner to provide persistent storage, which can also be facilitated through the cryptocurrency feature of the underlying blockchain.

2.3 Architecture

As illustrated in Figure 1, our design considers four actors and three system components: **data owner** is someone who owns a set of devices (e.g., wearables, appliances, services) which produce time-series data, i.e., **data producers**. In an industrial setting, the data owner can be an organization that owns a swarm of IoT devices. The generated data is stored on storage services, and data owners can decide to selectively expose their data to data consumers (i.e., **principals**) who can produce an added value from the data (e.g., fuse several streams for prediction tasks). Data is end-to-end encrypted at the data producer, and each principal computes the corresponding decryption keys locally based on an encrypted authorization token (i.e., embodies the access policy state) shared through *Droplet's decentralized authorization log*. Data owner, data producer, and data consumer run *Droplet's client library*, which covers the tasks of data serialization, enc/decryption, key management, and setting/viewing access permissions. Moreover, end-user applications (e.g., Fitbit/Strava) interact directly with Droplet's client API to facilitate sharing through Droplet. The **storage** node is in charge of storing encrypted data and providing access to principals as defined by the data owner. The storage node grants or denies access requests via Droplet, i.e., in accordance with user-defined access permissions. Access permissions are cryptographically bound to a specific principal's identity (public key). The storage node can take various forms, such as edge, decentralized (e.g., a node in a p2p storage service [65]), or cloud storage (e.g., Amazon's S3). The storage node runs Droplet's storage engine and can additionally run *Droplet's authorization agent* to handle access requests locally. *Droplet's authorization agent* bootstraps its state from the *decentralized authorization log*. As a matter of fact, anyone can run *Droplet's authorization agent* to either expose it as a service or to monitor the state of relevant access permissions. Note that Droplet's decentralized authorization agents are stateless and cache relevant access permissions for fast lookup, e.g., maintaining access permissions of resources stored by the storage node.

Droplet is, in essence, a new decentralized access control system that is materialized by coupling a new encryption-based access control scheme and a decentralized authorization service. In the following, we elaborate on our encryption-based data access construction. As the backbone of our encryption-based data access, we present the design of an efficient key-management construction. Afterward, we discuss Droplet's decentralized authorization service.

3 Encryption for Access Control

Goals. With our crypto-enforced data access construction, we pursue a design that fulfills the following goals: *(i) Flexible sharing abstractions*: support of the three common types of sharing modalities desired for time-series data, varying based on the role and purpose of the data consumer; *(a)* subscription, where the data consumer is granted continuous access to the data stream as it is generated, either temporarily or until revoked, (e.g., a visualization app rendering an overview of the user's daily activity based on wearable data), *(b)* sharing arbitrary intervals of past data (e.g., a practitioner app accessing and analyzing user's health data during past pregnancy), and *(c)* a combination of *i* and *ii*. *(ii) Efficiency:* computationally efficient crypto primitives to adhere to the constraint resources of IoT devices, *(iii) Scalability:* to cope with the velocity and large volume of time-series data.

Gist: A key aspect of our construction is tied to the observation that time-series data streams are continuous. Hence, we introduce time-encoded key-streams which map keys to temporal segments of the data stream, such that access to the data stream can be restricted by only sharing the corresponding range in the keystream with a principal. Based on the access policy, the principal gains access to the necessary decryption keys via an access token. Access tokens are encrypted with the principal's public key (hybrid encryption). To enable sharing without enumerating all the keys and expressing stream access policies in a succinct shared state, we design a key derivation construction that synthesizes the concepts underlying hash trees and dual-key regression.

3.1 Encryption-based Access Control

Each data chunk of a data stream is encrypted under a random symmetric key derived from our key derivation construction. Keys are rotated for each chunk permitting access permissions

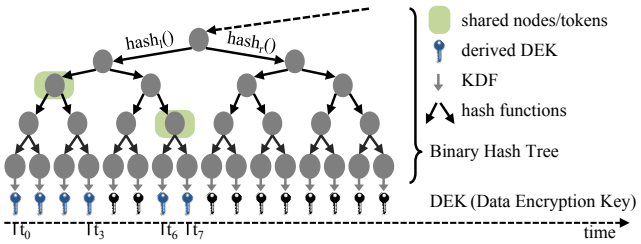


Figure 2: Droplet's key generation. Data Encryption Keys (DEKs) are managed through the hash tree, allowing efficient sharing of arbitrary intervals. An access policy contains several shared nodes as authorization tokens.

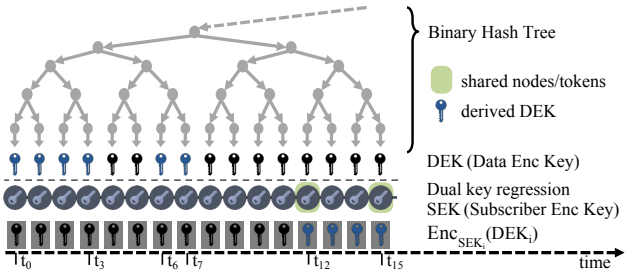


Figure 3: Droplet's hybrid key management supports sharing of arbitrary intervals (hash tree) and subscriptions (dual-key regression). Given the opening and end tokens (dual-key regression), one computes the interval Data Encryption Keys.

at the chunk level. This allows for flexible access policies for individual data consumers without the need for data re-encryption or introducing redundant data. The design of our key derivation construction in its core builds on hash trees [26] and key regression [50] to enable expressing *stream-specific access policies* and *efficient management of encryption keys.* Droplet supports computing a large segment of keys from a single shared state instead of sharing individual keys.

We now give a brief background on hash trees and key regression and their role in our encryption-based access control construction. We elaborate why these two components alone fall short in meeting our design requirements and describe how we leverage them to create our hybrid key management construction. We formalize the security guarantees of our key management in A.

Binary Hash Tree (BHT). A BHT [26] is a balanced binary tree, built top-down from a secret random seed as the root; using two cryptographic hash functions for the left and right child nodes, i.e., hash$_l$() and hash$_r$(), respectively. Initially the hash functions are applied to the root node. This procedure is applied recursively until the desired depth h in the tree is reached, as depicted in Figure 2. The leaf nodes represent the keystream $\{k_0, k_1, k_2, ..., k_{2^h-1}\}$. We select a large h such that the keystream is virtually infinite.

We encrypt each data chunk of the data stream with a unique key derived from the BHT. With this construction users can efficiently share any arbitrary time interval of their stream; by just sharing the inner nodes in the BHT necessary to compute the corresponding keys. For instance, in Figure 2, given the two highlighted inner nodes a data consumer is granted access to two disjoint intervals $t_{[0-3]}$ and $t_{[6-7]}$, and can compute the corresponding decryption keys. While consistent with our efficiency and low overhead requirements, this BHT-based construction lacks support for sharing in subscription mode, where data consumers have continuous access to data streams. Realizing this mode of sharing with BHT requires maintaining and sharing a growing state per individual data consumer.

Key Regression. Key regression [50] is a hash-chain based construction that enables sharing a large number of keys by only sharing a single state. Given a single hash token, one can derive all previous keys by applying the hash function successively, i.e., given key K_t in time t one can compute all keys until the initial key K_0, i.e., $\forall_{i\in[0..t]} K_i$. However, no future keys can be computed (forward-secrecy). This is not always desirable, as key regression enables sharing of all keys from the beginning until *current time* (all-or-none principle).

Dual-Key Regression. To overcome the all-or-none limitation of key regression, we design a hash chain construction that enables sharing with a defined lower time bound, e.g., access to data of a particular stream from *Nov'18* till revoked. To realize this, we extend key regression with an additional hash chain in the reverse order, to cryptographically enforce both boundaries of the shared interval (Figure 4). In simple key regression, hash tokens are consumed in the reverse order of chain generation as input to a key derivation function to derive the current key. Due to the pre-image resistance property of hash functions, it is computationally hard to compute future tokens and hence future keys. However, the reverse can be computed efficiently. We leverage this property of hash chains for defining the beginning of an interval through a secondary hash chain in the reverse order, as depicted in Figure 4. In the *dual-key regression*, the Key Derivation Function (KDF) takes a second token h'_i: $KDF(h_i||h'_i) = K_i$, with h'_i from the secondary hash chain (Figure 4). For instance, to share a data stream from time t_i to t_j, the user provides the tokens h'_i and h_j. Since it is infeasible to compute h_{j+1}, no key posterior to k_j can be computed. Conversely, since it is infeasible to compute h'_{i-1}, no key prior to k_i can be computed. With access to the two hash tokens (h_j, h'_i), indicating the beginning and end of the shared interval, one can compute all the encryption keys within this interval. We formalize and prove the security guarantees of dual-key regression in A.1.

3.2 Droplet's Key Management

We now discuss how our design compounds *dual-key regression* and BHT via a layered encryption technique to enable stream sharing abstractions. Dual-key regression resembles a linear chain of keys, where for a given state, i.e., *beginning* and *end* tokens, one can compute all the keys in between.

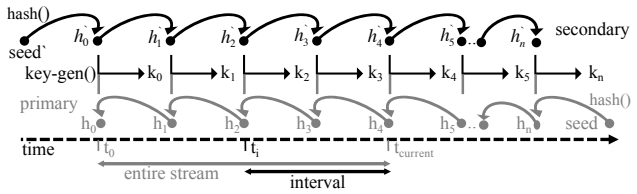


Figure 4: The dual-key regression supports time-bounded sharing via a secondary hash chain. The gray elements depict the standard key regression mechanism: Given current k_c, one can compute all keys up to k_0. Our construction allows the sharing of keys for an interval via a secondary hash chain.

Conceptually, we exploit the hash tree to allow arbitrary sharing of intervals and the dual-key regression to support sharing in subscription mode.

