Empowering Activism: Designing a Secure Protest Organization Application Using the Scuba Platform

Jason Oh

Advisor: Professor Amit Levy

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Computer Science
Princeton University

April 2024

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Jason Oh

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Jason Oh

Abstract

This report details the design and engineering of ProtestApp, an end-to-end encrypted and decentralized application for privately organizing protests, built on top of the Scuba platform. Additionally, this report details the 2019 Hong Kong Protests as motivation for this project, and provides a demonstration of ProtestApp.

Acknowledgements

Thank you to Professor Amit Levy for his advisory on this project. I also thank Natalie Popescu, Leon Schuermann, and Shai Caspin for their work on Scuba, without which this project would not be possible. Finally, thank you to my friends and family for all their support these past years.

Contents

Chapter 1

Introduction

End-to-end encryption is critical for maintaining privacy and security in digital communications. In particular, applications with end-to-end encrypted functionality provide secure platforms for free speech. This is especially important in nations where surveillance and censorship are embedded in their respective main digital communication platforms. Moreover, this project is primarily inspired by the 2019 Hong Kong protests and the associated need for means of communication and collaboration that avoids government surveillance.

This report details the engineering of an encrypted and decentralized application that enables protesters to coordinate their operations. In particular, the core functionalities includes a chat mechanism, location information database, and operation proposal feature. These functionalities are powered by the ability for users to create and join teams. Additionally, the report will provide an overview of how clients can best utilize the application. The application is built on top of a platform called Scuba, which provides the client and server implementations for synchronizing operations between client devices in a decentralized and encrypted manner.

Chapter 2

Problem Background

2.1 2019 Hong Kong Protests and Telegram

At the end of March 2019, the Hong Kong government proposed an extradition bill that would allow Hong Kong to send criminal suspects to Mainland China for trial. This sparked a series of organized demonstrations and acts of civil disobedience aimed against the extradition bill. These protests escalated into a broader movement that championed democratic freedoms and political autonomy. [15] As opposed to the 2014 Hong Kong protests, where Facebook and Twitter were widely used to mass mobilize citizens, [9] the 2019 protests were characterized by decentralized information flow and coordination through anonymous and encrypted messaging apps, namely WhatsApp, Signal, and most notably, Telegram. [15]

Telegram communication happened mainly through groups and channels, which are discussion threads (everyone can post in groups whereas only admins can post in channels) [9] that can support teams as large as $30,000$ members. [14] Telegram's encryption and anonymity guarantees allowed activists to discuss sensitive matters without having to fear detection and thus penal retribution enabled by the government's digital surveillance campaign. However, even if the government isn't able to

deanonymize users immediately, their digital footprint can be grounds for arresting and charging activists years later.

In 2020, the Hong Kong National Security Act was passed, criminalizing any activity that can be characterized as secession or subversion. This further disincentivised activism and contributed to a "climate of fear" that tanked Telegram activity. [6] [15] In the face of harsh offline measures, it has become even more crucial for messaging apps to be airtight regarding their privacy guarantees. Successful protests in the future can only work if there remains practical and private channels of communication that activists are not skeptical of using.

2.2 How Private Messaging Apps are Used

Telegram was used by the 2019 protesters mostly to share information regarding police presence, share protest related events, and deliberate. For instance, a reconaissance message might inform a Telegram group that "10 riot police have been deployed at Tin Shui Wai 1706 Crossroad". Additionally, information about planned protest events from high authority sources would spark deliberation in the group regarding what to do next [15]. Another usage was sharing resources. For instance, activists would provide access codes for buildings to hide in [13] and post requests for supplies such as riot gear [9] to discussion threads.

With Telegram groups potentially reaching the aforementioned size of thousands of users, it was expected that the police would infiltrate chats, resulting in attempts to identify the police and kick them out. [4] Although Telegram was the most prominent messaging app given its capacity for large group sizes, much of what has been mentioned so far can also be generalized to Signal and WhatsApp as well.

WeChat is the polar opposite of a private messaging app, as it is a CCP apparatus for surveillance and censorship. However, discreet activism still exists on the

platform, such as when Chinese anti-lockdown protests were taking place in 2020. Often, locations were given without context. For instance, "11.27, 9:30, Urumqi office" or map coordinates which communicate protest locations (e.g. $39.96698, 116.38804$) would be posted as clues. [7] This demonstrates the power of even just being able to share location based data, to enable activism.

