



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

*Distributed
Computing*



PolyPass - A Convenient Password Manager

Master's Thesis

Noah Studach

`studachn@ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:

Roland Schmid, Simon Tanner

Prof. Dr. Roger Wattenhofer

September 9, 2019

Abstract

Online services are getting more and more integrated into our daily lives. Users rarely concern themselves with strong security practice, even when confronted with massive user credential breaches. Weak or reused passwords are common practice. Password managers have not yet succeeded to become a widespread solution. PolyPass aims to improve the current situation and provides a password manager that keeps its data secure by using secret sharing. The relevant knowledge is spread among the user's devices and useless unless collectively restored. To retrieve a stored password sufficiently many devices need shares their knowledge. Security is ensured by devices only sharing knowledge if they are either prompted by the user or they confidently assume a local affiliation with the requesting device. The local affiliation gets evaluated with GPS locations and a novel approach of local network detection with the help of WebRTC.

Contents

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.2 Goal	1
2 Related Work	3
2.1 Local Approach	3
2.2 Dual Possession Approach	3
2.3 Distributed Approach	4
3 Background	5
3.1 Flowcharts	5
3.2 Password Manager	5
3.3 Progressive Web App	7
3.4 Hybrid Apps	7
3.5 Trusted Peer	7
3.6 Secret Sharing	7
3.7 WebRTC	8
3.8 Predecessors of PolyPass	9
4 Software Design	10
4.1 Key Challenges	10
4.2 PolyPass	10
4.3 Communication	11
4.4 Auto Unlocking	12
4.4.1 Proximity Parameters	12
4.4.2 Share Request	13
4.5 Android Application	14
4.6 User Study	16

5 Discussion	18
5.1 Performance	18
5.2 Implementation	19
5.3 Security	19
5.3.1 Proximity Parameters	19
5.3.2 Password Groups	20
5.3.3 Communication Channel	20
5.3.4 Denial of Service	21
6 Summary and Future Work	22
6.1 Summary	22
6.2 Future Work	22
Bibliography	24
A Pairing Process	A-1
A.1 Notation	A-1
A.2 Pairing	A-1
A.3 Data Exchange	A-2
A.4 NoKey Communication	A-2
A.4.1 Pairing	A-2
B Password Manager Analysis	B-1
C Elm	C-1
C.1 Data Types	C-1
C.2 Programming Pattern	C-2
D Performance Measurements	D-1

Introduction

1.1 Motivation

Passwords are still the most used form of user authentication. An average person has to manage passwords for 16 to 26 accounts [1]. With password policies getting more restricted, remembering a unique and high entropy password for each account is no easy task. Most users see authentication as a burden. Passwords containing personal data and the reuse of passwords are observed consequences. Reusing passwords is considered bad practice because even strong passwords can be leaked by data breaches [2, 3, 4, 5]. Massive data breaches such as the *Collection #1* [6] in January of 2019, which exposed more than 21 million passwords are no rarity. To the point that sites, like haveibeenpwned.com, exist that allow users to check whether their accounts might be compromised.

Password managers have been around for many years and are a helpful tool for managing credentials. The adoption rate of password managers is still surprisingly low, despite all the benefits they provide. Related research [1, 7, 8] indicates that for users the dangers of weak passwords are unclear and they do not relate to revealed security issues. Additional issues are the initial setup and little trust in a product that is not well understood by the average user. Most of them secure their storage with a master password. The user still faces some of the problems previously mentioned. A strong password is difficult to remember but required to keep the contents of the password manager safe. In this thesis, we apply the approach of splitting and distributing the master password among a few devices. Our solution to the problem of master passwords. In this case access to the manager is not granted by knowing a master password, but rather having control over a set of devices.

1.2 Goal

In this thesis, we create PolyPass, a convenient password manager. In contrast to popular password managers, the requirement for entering a master password is replaced by a distributed approach of user identification. PolyPass builds on an existing application called NoKey [9], but enhances it with certain features to deliver a more compelling user experience.

We reduce human interaction by adding the ability to automatically unlock passwords under certain conditions. With autofill for Android, the mobile experience will be massively improved. The user must be better informed about the operating principle of the password manager but also about potential security issues. By improving the UI and adding new features the user will better understand the application and potentially apply a more responsible

security practice. To reduce server usage we switch from WebSocket to peer-2-peer communication. We design a user study to thoroughly analyze the user experience of PolyPass. The user study will not be conducted in this thesis. At a later point we also plan to implement Bluetooth for local communication.

Related Work

Entrepreneurs and researchers have developed several solutions to improve the authentication process including password managers, graphical passwords, biometric and 2-factor authentication as well as password hashing and single sign-on systems [10, 11]. This thesis is focused on password managers. There are many different password managers available using different approaches to secure their content.

2.1 Local Approach

We selected and briefly analyzed popular password managers (Appendix B). All of them use the same underlying principle, which is to encrypt the sensitive data locally with a user-defined master password. All password managers provide some form of data synchronization.

Additionally, modern browsers, such as Chrome, Firefox, Opera, Safari, Internet Explorer and Brave, all have built-in password managers. They also implement the principle mentioned above, but the encryption key is typically derived from the operating system login credentials [12, 10].

In PolyPass the master password is shared among a set of trusted devices. To ensure security every device only stores a part of the master password.

2.2 Dual Possession Approach

The password manager Tapas implements a dual-possession authentication approach. The system consists of two devices, a *manager* and a *wallet*. The manager is implemented as a browser extension and the wallet runs on a smartphone. Both entities communicate over a web server.

The wallet stores the encrypted passwords and the corresponding meta data while the manager holds the key required to decrypt the passwords. To retrieve any password the encrypted data is securely sent to the from the wallet to the manager where the credentials are decrypted [13].

Tapas only uses two devices, each with a specific purpose. If an attacker steals both devices the have access to all Passwords. PolyPass improves over this by allowing countless equipotent devices. PolyPass separates the knowledge of the master secret instead of the master secret and the ciphertext.

2.3 Distributed Approach

The Convenient Password manager [14] and NoKey [9] both implement a distributed approach, where they split the master secret according to Shamir's secret sharing. The shares are distributed among all trusted peers. To access any password the secret must first be restored. The device requiring the passwords sends a request to each peer, asking for their share. As a protective measure, the user needs to accept the share request on each device. If the request is accepted, the device will send its share. Once the requesting device has received enough shares, it reconstructs the secret and acquires the password, after which the secret is destroyed.