The layered encryption consists of two steps: *(i)* the hash tree delivers time-encoded data encryption keys DEK_i, which we use to encrypt data generated during the time epoch i. *(ii)* the dual-key regression also delivers time-encoded subscriber encryption keys SEK_i for the epoch i. We use SEK_i to encapsulate the corresponding data encryption key: $ENC_{SEK_i}(DEK_i)$. For fast access, each encrypted data chunk holds the encapsulated DEK. With this construction, we can give access to data encryption keys either via the hash tree (arbitrary intervals) or dual-key regression (subscription), as depicted in Figure 3. To a subscriber, DEKs appear as random encryption keys. For principals with access to past data, DEKs are the leaf nodes of the BHT which they locally compute based on the shared inner nodes (e.g., root nodes of the corresponding subtrees). Note that a principal can be granted access in both modes simultaneously, as shown in the example of Figure 3. In this example, the data owner has granted the principal access to the intervals $t_{[0-3]}$ and $t_{[6-7]}$, which is realized through the hash tree. Also, the principal is granted a subscription from t_{12} which is realized over dual-key regression. We describe next how to handle long key chains efficiently and in constant space.

Key Distribution. An important aspect to address in crypto-based access control schemes is how to distribute keys efficiently. In Droplet, this is especially tricky for the subscription mode, where new data chunks arrive continuously, and each one is encrypted with a new key. We now describe our key distribution mechanism and refer to §4 for insights on obtaining the keying material over the decentralized authorization service. When a new data consumer is added, an authorization token encapsulating the defined access policies is issued which contains either: (i) the state to compute decryption keys for past data intervals (i.e., inner nodes of the hash tree) or (ii) in case of sharing in the subscription mode the hash token for the beginning of the interval h'_i (i.e., dual-key regression). For the subscription mode the challenge is to give the active set of subscribers continuous access to the latest token (i.e., h_t from the main chain), such that they can compute the current decryption key. If we were to encrypt the current hash token for each subscriber individually, this would incur communication/computation overheads in $O(s)$, given s subscribers.

To reduce this overhead, we distribute the latest dual-key regression token h_t within a digitally signed and encrypted lockbox. Authorized subscribers obtain the long-term *distribution key KD* to open the lockbox. This approach is more efficient than resorting to per subscriber encryption. When sharing access to a data stream, we share the distribution key encrypted for the new subscriber through the authorization service (§4). While *data encryption keys* and hence dual-key regression tokens are frequently updated at a defined interval, the distribution key is only updated after an access revocation event, as detailed next.

A subscriber decrypts the current data encryption key $ENC_{SEK_{j+1}}(DEK_{j+1})$ given the current token h_{j+1} and the opening token h'_i as:

$$KDF(h_{j+1}||h'_{j+1}) = SEK_{j+1}, \text{with } \mathrm{H}^{(j-i+1)}(h'_i) = h'_{j+1} \quad (1)$$

with H as a hash function. The secondary token is stored along the long-term per principal key information (§4).

Revocation. To revoke data stream access, the data owner updates the distribution key (i.e., crypto-based access) and issues a state update transaction (i.e., authorization) to evict the revoked service. The transaction includes a new distribution key KD' contained in the encrypted key information per subscriber. Hereafter, the new data encryption key is only available to the remaining authorized subscribers, protected with the new distribution key. The transaction confirmation time of the underlying blockchain determines the delay until Droplet's authorization state machine is updated. The end-to-end encryption, however, prevents revoked users from accessing new data instantly, due to the preceding key rotation.

With the newly issued transaction, the global access permission state is updated (§4). Droplet cryptographically prevents any future access to new data by the evicted subscriber. Any future access requests by the evicted subscriber to old data are declined during authorization.

Compact Hash Chains. Our key management, specifically dual-key regression, relies heavily on hash chains. The underlying chains can grow quickly due to frequent key updates. Given the memory-constraints of IoT devices, we revert to a combination of re-computing on-demand and storing a segment of the hash chain in memory, to achieve fast and efficient key rotations. We leverage hierarchical hash chains [61] which maintain the same security features as traditional hash chains but reduce the worst-case compute time to $O(\sqrt{n})$. In our evaluation in §8.1, we show how compact chains allow for a two-orders of magnitude key rotation speed-up.

4 Decentralized Authorization Service

So far, we have covered Droplet's encryption-based access control mechanism. Now we describe Droplet's authorization service which handles and manages access permissions.

At a high level, through Droplet's API, users can view their data streams, the associated sharing policies, and storage information, and can set/edit access permissions accordingly. Similar to today's authorization frameworks, e.g., OAuth2, our authorization service acts on behalf of users, forgoing direct interaction of individual services with the data owner. Storage providers query Droplet's authorization agent directly to validate access requests. Moreover, principals query the authorization agent to retrieve authorization tokens. The authorization agent falls under the same trust assumptions as the storage node, which enforces the authorization verdict. This means that the storage node can act maliciously, i.e., bypass the agent's authorization verdict, and hand out data to unauthorized parties. Similarly, an authorization agent can also act maliciously. However, due to Droplet's end-to-end encryption, these violations do not compromise data confidentiality (§6).

In our design, we employ a tamper-proof decentralized authorization log to enable anyone to bootstrap and presume the role of *authorization agent* and serve access permission lookups in a decentralized and verifiable manner. We realize the authorization log using a publicly verifiable blockchain to maintain an accountable distributed access control system *without a central trusted entity*. This allows us to move away from a single authorization server issuing and verifying access tokens, to where any resource owner can issue access permissions and any node can verify it. We now describe the owner-device pairing, blockchain-embedded access permissions, and how we protect the privacy of principals.

Owner-Device Pairing. The blockchain ecosystem relies on public key cryptography for identification and authentication of the involved principals. The hash digest of the public key serves as a unique pseudo-identity in the network. We leverage this feature to allow IoT devices to securely and autonomously interact with the storage service. This way we overcome the hurdle of passwords and rely on public-key crypto for authentication and authorization. During the bootstrap phase of a new device, it creates a pair of public-private keys locally, where the private key is stored securely and never leaves the device. Through an initial two-way multisignature registration transaction on the blockchain, Droplet allows the binding of IoT devices (PK_{device_i}, SK_{device_i}) to the owner (PK_{Owner}, SK_{Owner}). Henceforth, the owner can set access permissions (via the private key SK_{Owner}) and the IoT devices are permitted to store data (via the private key SK_{device_i}) securely. The necessary keying material for encryption (§3) on the data producer is also exchanged during the initial phase. Note that the data owner's private key is powerful in that it sets/updates access permissions. Droplet assumes a data owner private key management scheme to be in place (e.g., Human-Memorable Password-Protected Secret Sharing, backed with hardware security modules or multiple trusted devices [15, 63]), and a key recovery mechanism to be employed for handling a potential key loss (see §9).

In the event of device decommissioning, the new owner must issue a new multisignature device-binding transaction, to gain ownership of future data produced by the same device. Note that there is no need for the IoT device to interact with the blockchain directly. The owner creates the raw multisignature registration transaction and uses an out-of-band channel (e.g., Bluetooth Low Energy) to get the device's signature. After adding her signature, she broadcasts the register transaction to the network. Note that the channel between the IoT device and owner must be secure, otherwise we risk disclosure of the device's private key. In the absence of an out-of-band-channel or in the case where the device?s capabilities are limited, for instance, lack of secure key storage, a secure proxy can be leveraged to handle proper data serialization (§5).

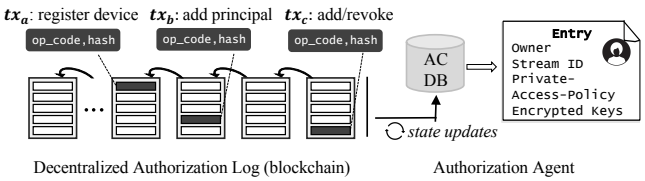


Figure 5: Droplet's authorization agent bootstraps the access control state machine (consolidated into the AC DB) from the transitions embedded in the decentralized log and accompanied off-chain access policies (not depicted for simplification).

Access Permissions. We utilize the blockchain to store access permissions in a secure, tamperproof, and time-ordered manner. Access permissions are granted per data stream. Initially, the data owner issues a transaction including the stream ID which creates the initial state. To change this state, e.g., grant read access permissions to a principal, the data owner issues a subsequent transaction which holds, among others, *(i)* the stream ID, *(ii)* the public key of the principal they want to share their data with, *(iii)* the temporal scope of access (e.g., intervals of past or open-end subscription), and *(iv)* encrypted keying material for data decryption (§3). For public key discovery of users, Droplet can leverage decentralized identity management solutions [38, 66, 84]. These efforts focus on establishing an open and standards-based decentralized identity ecosystem, removing any reliance on centralized systems of identifiers. Such solutions, e.g., Keybase [66], serve as a key directory that maps a user's online identities (e.g., Twitter, Github) to their public key in a publicly verifiable manner. The higher the dimensions of interlinked identities, the lower the probability of identity fraud.

Once a request to store or retrieve data is received at a storage node, it queries the Droplet's authorization agent (§4.2) for the corresponding access permissions, as illustrated in Figure 5. To enforce the permissions, the storage node verifies the identity of the requesting user via a signature-based authentication [31]. Data owners express and dynamically adjust permissions through Droplet client, which interacts with Droplet's authorization log only (i.e., blockchain) and not with individual services (i.e., asynchronicity). Data access

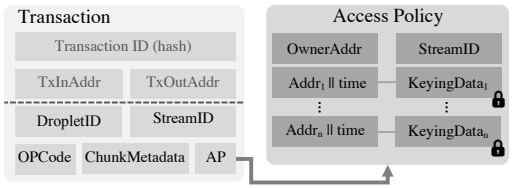


Figure 6: Overview of access control transactions, which embed transitions to the global access control state via an indirection (i.e., hash to the off-chain Access Policy).

is enforced cryptographically through end-to-end encryption. The storage node validates data access requests based on the embedded access permissions in the authorization log. The authorization log additionally protects storage nodes' network resources (i.e., bandwidth/memory) from unauthorized users. For instance, this mitigates an attack, where malicious parties flood the network with download/storage requests of large files. The storage node can terminate malicious sessions (e.g., data scraping and storage spamming attacks) after checking the access permissions (§4.2). Droplet supports privacy-preserving access permissions and audits by authorized entities, which we explain in §4.1.

