Sylo Protocol: Secure Group Messaging

Felix Schlitter John Carlo San Pedro Paul Freeman Callum Lowcay

March 4, 2020

Abstract

Privacy concerns, data collection, and hacking have driven demand for secure decentralised applications. Meanwhile, centralised social apps, such as WhatsApp, Instagram, and Facebook Messenger are some of the most used apps in the world. Development of social decentralised applications, those which include a feature set comparable to their centralised counterparts, is at the forefront of a larger decentralisation movement, but comes with it a unique set of challenges. We present the Sylo Protocol as a novel, decentralised, peer-to-peer group messaging protocol upon which powerful decentralised applications are built. The protocol provides the security necessary to maintain shared group state through the use of eventually-consistent operation logs.

1 Introduction

Messaging is a fundamental component of our growing digital landscape and the security of our communication is of vital importance. Secure messaging protocols (SMPs) were created to help ensure messaging is performed consistently and securely. Although a relatively recent technology, SMPs are fundamental to the online experience of today and have already established a strong history[19].

For many years, reliance on central authorities to oversee secure communication was an acceptable standard practice. A generation of applications has been built on the centralised model and when user data is sold or stolen, it is often tolerated as a flaw in an otherwise invaluable service. Today, many are scrutinising what is actually being done with these vast data stores and how safe it is in the hands of a few titan technology companies.

Research into decentralised technologies has provided an alternative to centralising user data. To serve a future where users are in control of their data, we require a new generation of decentralised applications built on a peer-to-peer, secure messaging protocol. Additionally, the protocol must provide demonstrated service at scale, with a user experience that minimises compromise.

1.1 Decentralised

If users are to have any semblance of authority over their data, it seems mandatory to provide users with a messaging protocol built on decentralisation and peer-to-peer (P2P) technology. Many of today's most successful protocols are still reliant on the presence of servers, operated by untrusted sources. End-to-end encryption (E2EE) using Signal, or a similar protocol, solves the problem of protecting the data within messages, but the flow of data across these centralised servers is full of metadata and the field of data mining is quickly learning how to extract valuable personal data from this unprotected resource.

Centralisation also restricts users' ability to *self organise* and create their own networks to circumvent disasters or censorship. When applications are reliant on a central authority, this provides an obvious attack point for bad actors wishing to disrupt services. By decentralising, attacks can be made more costly (or disincentivised completely), and a service's ability to survive catastrophic circumstances can be greatly improved.

1.2 Peer-to-Peer

A resilient group messaging protocol would benefit from the ability to send messages directly to intended peers, without the danger of being attacked

by entities along the travel path. P2P communication is a proven method for combating these attacks.

In P2P communication, data is sent directly to another peer by connecting via their Internet Protocol (IP) address. Combined with E2EE, this process is a well establish method for data transport. A P2P network can provide resiliency to many network attacks, as it does not inherently depend on intermediaries to perform messaging.

1.3 Scale and User Experience

Arguably one of the most important features of a group messaging protocol is its ability to survive as a messaging app in the real world. Consumers demand decentralised applications with a comparable user experience to their centralised counterparts. Sacrificing user experience for improved privacy may have a niche market, but the bulk of the market using the top social media apps is not willing to trade user experience for privacy.

For example, a messaging app built entirely on a P2P network may place an upper bound on the overall user experience. To rise above this boundary, to an experience that rivals centralised messaging apps, there is opportunity for an alternative networks to facilitate decentralised communication.

A valuable decentralised group messaging protocol is therefore one which not only achieves the goal discussed so far, but is able to provide it at scale, and with the user experience consumers have grown to expect.

1.4 Overview

The Sylo Protocol is a group messaging protocol targeted at a decentralised peer-to-peer network. Group state and the group's operation log is managed by members of the group without relying on external entities to support state replication or synchronisation. Messages are distributed directly over P2P connections and can also be supported by decentralised networks providing additional services. Operation logs are eventually consistent with no direct requirement for a consensus mechanism, such as a blockchain. Peers are assumed to be online infrequently and only for short durations, as is typical for most mobile applications. The protocol is highly tolerant of frequent asynchronous and/or concurrent operations while still being aware of, and minimising, the overall attack surface.

The protocol has already been demonstrated at scale, supporting hundreds of thousands of users. The Sylo mobile group messaging and smart wallet application has been well adopted and provides a similar experience to other leading social media apps.

2 Background

Historically, messaging has mostly been performed using instant messaging, short message service (SMS), and messaging apps. Today, the space is mostly dominated by messaging apps. Typically, messaging apps are designed for a specific messaging protocol. Many protocols for messaging are currently available or under development that attempt to address the privacy, security, interoperability, and robustness concerns associated with centralised messaging services. This section offers a brief review of recent developments in this space.

2.1 Encryption

One approach to improving privacy and security in messaging protocols is to employ end-to-end encryption. E2EE schemes for messaging protocols are expected to have three properties: confidentiality, forward secrecy, and repudiability [3].

2.1.1 Confidentiality

Only the intended recipient of a message should be able to read it. Confidentiality is achieved by encrypting messages with keys that are known to the sender and recipient, but not to the provider of the messaging app.

2.1.2 Forward Secrecy

Secret keys can be lost or compromised, especially in large groups. Strong encryption protocols should be resistant to compromised secret keys and, ideally, recover to a secure state. This ensures that messaging can be secure even when an encrypted conversation continues for a long time. Common encryption protocols for email such as Pretty Good Privacy (PGP) [4] do not have this property since they use long-lived encryption keys. If an attacker

can compromise the sender's PGP encryption key, then they can decrypt all of the senders past and future messages.