Even though PolyPass implements the same approach, it distinguishes itself from the Convenient Password Manager, by being a web-based retrieval password manager while the Convenient Password Manager generates passwords on demand and only runs on Android. PolyPass is built upon NoKey and adds convenience functions, such as automatically unlocking passwords, push notifications and autofill for Android. It also increased the communication secrecy by switching from WebSockets to WebRTC. The differences are thoroughly presented in Section 4.

Background

Before we begin presenting the application design, it is necessary to elaborate on commonly used terms and notations to prevent verbosely written explanations.

3.1 Flowcharts

This thesis contains flowcharts to better illustrate the inner working of the program code. Figure 3.1 helps reading them, by showing each flowcharts building block and its function.

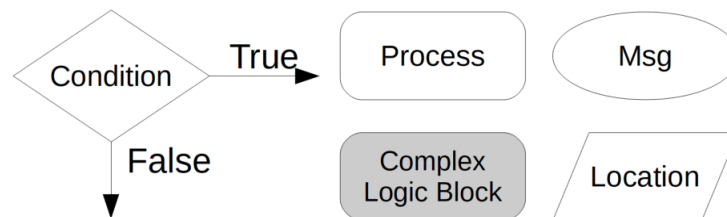


Figure 3.1: The building blocks and their function as used in the flowcharts.

3.2 Password Manager

User authorization is a daily challenge. Text-based passwords are still the most prevalent form of authentication [15]. Generally, a password is paired with an identifier to build user credentials. The user credentials serve as user authentication for a service. The service name and the user's credentials build the service credentials.

Weak passwords are more likely to be uncovered by attackers using various methods such as dictionary attacks, brute force or probabilistic context-free grammar. To prevent such attacks password policies for creating user credentials are getting more restrictive, forcing passwords to be more complex [4, 5]. Studies have shown that an average user has to manage authentication for a few dozen different services and are asked to provide multiple passwords a day. The vast majority can not remember that many strong passwords and is taking measures that weaken their security. Users still tend to use weaker passwords, with over half containing only letters or only numbers [16]. Other common occurrences are reusing the most frequently used passwords or creating passwords from personal or service related data [3, 2, 1].

Password managers are created to alleviate the difficulties of remembering lots of service credentials. They store all the user's passwords and keep the sensitive content save using encryption. The way passwords and its associated metadata is stored and the features password managers provide vary vastly among the available products. Most of the time a master password or master secret is used for authenticating the user to the password manager and prevents unauthorized access.

Retrieval Password Manager

Retrieval password managers store the user's passwords as text and retrieve them when they are needed to authenticate for a service. The sensitive data is protected by encryption. Typically, the master password is used to derive the key used for decrypting stored passwords and encrypting new additions. For practical reasons, most of the time the user identifier and the service name are not encrypted, since the existence of credentials for a specific service is preferably evaluated without requiring the master password. The benefits of retrieval password managers are that the user can store its existing password. However, if the master password is chosen poorly an attacker with access to the encrypted data might succeed in a dictionary or brute force attack [17].

Generative Password Manager

In generative password managers, the passwords are generated on demand. The password is derived from the master password and some identifier for the service such as an URL. This has the benefit that each generated password is a strong and unique password, eliminating password reuse. When setting up a generative password manager the user needs to change the passwords of each service he wants to have managed. This leads to huge friction on the initial setup and is the main drawback of this approach [17].

Cloud-based Password Manager

A password manager is called cloud-based if the data sets are stored in the cloud. The benefits of cloud-based password managers are that the data is accessible from anywhere and trustworthy storage providers are better at protecting the data than the average user [10, 11].

Browser-based Password Manager

Browser-based password managers are closely integrated into web browsers. Most web browsers provide a built-in solution and external password managers can be integrated by installing a browser extension. A high integration allows for convenient features such as seamless autofill and storage of service credentials. Browser-based managers are designed to offer as much convenient as possible. This is great in reducing friction, but also opens the user up to targeted attacks, such as redirect sweep attacks [18]. In built-in password managers, the encryption key is by default derived from the operating system login credentials, reducing the setup process even more [12, 10].

3.3 Progressive Web App

A progressive web app (PWA) is a website that behaves very similar to a native application. PWA's are created like traditional websites but they additionally require a service worker. The service worker provides the app with caching and offline capabilities. Using HTTPS is mandatory for running a PWA. Progressive web apps are an upcoming strategy to web-based cross-platform development [19, 20, 21]. Web applications and thus PWA's are getting more and more access to native functions. The state of implementation can be checked on whatwebcando.today.

3.4 Hybrid Apps

Hybrid applications are created by wrapping a web-based application in native code. In Android a *webview* is used to embed web-based content inside a native application. The webview sends data and events to the web app by initializing the execution of JavaScript code within the web app. The web app, in turn, sends data and events to the native app by calling any function the webview supplies in its *JavaScript interface*. Unfortunately, webview has not the same capabilities as the chrome browser for android and not all the functionality available to the browser is also available for webview.

3.5 Trusted Peer

Peers in terms of computing and distributed networks are equally privileged, equipotent participants of the same application [22]. A peer *A* calls another peer *B* trusted if *A* knows of the existence of *B* and can authenticate and read messages received from *B*.

3.6 Secret Sharing

In secret sharing, a secret is distributed among a group of peers such that it is only accessible under certain conditions. To achieve this the secret is split into n parts, equal to the size of the group. Each share on its own is meaningless. The secret can only be rebuilt collectively. To rebuild the secret the number of collected shares must surpass a previously defined threshold t . This system is also known as a (t, n) -threshold scheme. [23].

Shamir's Secret Sharing

Shamir's secret sharing [24] is form of secret sharing were the shares are created with the help of a polynomial function.

$$a_0 + a_1 * x + a_2 * x^2 + .. + a_n * x^n$$

A polynomial function of degree n is exactly defined given at least $(n + 1)$ points on the function. For a (t, n) -threshold scheme we therefore generate a random polynomial of degree $(t - 1)$. The parameter a_0 is set to the value of the secret S . Then n different points $P_1, P_2, .., P_n$ are sampled from the polynomial. Each point is given to one of the n peers for

safekeeping. The peers can only restore the polynomial and therefore the secret S , if they combine at least t out of the n points [14, 9].

3.7 WebRTC

Web real-time communication (WebRTC) provides browsers and mobile applications with a simple API for real-time communication. The communication takes place peer-to-peer and is secured by standardized and well-known encryption protocols. WebRTC was initially released in 2011 and is open-source.