Access Policy Indirections. Blockchain storage is scarce and expensive, as it is replicated and maintained by the blockchain network. This entails placing only the minimum necessary logic in the blockchain. To keep the number and more importantly the size of transactions as low as possible, our design incorporates off-chain storage of the Access Policy, as illustrated in Figure 6. Instead of holding the address information of all services, the transaction includes an indirection to the Access Policy via the hash digest of it. This allows managing access permissions with an unlimited number of services in a single transaction. Besides, the Access Policy can now contain advanced access control logic (e.g., XACML [4]), such as access groups and delegating parties. Any change to the Access Policy requires a new transaction. The hash digest serves as a data pointer and, more importantly, protects the integrity of the Access Policy. The Access Policy is stored off-chain. The time until an access permission change comes into effect is tied to the transaction confirmation time of the underlying blockchain, ranging from few seconds to minutes depending on the underlying blockchain.

4.1 Privacy-Preserving Sharing

In public blockchains, users are represented through virtual addresses, providing pseudonymity. However, advanced clustering heuristics can potentially lead to the de-anonymization of users [5, 77]. Access permissions in Droplet should be enforceable by storage nodes (i.e., verify authorization) and be auditable by authorized parties. However, we want to protect the privacy of sharing relationships from the public. To realize this, we leverage dual-key stealth addresses. With stealth addresses [36], we protect a principal's privacy, from any party who can view the access permissions, with regards to the resources they are granted access to. Moreover, different streams shared with the same principal are unlinkable. However, a data owner may learn whom they are sharing their data with. Note that if there is no channel between the data owner and data consumer to indicate requested or granted access permissions, the consumer needs inevitably to scan the permissions in Droplet's access permission state machine to identify any data that is shared with them.

Conceptually, each principal is represented by two public keys (main and viewer keys: PK_m, SK_m, PK_v, SK_v), which other parties use to generate a new unlinkable address PK_{new}. The viewer private key SK_v can be shared with an auditor to audit the permissions. However, access to both main and viewer private keys is required for data access, i.e., SK_m and SK_v are needed to compute SK_{new}, which only the principal is capable of (see B for technical details).

4.2 Access Control State Machine

Today, there are two main options developers can take for realizing decentralized applications that employ a blockchain as a ubiquitous trust network (i.e., a shared ground truth): *(i)* operating a new blockchain, or *(ii)* embedding the application logic into an existing secure blockchain deployment [81, 105]. We opt for the latter where we embed our logic without alternation of the underlying blockchain nor requiring the instantiation of a new blockchain. This allows us to benefit from an existing blockchain's security properties without introducing and running a new blockchain. Note that Droplet's state machine can alternatively employ a private authorization log, to address use-cases with a different trust model or in a closed ecosystem. We briefly discuss the reasons why we opt for this choice and detail on how we realize this efficiently.

Integrating a new application logic into a running blockchain typically results in consensus-breaking changes and hard forks, i.e., a new blockchain with only a subset of peers enforcing the new logic. While necessary for specific applications, this results in parallel blockchains which may not exhibit strong security properties due to a smaller network of peers (e.g., Namecoin's network became decentralized with one mining group controlling the majority of hashing power [3]). To benefit from the security properties of a strong and robust blockchain, new apps can embed their log of state changes in transactions. This is in turn used to bootstrap the global state in a secure and decentralized manner. We employ virtualchains [3, 81] to efficiently embed Droplet's access control logic in an existing global blockchain. A virtualchain is a fork*-consistent replicated state machine, allowing different applications to run on top of any production blockchain, without breaking the consensus. Droplet's authorization agent scans the underlying blockchain for the corresponding access permission transactions and maintains the global state in a database that can be queried for permissions of a given stream and principal.

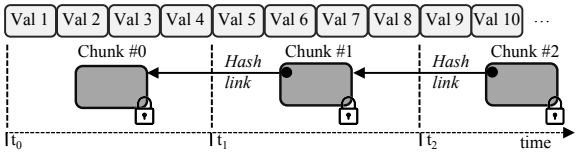


Figure 7: Data streams chunked at defined temporal intervals, and cryptographically linked together. For record lookup, the timestamp is mapped to the chunk identifier.

5 Data Serialization

In Droplet's data model, a data stream is divided into chunks of predefined time intervals (Figure 7); chunking and batching are common techniques for time-series data [49, 58, 72, 111]. Although chunking prevents random access at the record level, it results in a positive performance gain for data retrieval as in time-series data most queries require access to temporally co-located data [58, 111]. E.g., data analytic apps work with temporal data records (e.g., all records of a day).

Encryption. Each data chunk is initially compressed and then encrypted at the source with an efficient symmetric cipher[3]. We rely on AES-GCM, as an authenticated encryption scheme. Note that NIST bounds the use of AES-GCM to 2^{32} encryptions for a given key/nonce pair. Due to our frequent key rotations, we stay far below this threshold. The chunks have a metadata segment containing, among others, the chunk identifier, the owner's address, hashes to previous chunks (§5), and the stream identifier. The data field contains the encrypted and compressed data records. Services with access to the encryption key can verify the integrity of the chunk and perform an authenticated decryption. To ensure data ownership, each chunk is also digitally signed. This allows parties without access to the encryption key to verify the owner of the data stream, albeit at a higher computation cost.

Storage Interface. The storage nodes expose a key-value interface, with a common *store*/*get* interface with various flavors of *get*, such as *getAll* or *getRange*. For each incoming request, the storage node first verifies the identity of the client (i.e., authentication) and looks up the corresponding access permissions regarding the client's identity (i.e., authorization). Each request is accompanied with a universally unique identifier (UUID), defined as the hash of the tuple: <`owner address, streamID, counter`>, where `streamID` is a unique identifier of an owner's data stream. Traditional indexing for data retrieval cannot be applied here as data chunks are encrypted. Hence, we need to devise a mechanism to perform temporal range queries over encrypted data efficiently. To avoid consistency issues of a shared index, we exploit a simple local lookup mechanism to enable temporal range queries. For a constant lookup time of a record with timestamp t_i, we compute the counter of the chunk holding it based on the known time interval Δ of the chunks: $\lfloor (t_i - t_0)/\Delta \rfloor$. For instance, we can map the lookup of value 7 in Figure 7 to the identifier of chunk #1. The chunk metadata is included in the initial stream registering transaction, as depicted in Figure 6. Note that the chunk metadata additionally enables freshness checks for chunks, since the chunk interval indicates the frequency and time at which new data chunks are generated.

[3] Note that it is important to apply padding to prevent inference attacks based on the varying sizes of the chunks.

Strong Data Immutability. While Droplet provides integrity protection via authenticated encryption and digital signatures, the data owner can still modify old data. Specific applications might require a stronger notion of immutability such that even the data owner can no longer modify the data (e.g., contractual agreements in logistics). Droplet enables such a notion of immutability through blockchain's append-only property [25]. The application developer can define a grace period, after which data chunks become immutable. For sensitive applications, this can be per chunk. Otherwise, a more extended period can be selected. To accommodate for the narrow bandwidth of blockchains, we leverage an anchoring technique, where data immutability transactions are reduced to the level of the grace period. To realize this, the first data chunk holds a pointer to the registration transaction, and after the grace period, a transaction with a pointer to the latest chunk is issued, as depicted in Figure 8. Since all data chunks are cryptographically linked via hashes, all data chunks in the grace period become immutable at once, forming a chain of data chunks. To avoid a linear verification time, chunks hold hashes to several previous chunks, forming a geometric series (i.e., logarithmic verification time).

6 Privacy and Security Analysis

Authorization. In conventional authorization frameworks, i.e., OAuth, any entity in possession of the bearer token can assume the same access permissions granted to the token [100]. In case of token theft, the adversary in possession of the token can gain unauthorized access to the user's resources (i.e., impersonation attack). Moreover, the compromise of an authorization server enables the issuance of unauthorized access tokens for all registered resources at the authorization server. Droplet is not susceptible to these attacks. In Droplet, an authorization claim with the scope of access is logged in the blockchain in a privacy- preserving manner, such that only the authorized party in possession of the correct private key can claim ownership for data access, in a publicly-verifiable manner. For an adversary to alter access permissions in the blockchain, it requires forging a digital signature (i.e., breaking public key cryptography with a 128-bit security level) or gaining control over the majority of the computing power in the blockchain network (i.e., 51% attack [3]). Existing production blockchains, e.g., Bitcoin or Ethereum, can be subject to security attacks, such as routing [7] and selfish mining [46], which can lead to access permission state update transactions to be dropped, delayed, or excluded. An active adversary can employ these attacks to prevent/delay access permission modifications of victims from taking effect. However, none of these attacks can lead to unauthorized access permission.

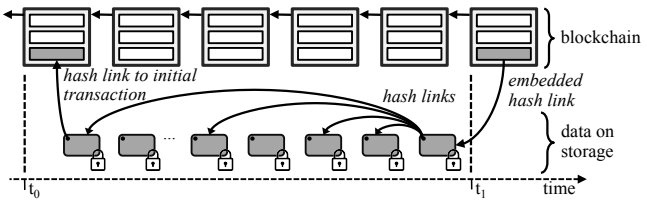


Figure 8: Example of immutable chunks, with a grace period ($t_1 - t_0$). Chunks are cryptographically linked together, forming a geometric series, enabling faster integrity checks.