2.3 Security Review of Private Messaging Apps

2.3.1 Telegram

Telegram's messages are encrypted in transit and at rest via its proprietary MTProto Protocol. Unlike the open source Signal Protocol, independent security experts are not able to freely examine it. Additionally, concerns have been raised regarding Telegram's usage of a novel and not yet rigorously evaluated encryption protocol rather than a widely accepted encryption protocol. [12]

Telegram is a cloud service by default, which means that encrypted messages and their decryption keys are stored on Telegram servers. The centralized architecture allows for users to access their messages synchronously through multiple devices, and even when they've lost their device. However, this application architecture is vulnerable; any entity that can compromise the Telegram servers containing an encrypted message and its corresponding decryption key can decrypt the said message. In practice, this should be impossible because Telegram distributes data between many different legal jurisdictions, but there is no technical guarantee that the servers are inaccessible to untrustworthy entities. [1]

Groups and channels in Telegram are always cloud based, but direct (2 person) conversations can optionally occur through "Secret Chats", which implement end-to-end encryption via the MTProto Protocol. The encrypted messages are only stored on the clients' devices (i.e. state lives on the edge) and can only be accessed by the

clients through their private keys. [3] It is problematic that end-to-end encryption in Telegram is opt in and unavailable for groups and channels, given that many users may be under the false impression that their cloud messages are 100% inaccessible to third parties. Additionally, the option to create a "Secret Chat" is hidden in the UI.

2.3.2 Signal

Signal is engineered for messages to always be end-to-end encrypted and stored on the clients' devices (i.e. state lives on the edge). Thus, unlike Telegram, 2-person and group communications are completely inaccessible to any untrustworthy entity that cannot access the clients' devices. [2]

The Signal Protocol is an open source encryption standard. Unlike Telegram's MTProto Protocol, the Signal Protocol has been heavily audited by security experts and has thus gained enough trust to become the defacto end-to-end encryption standard for messaging applications. For instance, WhatsApp, Facebook Messenger, and Google Messages have adopted the Signal Protocol to implement end-to-end encryption options on their applications. [11]

Aside from its transparency and established trust, the salient feature of the Signal Protocol is its Double Ratchet algorithm, which ensures that messages are forward secret. To elaborate, the Double Ratchet algorithm guarantees that in the case that an adversary gains access to a client's current message and the private session key corresponding to that message, the said adversary is unable to decrypt any prior or future messages. Even in the case that an adversary gains access to the client's permanent private key, the adversary is unable to decrypt prior messages. [11] [8] Signal allows users to capitalize on forward secrecy by providing an option for messages to automatically disappear from clients' devices after a certain period of time.

Chapter 3

Application Background

3.1 The Scuba Platform

3.1.1 Platform Objectives

Scuba [5] is a common core platform for building end-to-end encrypted applications. Such applications eschew the storage of client data in a central server, and instead maintain application state through the clients' devices. Although inter-client communications are routed (but not permanently stored) through a central server, the server cannot observe the contents of such communications because they are encrypted with the clients' private keys. Ultimately, this ensures that Scuba powered applications have strong privacy guarantees that are within the control of clients.

Whereas the systems for end-to-end encryption messaging applications are engineered expressly for the purpose of messaging (i.e. group texting and voice calling), Scuba aims to be application agnostic. In particular, the Scuba core offers a abstract interface for receiving, ordering, and sending operations defined on objects. Concrete data models such as key-value stores and relational databases are then meant to be built on top of the Scuba core, powering a diverse set of applications. These applications will be able to run concurrently on top of a shared Scuba server.

Scuba also provides a client enabled mechanism for proving server misbehavior. Whereas private messaging services such as Signal and Telegram assume that the application servers are trustworthy, Scuba accounts for the hypothetical case of a malicious server, one that might drop and reorder messages in violation of the canonical server protocol. When clients deduce that their server is misbehaving, they can choose to migrate their data to another third party server.

3.1.2 System Design

End-to-end encryption between pairs of clients in Scuba is achieved through the Double Ratchet algorithm. In the same way that messages are forward secret in Signal, in the case that the privacy of a client operation is compromised, the vulnerability does not extend to any prior or future operations from that client. To elaborate further on the Double Ratchet algorithm, a new symmetric private key is generated each time a client forwards an operation to another client. This key is not only unique to the operation, but for each pair of clients. The cryptographic properties of the algorithm ensure that the generation of this key cannot be reproduced.

Operations on a distributed object must be synchronized in order to enable the engineering of useful applications. The centralized Scuba server establishes a canonical total ordering of all operations, these operations being ordered in the real time sequence the server receives them from clients. This enables client reads and writes on objects to be linearizable. To do so, client read and write operations must be processed through and received from the server before being actually applied to the client's data store. Via this method, reads come after all prior writes on an object are observed, and writes are synchronized between clients. Simply put, this behavior is identical to as if clients were interacting with a single, centralized data store, thus allowing the sane development of apps built on top of a linearizable data store.

Server misbehavior is detected through a client side mechanism called Pairwise

Order Validation Scheme. Each client maintains a vector of hashes for each peer that it communicates with. The hash vector reflects the order of shared operations, which should be consistent between pairs of clients. Thus, conflicting hash vectors indicate that operations were shared incorrectly, which may be the result of the server dropping or misordering said operations. Migrating to another server is trivial because the clients, not the server, store all of a Scuba application's state.

3.1.3 Scuba Key Value Data Store

A Key Value data store implementation built on top of the Scuba core is provided to application developers. Objects can be created and written to via a `set(id, value)` API function, and read via a `get(id)` API function. The Key Value store guarantees linearizable operations as long as client reads and writes happen through the API, and they are called sequentually by the client (i.e. one operation at a time).