The establishment of the connection is called signaling. The way signaling messages are exchanged is not defined in the WebRTC implementation, but in most cases, a relay server is put in action. For the signaling process one peer must act as the initiator. To start the signaling, the initiating peer sends a session description protocol (SDP) offer to the passive peer expressing the desire to communicate and parameter negotiation. If this desire to communicate is met the passive peer derives an SDP answer from the received offer and returns it to the initiating peer. The SDP exchange defines the communication parameters used in the connection. Once both parties agree on the extent of the connection they need to negotiate the package routing. WebRTC uses interactive connectivity establishment (ICE) to determine the shortest path. For each node the path passes, an ICE candidate is created. The ICE candidate contains a communication protocol to be used (UDP, TCP), an IP address with a port number, a connection type (see Table 3.1), alongside other information. Every new candidate is sent to the peer. A STUN server is used to determine the public IP address for devices behind a NAT. The first node on the path both peers visit is used to build the connection and is also the shortest path. Figure 3.2 represents the ICE negotiation visually.

Type	Discovery	IP Address Representation
Host	From network interface of host	Host, VPN
Server reflexive (srflx)	Contacting a STUN server	Public IP address
Peer reflexive (prflx)	Contacting the peer	IP address of NAT
Relay	Contacting TURN server	TURN Server

Table 3.1: All ICE types and how they are obtained.

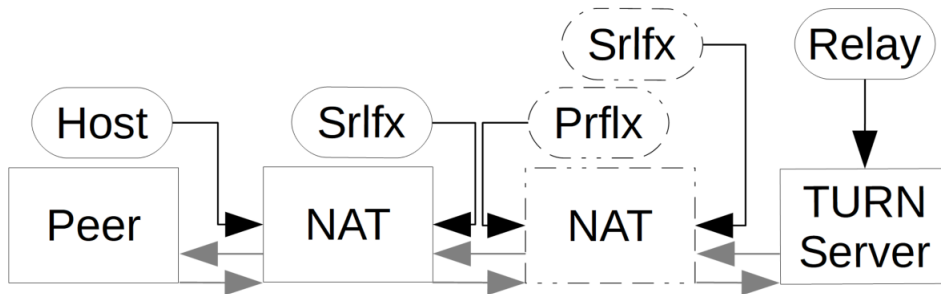


Figure 3.2: ICE candidates in a single and double NAT configuration. In case of multiple NATs only the outer most NAT generates a candidate of type srflx.

3.8 Predecessors of PolyPass

At ETH Zürich, the idea of a distributed password manager was first realized with the Convenient Password Manager. NoKey was follow up project and PolyPass is the third instance of this idea. The goal of these projects is to provide a secure password manager, where the stored passwords are accessible without remembering any password. To achieve this the distributed approach presented in Section 2.3 is applied in combination with Shamir's secret sharing.

Convenient Password Manager

The Convenient Password Manager (CPM) [14] is a generative password manager, that runs on a set of mobile devices in possession of the user. Each device runs an instance of the program and can communicate with all other devices over Bluetooth. The passwords are generated by hashing the concatenation of a secret, service name and login name. All devices have a complete list of service and login names.

Password Groups

To better manage passwords, the Convenient Password Manager introduced the idea of password groups. Each group has a unique secret and threshold to restore it. To access a password group with threshold t_g , t_g devices must be controlled. Password groups make the application more flexible. Valuable passwords are put in groups with higher t_g , while less important passwords are put in groups with a low t_g . A higher t_g requires more devices and more effort to generate the password. However, groups with a high t_g are more resilient to devices being stolen, since an attacker must control at least t_g devices to restore the password.

NoKey

NoKey is a web-based distributed retrieval password manager [9]. The application is implemented as a progressive web application and mainly written in Elm and JavaScript. NoKey is also available as a browser extension for Chrome and Firefox as well as an Android Application.

NoKey was built as a new take on the CPM, with the goal of better usability and user experience. Instead of a generative password manager, NoKey works as a retrieval password manager, since this approach seemed more usable. To make the application available to more devices, the platform was switched from Android to a web-based application. The communication is moved from Bluetooth to a relay server, a more reliable and performant solution. NoKey keeps the concept of password groups but introduces a new UI and adds many additional features from which we will only name a few. For one, NoKey creates a unique icon from the id of the device. This helps in recognizing the devices but also helps to prevent attacks on the pairing process. Also, a tutorial was added, informing the user about the functionality of NoKey. As an additional feature, they introduce the concept of key boxes. A key box holds a unique share for any password group, just like a regular device. The box is protected by a password. The purpose of the key boxes is to provide an additional share in case too few devices are accessible.

Software Design

4.1 Key Challenges

We observed the key challenges of this thesis to be the following:

1. The locality assessment of trusted peer, while conforming to the limitations of a web-based application.
2. Finding the right balance between convenience and security when designing the sharing concept for automatically approving share requests.
3. Improving the UI and provide relevant information but not overwhelm the user.

4.2 PolyPass

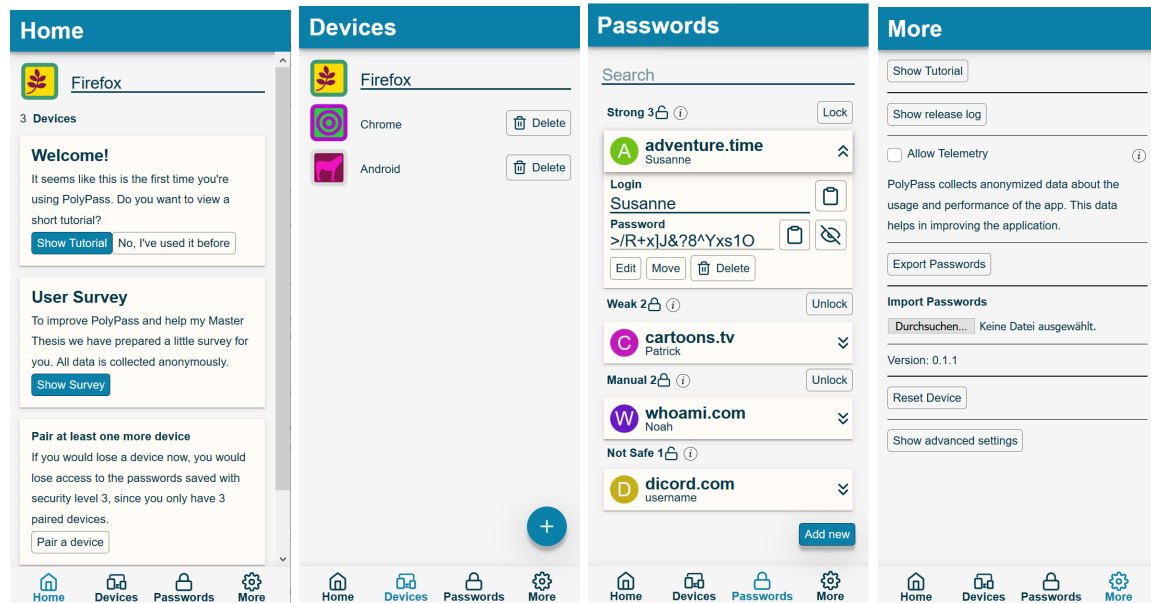
PolyPass is a distributed retrieval password manager and developed as a fork of NoKey. A detailed exposition of NoKey is available in its original paper [9] but we also summarized its concept in Section 3.8. Many parts of PolyPass were inherited from NoKey. The cryptographic operations, the key boxes, the random password generator, and the browser extension to name a few. The main areas PolyPass improved on NoKey are the design of the password groups, the UI, the communication secrecy and the pairing process. We give a brief insight into the general software design of PolyPass, before explaining the more intense changes in the following Sections.