2.1.3 Repudiability

The sender of a message must be able to prove their identity to the recipient, and the recipient must be able to prove that the message was not altered during transit. Repudiability ensure that the sender's identity is verifiable only to the recipient [3].

One *non-repudiable* authentication method is to provide a digital signature with each message, as in PGP. While this allows a recipient to verify the identity of a sender, it also exposes the sender's identity to any third party that can intercept the messages, creating a privacy risk for the messaging protocol.

The properties of forward secrecy and repudiability can be achieved by encrypting messages with short-lived keys derived through a suitable key agreement protocol. The Signal protocol [10][9] is a popular implementation of this idea.

The Signal protocol is used by several centralised messaging apps including the Signal application itself, WhatsApp [23], Facebook Messenger [11], and Skype [18]. Telegram, another centralised messaging app, uses a home-grown E2EE protocol called MTProto [12].

While end-to-end encryption is necessary to protect users from eavesdropping, it does not prevent other attacks. Accounts on centralised services are vulnerable to hijacking. Additionally, centralised services may suffer denial of service attacks, outages due to technical problems, or they may be blocked outright. When a centralised service is compromised, shut down, or otherwise becomes unavailable, then users will no longer be able to communicate securely through that service. Network effects associated with centralised services make it difficult for users to migrate to other services, increasing the risk to users of losing connectivity with their contacts.

2.2 Peer-to-Peer Communication

Peer-to-peer systems avoid centralisation by allowing users to communicate directly with each other. A server is not required to mediate the exchange. However, when two peers are communicating over the Internet, there may be any number of routers or other middleware devices along the route between the two peers. End-to-end encryption with forward secrecy and repudiability is thus required to ensure privacy in P2P systems.

There are many technical challenges associated with P2P systems. Most networks use Network Address Translation (NAT) which makes it difficult for devices on those networks to accept incoming connections. Most consumer devices are assigned network addresses dynamically, so a mechanism is required to locate peers and confirm their identity. Efficient broadcasting of messages (required for group messaging) is difficult without a network of servers. These challenges have slowed the development of P2P applications.

Recently, however, there has been a resurgence of interest in P2P systems. Algorithms for peer discovery, routing, and NAT traversal are now available in open source libraries such as libp2p. Applications built on this technology include Textile for social data storage, and Berty, a P2P secure messaging protocol built on IPFS [2].

Blockchains such as Ethereum are synchronised peer-to-peer, and there has been some interest in reusing these protocols for other applications. Status, for example, uses Ethereum's whisper protocol. The whisper protocol is vague and poorly specified [5]. It appears to work by encrypting messages so that only the intended recipients can recognise and decode those messages, then distributing those messages to every peer on the network. Bloom filters are employed in an attempt to reduce the bandwidth required to distribute all these messages, but there is evidence that this protocol will not scale to more than a few thousand users [22].

Authenticating users' identities is a particularly important issue in peer-to-peer systems. In P2P systems, unlike most centralised services, there is no central authority to authenticate users' identities. The usual solution is for users to directly authenticate each other using public key cryptography. This approach is implemented in Scuttlebutt, a protocol for asynchronous social networking applications. Berty uses a similar scheme.

Peer-to-peer authentication requires users to manage their own identity keys. If a user's identity keys are lost or compromised, their identity is also lost, without any possibility of recovery. This is in contrast to a centralised service where a user

may be able to recover a lost or stolen account by providing identifying information to their service provider.

2.3 Federation

Federated networks are a middle-ground between centralised services and peer-to-peer networks. Unlike peer-to-peer networks, federated networks *do* make a distinction between client and server devices. Each user must create an account on a server. When a user wants to send a message to another user, their client device first sends the message to the server that hosts their account, then that server forwards the message on to the recipient's client device. In a federated system, the sender and the recipient of a message need not have accounts on the same server. The servers exchange message traffic with each other through open protocols. Anyone can set up a new server and add it to the federation.

A familiar example of a federated network is email. A user can create an email address with any provider, then send emails to any other email user even if the recipient is using a different email provider.

Two prominent federated networks for messaging are Extensible Messaging and Presence Protocol (XMPP) [14] and the more recent Matrix [1].

XMPP is a minimal protocol that specifies how to establish, authenticate, and encrypt streams for the near-real-time exchange of Extensible Markup Language (XML) data [14]. Messaging functionality is provided as an extension [15]. XMPP also has extensions for group conversations [16] and E2EE [20].

Matrix, on the other hand, specifies a protocol for securely replicating sequences of events between different servers. In this paradigm, a message is an event that needs to be replicated from the sender's server to the recipient's server. Group conversations are supported by replicating messages to all of the servers for all of the users in the chat group. Matrix supports E2EE through their Megolm [6] protocol, which is derived from an early version of the Signal protocol.

Federated networks provide the interoperability that is lacking in centralised services. They also give users the freedom to host their identity information with a provider that they trust, rather than forcing all users to have an account with the same service provider. Federated networks also avoid the technical challenges of peer-to-peer applications. It is easier to develop feature-rich client applications for federated networks: examples include Riot for Matrix and Conversations for XMPP.