An adversary is not capable of learning sensitive information from the public blockchain, since only unlinkable pseudo-identities and stream identifiers are stored. In profiling attacks, the adversary creates profiles of all user identifiers and the network of users [77]. An adversary can break the pseudonymity of specific users. Hence, a large body of research aims at concealing identity and relationships in public blockchains while maintaining verifiability [27, 57, 92]. Droplet employs dual-key stealth addresses, where the anonymity set is equal to the set of users using non-spendable stealth addresses.

A malicious storage node (or authorization agent) could hand out data without permission or data leakage might take place due to system compromise. However, the impact of this action is limited since data is end-to-end encrypted. Moreover, leakage of a data encryption key results only in the disclosure of the data stream segment associated with it. The compromised key cannot be used to disclose old data nor can it be used to gain access to future data due to pre-image resistance property of hash functions. The distribution key (KD) for continuous stream subscription gives access to the latest token from the primary chain. The compromise of KD has no impact without access to the aligned token from the secondary chain (Figure 4) since both tokens are required to compute data encryption keys. An attacker needs to compromise an authorized user's private key to gain access to tokens from the secondary chain. The blockchain provides auditable information about when a stream was shared with whom; a crucial piece of evidence to prove/disprove access rights violations should the need arise.

Data Serialization. Data chunks are encrypted, integrity protected, and authenticated. Any data chunk manipulations are detectable via the digital signature and authenticated encryption. Note that while a property of AES-GCM can be exploited to find collisions within ciphertexts that decrypt to different valid plaintexts [39], the per chunk signature in Droplet protects us from such an attack. The optional data immutability is based on the security and immutability of blockchain. The secure channel (i.e., TLS) for storing and fetching data prevents replay attacks, in addition to ensuring an authenticated and confidential channel. An adversary with access to disclosed encryption keys cannot alter old data, as it requires access to the signing private key.

	AES Encrypt		SHA Hash		ECDSA Sign	
	[μs]	[op/s]	[μs]	[op/s]	[ms]	[op/s]
IoT SW	298	3.4k	297	3.4k	270	3.7
IoT HW	42	23.8k	17	58.8k	174	5.7
Phone	50	20k	45	22.2k	4.4	227
Laptop	5.4	185k	1.6	623k	1.3	770
Cloud	2.6	384k	1.2	833k	1.1	909

Table 1: Performance of security operations – 128-bit security. For IoT devices, we use OpenMote microcontrollers with software (SW) computations or crypto accelerators (HW). As a smartphone, we use a Nexus 5. As a laptop, we use Macbook Pro. For the cloud, we use an Amazon t2.micro instance.

7 Implementation

Our reference implementation of Droplet is composed of three entities implemented in Python: the client engine, the storage-node engine, and the authorization agent. The client engine is implemented in 1700 sloc. We utilize Pythons's `cryptography` library [89] for our crypto functions. For compression, we use `Lepton` [41] for images and `zlib` [34] for all other value types.

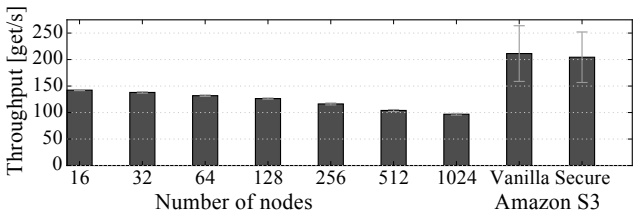
The storage engine can either run on the cloud or nodes of a p2p storage network. For the cloud, we have integrated a driver for Amazon's S3 storage service.

We have as well a realization of Droplet with a serverless computing platform with ASW Lambda serving as the interface to the storage (i.e., S3). Once Lambda is invoked, it performs a lookup in the access control state machine to process the authorization request. For comparison, we implement as well an OAuth2 authorization, based on AWS Cognito [11]. For the distributed storage, we build a DHT-based storage network. We instantiate a Kademlia library [90] and extend it with the security features of S/Kademlia [17]. On the p2p storage nodes, we employ LevelDB [74]. Our extensions amount to 2400 sloc.

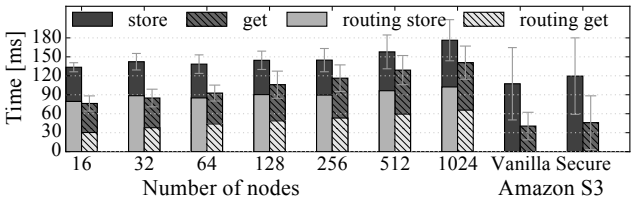
The authorization agent is implemented with the virtualchain library [3] to maintain the access control state machine. The virtualchain scans the blockchain, filters relevant transactions, validates the encoded operations, and applies the outcome to the global state. The state is persisted in an SQLite database. The global state can either be queried through a REST API or accessed directly through the SQLite database. Our extensions to the virtualchain amount to 1400 sloc. As the underlying blockchain, we employ a Bitcoin test-network with a block generation time of 15 s.

8 Evaluation

Goals. One of our primary goals was to develop Droplet as a practical system, which translates to ensuring: that Droplet can *(i)* be supported by existing resource-constrained IoT devices, *(ii)* sustain a high access permission lookup and verification throughput, and *(iii)* that the overhead to both



(a) Average throughput for *get*.



(b) Latency for single *store* and *get* requests.

Figure 9: *store*/*get* performance for centralized and decentralized storage layers. The latency for the decentralized storage is dominated by network routing. For fairness, all settings, including Vanilla S3 (w/o Droplet) operate on compressed data chunks.

data owners and consumers is low, allowing consumers to process large volumes of data streams. Hence, our evaluation metrics include the overheads (CPU, memory) that Droplet imposes on each party, as well as the end-to-end throughput and latency that apps experience with Droplet. Our evaluation is conducted in the context of real-world devices, datasets, and runtime environments.

Devices. We perform our evaluation on the following four device classes: *(i)* IoT: OpenMotes equipped with 32-bit ARM Cortex-M3 SoC at 32 MHz, a public-key crypto accelerator running up to 250 MHz. Fitbit trackers utilize a similar class of micro-controllers; *(ii)* smartphone: LG Nexus5 equipped with a 2.3 GHz quad-core 64-bit CPU, 2 GiB RAM; *(iii)* laptop: MacBook Pro equipped with 2.2 GHz Intel i7, 8 GiB RAM; *(iv)* Cloud: EC2 t2.micro (1 vCPU, 1 GiB RAM).

Datasets. We validate the applicability of Droplet by deploying three real-world IoT applications atop of Droplet and quantifying the end-to-end overhead due to our system; *(i)* for the Fitbit activity tracker, we use the anonymized fitness tracker data of the co-authors over one year (16 data types, 130 MB), which we use to synthesize data for an arbitrary number of users. *(ii)* for the Ava health tracker [9], we use an anonymized dataset from Ava [9] (10 s intervals, 13 sensors, 1.3 GB). *(iii)* for the ECOviz smart meter dashboard, we use the publicly available anonymized ECO dataset (1.85 GB) for 6 households over 8 months [18].

Storage Infrastructure and Runtime Environment. We run Droplet on both centralized and decentralized storage layers. For the former, we use Amazon's S3 service, and for the latter, we implement and run several DHT nodes in real-time on an emulated network (e.g., using `netem` [82]). Evaluating Droplet in a decentralized storage setting is a compelling case, as peer-to-peer storage networks could become a viable solution for the IoT [110]. Additionally, this setup resembles storage-oriented blockchains (e.g., Storj [103], Filecoin [102]), which still lack adequate mechanisms for secure data sharing, where Droplet can be helpful. We also evaluate Droplet's performance in a serverless setting (Lambda [12]) and compare it to OAuth2 authorization. Emerging serverless platforms require request-level authorization [1], where Droplet can serve as an *Authorization as a Service*.

8.1 Microbenchmark

We instrument the client engine to perform the microbenchmark in isolation with up to 1000 repetitions.

Cryptographic Operations. Table 1 summarizes the costs of the crypto operations involved in Droplet on four different platforms. All these operations, namely AES encryption, SHA hash, and ECDSA signature are performed once per chunk for store requests. For data retrieval, the client does not perform a signature verification, since AES-GCM has built-in authentication. Running the crypto operations only in software on the IoT devices shows the highest cost, with 3.4k encryptions/hashes per sec and only 3.7 signatures per sec. With the onboard hardware crypto, the cost of AES and SHA is improved by one order of magnitude and approaches that of smartphones. Note that overall signatures are three orders of magnitude slower than symmetric key operations.

Crypto-based Access. Hash computations are the basis for dual-key regression. The computation occurs at the initial setup and each key update if the client chooses to re-compute keys on-demand rather than store them. Assuming a chain length of 9000 (hourly key updates for one year), it takes 405 ms to compute the entire chain on smartphones and 2.7 s on an IoT device without a hardware crypto engine. With compact hash chains, we reduce this worst-case compute time to 4.3 and 28.2 ms, respectively. The performance gains become pronounced with smaller epoch intervals. The hash tree induces $O(log\ n)$ computations for n keys, which amounts to 48 μs (laptop) with 2^{30} keys.

The per chunk overhead consists of key computation (hash tree and dual-key regression), chunk encryption, key encryption, and signature, which amounts to 1.5 ms (laptop) without caching. Compared to ABE (§10), Droplet's crypto-based data access is by a factor of 57x faster. E.g., with ABE per chunk overhead with only two attributes (timestamp for temporal access and data type) amounts to 86 ms (laptop).