PolyPass secures the user's service credentials not with a master password but by distributing the decryption secret among a set of trusted peers. Each credential is added to one of a few predefined password groups. Each group is encrypted with a group secret, that is split into a (t, n) -threshold scheme and shared among peers. To restore the secret at least t shares are needed. The application is provided as an Android application, browser extension for Chrome and Firefox, and as a progressive web app. The application UI is split into four sections and screenshots are displayed in Figure 4.1.

- The Home screen informs the user about pending tasks and security issues.
- The Devices tab displays and manages all trusted peers.
- The Passwords tab lets the user interact with all the stored passwords.

- The More tab hosts the tutorial, the settings, and the password import and export feature.

The tutorial contains information about PolyPass’s concept and step-by-step instructions. Before PolyPass can be used to its full potential additional devices must be paired to become trusted peers. The pairing and communication process is presented in Appendix A.



(a) Home screen.

(b) Devices tab.

(c) Passwords tab.

(d) More tab.

Figure 4.1: The four main parts of the PolyPass UI.

4.3 Communication

In NoKey, peers use WebSockets and a relay server to communicate. In PolyPass the peers communicate directly with each other, using the WebRTC peer-to-peer protocol. The reason for switching the communication protocol is manifold. First, a peer-to-peer reduces the strain on the relay server. Only the initial connection buildup is sent over the server, which in turn results in a more secretive communication channel and is the second benefit. The server only serves two functions, keeping a state of all connected clients and relaying signaling messages between clients. Whenever a client gets online it first connects to the server via a WebSocket and sends its ID. The server stores the socket identifier and the device ID until the device disconnects again. On receiving a message, the server looks up the recipient’s ID and relays the message on the corresponding socket.

Signaling Process

Before establishing the WebRTC connection the signaling process must be completed. The peers must mutually agree which peer initiates the communication. In case both peers initiate a communication channel at the same time, the signaling process can get stuck. We prevent

this by using the local time as decider when choosing the initiating peer. If A wants send a message to B , at first, A requests the online status of B from the server. If peer B is online A sends its local time, $time_A$, and stores a copy. When B receives $time_A$ there are two possibilities. Either peer B also has tried to initiate the communication or B is idle. In case B is idle it returns $time_A - 1$ and waits for A to initiate. Otherwise B has also sent its time, $time_B$. Now both peers have $time_A$ and $time_B$ and compare the time values. The peer with the lower time value is the initiator. In the unlikely case that both peers have the same time, they repeat this process. Once the peers reach consensus the signaling process is completed by sending the SDP and ICE messages over the server as described in Section 3.7.

4.4 Auto Unlocking

From the Section 3.8 we know that NoKey stores the service credentials in *password groups*. Each of these groups has different security strength. PolyPass extends those groups with a set of new groups that we henceforth call *auto groups*, while the original NoKey groups are referred to as *manual groups*. As the name implies the manual groups are only unlockable with explicit user consent. The auto groups can be unlocked automatically if certain conditions apply. The auto groups can always be unlocked with a user interaction if the circumstances permit an automatic unlock.

4.4.1 Proximity Parameters

Auto groups must only be unlocked automatically in a secure context. A set of parameters called *proximity parameters* is defined to quantify the security of the context. The proximity parameters are evaluated between two devices. The parameters approximate the validity of the assumption that the same person is in possession of both peers. We define proximity by evaluating the device's locality with the help of GPS and the affiliation to a local network. The value of the parameter can be one of four possibilities.

1. **No Proximity:** The other device is not nearby
2. **GPS Proximity:** The GPS location of the devices are within a certain range of each other
3. **Network Proximity:** The devices are in the same network and the network is a local network
4. **Mixed Proximity:** The network proximity is fulfilled and the GPS location of the evaluating device is within the range of a user-set location.

Since PolyPass is running as a web application, it is limited to functions accessible within the browser environment. In Section 3.3 we mentioned that PWA's have access to native functions and the GPS is one of them. The coordinates are provided by the browser via the *navigator.geolocation* [25]. Figuring out the network proximity is not that obvious. The browser cannot access any command-line interface to probe its network. Fortunately, WebRTC provides connection statistics [26] that disclose the local IP addresses of both devices.

The statistics also reveal the selected connection, which we use to evaluate the network proximity. We can extract the ICE candidates used to establish the connection and extract the IP address from each candidate [27]. If the IP addresses are local addresses the connection is within a local network. During our research, we have not seen this approach used before.

4.4.2 Share Request

Whenever a peer needs to restore a secret, a share request is sent to every trusted peer. This process is illustrated in Figure 4.2. If the requested group is an auto group special steps are taken to determine the proximity parameters. When sending a share request for an auto group the current device location is sent to the peer alongside the share request. If the received request is for a manual group, a notification with the request gets displayed to the user. Otherwise, the proximity parameters are obtained and evaluated against the values required to unlock the requested group. In case the parameters suffice, the requested share is sent immediately else the user needs to confirm the request manually. A trusted peer only receives share requests when online. This fact is not obvious to all users, therefore we implemented push notifications that appear upon a share request when PolyPass is closed.