However, federation does not solve all of the problems of centralised instant messaging. There is evidence that federated systems tend towards centralisation over time [13]. Gmail has become by far the largest provider for personal email addresses, for example. Another danger is that providers in a federated network may opt-out of the federation when they become large enough to be sustained by their own network effects. Facebook, Google, and WhatsApp all supported XMPP federation in earlier versions of their instant messaging services, but have since dropped it.

3 Goals

3.1 Decentralised

Sylo Protocol does not rely on the existence of any centralised infrastructure to perform secure group messaging. All required operations can be performed in a P2P setting. Therefore, the only requirement of Sylo Protocol is a network capable of delivering P2P messages. The Sylo Network[21] is an example of a decentralised network providing P2P message delivery.

3.2 Secure and Confidential

Sylo Protocol aims to provide a high level of security and confidentiality by purposely removing itself from the equation. Since everything happens in a peer-to-peer setting, state and state transitions are propagated autonomously through the Sylo Network. Aside from removing the intermediaries, however, Sylo Protocol also goes to great length to protect communications and profiles in the unlikely event that a single message or session key gets compromised by responsibly securing all communications using Signal Protocol.

3.3 Highly Performant

In order to be adopted, decentralised messaging apps need to provide a user experience that meets

or exceeds existing centralised competitors. Centralised apps can leverage servers to perform expensive computations, but in decentralised setting, messaging apps must perform the same tasks on the clients device, making performance a first-class feature.

It is therefore expected that the Sylo Protocol will consume a reasonable amount of CPU, memory, and bandwidth on the user's device.

3.4 Feature Rich

As with application performance, the Sylo Protocol will only see wide adoption if the experience meets or exceeds what is currently available and in use by the target audience. Hence, Sylo Protocol aims to provide a feature set that is capable of matching user expectations for a centralised group secure messaging protocol, while providing a decentralised, P2P experience.

3.5 Fault Tolerant

By the very nature of how software operates and integrates, failure is inevitable. In fact, shifting business logic from central servers onto clients themselves only pronounces the need for robust and responsible fault management. Recovering from disaster is not the responsibility of a dedicated operations team, but that of the very application itself.

Fault-tolerance and recovery must be a first-class feature that is deeply ingrained in all technical design decisions within the Sylo Protocol.

3.6 Resilient

In the spirit of being generally fault-tolerant, it is assumed that the messaging app itself will be operating infrequently and only for short durations. Similarly, network connections to peers are assumed to be slow, rarely available, and highly volatile. Rather than waiting for network connections, state changing operations should be scheduled and performed when possible during network uptime.

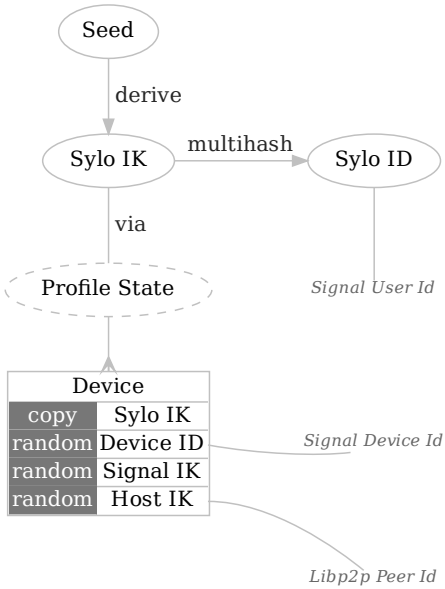

Figure 1: The Sylo Identity

4 Sylo Users

4.1 Identity

The identity used by the Sylo Protocol is a public/private key-pair derived from a mnemonic seed phrase. It is referred to as the *Sylo Identity Key (IK)*.

The Sylo IK is the single most important piece of information a user must safeguard from both being lost and being compromised. A lost Sylo IK is irrecoverable and a compromised Sylo IK is irrevocable. The derived Sylo IK should never leave the user's device under normal operation. When not in use, the Sylo IK should be symmetrically encrypted, and stored using the target platform's most appropriate means of doing so.

It is advised to store the mnemonic seed phrase underpinning the Sylo IK on an alternative medium altogether, as it may be required when configuring an additional device.

4.1.1 Details

The Sylo IK employs a Edwards-curve Digital Signature Algorithm (EdDSA) scheme with Ed25519[1] parameters [7]. Ed25519 has the following attractive properties:

High performance The signature scheme uses a 32 byte public key and a 64 byte signature, leading to high performance across a variety of platforms. This makes the scheme suitable to work on mobile and similarly constrained platforms.

Asymmetric encryption Key-pairs generated for Ed25519 can conveniently be converted for use with X25519, which is a Elliptical Curve Diffie-Hellman scheme, employed by the Sylo Protocol for asymmetric encryption.

4.2 Profile

User profiles are either private or public, and users will typically have one of each. At the application-level, the profiles may be extended to contain application or user-defined values.

4.2.1 Public Profiles

Public profiles serve as a mechanism to actively advertise and learn self-proclaimed information about other peers. The application is most responsible for the public profiles, and should determine policies regarding the handling of keys for any specific application-level concern. The operations log associated with the public profile is discussed further in Section 5.6.1.

4.2.2 Private Profile

Users maintain their own private profile as a way to synchronise profile data between their devices. The private profile is a commutative replicated data type (CmRDT) operation log. All keys and values in the log are symmetrically encrypted using the Sylo IK_{SK}. The Sylo Protocol uses the private profile internally to facilitate the automatic synchronisation of group membership, device ownership, and for managing contacts. The operation log associated with the private profile is discussed further in Section 5.6.2.