Feasibility for IoT. To assess if Droplet is viable for the IoT, we validate its practicality for low-power devices, concerning their constraint resources (Table 1). Crypto operations are the most expensive ones on a data producer, and beyond that, no connectivity to the authorization services is required. Today, most IoT devices are equipped with crypto accelerators for AES encryption integrated with their radios; however, accelerators for hash functions and signatures have yet to

become the norm. Nevertheless, Droplet is feasible on legacy IoT devices without accelerators despited 1.5x slower signing operations. In terms of impact on the energy budget, the signature consumes only 9 to 25mJ. Considering a wearable's lithium-polymer battery capacity of 1.2 Wh (4.32 kJ), and a 48h charge cycle, 3 signatures/minute (8.6 with accelerator) can be computed with 5% of the energy budget.

8.2 System Performance

To model the real-world performance of Droplet, we constructed an end-to-end system setup, where we use our three apps datasets. Note that we do not cache any data to emulate worst-case scenarios. The stream chunk size is set to 8 KiB. We evaluate `get` and `store` requests to the storage layer, which include the overhead of Droplet's access control.

Serverless Computing. In the serverless setting, Lambda either runs Droplet for the access control or uses the AWS Cognito service, which runs OAuth2, as the baseline. Lambda with both Droplet and Cognito exhibits a latency of around 118 ms (0.4% longer with Droplet). Note that with OAuth2, to reach the same level of access granularity as with Droplet, separate access tokens are required for each data chunk, which is impractical. This is why in practice, long-lived and more broadly-scoped access tokens are granted.

Cloud. We extend AWS S3 storage with Droplet and compare its performance against vanilla S3. Figure 9(a) shows the throughput for different request types. We follow Amazon's guidelines to maximize throughput: e.g., the chunk names are inherently well distributed allowing the best performance of the underlying hash-table lookup. The vanilla S3 throughput of 211 gets/s is within Amazon's optimal range (100-300). With Droplet, we maintain an average rate of 204 get/s (3% drop). Figure 9(b) shows the latency for individual *store* and *get* operations. In Droplet, the latency overhead is 13% for *get* and 11% for *store* (incl. crypto). Part of the overhead is due to the expensive signature operation. Also, there is an overhead for a fresh lookup of access permissions at the access control DB of the virtualchain.

Distributed Storage We measure the performance of *get* and *store* requests on a secure DHT with Droplet, with varying network sizes, from 16 to 1024 nodes. Figure 9(a) shows the throughput results. As the number of nodes increases from 16 to 1024, the performance decreases from 142 to 96 get/s. Figure 9(b) shows the latency results, divided into routing and retrieval. The total *get* latency increases from 76 to 140 ms as the number of nodes grows. This is about 3 times slower than S3's centralized storage. However, note that the routing cost dominates this slowdown. After resolving the address of the storage node, which holds the data chunk, the secure retrieval time is similar to that of S3. Also, note that *get* requests have a lower routing overhead than store requests. This is because for get requests, the routing process is aborted as soon as a node holding the data chunk is found.

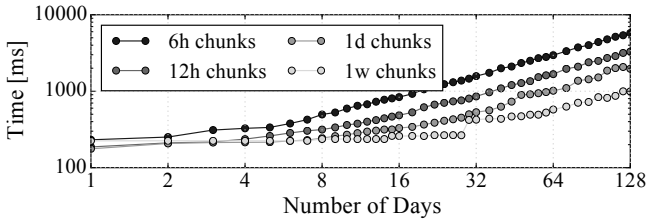


Figure 10: EccoViz app results. Retrieving records from the energy data set in the EccoViz dashboard app (p2p storage).

Applications. The three applications we deploy atop Droplet, vary in terms of type, size, and granularity of collected data. Fitbit and Ava are both smartphone apps, where users view visualized summarizes about their collected data and set goals. We enhance both demo apps, with additional views where the user can selectively share parts of their data (e.g., heart-rate/body-temperature/steps) with friends or services over Droplet. ECOViz dashboard is a web app that visualizes energy consumption from smart-meters. Users can set access permissions per data stream, and they can only view streams to which they have been granted access. The user experience of sharing via Droplet remains similar to that of existing sharing methods. Users initially register a data stream either consisting of a single or multiple data types (e.g., sensitive data types can be highlighted to prevent accidental sharing). Afterward, they can add or remove users to/from their data streams (e.g., the iOS native Health app allows per data stream sharing decisions for third-party apps, similar to our subscription mode).

To measure the overhead induced by Droplet, we quantify the overhead of *store* and *get* data requests for different views (i.e., each access requires cryptographic operations and access permission checks). We now discuss the decentralized storage setting with 1024 nodes. Due to memory constraints, data synchronization is required at least weekly for Fitbit and daily for Ava devices. This results in an average *store* latency of 176 ms and 1.2 s for Fitbit and Ava, respectively. Note that *store* operations run in the background. For different views, the maximum *get* latency is below 150 ms. Hence, the user experience remains unaltered.

In contrast to Fitbit and Ava, the smart meter node has direct Internet connectivity. Instead of synchronizing periodically, it stores chunks after generation. This takes 176 ms per chunk. The most comprehensive view in the ECOViz dashboard can visualize the entire data stream. Figure 10 shows the latency to fetch chunks dependent on the number of days requested. Fetching data for 128 days of 6 h chunk size takes about 10 s, whereas the one-week size takes less than 1 s.

Scalability. Droplet's scalability can be examined from three angles; *(i) Read throughput of authorization*; read operations are performed in $O(1)$, after the authorization agent bootstraps the Access Control State Machine DB. Scaling to handle high read throughputs, is a matter of increasing the number of authorization agents. *(ii) Storage of access permissions*; Droplet

anchors indirections in the blockchain (§4), as we store access policies and metadata off-chain. Hence, to scale with the growing number of access permissions, the allocated off-chain storage is dynamically increased. As Droplet scales to a more significant number of data streams, the access permission logic consequently grows. The individual authorization agents are not impacted by this growth, as they only store the state for the resources they serve. The annual meta-data storage costs[4] for a billion user-base with an average of 100 streams and 100 consumers per stream, would amount to less than $0.001 per user today, which accounts for a fraction of the actual storage costs of streams. *(iii) Write throughput*; represents the scaling bottleneck of Droplet, as access permission updates are bound to the write throughput of the underlying blockchain. Although we consider several optimizations (e.g., grouping access updates) to contain this constraint, it remains a bottleneck. In our current prototype, the transaction confirmation time is set to 15 s, similar to that of Ethereum. The slow blockchain writes have a direct impact on the time until new access permissions take effect, which is significantly higher compared to OAuth2 protocol. Read-throughput is, however, fast and comparable to that of OAuth2. Data stream registrations and access permission adjustments (e.g., grant/revoke access) require transaction writes. To understand the extent of scaling authorization writes in Droplet with an example, consider Fitbit with 25 Million active users, which logged 4.7 million group-join events in 2017 [48], which would require 0.14 transactions per (tps). However, to scale Droplet to billions of data streams, a blockchain throughput of a few thousand tps is necessary (assuming 25% of streams require an access permission modification per day). While currently deployed blockchains achieve only a fraction of this throughput, scaling to higher throughput is an active area of research, and next-generation blockchains already support several thousand tps [69](§9).

9 Discussion

We highlight some research questions that remain open.

Beyond IoT. An authorization service with Droplet's properties is crucial for systems that advocate for data sovereignty [44, 104, 110] or handle privacy-sensitive data, e.g., sharing medical records [13], and humanitarian aid [73]. The storm of recent privacy incidents [20, 35] has prompted a rethinking of this space. Moreover, decentralized storage services that run on blockchain (e.g., Filecoin) can integrate Droplet for data sharing. Services with varying trust assumptions can, however, run Droplet's authorization log instead by a federated set of servers.

Usability. Droplet is a user-centric system that empowers data owners with control over their data. While we design Droplet's API to abstract away system complexities from users and mimic current data sharing abstractions, some usability considerations remain open in this user-centric paradigm. In this paradigm, users will potentially be confronted with more decisions to make regarding their data. Hence, it is essential to study and design abstractions and interfaces that mitigate usability concerns that might arise in this paradigm. In an end-to-end encryption model, protection and recovery mechanisms for private master keys should be addressed with adequate solutions. For instance, Shamir's secret sharing scheme [95] allows reconstruction of the secret from a set of recovery keys which are, e.g., distributed among the data owner's devices [106] or a group of friends [87]. The recovery keys collectively reconstruct a master secret key.

Blockchain Scalability. In §8, we discussed scalability aspects of Droplet and how the underlying blockchain, which realizes the decentralized authorization log, can impact the write throughput within Droplet. Next-generation blockchains [28, 45, 52, 67, 69, 78] particularly tackle the scalability aspects and promise higher throughputs and lower latencies, which is crucial for the adoption of blockchain-based systems in retail payments and financial sector, and for realizing large-scale decentralized applications. Recent works [67, 69] introduce a hybrid consensus by combining the slow PoW to bootstrap the faster PBFT algorithm, where for each epoch, a random set of validators is selected. Hence, they bring both worlds' best: secure open enrollment and high throughput and low latency. These scalable blockchain protocols, e.g., OmniLedger [69], lay the groundwork enabling practical advanced decentralized services, such as Droplet. Droplet can be deployed on top of any blockchain that supports the total ordering of transactions, as elaborated in §4.2.

10 Related Work

We now briefly discuss key relevant works to Droplet.

Crypto-enforced Data Access. End-to-end encryption provides the strongest level of protection for data stored in the cloud, as data remains encrypted and only authorized entities are trusted with decryption keys. However, fine-grained access and sharing of data is a challenge here. A simple approach to selective sharing of encrypted data is to encrypt the target data towards the principal's public key; although simple this approach suffers from three drawbacks: *(i)* hard-coded access control [73]; at encryption time the access permission is defined and cannot subsequently be altered or revoked, *(ii)* storage overhead; if the same data is shared with multiple principals, the user ends up storing redundant data as she needs to encrypt the same data under each principal's public key, and *(iii)* scalability and practicality issues particularly when considering fine-grained access policies. These drawbacks are pronounced with time series-data, where high volume of data is continuously produced and a high key-rotation is necessary to ensure flexible access control.