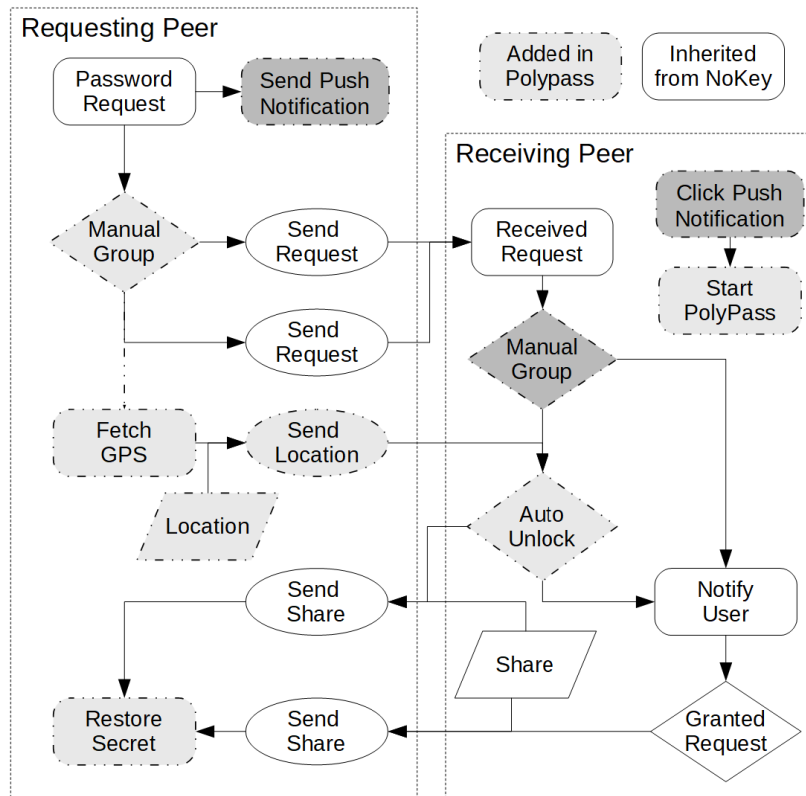


Figure 4.2: High-level structure of a share request. Additions by PolyPass are highlighted in grey.

In PolyPass we have two kinds of password groups. The simple groups and the complex groups. The category of simple groups contains all the manual groups and two auto groups. Manual groups can be created in different security strengths, by changing the threshold t when splitting the group secret into a (t, n) -scheme. The lowest possible t is 1, meaning the

group is not protected since every device already has one share. The simple auto groups are *strong* with a threshold of 3 and *weak* with a threshold of 2. Each of the simple auto groups releases the share if one of their associated proximity parameters is evaluated *true* or the user confirms the request manually. The strong group only grants the share if the mixed proximity is fulfilled. The weak group releases its share if any proximity is valid. The associated parameters for each group are listed in Table 4.1a.

Password Group		Proximity Parameter		
Name	Required Shares	Mixed	Network	GPS
Manual Groups	1 to n			
Strong	3	✓		
Weak	2	✓	✓	✓

(a) Each share for a simple password groups is released under different conditions.

Sub-Group		Proximity Parameter		
Name	Required Shares	Mixed	Network	GPS
Strong _{sub}	2	✓		
Weak _{sub}	3	✓	✓	
GPS Group	2			✓

(b) The medium group contains 3 sub-groups that release their share under different conditions.

Table 4.1: Password groups and their unlock conditions

To showcase the structure of a complex password group we look at the design of the *medium* password group in more detail. Complex groups allow for a finer grading of the security strength applied to the password group. The medium group is composed of two sub-groups, a simple group *strong_{sub}* with a threshold of 2 and another complex group called *weak_{sub}* with a threshold of 3. Unlike the simple groups, for the *weak_{sub}* group the group secret is not split into n shares but rather $n + 1$ shares. Each of the n trusted peers needs one share. The remaining share is split into n shares with a threshold of 2, creating the *GPS* group. The process is summarized in Figure 4.3. The different sub-groups and the conditions under which they release their share is outlined in Table 4.1b. The *Strong_{sub}* group grants the share request if the devices are in mixed proximity to each other. *weak_{sub}* releases its share in case of network proximity and therefore also the mixed proximity. The share of the GPS group is sent when the GPS proximity holds true. The GPS group exists to prevent GPS spoofing attacks, since only one share of the *weak_{sub}* group can be obtained using the GPS proximity.

4.5 Android Application

Even though NoKey is a progressive web app and does not necessarily need a native application to run on Android, it still provides one. The native application displays the web app in a webview and provides access to the camera and the file system. However, PolyPass needs a native app to access the Android autofill service and the Android accessibility service. The autofill service is used to fill credentials for native applications. Since the autofill service cannot fill credentials into websites, we also need the accessibility service. Both services need to use an activity as a bridge to the web app because services can not run a webview. A

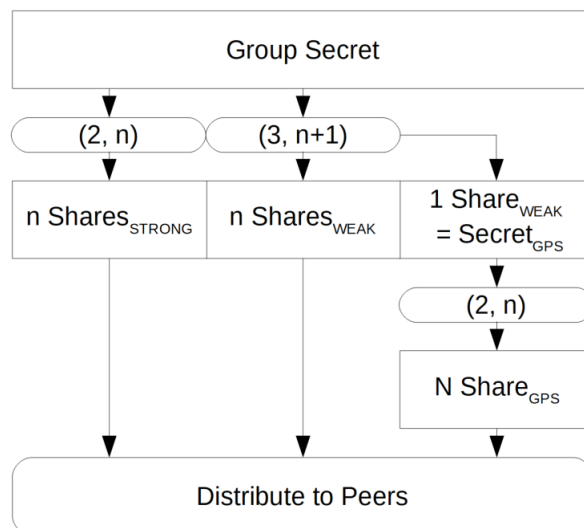


Figure 4.3: The complex password group medium is composed of multiple sub-groups. (t, n) represents splitting the secret into a (t, n) -threshold scheme according to Section 3.6. The total number of trusted peers is n .

webview is required to interact with the web-based application through a JavaScript interface. The *autofill activity* has the responsibility of relaying messages but also displays the web app. The interaction between these components is depicted in Figure 4.4 and Figure 4.5. Both services run in the background and receive events from the operating system.

Figure 4.4 shows the high-level implementation of the autofill service. Whenever the Android system wants our service to potentially autofill a native application it calls the service's *onFillRequest* function. At first, the view is searched for any field that can be filled. If some fields are detected the user is notified and PolyPass awaits confirmation before proceeding. The appearing notification can be seen in Figure 4.6b. On confirmation, the autofill activity is opened, which is also displayed in Figure 4.6c. The search box is immediately filled with the service name and only relevant credentials are listed. The user then selects the correct set of credentials. As soon as the group containing the credentials is unlocked, the now decrypted credentials are delivered to the autofill activity. Finally, the activity fills the username and password into the corresponding fields previously discovered and terminates.