Replication Replication of the private profile is challenging, since users will not regularly be expected to be using multiple devices simultaneously. Additionally, there is no direct incentive for peers on the network to help propagate a private profile. In a P2P setting, users may need to manually bring multiple devices onto the network to initiate the sync. Alternatively, users could rely on an incentivised service built into the Sylo Network.

4.3 Device

Users may operate one or more devices. Ownership over a device is claimed via entries maintained in the user's profile log.

Operating a device provides the user with the following capabilities:

Connecting to peers on the Sylo Network A device maps onto a Host Identity Key. P2P communication within the Sylo Network happens exclusively in terms of these host keys. The public key acts as a unique Device ID. The Host IK not only provides a means for authentication, but also plays a critical role in establishing a shared secret used to provide secure P2P connections.

End-to-end encryption A device maps onto a Signal IK used for end-to-end encryption. Though communication is primarily peer to peer, content may be stored outside of the user's device for asynchronous retrieval. This leads to a need for an additional layer of encryption. The Signal IK is generated randomly during device setup. The Sylo IK, in combination with the Device ID, derives a valid Signal Protocol address which is used for establishing Signal Protocol sessions between peers' devices.

4.3.1 Device Initialisation

A device is comprised of the following components:

The Host Identity Key A randomly generated key-pair used for P2P communication on the Sylo Network.

[1] Ed25519 is an EdDSA signature scheme using SHA-512 (SHA-2) and Curve25519

The Signal Identity Key A randomly generated key-pair used as the E2EE Signal Protocol identity key for asynchronous exchange of information.

The Device ID An identifier derived from the public key of the Host. Identifies a particular device owned by a user.

Registering a device is performed by creating the necessary IKs and then appending an entry in the operation log for the peer's private profile. The operation log entry advertises public key values as a device operated under the user's Sylo IK.

4.3.2 Device Security

A device represents any unique instance of a user on the Sylo Network. For example, a user could operate a mobile and desktop client simultaneously. For improved security, it's important to use device specific keys for the Host IK and the Signal IK, and for these keys to be distinct from the global Sylo IK. This ensures that compromising a device's keys will isolate the damage to that particular device.

Similar to the Sylo IK, it is the application developer's responsibility to store these keys securely.

5 Operation Logs

A fundamental problem that exists in distributed systems is handling mutations on some shared state, and synchronising that state across each actor.

Consider a simple example where Alice and Bob wish to add and remove items from the same list. It's easy to understand that the state of the final list will be dependent on the order of the operations applied. For traditional applications, a centralised actor, such as a server, is responsible for coordinating these mutations, essentially acting as a source of canonical truth for the final state. In the distributed setting, Alice and Bob each maintain their own view of the list, and need to exchange messages frequently to improve state consistency. Not only would this be untenable in the decentralised, P2P environment of the Sylo Network, but an ad-hoc approach to synchronisation could lead to conflicts in state. This severely impacts the user experience, and potentially exposes opportunities for misplay.

In order to facilitate rich application features, such as user profiles or groups, the Sylo Protocol employs the use of operation logs.

5.1 Concepts

Before describing the operation log in detail, it will be helpful to define some of the common terminology.

5.1.1 Directed Acyclic Graph

Operation logs can be mapped to a directed acyclic graph (DAG), where the nodes contain the data of operations and the referenced operations map to outgoing edges. Operations can only reference existing operations, so the graph is directed. Operations cannot reference themselves, so the graph is also acyclic.

Once an operation has been appended to an operation log, it is immutable. The data within the operation cannot be changed and the references to other operations cannot be altered. An immutable sub-DAG of operations extending to the root operation can be defined for any operation node by following edges recursively until the root operation is reached.

5.1.2 Lamport Clock

A Lamport clock[8] is used to provide additional ordering requirements to the operation log. Peers order operations by assigning each new operation with a Lamport clock values stepped according to the maximum clock value of all nodes pointed to by outgoing edges.

Operations with improperly stepped values should be rejected by peers. We assume that the first operation in the log will have a clock value of 0. If the latest operation in the log is at clock n, we can infer that there should exist at least one operation with every clock value $0 \leq c \leq n$ in the operation log.

5.1.3 Headset

The *headset* of an operation log is the *leaf set*[2] of operations in the current DAG. The operations in

[2]The leaf set of a DAG is the set of nodes without any incoming edges.

the headset are the most recent operations known within the current state of the operation log and ordering of the operations in the headset cannot be performed[3].

5.2 Operation Log Model

An operation log is modeled after a conflict-free replicated data type (CRDT). CRDTs are an approach to replicating state amongst distributed peers that guarantees strong eventual consistency (SEC)[4], avoiding the need for frequent remote synchronisation [17]. In particular, operation logs are a CmRDT, with the main components to consider being:

- A state, or data structure, which is to be replicated and consistent amongst each peer.
- A defined set of mutations on the state, each with its own parameters. Mutations must be *commutative*. This ensures that the order in which they are applied will have no bearing on the final state.
- A function to apply a mutation given an existing state, returning a new state. This function should be *side-effect free*, only considering the current state and the parameters given to it.

Having a commutative set of mutations allows users to make changes to their local state without needing to be in constant coordination with other peers. This is critical for the P2P Sylo Network, where bandwidth usage should be minimal, and peers are expected to frequently drop their connections. Modeling the replicated state as a CRDTs allows users to converge on the same state asynchronously. With the CmRDT properties in mind, an operation log can be defined to be a

- A directed acyclic graph, where each vertex represents an operation from an operation log scheme.
- An append function, acting as the sole mutation on the CmRDT. The parameter to the append function must be a valid operation from the operation log scheme.
- An insertion function, using the defined links in the operation to appropriately insert it into the DAG.