The Key Value data store also implements access control, meaning that clients are able to expose objects to peers while enforcing different permissions on them. More specifically, access control rules for data objects are encoded through their relevant permissions objects. Clients can interact with data objects as either an admin, writer, or reader. Admins are allowed to grant read and write privileges to other clients through the `add_writers()` and `add_readers()` API functions. These functions connect pairs of clients in the context of data objects, thus establishing that encrypted updates will be shared between them when these objects are written to. Moreover, the access control mechanism verifies whether an incoming write is sent from a permissioned writer, in which case it is applied.

The Key Value data store provides `start_transaction()` and `end_transaction()` API functions to allow atomic commits of multiple writes involving different keys. Transactions are useful in the case that multiple writes need to occur at the same time, or not at all, in order to guarantee a correct application state. Moreover, trans-

actions allow for operations over data objects to be characterized as serializable.

The Key Value data store provides a data invariant checking mechanism for incoming writes. More specifically, data objects are associated with a `type` field. An invariant checking function can then be registered with the said type; if that function returns that a write violates its corresponding object's type invariant, the client application is blocked from applying that write on its local store.

3.2 Existing Applications Built on Scuba

A family social media application is built on top of Scuba's Key Value store. It allows family members to chat with each other and create posts. Family members can then comment on or react to these posts. In light of social media platforms' privacy violations, such as Facebook's provision of user data to the political consulting firm Cambridge Analytica, there is a strong need for end-to-end encrypted alternatives.

A patient-provider appointment scheduling application is built on top of Scuba's Key Value store. Healthcare providers can add patients as contacts, revealing their calendar availability to patients. Patients can then request an appointment within the provider's availibility slots, which must then be confirmed by the provider. The need for privacy in a healthcare setting is evident, but the need for tight encryption guarantees is further augmented given the 2022 reversal of *Roe v. Wade* by the U.S. Supreme Court and the subsequent criminalization of abortion in states like Texas.

Additional applications have been built on top of Scuba's Key Value store, but the aforementioned apps demonstrate some of what can be achieved with the platform.

3.3 Regarding the Usage of Rust

End-to-end encryption is not only as safe as its cryptographic guarantees, but it is also only as safe as its engineered implementation. To elaborate, traditional

systems programming languages such as C and C++ are notorious for containing many security pitfalls that must be consciously avoided by developers, and are often only discovered at program runtime.

It is possible in these languages to compile code that violates safe memory usage. An iconic example is the access of memory through a pointer variable after `free()` has been called on it (an analogy would be like entering your old apartment even though it belongs to the new tenants, the apartment representing computer memory). Another class of memory vulnerabilites is buffer overflows, where a program allocates a fixed chunk of memory, but then later writes to or reads from beyond its bounds. For instance, the 2014 uncovering of the Heartbleed bug in OpenSSL, a software library for establishing encrypted communications over computer networks, revealed that adversaries could trick servers into retreiving memory beyond the bound of an allocated buffer and sending it, potentially revealing client secrets. [10]

Rust provides memory safety guarantees at compile time, ensuring that these types of vulnerabilies are not possible in compiled Rust code. The core enabler of this feature is the Rust compiler's borrow checker, which enforces a strict ruleset (i.e. ownership and borrowing) for how Rust code interacts with memory, one that must be understood and followed by the developer. Scuba's core, key-value data store, and example applications are programmed using Rust, thus providing full stack system security guarantees for the private applications.

To capitalize on its security, ProtestApp is also engineered using Rust. While writing code that follows ownership and borrowing rules is sometimes frustrating, especially when compared to automatic memory management, the compiler error messages and static code analyzer make writing in Rust quite sane in practice.

Chapter 4

Approach and Implementation

4.1 Application Functionality

4.1.1 ProtestApp Overview

This report introduces ProtestApp, an application designed for privately coordinating protests. The core functionalites of this application are inspired by how Telegram has been used by activists to power the 2019 Hong Kong protests. Namely, ProtestApp implements private chatting, public chatting, a location database, and an operation voting feature. These functionalities work under the context of teams. Once an agent (i.e. application client) has joined a team, they can establish communications between each other through their chosen aliases.

Agents signup to the application by choosing an alias for themselves. This alias could be the user's real name or a nickname of their choice depending on their privacy tolerance. Upon signing up, the agent receives an id-key which is their passcode for reentering the client application state. Additionally, they are assigned a *Follower* role by default. Agents can elevate their role from *Follower* to *Coordinator* by creating a team. A team coordinator is responsible for accepting requests from other agents to join their team, and can also remove agents from their team if necessary.

Once an agent has joined a team, they can access a directory containing the aliases of the team's agents. An agent can send a *private_message* to another agent in the team by specifying a message and the alias of the agent to send it to. The private message thread between a pair of clients is only accessible to those clients. Moreover, an agent can also send a *public_message* to the team. Although the public message is visible to all agents via the public message thread, clients that are not part of the team cannot access the thread. ProtestApp's private and public messaging should be effective on their own, as demonstrated by Telegram's successful role in coordinating the 2019 Hong Kong protests. However, the app still provides additional features.