The accessibility service is designed to aid users with disabilities, but we re-purpose it to function as an autofill service. The accessibility service gets called on different events, but we are only interested in those indicating a change of the user interface. Events that originate from the system or that are otherwise undesired are ignored. If the event indicates an opportunity to assist the user, the UI is scanned for text fields for credentials. If we find such fields the service creates a notification as seen in Figure 4.6a to promote the occasion of auto-filling. In case no fields are found any foregone notification is removed. Once the notification is selected, the autofill activity starts and the process continues as mentioned beforehand. Filling the credentials is slightly different. The autofill activity does not fill the credentials but sends them to the service, which finishes the job.

In case the user has exactly one password that fits the service name the credentials can

be retrieved and filled without the user ever interacting. One can argue that this is the most convenient solution. Considering that filling automatically is considered a major security gap, PolyPass deliberately forgoes this path. When PolyPass is launched from the home screen, the web app rendered in a webview. The webview is run in the *main activity*, which also provides camera, storage, and GPS access as well as push notifications. The main activity and autofill activity do not share any function and are separated to reduce code clutter.

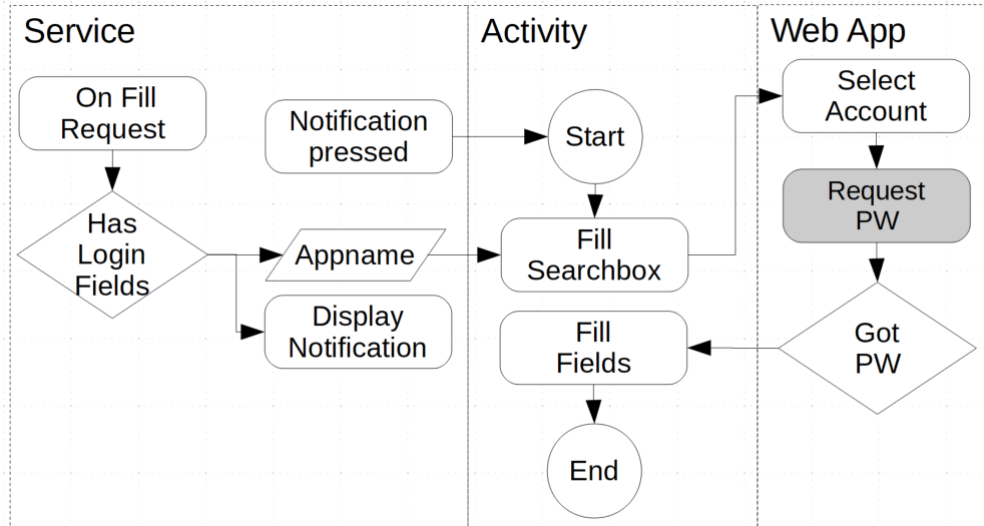


Figure 4.4: The high-level logic of the Android autofill service.

4.6 User Study

In this thesis, we initially planned to conduct a user study to provide some insight into the performance and usage of PolyPass but also to collect user feedback and past experiences with different approaches of password management. We incorporated a survey into the web application which, for the user, reduce the inconvenience of opening an external tool just to submit a survey. We anonymize the collected data before it is sent to our database. To measure the performance we added the ability to continuously submit telemetry data. The telemetry registers any share request and response with a timestamp, the requested group, the result of the request, and the involved devices. It also collects the number of stored passwords per group and the number and type of paired devices.

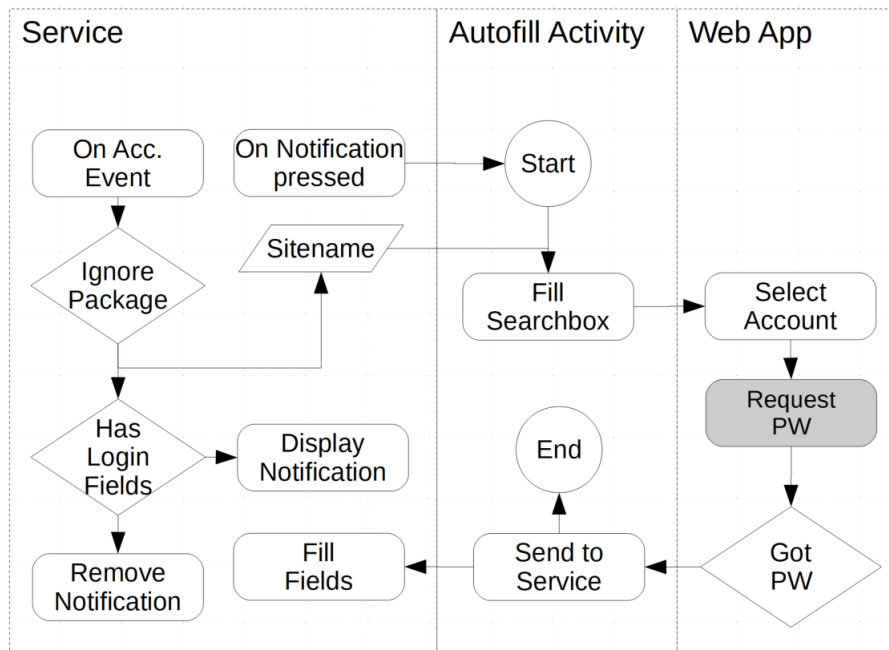
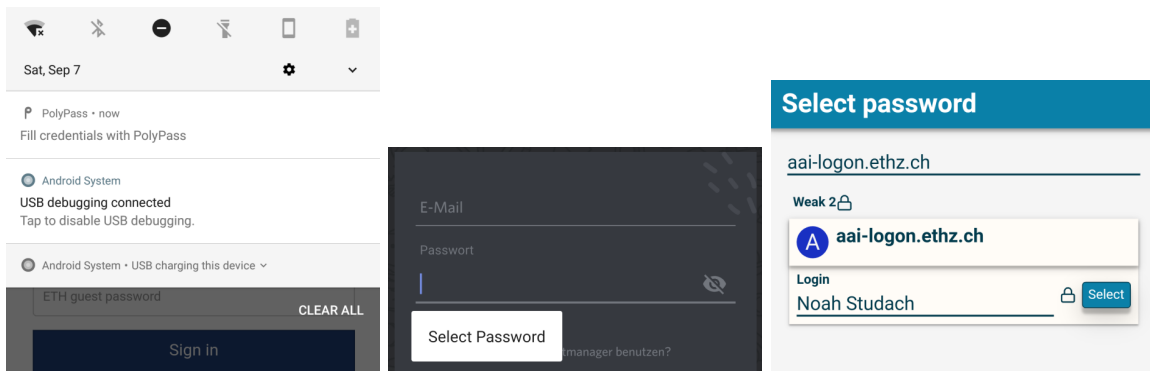


Figure 4.5: The process flow of filling a password with the Android accessibility service.