Any two peers that have inserted the same set of operations, regardless of order, will form the same DAG. The reachability relationship of the DAG provides a partial ordering of the operations. With the addition of a conflict resolution strategy, concurrent operations can be resolved to achieve a total ordering of the operations.

An application can leverage the properties of the conflict-free replicated DAG to deterministically reduce the operations to a secondary state. By having an operation log that can be replicated in such a way amongst peers, the Sylo Protocol is able to provide custom user profiles and group membership in a distributed setting.

5.3 Transformations

The operation log is "append-only", and therefore has an *append* function. An *insert* function is also provided. The two functions differ in their restrictions. `Append` should be applied locally by a peer to add a new operation, with links and a clock value, to the operation log. `Insert` should be used to apply *patches*, which are made up of existing operations, and distributed between peers.

5.3.1 Append

The *append* function is used locally by a peer to create and add new operations to the operation log. When appending an operation, links are create from the new operation to all the operations in the peer's current headset. The Lamport clock value for the new operation is obtained by stepping the maximum Lamport clock value in the current headset. After an append is performed, the new headset should contain *only* the newly appended operation.

Peers provide new operations to their peers with a *patch*, which can be applied using the *insert* function.

[3] An ordering may still be possible using information contained within the node, but this is left to the application.

[4] Strong eventual consistency informally guarantees that, given enough time, any two nodes will converge at the same set of updates and, regardless of the order the updates are received, both peers will compute the same state.

5.3.2 Patches and Insertions

Patches are the primary mechanism for distributing operations between peers. When patches are received by peers, the *insert* function is used to apply each operation in the patch to the peer's current operation log and may provide new operations. Patches are discussed further in Section 5.5.2.

The purpose of the insert function is distinct from the append function. The insert function *also* adds operations to the operation log, but the operations themselves have already been created, and have preexisting outgoing edges and Lamport clock values. Appending is for something that *happens*, and inserting is for something that *happened*.

Peers should validate new operations received in a patch. While it may be difficult to determine a peer's headset at the time an operation was appended, it is possible to validate the Lamport clock was stepped appropriately for the set of operations to which the appended operation is linked.

Where possible, operation validation is performed within Sylo Protocol, but it may be appropriate for applications to perform more extensive validation.

5.4 Operations

An operation object consists of:

- **Links** to parent operations.
- A properly stepped Lamport **clock**.
- Application defined **data**.
- A digital **signature**, derived from the links, clock, and data.

Signing operations with a user's public key can uniquely identify an operation within a DAG, provided a user does not concurrently appending the same data. This signature method is not vulnerable to replay attacks, as inserting the same operation has no effect on the DAG.

The above properties are then hashed together to create an *Operation ID*. Deriving an identifier from the operation contents is known as *content addressing* and plays an important part in securely retrieving operations from untrusted sources. The content address itself is a simple validation of data integrity.

The first operation in an operation log is the *root* operation. It is the only operation in the operation log that does not contain links to any prior operations. Following the creation of the root operation, when a new operation is appended to an operation log, it must include links to all operations in the current headset.

5.5 Synchronisation

5.5.1 Synchronisation Points

The operation log must be shared between peers to ensure the requirement of eventual consistency is met. Before the exchange of operations can occur, it is useful to designate specific points in the operation log as *synchronisation points*. Sylo Protocol uses *clock sets* as synchronisation points. A clock set is a set of operations in the operation log with the same Lamport clock value.

Algorithm 1 Calculate the synchronisation points for n

Require: $n \geq 0$

procedure SYNCPOINTS(n)
  $s \leftarrow \{0\}$
  if $n \neq 0$ **then**
    $x \leftarrow 1$
    while $x \leq n$ **do**
      $y \leftarrow \lfloor n/x \rfloor$
      if $y \bmod 2 = 1$ **then**
        $s \leftarrow s \cup \{x \cdot y\}$
      else
        $s \leftarrow s \cup \{n - x - n \bmod x\}$
      end if
      $x \leftarrow 2x$
    end while
  end if
  return s
end procedure

Synchronisation points must be created for deterministic clock values. Using deterministic synchronisation points allows peers to arrive at the same synchronisation points independently, leading to efficient communicate regarding the shared operation log state. When two peers' clock sets differ, they will be able to request a patch from the other peer containing the operations needed. To support scaling, the number of synchronisation