Agents can contribute to a public *location_database*. An entry in the database is a location with a name, type, and information. A gps coordinate (longitude, latitude) is also associated with the location. Agents can query for locations in the database by their type. Additionally, they can set their personal coordinates, and then query for locations based on their distance from the agent's personal coordinates. This feature is meant to enhance the experience of using the application's messaging functionality. In particular, the public message thread might become a disorganized list of location based information without this feature. The location database is an alternative place to post such information, which can then be referred to in the messages.

Finally, the operation voting feature is meant to enhance the experience of deliberating on potential operations. Any agent can propose a named operation, which must be associated with a time range, and is optionally associated with a coordinate. Agents can then cast their votes for the operation; the voting status is visible to the team. The team coordinator has the final power to commit the operation, and the application ensures that the dates of committed operations do not conflict with each other. It is expected that the public message thread will be used adjacently in order to deliberate on potential operations.

4.1.2 Commands List

This is a list of ProtestApp's commands. `help <command_name>` can be used in the application to fetch instructions on how to use the command. In particular, `help <command_name>` shows which input fields are required, and which input fields are optional. In addition to the commands listed below, the debug command `dump_data` can be used to print all key-value objects that are associated with the client.

General Agent Commands:

- signup_agent <alias>
- login_agent <id>
- get_agent_info (+ specific get_agent_xxx commands)
- update_agent_alias <alias>

Team Related Commands:

- get_agent_alias_list
- create_team
- join_team_request <coordinator_id> <coordinator_name>
- join_team_accept_request <agent_alias>
- check_join_team_request
- remove_from_team <agent_alias>

Private Messaging Commands:

- send_private_message <agent_to_alias> <message> <message_type>
  (message_type: Message, Alert, Announcement)
- get_private_messages <agent_to_alias> <num_last_messages>
- get_new_private_messages <agent_to_alias>
- delete_private_message <agent_to_alias> <message_index>

Public Messaging Commands:

- send_public_message <message> <message_type>
  (message_type: Message, Alert, Announcement)
- get_public_messages <num_last_messages>
- delete_public_message <message_index>

Location Database Commands:

- update_agent_location <longitude> <latitude> or
  update_agent_location Auto
- get_agent_location
- add_to_location_database <name> <longitude> <latitude> <type> <info>
- remove_from_location_database <index>
- get_own_location_database
- get_location_database <location_type> <distance_from_agent (in meters)>
  (location_type: All or xxx)

Operation Voting Commands:

- make_operation_proposal <name> <start > <end> <info> <coord>
- vote_for_operation_proposal <agent_alias> <operation_name> <vote>
  (vote: Yes, No, N/A)
- operation_proposal_status <agent_alias> <operation_name>
- commit_operation_proposal <agent_alias> <operation_name>
- get_committed_operations

4.2 Implementation Details

4.2.1 Implementation Overview

The existing applications *calendar* and *family-app* in the Scuba repository were used as references for understanding how to use the Scuba Key Value API and pro-

vide a CLI interface to the application via the *reedline-repl-rs* crate. Additionally, ProtestApp imports the *time*, *geo*, and *chrono* crates for their implementation of date-time and coordinate data types. The *serde* crate allows for Rust data types to be converted to and from JSON before being set and retreived from the key-value store respectively. Overall, ProtestApp is implemented using $4,990$ lines of code (including comments). The application is available via the following fork of the Scuba repository: https://github.com/Jason0h/SeniorThesis, and is under the SeniorThesis/apps/cmdline/senior-thesis directory.

4.2.2 Key Naming Scheme

A drawback of the existing applications is their limited attention to client ergonomics. For instance, in the *family-app* application, when client A makes a post, they receive a <post-id> associated with the post. The <post-id> is the post object's key in the key-value store. When client B wants to comment on the post, they then have a provide <post-id> as an input. The application provides no ergonomic mechanism for client B to be made aware of <post-id>'s existence. Unless client A informs client B about <post-id> through some external channel, client B would have to call `dump_data` and recognize the existence of a new object in the key-value store called <post-id> to realize that client A has made a new post. Additionally, clients must share their client-id and client-name through an external channel in order to bootstrap contact and shared access to key-value objects.

ProtestApp abstracts away the key-value store mechanism and the client-id / client-name mechanism as much as possible from the user. In particular, clients never have to explicitly interact with object keys nor explicitly share these keys with other clients; this is instead handled internally in ProtestApp. Sharing client-id and client-name through an external channel is still necessary to bootstrap client communication, but in ProtestApp, only the coordinator needs to externally share their

id and name. Once an agent joins a team, the application automatically handles establishing communications between the agent and all other agents in the team. Additionally, upon joining a team, an agent can get the aliases of all team members via `get_agent_alias_list`. These aliases are all an agent needs to establish communication with team members. Since client-id and client-name are random character strings, and thus non-semantic, it is easier for clients to interact with aliases instead.