(a) The accessibility service creates a notification in the notification drawer
 (b) The notification from the autofill service appear in the currently active app.
 (c) The password selection screen within the autofill activity.

Figure 4.6: Android service notifications and the autofill activity.

Discussion

5.1 Performance

We advocate that the addition of auto groups improves the user experience of PolyPass, however, the convenience of not manually confirming every share request is easily outweighed if the unlocking is not fast enough. We measured the duration and success rate of share request. For the measurement, three devices were paired and we created one password in the weak, medium, strong, and manual group with threshold 2. We unlocked the groups a total of 51 times, with the devices in the same network, in the same location, and a known location. Under these conditions, all auto shares should release their share without any interaction. Only shares of the group strong failed to release and did so twice. The average duration correlated with the restrictions opposed by the auto groups and is displayed in Table 5.1. We observed that the average performance is respectable and in line with our expectations. It must be noted that consecutive request for auto groups benefit from cached GPS data and that the duration of the manual group is heavily dependent on the reaction time of the user. Nevertheless, in 3 instances the auto group strong took significantly longer, 10, 14 and 32 seconds. This shows that there is room for improvement. In our current implementation, the request evaluation is controlled by a timer. On every tick, the proximity parameters are checked for completeness. In case all proximity parameters are collected, they are evaluated and either the share is released or the user is notified. This process can be improved by running it sequentially and also by informing the user instantly such that the request could be confirmed before the proximity parameters are evaluated. The measurement data can be found in Appendix D.

Group	Weak	Average	Strong	Manual
Avg. Duration [ms]	2025	2704	4297	4478

Table 5.1: The average unlock duration in milliseconds per password group.

Our test was conducted with all proximity conditions met but during our sporadic real-world testing and our research, we identified an issue with the GPS accuracy. On Windows, only the Edge browser can request a high precision GPS request. Since PolyPass is designed to run on Chrome and Firefox it will not receive accurate GPS locations unless they are cached from a previous request by Edge. To our knowledge, there is currently no elegant way of reliably providing high precision. With wider adoption of IPv6, it is likely that a peer connect to a network with a public IP address. In this case, the network proximity is always evaluated as false even if both peers are within the same local network. In Section 5.3.1 we

show that a public IP must lead to declining network proximity. We accept the occurrence of this false negative in favor of increased security. During the creation of PolyPass, we also encountered issues when establishing a WebRTC connection. The issues resulted in the signaling process not completing. We believe this issue is fixed but we did not sufficiently test the current implementation on all possible platforms and circumstances. On the other hand, the peer-to-peer communication introduces some overhead for establishing the connection, however, once the channel is opened the peer-to-peer communication is assumed to be faster than relaying messages through a server. To minimize the overhead, on start-up, every device instantly tries to initiate a communication channel to each trusted peer.

5.2 Implementation

PolyPass spans over multiple programming languages and involves many different technologies to achieve its goals. During this thesis, the effort to read up on all involved architectures and combining the selected technologies has been underestimated. Also, NoKey is missing in-depth documentation and as source code is difficult to follow. Those circumstances have shifted the timetable to an extent where parts of the project had to be omitted. Even though the survey mentioned in Section 4.6 was ready to be distributed, the time constraint would have limited the quantity and quality of the results beyond reason. In our opinion, the implementation generally requires more testing and polishing, which are both time-intensive tasks.

5.3 Security

For the security assessment, we assume the user has not yet removed any stolen device or else all communication attempts from the compromised device will be ignored. We evaluate the security of the proximity parameters under the following assumption. An adversary can steal a trusted device. Any stolen device is fully controlled by the adversary and he can modify any part of the program to gain an advantage. Additionally, the attacked device is online but none of the proximity conditions are met unless otherwise stated. We also assume that the user is not manually allowing share requests.

5.3.1 Proximity Parameters

The adversary cannot influence the calculation of any proximity parameter because this process is executed locally. In some cases, the adversary can send modified messages that trick peers into wrongfully releasing their share. We first look at the GPS proximity, which is an easy target for any skilled adversary. To send a fake location the adversary can either modify the code or simply spoof the location in the browser. The network proximity, on the other hand, is safe. A device can not be tricked into believing the network proximity is valid if the peer is not in the same network. The network proximity is evaluated through the selected ICE pair of the WebRTC connection. The network proximity is true only if both ICE candidates are of type host and both associated IP addresses fall within the range dedicated to local addresses [28]. If the attacker uses a valid ICE candidate but changes the type to host, the associated IP address is not a local address. If the adversary modifies its

ICE candidate and changes the associated IP address to a fake local address, the connection can not be established. Thus the network proximity cannot be used to release any share.

If the network proximity can not be forced the mixed proximity can also not be forced unless the adversary is connected to the local network. We assume the adversary is in the same network, but it is not a known location. Otherwise, the device would correctly assume mixed proximity. The adversary must trick the peer to be in a known location. Since the evaluation of any parameter is performed locally, the adversary must push a new known location with the target’s coordinates to the target. We implemented the concept of safe settings to protect against this attack. The known locations are stored in the safe settings. If a change to the safe settings is received from a peer, the user must confirm the changes before they are applied. Unless the user accepts a malicious location, the mixed proximity is protected against attacks.

5.3.2 Password Groups

For manual groups the security assessment made in the NoKey’s paper still hold. To gain access to a manual group with threshold t at least t devices must be controlled. Since any share can be released with manual confirmation, by inversion of argument, controlling k devices grants access to any group with threshold $t = < k$. We know that only the GPS proximity can be spoofed and be used to obtain all the shares of the weak group and exactly one share of the $weak_{sub}$ sub-group of the medium group. The weak group only requires one additional share and is therefore accessible to the adversary if he controls one device. The $weak_{sub}$ group requires 3 shares. One is on the stolen device and is obtained by GPS spoofing. The third share cannot be spoofed because it only releases under network proximity. If the adversary either controls another device or is within the local network, he gains access to the $weak_{sub}$ and ergo the medium group. The medium group can also be unlocked via $strong_{sub}$ but only if the adversary gains access to the local network in that particular known location. The same holds true for the strong password group.