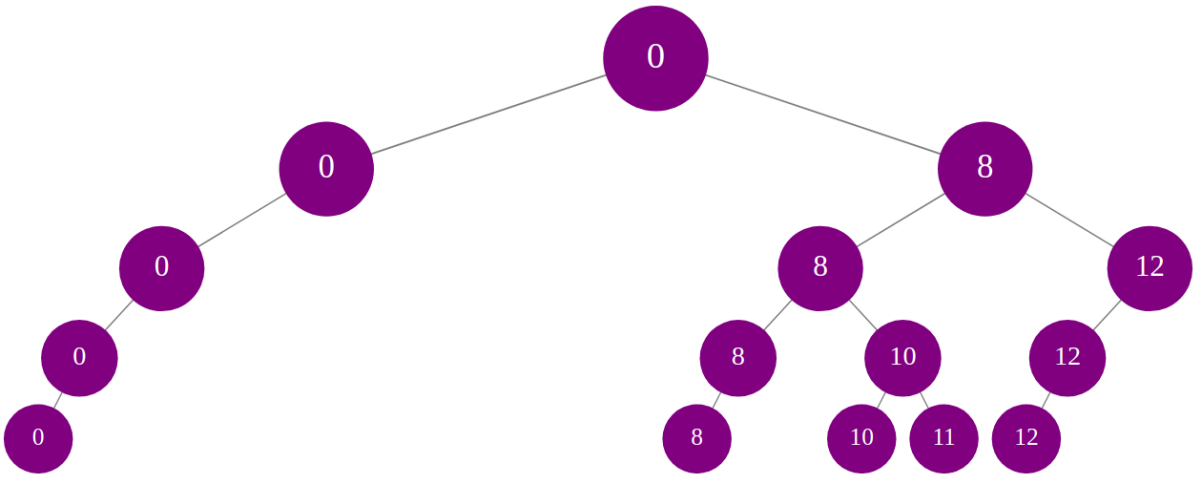

Figure 2: The binary representation of 12 is 1100. Synchronisation points are computed by descending a binary tree. The resulting set of synchronisation points for 12 is $\{0, 8, 10, 11, 12\}$.

points should grow at $O(\log n)$, where n is the number of operations in the operation log. Algorithm 1 demonstrates how synchronisation points are calculated. A peer need only pass the clock of the most recent operation into the algorithm and a set of synchronisation points is returned. Figure 2 shows the binary tree used for computing synchronisation points for the value 12.

Conceptually, the algorithm descends a binary tree using specific rules to arrive at synchronisation points. It computes a synchronisation point for each bit in the binary representation of the input. Zero is also added to all results as a "worst case" synchronisation point.

For a given input, n, the algorithm is designed against the following properties:

- the cardinality of the set of synchronisation points is $\lceil \log n \rceil + 1$
- 0 and n are both included in the set of synchronisation points
- for each synchronisation point, p, it holds that $0 \leq p \leq n$
- a peers who is n ops "behind" another peer is guaranteed to share a sync point within $2n$ clocks of their current clock value

The last property allows us to place an upper bound on the number of clock sets a peer needs to retrieve to be in sync. An example is provided in Figure 3.

5.5.2 Patches

Operations are sent between peers using *patches*. A patch is a sub-set of operations in an operation log and can be applied by peers using the insert function. Peers generate patches based on the synchronisation points of their operation log.

Consider an operation log where the most recent operation has Lamport clock value 63. Using Algorithm 1, synchronisation points can be calculated:

$$\texttt{syncPoints}(63) = \{0, 32, 48, 56, 60, 62, 63\}$$

The synchronisation points are then used to split the entire operation log into patches.

Patch	Hash
0-32	2824949852579646517
32-48	6994901160385823322
48-56	1659644955901676156
56-60	3952200542213080601
60-62	8193115102924706494
62-63	4638107771511484556

Patch hashes are created from the operations making up the clock set of the synchronisations point. Hashes are created for each patch to allow other peers to quickly validate operation log history.

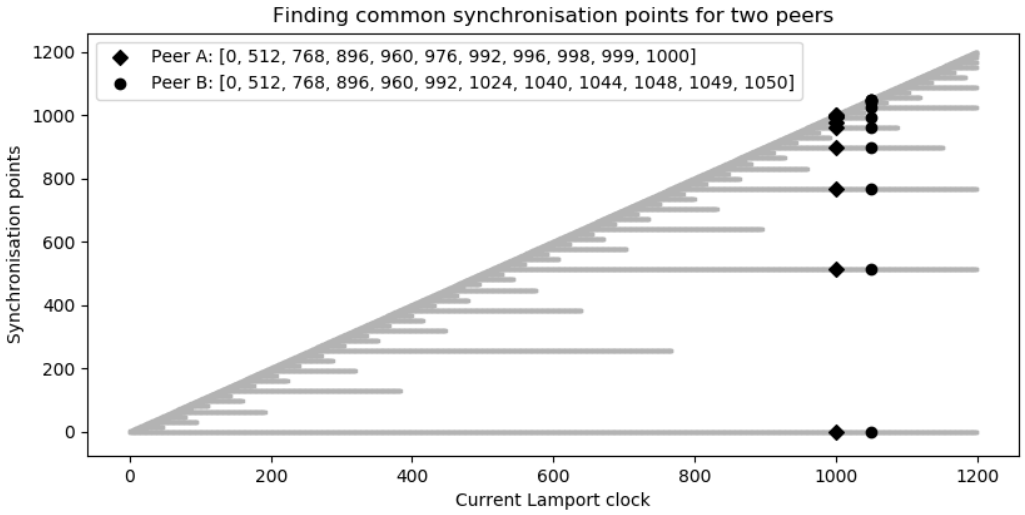

Figure 3: Peer A is at Lamport clock 1000 and Peer B is 50 operations ahead of them, at Lamport clock 1050. The `syncPoints` algorithm guarantees that they will share a synchronisation point between 900 and 1000. In this case, the common synchronisation point happens to be at 992.

Consider another peer, at Lamport clock 66.

Patch	Hash
0-32	2824949852579646517
32-48	6994901160385823322
48-56	1659644955901676156
56-60	3952200542213080601
60-64	7261960480495164932
64-65	5269001705850208932
65-66	3579327145562378258

Using the patch hashes, it can be verified that the operation log for both peers is identical between clock sets 0 and their shared synchronisation point at clock set 60. Using this information, the peer could acquire only the patches for clock sets 60-64, 64-65, and 65-66 and be assured of having all missing operations known to the other peer.