ProtestApp employs a coordinated key naming scheme so that client applications automatically know how to access each others' key-value objects. The key naming convention is detailed comprehensively in the comments starting at line 30 of *main.rs*. For instance, when agent A sends a message to agent B, agent A will create / set the object named "private_messages/{coordinator_alias}/{agent_A_alias}/{agent_B_alias}". Moreoever, when agent B wants to retreive the messages that agent A has sent to them, agent B can use the aforementioned key to get the same object. Thus, the command the client calls and the associated client inputs are sufficient enough clues to infer the appropriate key. Additionally, the keys are guaranteed to not collide as long as the clients respect the key naming scheme.

4.2.3 Access Control

For the most part, an object in ProtestApp has an admin (share and write privileges) who grants read-only privilege for the object to other agents. Agents without read-only privilege are unable to access the object because operations on that object are not routed to them (i.e. a `get()` query on that object will return None). To illustrate, consider private messaging between agent A and agent B. agent A creates and read-only shares a vector of messages to agent B. agent B creates and read-only shares a vector of messages to agent A. Thus, the private message thread is composed of two threads where each agent is able to read but not write to their peer's thread. Additionally, consider public messaging. An agent creates a vector of messages and

read-only shares it with all agents in the team to prevent tampering. Moreoever, when an agent retreives public messages, they are retreiving each agent's respective public message thread. Much of the other application functionalities operate similarly regarding access control. For instance, the agent_alias_list is created by the coordinator and read-only shared with all agents in the team.

ProtestApp provides various "deletion" commands. For instance, delete_public_message <message_index> allows an agent to remove a message from their public message thread. This removal can only be done by that agent (who has write privilege), and is complete once the removal has been propogated to all agents in the team. Scuba's forward secrecy property ensures that "deletions" cannot be reversed.

ProtestApp does not fully protect against byzantine clients. For instance, there is nothing stopping agent C from setting the "private_messages/{coordinator_alias}/{agent_A_alias}/{agent_B_alias}" key first even though this key "belongs" to agent A. Thus, agent B may be reading messages that they believe is coming from agent A, but in reality is coming from agent C. What would help is a *creator_name* field for the `get(id)` function; in particular, that the amended `get(id, creator_name)` function retreives the object whose key is id, **and** whose creator is *creator_name*.

4.2.4 Consistency

All reads and writes happen through the `get(id)` and `set(id, value)` key-value API functions respectively. Additionally, these operations occur sequentialy. Accordingly, ProtestApp's client outputs will observe linearizability. There are some commands that involve writes to more than one object. In these cases, the writes are wrapped under a single transaction to ensure that they are applied atomically. Thus, ProtestApp's outputs will observe serializability as well.

4.2.5 Error Handling

ProtestApp strives to handle client and system errors gracefully. In particular, it propogates errors upwards in the reverse order of function calls until they're returned to the client without exiting the application. For instance, if an agent tries to send a private message to a nonexisting alias, the application will catch the error and return the message "Client Error: <agent_alias> Is Not A Part Of The Team". Moreover, if the Scuba client core returns an error, the client will see a message such as "System Error: Unable To Add Agent As Contact".

ProtestApp ensures that inputs to the key-value store, and hence the application state is correct. For instance, if the client tries to write a public message that is longer than the character limit for messages, ProtestApp will return a Client Error. Additionally, if a coordinator tries to commit an operation whose date range conflicts with already commited operations, ProtestApp will return a Client Error. There are many more similar instances of input validation implemented throughout ProtestApp.

ProtestApp also utilizes Scuba key-value's data invariant checking mechanism to ensure a correct application state in the face of byzantine clients who might set improper values. For instance, it converts objects from JSON and ensures that they're the proper data type. Additionally, the timestamps of private messages are checked to make sure that they're in order. There are more instances of invariant checking, and violations of registered invariants block writes from being applied.

A limitation of ProtestApp's error handling is that it doesn't gracefully handle panics from Scuba's client core. When this happens, the application exits and the client sees at which line the client core panicked. Although ProtestApp runs without issues for the most part, there may be instances when this occurs. in particular, there is an out of bounds access issue at client/core/src/hash_vectors.rs:404:34.

4.2.6 Other Implementation Problems

A remaining problem with ProtestApp is its inclusion of 1 second timeouts. In particular, 1 second timeouts are included before adding clients as contacts, and before adding clients as readers / writers to a data object. This avoids the Scuba client core from panicking. These timeouts cause a noticable delay whenever an agent shares something with the entire team. In the current implementation of ProtestApp, this delay scales with the number of agents in the team. For instance, in a team of 3 agents, the delay will be at least 3 seconds. Eliminating these timeouts (once the underlying issue is fixed) should allow ProtestApp to run without delay.

The forward secret property of Scuba requires agents to regularly delete old data for it to be truly useful. ProtestApp expects agents to do this manually. Perhaps in a better implementation of ProtestApp we'd have an asynchronous function automatically deleting old entries as per client settings. Moreoever, a suggestion for Scuba key-value is the inclusion of an automatic deletion mechanism within the platform.