Various cryptographic schemes [8, 23] have been introduced to overcome some of these challenges, among which

[4] S3 frequent access tier, over 500 TB/Month, $0.021/GB, May 2020.

attribute-based encryption (ABE) [2, 55, 56, 91, 106] offers high expressiveness. Several ABE-based systems [106, 108] introduce crypto-based access control. However, ABE suffers from expensive crypto operations and the costs grow linearly with the number of attributes, limiting the granularity of access due to computational burdens [2, 51]. The overhead dominates even with a hybrid encryption technique [106, 108], where data is encrypted with symmetric encryption and only encryption keys are encrypted with the expensive ABE, e.g., only two attributes result in 100 ms for enc/decryption on desktops and few seconds on IoT devices [107]. FAME [2] exhibits a constant decryption time (60 ms), however, encryption time increases linearly with the number of attributes.

The notion of *time-encoded keys* in our access control is similar to Time-Specific Encryption (TSE) [29, 85]. TSE assigns objects to temporal intervals and for each time instance a unique key is generated. Our scheme differs from TSE in that no central trusted time server is required for the generation and broadcast of epoch keys. In Droplet, each data source generates the data encryption keys per epoch locally, and key distribution is handled over Droplet's decentralized network.

Distributed Authorization. Current distributed authorization protocols, such as OAuth2 [75] and Macaroons [21], suffer from several limitations, as highlighted in §6. Signature-based schemes (e.g., public-key certificates [22, 42]) require means for distributing public keys for verification. Today, conventional approaches to attest to public keys are to rely on internal key servers or at the Internet-scale, hierarchical network of certificate authorities (CA) issuing X.509 certificates, which come with their weaknesses [76], (e.g., Symantec's issuance of unauthorized certificates for Google [97], lack of support for non-domain identities). Alternative public-key based approaches, e.g., SPKI/SDSI [42] and follow-up schemes [43], eliminate the need for complex X.509 public-key infrastructure and CAs. However, these schemes are either based on the idea of local names and suitable for deployments under a single administrative domain (e.g., smart home) or build upon an organically growing trust model [112] (e.g., PGP's web of trust). While the key idea of signature-based schemes underpins Droplet, our system neither suffers from certificate-chain discovery nor requires a complex certificate infrastructure (§4). Droplet's current prototype supports pseudonyms and can be extended with a publicly-auditable directory of keys and identity proofs, such as Keybase, which maps digital identities (e.g., Twitter) to public keys in a verifiable manner [66].

Blockchain-based Systems. Decentralized blockchain-based applications (i.e., without trusted intermediaries) beyond cryptocurrencies have gained more attention in recent years. Example applications include; medical data access [13], IoT device commissioning and management [60], financial auditing [80], name and identity management [3, 14], software-update transparency and verifiability [83] and preventing unauthorized certificate issuance [76]. Closest to our work are; Enigma [113, 114] which envisions a decentralized personal data management and secure multi-party computation platform for multilateral sharing. They use a single data encryption key among the sharing parties (i.e., no fine-grained crypto-based access) and require blockchain transactions for each read/write request (i.e., limited scalability). Calypso [68] introduces on-chain encrypted secrets, with associated access policies. A set of trustees collectively enforces the policies via threshold encryption and distributed key generation, which ranges for each key access request from 0.2 to 8 s, depending on the number of trustees. None of the above systems addresses the challenge of fine-grained access control for encrypted time-series data. Moreover, our design mimics the flow of authorization services in production, so that Droplet can seamlessly be integrated to support current services, as we show through deployments of several case-studies (e.g., serverless computing, §8.2).

11 Conclusion

This paper introduces Droplet, a decentralized access control system that enables secure, selective, and flexible access control that empowers users with full control of their data. With Droplet we present a design that marries a decentralized authorization service and a novel encryption-based access control scheme tailored for time-series data. Our prototype implementation and experimental results show the feasibility and applicability of Droplet as a decentralized authorization service for end-to-end encrypted data streams.

12 Acknowledgments

We thank our shepherd Ariel Feldman, the anonymous reviewers, Alexander Viand, Dinesh Bharadia, and Friedemann Mattern for their valuable feedback. We thank Simon Duquennoy for his valuable input on earlier versions of this paper. This work was supported in part by the Swiss National Science Foundation Ambizione Grant, VMware, Intel, and the National Science Foundation under Grant No.1553747.

References

[1] Gojko Adzic and Robert Chatley. Serverless Computing: Economic and Architectural Impact. In *ACM FSE*, 2017.

[2] Shashank Agrawal and Melissa Chase. FAME: Fast Attribute-based Message Encryption. In *ACM CCS*, 2017.

[3] Muneeb Ali et al. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *USENIX ATC*, 2016.

[4] Anne Anderson et al. eXtensible Access Control Markup Language (XACML). *OASIS*, 2003.

[5] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating User Privacy in Bitcoin. In *FC*, 2013.

[6] Marcin Andrychowicz et al. Secure Multiparty Computations on Bitcoin. In *IEEE S&P*, 2014.

[7] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE S&P*, 2017.

[8] Giuseppe Ateniese et al. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*, 2005.

[9] Ava. Fertility Tracking Bracelet. Online: avawomen.com, 2016.

[10] AWS. Identity and Access Management (IAM). https://aws.amazon.com/iam/.

[11] AWS Cognito. https://aws.amazon.com/cognito/.

[12] AWS Lambda. https://aws.amazon.com/lambda/.

[13] Asaph Azaria et al. Medrec: Using Blockchain for Medical Data Access and Permission Management. In *IEEE OBD*, 2016.

[14] Sarah Azouvi et al. Who am I? Secure Identity Registration on Distributed Ledgers. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. 2017.

[15] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In *ACM CCS*, 2011.

[16] Shehar Bano et al. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*, 2017.

[17] Ingmar Baumgart et al. S/Kademlia: A Practicable Approach Towards Secure Key-based Routing. In *IEEE ICPADS*, 2007.

[18] Christian Beckel et al. The ECO Data Set and the Performance of Non-Intrusive Load Monitoring Algorithms. In *ACM BuildSys*, 2014.

[19] Iddo Bentov and Ranjit Kumaresan. How to use Bitcoin to Design Fair Protocols. In *International Cryptology Conference*, 2014.

[20] John Biggs. It's time to build our own Equifax with blackjack and crypto, 2017. https://techcrunch.com/2017/09/08/its-time-to-build-our-own-equifax-with-blackjack-and-crypto/.

[21] Arnar Birgisson et al. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *NDSS*, 2014.

[22] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *IEEE S&P*, 1996.

[23] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based Encryption with Efficient Revocation. In *ACM CCS*, 2008.

[24] Dan Boneh and Brent Waters. Constrained Pseudorandom Functions and Their Applications. In *ASIACRYPT*, 2013.

[25] J. Bonneau et al. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE S&P*, 2015.

[26] Bob Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. *Networked Group Communication*, 1736:301–320, 1999.

[27] Benedikt Bünz et al. Bulletproofs: Short Proofs for Confidential Transactions and More. In *IEEE S&P*, 2018.

[28] Vitalik Buterin and Virgil Griffith. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[29] Julien Cathalo, Benoît Libert, and Jean-Jacques Quisquater. Efficient and non-interactive Timed-release Encryption. In *Conference on Information and Communications Security*, 2005.

[30] Tej Chajed et al. Amber: Decoupling User Data from Web Applications. In *ACM HotOS*, 2015.

[31] David Chaum and Hans Van Antwerpen. Undeniable Signatures. In *ASIACRYPT*, 1989.

[32] Eric Y Chen et al. OAuth demystified for Mobile Application Developers. In *ACM CCS*, 2014.

[33] Google Cloud. Identity and Access Management (IAM). https://cloud.google.com/iam/.

[34] Compression Library zlib. https://zlib.net/.

[35] Nicholas Confessore. Cambridge Analytica and Facebook: The Scandal and the Fallout So Far. The New York Times, Online: https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html, 2018.

[36] Nicolas T Courtois and Rebekah Mercer. Stealth Address and Key Management Techniques in Blockchain Systems. In *ICISSP*, 2017.

[37] Brian Desmond et al. *Active Directory: Designing, Deploying, and Running Active Directory*. O'Reilly Media, Inc., 2008.

[38] DIF. Decentralized Identity Foundation. Online: https://identity.foundation, (accessed May 2020, 2019).

[39] Yevgeniy Dodis et al. Fast Message Franking: From Invisible Salamanders to Encryptment. In *Crypto*, 2018.

[40] Stuart Dredge. Yes, those Free Health Apps are Sharing your Data with other Companies. Guardian, Online: theguardian.com/technology/appsblog/2013/sep/03/fitness-health-apps-sharing-data-insurance, 2013.

[41] Dropbox Compression. github.com/dropbox/lepton.

[42] Carl M Ellison et al. SPKI Certificate Theory. RFC 2693 (Sep 1999), Online: https://www.ietf.org/rfc/rfc2693.txt, 1999.

[43] Andres Erbsen, Asim Shankar, and Ankur Taly. Distributed Authorization in Vanadium. *arXiv preprint arXiv:1607.02192*, 2016.

[44] European Union. GDPR: Council regulation (EU) no 679/2016. GDPR, Online: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&rid=1, 2016.

[45] Ittay Eyal et al. Bitcoin-NG: A Scalable Blockchain Protocol. In *USENIX NSDI*, 2016.