5.6 Operation Log Schemes

The Sylo Protocol defines operation logs with a set of predefined operations. Appending certain operations may only be valid when pointing to a sub-DAG containing a prerequisite operations.

The set of operations and validation criteria that define an operation log is referred to as the *scheme*. The Sylo Protocol currently defines three operation log schemes: a public profile scheme, a private profile scheme, and a group scheme.

5.6.1 Public Profile Scheme

Operations supported:

- add device
- remove device
- update metadata
- remove metadata

The public profile scheme defines an operation log for peer data that should be made available to contacts. It is specifically designed to store device information for the peer, but can also store generic metadata.

The scheme defines operations for adding and removing devices, via `AddDevice` and `RemoveDevice`. Metadata is updated using `UpdateMeta` or removed using `RemoveMeta`.

5.6.2 Private Profile Scheme

Operations supported:

- add device
- remove device
- update metadata

- remove metadata
- add contact
- delete contact
- edit contact
- add group
- remove group
- add invitation
- remove invitation

The private profile scheme is intended for use across a peer's devices. It includes the same operations used in the public profile scheme described in Section 5.6.1, but extends the operation set with operations for managing contacts, groups, and invitations.

Contacts `AddContact`, `DeleteContact`, and `EditContact` are defined by the scheme. They are used for storing a contacts arbitrary metadata associated with each contact.

Groups The current set of groups for a peer is managed using `AddGroup` and `RemoveGroup`. Note that, unlike the contact operations, no metadata is stored for groups.

Invitations Invitations are associated with group membership. The group operation log must manage peer invitations. However, peers cannot see invitations in group before joining the group. This is resolved by storing invitations to new groups in the private profile operation log. Invitations are added using `AddInvitation` and removed using `RemoveInvitation`.

5.6.3 Group Scheme

Operations supported:

- post
- invite user
- remove user
- accept invitation
- revoke invitation
- define role
- grant role
- revoke role

The group scheme is used to create operation logs that manage group state along with group data operations. The group state tracks membership in the group and also provides operations for creating and assigning roles to group members.

Post Data is added to the operation log using the `Post` operation. The post operation is highly abstract and supports a wide range of applications. It contains only the following information:

- *tag* – An application defined tag.
- *meta* – A set of key value pairs.
- *encryption* – The encryption scheme that is applied to the message content field.
- *content* – The message content, which can also be empty.

Membership The group scheme provides four operations to track group membership: `InviteUser`, `RevokeInvitation`, `AcceptInvitation`, and `RemoveUser`.

With these operations, it is possible to project the group membership state for any position in the operation log. The group state is used to validate append operations and to encrypt messages sent between members of the group.

For instance, it may only be valid for peer A to append a `message` operation if the sub-DAG contains an `accepted-invite` operation for peer A.

Roles Group members can be assigned roles within a group, allowing them to append operations to the group operation log. Roles are defined using the `DefineRole` operation and roles for a member are modified using the `GrantRole` and `RevokeRole` operations.

5.6.4 Conflict Resolution

Concurrent operations can produce conflicts in an operation log. For instance, an invitation in the group scheme can be concurrently accepted *and* revoked. Conflict resolution strategies are used to determine the operation log state in these cases. Revoking an invitation, for instance, is a "best-effort" operation. If the invitation was already accepted concurrently, the revoke operation will have no effect on the group state.

Conflict resolution will be further explored in future work and is discussed in Section 6.2.

6 Future Work

The Sylo Protocol is in ongoing development and several areas are still actively be explored. This section highlights some of the interesting, challenging, or promising areas of research. It makes an effort to be thorough, but should not be seen as exhaustive.

6.1 Metadata Security Analysis

The exposure of metadata while communicating over distributed or P2P networks is valid and worthy of consideration. Identifying information is frequently directly accessible, such as; sender/receiver IP addresses, geolocation, time, and message frequency/size. This data can be further mined to indirectly predict other data, such as; home address, work address, movement patterns, shopping habits, and hobbies. Other methods for extracting useful data from metadata is emerging all the time.

There are many avenues of exploration available to protect the Sylo Protocol from metadata leaks. Research in this area will likely be ongoing, with problems corrected as they are encountered.

6.2 Improved Operation Log Conflict Resolution

The operation log is a partial ordering of operations. But applications frequently need a total ordering of operations. Topological sorting is the process of transforming a partially ordered set into a totally ordered set.

To achieve total ordering, we need a method for selecting an ordering for two operations that happen concurrently. This problem is considered a conflict and the process for choosing an ordering for the conflicted operations is known as conflict resolution.

A naive approach to conflict resolution, such as choosing one operation at random, can always be performed. Naive approaches can suffer an number of problems, so effort should be taken to resolve conflicts as logically as possible.

The current operation logs in the Sylo Protocol implement a simple, deterministic form of conflict resolution. This approach ensures that all peers will resolve conflicts in the same way, thus ensuring consistency of total ordering of an operation log across all peers. Additionally, work is being done to employ more efficient and logical conflict resolution strategies. This work should improve the overall user experience by aligning with user expectations for conflict resolution.