4.2.7 Memory Usage

The ProtestApp CLI application uses 3.8 Mb of memory when opened. For comparison, the Scuba social media application also uses 3.8 of memory when opened. ProtestApp is neither more nor less memory performant than existing Scuba apps.

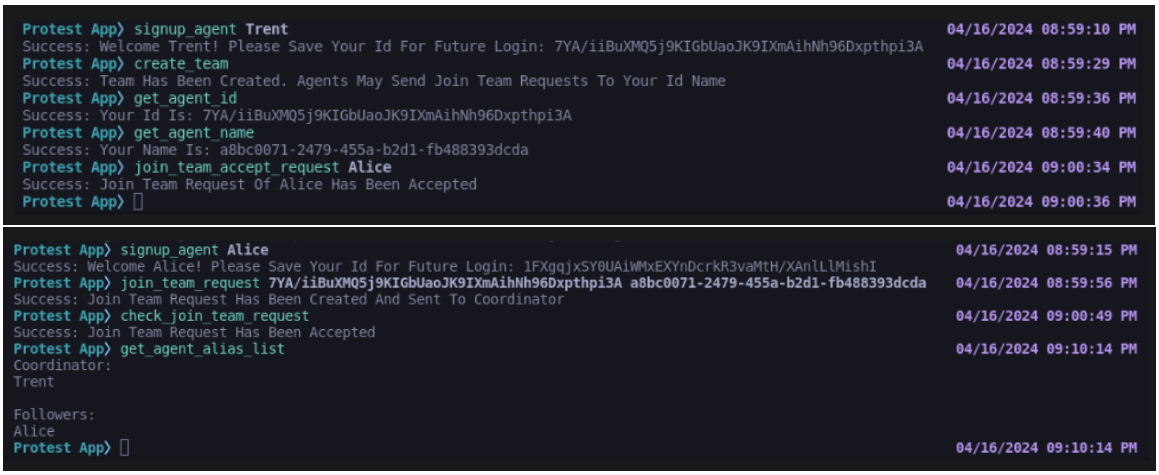
Chapter 5

Application Demonstration

5.1 Starting The Application

This demonstration requires the installation of Rust, and 6 terminal windows will need to be opened at the SeniorThesis/apps/cmdline/senior-thesis directory. On the first 2 terminal windows, separately call `./startshards.sh` and `./startsequencer.sh`. This will spin up a local Scuba server. The remaining 4 terminal windows will host the clients of ProtestApp. A client can open ProtestApp by running `cargo run`.

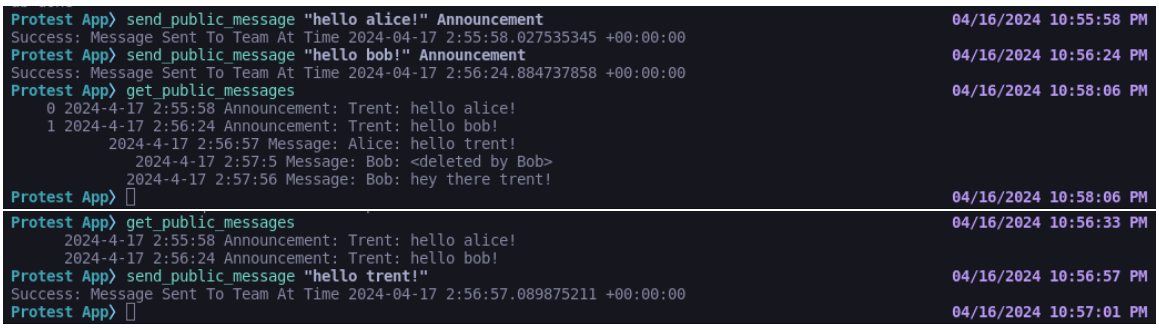
5.2 Creating Agents and Teams



The 2 terminal sessions above demonstrate how clients can create ProtestApp agents and how these agents can establish connection via a team. Trent takes the role of a coordinator and Alice takes the role of a follower. Trent elevates his role to coordinator by creating a team. Trent must share his <id> and <name> to Alice through an external channel such as Signal, after which Alice will be able to request to join the team. Once Trent has accepted Alice's team request, Alice can check her request status, which confirms that her request has been accepted by Trent.

ProtestApp supports multiple coexisting teams. However, there are limitations to the current implementation. Coordinators cannot join other teams as followers and followers can only belong to one team. Overcoming these limitations is definitely possible, but will require further work on the application.