[46] Ittay Eyal and Emin Gün Sirer. Majority is not Enough: Bitcoin Mining is Vulnerable. In *FC*, 2014.

[47] Ariel J. Feldman et al. SPORC: Group Collaboration Using Untrusted Cloud Resources. In *USENIX OSDI*, 2010.

[48] Fitbit Business Release. https://investor.fitbit.com/press/press-releases/press-release-details/2018/Fitbit-Community-Grows-to-More-Than-25-Million-Active-Users-in-2017/default.aspx.

[49] Mike Freedman. Time-series data: Why (and how) to Use a Relational Database Instead of NoSQL. Timescale, Online: https://blog.timescale.com/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c, 2017.

[50] Kevin Fu et al. Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage. In *NDSS*, 2006.

[51] W.C. Garrison et al. On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud. In *IEEE S&P*, 2016.

[52] Yossi Gilad et al. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *ACM SOSP*, 2017.

[53] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *J. ACM*, 33(4):792–807, 1986.

[54] Dieter Gollmann. *Computer Security*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[55] Vipul Goyal et al. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *ACM CCS*, 2006.

[56] Vipul Goyal et al. Bounded Ciphertext Policy Attribute Based Encryption. In *ICALP*, 2008.

[57] Matthew Green and Ian Miers. Bolt: Anonymous Payment Channels for Decentralized Currencies. In *ACM CCS*, 2017.

[58] Trinabh Gupta et al. Bolt: Data Management for Connected Homes. In *USENIX NSDI*, 2014.

[59] Paul Handy. How Storj Increases Object Storage Security Exponentially. Sorj Blog, Online: https://blog.storj.io/post/145305561698/how-storj-increases-object-storage-security, June 2016.

[60] Thomas Hardjono and Ned Smith. Cloud-based Commissioning of Constrained Devices using Permissioned Blockchains. In *Workshop on IoT Privacy, Trust, and Security*, 2016.

[61] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. Efficient Constructions for One-way Hash Chains. In *ACNS*, 2005.

[62] Urs Hunkeler et al. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *IEEE COMSWARE*, 2008.

[63] Stanislaw Jarecki et al. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *AsiaCrypt*, 2014.

[64] Yaoqi Jia et al. OblivP2P: An Oblivious Peer-to-Peer Content Sharing System. In *USENIX Security*, 2016.

[65] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3). https://github.com/ipfs/papers, 2017.

[66] Keybase. Publicly Auditable Proofs of Identity. Online: https://keybase.io/, (accessed June, 2020).

[67] Eleftherios Kokoris-Kogias et al. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *USENIX Security*, 2016.

[68] Eleftherios Kokoris-Kogias et al. CALYPSO: Auditable Sharing of Private Data over Blockchains. *Cryptology ePrint Archive:209 https://eprint.iacr.org/2018/209.pdf*, 2018.

[69] Eleftherios Kokoris-Kogias et al. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE S&P*, 2018.

[70] John Kolb, Kaifei Chen, and Randy H. Katz. The Case for a Local Tier in the Internet of Things. In *Technical Report No. UCB/EECS-2016-222*, 2016.

[71] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. *IETF, RFC 6962*, 2013.

[72] Florian Lautenschlager et al. Chronix: Long Term Storage and Retrieval Technology for Anomaly Detection in Operational Data. In *USENIX FAST*, 2017.

[73] Stevens Le Blond et al. On Enforcing the Digital Immunity of a Large Humanitarian Organization. In *IEEE S&P*, 2018.

[74] LevelDB by Google. https://github.com/google/leveldb.

[75] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. OAuth 2.0 Threat Model and Security Considerations. *IETF, RFC 6819*, January 2013.

[76] Stephanos Matsumoto et al. IKP: Turning a PKI Around with Decentralized Automated Incentives. In *IEEE S&P*, 2017.

[77] Sarah Meiklejohn et al. A Fistful of Bitcoins: Characterizing Payments Among Men with no Names. In *ACM IMC*, 2013.

[78] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *ACM CCS*, 2016.

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

[80] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In *USENIX NSDI*, 2018.

[81] Jude Nelson et al. Extending Existing Blockchains with Virtualchain. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[82] Netem. https://wiki.linuxfoundation.org/networking/netem.

[83] Kirill Nikitin et al. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *USENIX Security*, 2017.

[84] onename. Decentralized Registrar and and Identity Manager. Online: https://onename.com, (accessed May 2020, 2020).

[85] Kenneth G Paterson and Elizabeth A Quaglia. Time-Specific Encryption. In *Security and Cryptography for Networks*, 2010.

[86] Raluca A. Popa et al. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM SOSP*, 2011.

[87] Raluca Ada Popa. The Importance of Eliminating Central Points of Attack. Preveil, Online: https://www.preveil.com/blog/importance-eliminating-central-points-attack/, 2017.

[88] Raluca Ada Popa et al. Enabling Security in Cloud Storage SLAs with CloudProof. In *USENIX ATC*, 2011.

[89] Python Crypto Library. https://cryptography.io/.

[90] Python DHT library (Kademlia). https://github.com/bmuller/kademlia.

[91] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *EUROCRYPT*, 2005.

[92] Eli Ben Sasson et al. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE S&P*, 2014.

[93] Hossein Shafagh et al. Talos: Encrypted Query Processing for the Internet of Things. In *ACM SenSys*, 2015.

[94] Hossein Shafagh et al. Secure Sharing of Partially Homomorphic Encrypted IoT Data. In *ACM SenSys*, 2017.

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

[96] R Shirey. Internet security glossary. *IETF, RFC 4949*, 2007.

[97] S. Somogyi and A. Eijdenberg. Improved Digital Certificate Security. Online: https://googleonlinesecurity.blogspot.com/2015/09/improved-digital-certificate-security.html, 2015.

[98] Mark Stamp. *Information Security: Principles and Practice*. Wiley Publishing, 2nd edition, 2011.

[99] Emil Stefanov and Elaine Shi. Oblivistore: High Performance Oblivious Cloud Storage. In *IEEE S&P*, 2013.

[100] San-Tsai Sun et al. The Devil is in the (Implementation) Details: an Empirical Analysis of OAuth SSO Systems. In *ACM CCS*, 2012.

[101] Anuchart Tassanaviboon and Guang Gong. OAuth and ABE based Authorization in Semi-trusted Cloud Computing. In *ACM Workshop on Data Intensive Computing in the Clouds*, 2011.

[102] Techical Report. Filecoin: A Cryptocurrency Operated File Network. http://filecoin.io/filecoin.pdf, 2014.

[103] Techical Report. Storj: A Peer-to-Peer Cloud Storage Network. https://storj.io/storj.pdf, 2016.

[104] Sam Thielman. Your Private Medical Data is for Sale and it is Driving a Business Worth Billions. The Guardian, Online: https://www.theguardian.com/technology/2017/jan/10/medical-data-multibillion-dollar-business-report-warns, 2018.

[105] Alin Tomescu and Srinivas Devadas. Catena: Efficient Non-Equivocation via Bitcoin. In *IEEE S&P*, 2017.

[106] Frank Wang et al. Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds. In *USENIX NSDI*, 2016.

[107] Xinlei Wang et al. Performance Evaluation of Attribute-based Encryption: Toward Data Privacy in the IoT. In *IEEE ICC*, 2014.

[108] Shucheng Yu et al. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *IEEE INFOCOM*, 2010.

[109] Thomas Zachariah et al. The Internet of Things has a Gateway Problem. In *HotMobile*, 2015.

[110] Ben Zhang et al. The Cloud is Not Enough: Saving IoT from the Cloud. In *USENIX HotCloud*, 2015.

[111] Wenting Zheng et al. Minicrypt: Reconciling Encryption and Compression for Big Data Stores. In *EuroSys*, 2015.

[112] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.

[113] Guy Zyskind et al. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *IEEE SPW*, 2015.

[114] Guy Zyskind et al. Enigma: Decentralized Computation Platform with Guaranteed Privacy. arXiv (whitepaper) http://www.enigma.co/enigma_full.pdf, 2015.

A Crypto-based Access Control

A.1 Dual-Key Regression

A key regression scheme [50] enables the efficient sharing of past keys. If an entity is in possession of the key regression state s_i, the entity can derive all keys k_j with $j \leq i$ for $i \in \{0,1,...,n\}$. However, the entity cannot infer any information about the keys k_l with $l > i$.

In our constructions, we make use of a Pseudorandom Generator (PRG) defined as follows.

Pseudorandom Generator (PRG). $G : \{0,1\}^n \rightarrow \{0,1\}^m$ is a pseudorandom generator, if $m > n$ and no probabilistic polynomial-time (PTT) distinguisher can distinguish the output $G(x)$ from a uniform choice $r \in \{0,1\}^m$ with non-negligible probability [53].

Using a pseudorandom generator $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+l}$, a client constructs a key regression scheme as follows. First, the client generates all the possible states s_i $0 \leq i \leq n$ in reverse order from an initially randomly chosen seed s_n. The seed s_{i-1} is computed as the first λ bits of the output of $G(s_i)$. To derive key k_i from the corresponding state s_i, the client computes $G(s_i)$ and takes the last l bits (i.e., applies the key derivation function). For sharing the keys to the i-th key, the client shares state s_i with the other entity. With state s_i, the entity can compute all pervious states s_x with $0 \leq x \leq i$ by applying the pseudorandom generator function G. Because of the one-way property of G the client is not able to compute or infer any information about s_{j+1} or any s_x with $x > j$. Since the entity owns states $\{s_0,...,s_i\}$, the entity can derive the keys $\{k_0,...,k_i\}$ with the key derivation function.