6.3 Development of Additional Operation Log Schemes

The Sylo Protocol defines three operation log schemes, as discussed in Section 5.6. The existing operation logs (and specifically the group operation log) are designed to be flexible for many use cases. However, as development of dApps for the Sylo Protocol continues, a need for additional operation log schemes could emerge. It could even be possible for external developers to define their own custom schemes. For example, an operation log scheme could be designed to collaboratively edit documents using the Sylo Protocol.

Custom operation log schemes would makes it possible for custom validation to be performed within the Sylo Protocol itself. More importantly, validation by the protocol level can prevent invalid operations from being appended to the operation entirely, thus conserving storage space.

It would seem likely that many dApps would benefit from a custom operation log scheme and may further drive external developer interest in the Sylo Protocol.

7 Summary

The Sylo Protocol brings together decentralisation, peer-to-peer communication, security, and distributed operation logs, to provide a group secure messaging protocol for the Sylo Network and external decentralised application development. Through the Sylo app, a large number of users already use the Sylo Protocol for their group messaging. Future releases of the protocol will bring new features and experiences to the decentralised community and compete with popular centralised applications in terms of both scale and user experience, all while maintaining privacy and data autonomy.

The decentralised Internet of the future demands solutions to some very challenging problems. The

Sylo Protocol addresses some of larger ones by giving developers a protocol to build upon and, hopefully, making this exciting space more accessible to everyone.

8 Acknowledgements

The Sylo Protocol is being developed for everyone and we would like to thank the incredible community of humans who share our vision of a decentralised future. If you work in this space, chances are your work has inspired us in some way, and for that we thank you.

References

[1] Brendan Abolivier. *Enter the Matrix*. May 13, 2018. URL: https://brendan.abolivier.bzh/enter-the-matrix/.

[2] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: (July 2014).

[3] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-record communication, or, why not to use PGP". In: Jan. 2004, pp. 77–84. DOI: 10.1145/1029179.1029200.

[4] J. Callas et al. *OpenPGP Message Format*. Nov. 2007. URL: https://tools.ietf.org/html/rfc4880.

[5] Ethereum. *Whisper PoC 2 Protocol Spec*. Aug. 22, 2018. URL: https://github.com/ethereum/wiki/wiki/Whisper-PoC-2-Protocol-Spec.

[6] Richard van der Hoff. *Megolm Group Ratchet*. Nov. 8, 2019. URL: https://gitlab.matrix.org/matrix-org/olm/blob/master/docs/megolm.md.

[7] S. Josefsson and I.Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. URL: https://tools.ietf.org/html/rfc8032.

[8] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Commun. ACM* 21.7 (July 1978), pp. 558–565. ISSN: 0001-0782. DOI: 10.1145/359545.359563. URL: https://doi.org/10.1145/359545.359563.

[9] Maxie Marlinspike. *The Double Ratchet Algorithm*. Ed. by Trevor Perrin. Nov. 20, 2016. URL: https://signal.org/docs/specifications/doubleratchet/.

[10] Maxie Marlinspike. *The X3DH Key Agreement Protocol*. Ed. by Trevor Perrin. Nov. 4, 2016. URL: https://signal.org/docs/specifications/x3dh/.

[11] *Messenger Secret Conversations*. Facebook, Inc. May 18, 2017. URL: https://fbnewsroomus.files.wordpress.com/2016/07/messenger-secret-conversations-technical-whitepaper.pdf.

[12] *Mobile Protocol: Detailed Description*. Telegram. URL: https://core.telegram.org/mtproto/description.

[13] Aravindh Raman et al. "Challenges in the Decentralised Web: The Mastodon Case". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 217–229. ISBN: 9781450369480. DOI: 10.1145/3355369.3355572. URL: https://doi.org/10.1145/3355369.3355572.

[14] P. Saint-Andre, ed. *Extensible Messaging and Presence Protocol (XMPP): Core*. Oct. 2004. URL: https://tools.ietf.org/html/rfc3920.

[15] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. Mar. 2011. URL: https://tools.ietf.org/html/rfc6121.

[16] Peter Saint-Andre. *Multi-User Chat*. May 15, 2019. URL: https://xmpp.org/extensions/xep-0045.html.

[17] Marc Shapiro et al. "Conflict-free Replicated Data Types". In: *SSS 2011 - 13th International Symposium Stabilization, Safety, and Security of Distributed Systems*. Ed. by Xavier Défago, Franck Petit, and Vincent Villain. Vol. 6976. Lecture Notes in Computer Science. Grenoble, France: Springer, Oct. 2011, pp. 386–400. DOI: 10.1007/978-3-642-24550-3_29. URL: https://hal.inria.fr/hal-00932836.

[18] *Skype Private Conversation*. Microsoft. June 20, 2018. URL: https://az705183.vo.msecnd.net/onlinesupportmedia/onlinesupport/media/skype/documents/skype-private-conversation-white-paper.pdf.

[19] Wickr Staff. *Secure Messaging Protocols Part 1: A Brief History*. June 14, 2019. URL: https://wickr.com/secure-messaging-protocols-part-1-a-brief-history/.

[20] Andreas Straub. *OMEMO Encryption*. July 31, 2018. URL: https://xmpp.org/extensions/xep-0384.html.

[21] Sylo. *Sylo Network: An incentivised peer-to-peer network*. URL: https://developers.sylo.io.

[22] Oskar Thoren. *Fixing Whisper with Waku*. Dec. 3, 2019. URL: https://vac.dev/fixing-whisper-with-waku.

[23] *WhatsApp Encryption Overview*. WhatsApp. Dec. 19, 2017. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.