5.3 Private and Public Messaging



```
Protest App) get_public_messages                                                    04/16/2024 10:56:39 PM
      2024-4-17 2:55:58 Announcement: Trent: hello alice!
      2024-4-17 2:56:24 Announcement: Trent: hello bob!
Protest App) send_public_message "hello trent!"                                     04/16/2024 10:57:05 PM
Success: Message Sent To Team At Time 2024-04-17 2:57:05.385703625 +00:00:00
Protest App) delete_public_message 0                                                04/16/2024 10:57:47 PM
Success: Message 0 Deleted
Protest App) send_public_message "hey there trent!"                                 04/16/2024 10:57:56 PM
Success: Message Sent To Team At Time 2024-04-17 2:57:56.787231721 +00:00:00
Protest App)                                                                        04/16/2024 10:58:00 PM
```

We have Trent, Alice, and Bob respectively in the 3 terminal sessions above. Trent is the coordinator of the team whereas Alice and Bob are followers in the team. The agents are greeting each other through the public message thread. Afterwards, agents may want to deliberate on potential operations, share important news, or voice political sentiments through the public thread.

```
Protest App) send_private_message Bob "hey bob! do you have an extra first aid kit?" Alert  04/16/2024 11:30:55 PM
Success: Message Sent To Bob At Time 2024-04-17 3:30:55.784501122 +00:00:00
Protest App) get_new_private_messages Bob                                           04/16/2024 11:31:55 PM
          0 2024-4-17 3:30:55 Alert: Alice: hey bob! do you have an extra first aid kit?
            2024-4-17 3:31:40 Message: Bob: give me a sec. looking
Protest App) get_new_private_messages Bob                                           04/16/2024 11:32:07 PM

Protest App) get_new_private_messages Bob                                           04/16/2024 11:32:57 PM
            2024-4-17 3:32:49 Message: Bob: found one. meet me at 202 Queens' Road East
Protest App)                                                                        04/16/2024 11:32:57 PM
```

```
Protest App) get_private_messages Alice                                             04/16/2024 11:31:14 PM
            2024-4-17 3:30:55 Alert: Alice: hey bob! do you have an extra first aid kit?
Protest App) send_private_message Alice "give me a sec. looking"                    04/16/2024 11:31:40 PM
Success: Message Sent To Alice At Time 2024-04-17 3:31:40.52660874 +00:00:00
Protest App) send_private_message Alice "found one. meet me at 202 Queens' Road East"  04/16/2024 11:32:49 PM
Success: Message Sent To Alice At Time 2024-04-17 3:32:49.888274742 +00:00:00
Protest App)                                                                        04/16/2024 11:32:49 PM
```

Alternatively, agents can communicate privately. In this scenario, Alice has a request specifically for Bob. Sending this request through a private message channel rather than the public message channel ensures that it won't go unnoticed, especially when there are many agents talking concurrently through public messages. Also, Alice may not want to disclose the location "202 Queen's Road East" to every agent.

Messaging, both private and public only works for agents who belong to the same team. This limitation also extends to the rest of ProtestApp's functionalities. Therefore, the current implementation of ProtestApp does not support communication between teams or agents who have not joined a team.

5.4 Location Database

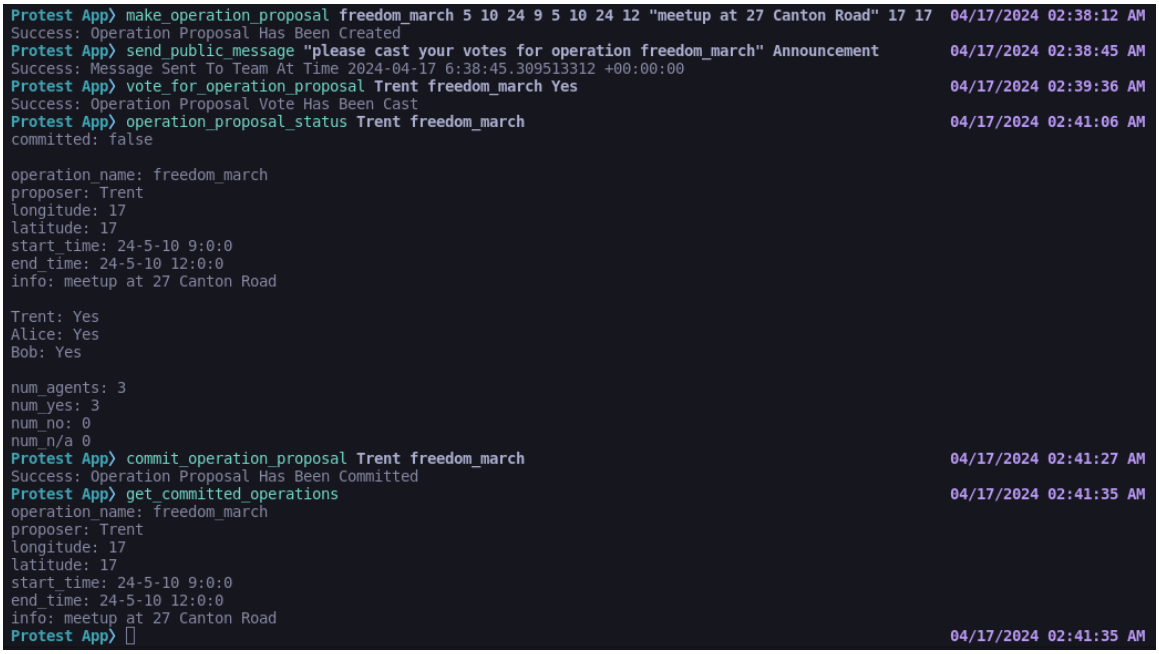
We have Trent's terminal session above. The scenario is that the team has decided to conduct reconaissance for police presence and collect their findings in the location database. Trent, Alice, and Bob have contributed entries to the database. The agents have posted the coordinates of police cars, along the associated addresses and license plate numbers of the police cars. Trent can query for "PoliceCar" to get entries in the database of type "PoliceCar". Additionally, Trent can update his location and further refine the query to return police cars that are within 500 meters of him. This would come in handy if Trent for instance wanted to give a public speech, but wanted it to be reasonably far away from police attention.