The key regression scheme based on a single series of states has the drawback that given the current state s_i an entity can compute all the previous states and keys. Hence, a client is not able to define a lower bound to restrict access on past keys (e.g., k_j, $low \leq j \leq cur$). To address this problem, we combine two sequences of states to derive the keys, similar to [26]. We denote the i-th state of the first sequence as $s_{1,i}$ and the second sequence as $s_{2,i}$ for $i \in \{0,\ldots,n\}$ where $n+1$ is the length of each sequence.

In the bootstrapping phase, the client generates the states $s_{1,i}$ as previously from a randomly chosen seed $s_{1,n}$ and computes the other states $s_{1,i-1} = MSB_\lambda(G(s_{1,i}))$ where MSB_λ denotes the mapping to the λ least significant bits of the input. The second sequence is generated from the opposite direction to enable a lower restriction level. The second sequence starts with the random seed $s_{2,0}$ and the corresponding next state is computed as $s_{2,i+1} = MSB_\lambda(G(s_{2,i}))$. To derive the key k_j where $j \in \{0,\ldots,n\}$, the states $s_{1,j}$ and $s_{2,j}$ serve as an input to the key derivation function which is defined as $k_j = LSB_l(G(s_{1,j} xor s_{2,j}))$ where LSB_l denotes the mapping to the l most significant bits of the input. If an entity is in possession of states $s_{1,i}$ and $s_{2,j}$ where $0 \leq j < i \leq n$, it can compute the states $\{s_{1,0}, s_{1,1}, \ldots, s_{1,i}\}$ and $\{s_{2,j}, s_{2,j+1}, \ldots, s_{2,n}\}$ with G. Since pairs of states are required for deriving the keys, the entity can only compute the keys for which it possesses the corresponding state pairs. Considering the states computed above, the entity knows the state pairs $\{(s_{1,j}, s_{2,j}), (s_{1,j+1}, s_{2,j+1}) \ldots (s_{1,i}, s_{2,i})\}$ and can compute $\{k_j, k_{j+1}, ..., k_i\}$ but no other keys. Therefore, dual key regression can restrict access based on ranges of keys by sharing the corresponding state of each state sequence.

A.2 Key Derivation Tree

Droplet's key-derivation tree is based on the Goldreich-Goldwasser-Micali (GGM) construction [53]. The GGM construction is a binary tree of height h where each node contains a unique binary label v and an associated key k'. The label of a node encodes the path from the root to the current node where the label of the left child is encoded as $v||0$ and the right child as $v||1$. The key of a node is computed based on the label $v = v_1, v_2, .., v_l$ as $G_{v_l}(...(G_{v_2}(G_{v_1}(k'))))$ where $G(k') = G_0(k')||G_1(k')$ is a PRG. The GGM tree is a construction that builds a pseudorandom function (PRF) [53]. The PRF T takes as an input a master key k and a leaf label v and outputs a key $T(k,v) = k_v$. In GGM, k is the key of the root node, v the label of a leaf node, and the output k_v the key associated with the leaf node with label v. In Droplet, the keystream for encryption is derived using T, which leads to the keystream $\{T(k,0), T(k,1), ..., T(k,2^h-1)\}$.

To enable access control on the output keys, T offers the following additional algorithms:

- **T.constrain**(k,S) takes as an input the master secret of the root node k and a set of labels of leaf nodes S. The algorithm outputs a set of constrained keys k_S that contains the keys from the inner-nodes. These inner-node keys are selected so that they facilitate the computation of the keys of the nodes with labels in S but no other leaf node keys.
- **T.eval**(k_S, v) takes as an input the set of constrained keys k_S and a label v of a leaf node. The algorithm outputs the leaf node key k_v if $v \in S$ else outputs $\perp$.

With the two additional algorithms for access control, the construction is a constrained PRF [24]. For the detailed security analysis, we refer to [24].

B Dual-Key Stealth Addresses

To protect the privacy of access permissions, Droplet employs dual-key stealth addresses [36]. Let us consider the case of a data owner Alice giving access permission to a subscriber Bob. Bob has initially constructed and published his dual public keys (B, V): $B = bG$ and $V = vG$, with G as the elliptic curve group generator and the private keys b and v. Alice constructs a new address P using Bob's stealth addresses by using a hashing function H, and generating a random salt r:

$$P = H(rV)G + B \tag{2}$$

Alice embeds the tuple (P,R) in the access permissions, with $R = rG$ (r is protected and not recoverable from R). Only Bob can claim the address P, as he is the only one capable of recovering the private key x, such that $P = xG$, as follows:

$$x := H(vR) + b \tag{3}$$

Hence, he can prove (e.g., with a signature) to the storage node that he is the rightful principal. Note that guessing x, given G and P, is equivalent to solving the elliptic curve discrete log problem, which is computationally intractable for large integers. The correctness of x from Equation 3 can be shown as:

$$\begin{aligned} xG = (H(vR)+b)G = H(vR)G + bG = \\ H(vrG)G + B = H(rvG)G + B = H(rV)G + B = P \end{aligned} \tag{4}$$

Except Alice and Bob no other party can learn that P is associated with Bob's stealth addresses. Moreover, the randomness r in the address generation ensures the uniqueness and unlinkability of new addresses. Bob discloses the private viewer key v to the auditor to enable an authorized auditor to audit the sharing. The auditor can verify the mapping of the tuple (P,R) to Bob's main key address B as:

$$\begin{aligned} P - H(vR)G = P - H(vrG)G = \\ H(rV)G + B - H(rV)G = B \end{aligned} \tag{5}$$

Note that the auditor is cryptographically prevented from using v to compute Bob's private key x.

C Security Guarantees

Droplet consists of the following entities: *the data owner, data producer, data consumer, storage node, authorization agent, and decentralized authorization log (a public blockchain)*, as described in §2. Under the trust assumptions laid out in §2.2, we now elaborate on the security guarantees of Droplet.

Guarantee 1.1 *An* `Adv` *is not able to access or manipulate data chunks except by compromising data producers/consumers.* Droplet ensures this by end-to-end encryption. Each data chunk is encrypted with a fresh key (§A.2) on the client-side with AES in GCM mode, which is an authenticated block-cipher, providing confidentiality, integrity, and authenticity guarantees:

$$\begin{aligned} \text{AES-GCM}.Enc(K_i, IV, M_i) \rightarrow C_i \\ \text{AES-GCM}.Dec(K_i, IV, C_i) \rightarrow M_i \end{aligned} \tag{6}$$

Given the $i-th$ key, IV, and $i-th$ message, it computes the $i-th$ ciphertext. Given the $i-th$ key, IV, and $i-th$ ciphertext, it computes the $i-th$ message or fails with an error.

For proof of ownership, each chunk is digitally signed:

$$\begin{aligned} \text{ECDSA.KeyGen}(curve) \rightarrow (PK_{device_{id}}, SK_{device_{id}}) \\ \text{ECDSA.Sign}(SK_{device_{id}}, C_i) \rightarrow Sig_{C_i} \\ \text{ECDSA.Verify}(PK_{device_{id}}, C_i, Sig_{C_i}) \rightarrow (\mathsf{true}, \mathsf{false}) \end{aligned} \tag{7}$$

After generating the per device private and public ECDSA key pair , Droplet signs the encrypted message and generates the signature, which can be verified given the public key and the ciphertext. As long as the `Adv` does not compromise the private key, a polynomial-time `Adv` cannot forge the signature.

Guarantee 1.2 *For streams with strong immutability requirements, an* `Adv` *is not able to modify the stream without compromising the authorization log.* The `Adv` must control a large threshold of nodes/computing power to compromise the authorization log to change a committed hash link.

Guarantee 2.1. *If an* `Adv` *compromises data consumers that had access to intervals of a data stream, the* `Adv` *is not able to access any other data than the data the compromised data consumers previously had access.* Each data chunk in a stream is encrypted with a fresh key K_i. If an `Adv` compromises a data consumer, the `Adv` gains access to the subset of the decryption keys which the consumer had access. Hence, it can only decrypt the data chunks where it possesses the decryption keys. In Droplet, the keys for encryption are derived with a PRF that is constructed from the key derivation tree T (§A.2). With the master secret k, the i-th key is derived as $T(k,i) \rightarrow K_i$. Instead of sharing the keys for range $K_i,...,K_j$ individually with a data consumer, Droplet shares constrained keys $T.constrain(k, S := \{i,..,j\}) \rightarrow k_S$ (i.e., inner-nodes of the tree). $K_i,...,K_j$ can be derived as $T.eval(k_S,i),..,T.eval(k_S,j)$ but no other keys. This guarantees that an `Adv` in possession of k_S can derive keys outside of the interval $K_i,...,K_j$ with negligible probability.

Guarantee 2.2. *In addition to Guarantee 2.1, an* `Adv` *in control of a compromised data consumer can access data that was previously revoked, if the* `Adv` *controls the respective storage node or authorization agent.* Beyond end-to-end encryption, the storage node enforces access based on the authorization log. To retrieve data after revoked access, the `Adv` must compromise the storage node or authorization agent.

Guarantee 3.1. *An* `Adv` *cannot link data permissions of a data consumer from the publicly accessible authorization log unless the* `Adv` *compromises the audit key of the data owner.* Dual-key stealth addresses hide any linkability between the consumer identities included in the access permissions (§B). A data consumer proves legitimate access to the storage node via a zero-knowledge proof, where the data consumer proves it controls the private key associated with the public key, whose hash digest is included in the access permission.

Guarantee 3.2. *An* `Adv` *compromising the authorization agent cannot compromise data confidentially nor break the non-linkability from Guarantee 3.1, but it can prevent data availability.* An `Adv` can maliciously give access to encrypted data, which does not impact data confidentiality as data is end-to-end encrypted. An `Adv` does not learn anything about the data consumer from their request to access their data, other than that they control the private key corresponding to the public key included in the access permission.