Trent can also delete his entry from the location database once he has noticed that the threat is gone. ProtestApp prevents other agents from deleting Trent's entries, but an agent can request for Trent to remove his entry via a private message.

There are countless other ways the location database can be utilized. For instance,

agents could post the coordinates of safe spots, and the associated information necessary to enter those safe spots. Additionally, agents could post the coordinates of surveillance cameras. Agents can then avoid locations with surveillance cameras present, and even plan and vote on operation to disable the surveillance cameras that have been listed in the location database.

5.5 Operation Voting



We have Trent's terminal session above. Trent has made a proposal for an operation "freedom_march" scheduled between 5/10/24 9:00 and 5/10/24 12:00. Afterwards, all the agents including Trent have casted a "Yes" vote for the operation. These votes are publicly visible to all agents in the team. Depending on the vote status, Trent (the coordinator) can decide to commit the proposed operation, upon which agents will see that it has been committed. The times of committed operations do not intersect. Operation voting is meant to help the deliberation process that is expected to occur in the public messaging thread, by summarazing the state of agents' opinions.

Chapter 6

Conclusion and Future Work

6.1 Concluding Remarks

Overall, this report has demonstrated the need for end-to-end encrypted channels of communication in the context of the 2019 Hong Kong protests. In particular, it has examined the existing usage of private applications to coordinate protests, Telegram being the most prominent amongst these applications. This report addresses the limitation of coordinating protests via a strictly messaging based application. Therefore, it introduces the previous work of Scuba, a platform for creating decentralized and end-to-end encrypted applications. Moreover, this report details the engineering of ProtestApp on top of the Scuba platform. ProtestApp not only implements messaging functionalities, but also provides location database and operation voting features. ProtestApp's different functionalities can be used in tandem to enable a more efficient experience of organizing protests than merely coordinating via a group chat.

6.2 Future Work

Currently, ProtestApp's user interface is the command line. Ideally, ProtestApp would be a mobile application, making it comparable to existing private applications

on the market. Therefore, further work on ProtestApp could include porting its inner logic onto a mobile interface. Furthermore, further work on ProtestApp could include allowing communication between teams and agents who have not yet joined a team.

Additionally, there are some recommendations for the development of the Scuba platform, coming from an application developer's pov. As mentioned before, it would help for the Scuba key-value to have a *name* field along with the *key* field, to verify the creator / admin of an object retreived with `get()`. Also, it may be useful to implement a mechanism to register the automatic deletion of objects in the key-value store; this would make it convenient to capitalize on Scuba's forward secrecy. Finally, this report has outlined some bugs in the Scuba client core that should be addressed.

Bibliography

[1] End-to-end encryption faq. *Telegram Support Force*.

[2] Signal terms privacy policy. *Signal*.

[3] Telegram privacy policy. *Telegram*.

[4] How hong kong protesters evade surveillance with tech wsj. *The Wall Street Journal*, 2019.

[5] Enabling privacy for decentralized applications with noise. Working paper, Princeton University, 2024.

[6] Hong kong national security law: What is it and is it worrying? *BBC*, 2024.

[7] E. Baptista. Dating apps and telegram: How china protesters are defying authorities. *Reuters*, 2022.

[8] Ch0pin. The signal protocol and the double ratchet algorithm. *Wired*, 2021.

[9] K. Chiu and L. Lew. From facebook and twitter to telegram, whatsapp and signal: how protest technology has evolved since occupy central. *South China Morning Post*, 2019.

[10] J. Fruhlinger. The heartbleed bug: How a flaw in openssl caused a security crisis. *CSO*, 2022.

[11] A. Greenberg. Hacker lexicon: What is the signal encryption protocol? *Wired*, 2020.

[12] M. Grigutyte. Is telegram safe? *NordVPN*, 2024.

[13] J. Schectman. Exclusive: Messaging app telegram moves to protect identity of hong kong protesters. *Reuters*, 2019.

[14] Z. Soo. What is telegram and why did the messaging app prove so popular during the hong kong protests? *South China Morning Post*, 2019.

[15] A. Urman, J. C. Ho, and S. Katz. "no central stage": Telegram-based activity during the 2019 protests in hong kong. *SocArXiv*, 2020.