5.3.3 Communication Channel

We evaluate the security of our communication channel. All communication between trusted peers is secured twice. Once by the WebRTC protocol and once by PolyPass using two RSA key pair. One key pair verifies the authenticity and integrity of the message, the other is used to encrypt confidential data. More details can be found in Appendix A.3. The RSA layer is identical to NoKey and its security is not discussed further.

We first evaluate attacks on an already established WebRTC connection. All WebRTC connections are protected via DTLS and support perfect forward secrecy (PFS) depending on the selected cipher suite. For example, the WebRTC implementation of Firefox mandates a cipher suite with PFS. In the rare case of non-PFS, recorded messages can be decrypted once access to one of the involved peers is gained. The connection to the signaling server takes place over a secure WebSocket with SSL/TLS and is subject to the same attacks.

When establishing a new connection a man-in-the-middle can modify the ICE candidate collection and redirect the connection to a malicious host [29]. The same holds true if an attacker gains control over the signaling server and modifies the signaling messages. Since all non-signal messages are encrypted by PolyPass, sensitive data is not compromised. Any

message sent by an attacker impersonated as a trusted peer are not authenticated correctly and are ignored.

Finally, the adversary can launch an attack on the pairing process. If the pairing server acts maliciously the peers can be paired with the adversary's device. For this reason, NoKey implemented the concept of hash icons that represent the device ID. If the user pays attention, the attack is spotted and the malicious device can be removed.

5.3.4 Denial of Service

An attacker can cause a denial of service by target the signaling server as previously mentioned. In case the adversary managed to become a trusted peer, he can remove all trusted peers. To revert this attack the user needs to pair all devices again. Another attack a trusted peer can launch is to modify the application state and remove all passwords, before synchronizing the state with all peers. PolyPass does not prevent this attack.

Summary and Future Work

6.1 Summary

In this day and age, online services have evolved to the point that they are an integral part of many peoples daily lives. All the more it is in their interest to keep their online accounts safe. But remembering complex passwords is hard and users often choose to skip on recommended security practices, leaving them vulnerable to attacks on their credentials. Popular password managers still require a master password or biometric authentication. Users are still forced to remember one complex password and the common biometric authentication with fingerprint scanners are not particularly secure[30]. We presented our web-based password manager PolyPass, a solution to this problem. PolyPass implements a distributed approach to data security. Instead of a master password, PolyPass utilizes the user's devices as proof of ownership. The user data is encrypted as usual but the encryption secret is split with Shamir's secret sharing and each device stores a part of it. The passwords can only be decrypted once shares are combined to restore the initial secret. The devices communicate peer-to-peer using WebRTC, which increases the communication secrecy in regards to prior projects. To enhance the user experience some password PolyPass unlocks selected passwords under secure circumstances. The secure context is evaluated by utilizing GPS positioning and a novel approach of local network detection with the help of WebRTC. The PolyPass Android app implements autofill functionality for both native applications and websites. PolyPass has not yet received the amount of testing such a security-relevant application deserves. Even though PolyPass is a functioning application, many improvements can still be made towards increased convenience.

6.2 Future Work

The future development process can be heavily improved by refactoring the Elm codebase to better match existing data structure with the techniques presented in Appendix C. Also continuing to improve the documentation in and outside of the code base, reduces the initial hurdle for new contributors. Another point for improvement is the support for Apple devices. Safari does not support the web crypto API that PolyPass is currently using. Updating or replacing it would open up the accessibility to many potential users. We can increase the reliability by offering to set multiple servers for WebRTC signaling or provide Bluetooth as a local communication channel. Browsers support Bluetooth LE and can interact with Android devices. Finally, conducting the survey would provide the necessary insight to plan

the long-term development of PolyPass.

Bibliography

- [1] S. A. Zhang, S. Pearman, L. Bauer, and N. Christin, “Why people (don’t) use password managers effectively,” in *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: <https://www.usenix.org/conference/soups2019/presentation/pearman>
- [2] Y. Li, H. Wang, and K. Sun, “A study of personal information in human-chosen passwords and its security implications,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.
- [3] R. Wash, E. Rader, R. Berman, and Z. Wellmer, “Understanding password choices: How frequently entered passwords are re-used across websites,” in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. Denver, CO: USENIX Association, Jun. 2016, pp. 175–188. [Online]. Available: <https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash>
- [4] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, “Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms,” in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 523–537.
- [5] M. Dell’ Amico, P. Michiardi, and Y. Roudier, “Password strength: An empirical analysis,” in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.
- [6] A. Ng, “Massive breach leaks 773 million email addresses, 21 million passwords,” <https://www.cnet.com/news/massive-breach-leaks-773-million-emails-21-million-passwords/>, Accessed: 02.09.2019.
- [7] N. Alkaldi, K. Renaud, and L. Mackenzie, “Encouraging password manager adoption by meeting adopter self-determination needs,” pp. 4824–4833, January 2019. [Online]. Available: <http://eprints.gla.ac.uk/169744/>
- [8] N. Alkaldi and K. Renaud, “Why do people adopt, or reject, smartphone password managers?” August 2016. [Online]. Available: <http://eprints.gla.ac.uk/120760/>
- [9] F. Zinggeler, “Nokey - a distributed password manager,” 2018. [Online]. Available: <https://pub.tik.ee.ethz.ch/students/2017-HS/MA-2017-24.pdf>
- [10] R. Zhao and C. Yue, “Toward a secure and usable cloud-based password manager for web browsers,” *Computers & Security*, vol. 46, pp. 32 – 47, 2014. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167404814001059>
- [11] R. Zhao, C. Yue, and K. Sun, “Vulnerability and risk analysis of two commercial browser and cloud based password managers,” *ASE Sci J*, vol. 1, pp. 1–15, 01 2013.
- [12] C. Boja, “Security survey of internet browsers data managers,” *CoRR*, vol. abs/1112.5760, 2011. [Online]. Available: <http://arxiv.org/abs/1112.5760>

- [13] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot, “Tapas: Design, implementation, and usability evaluation of a password manager,” in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 89–98. [Online]. Available: <http://doi.acm.org/10.1145/2420950.2420964>
- [14] C. G. Manuel Eggimann, “Convenient password manager,” 2016. [Online]. Available: <https://pub.tik.ee.ethz.ch/students/2015-HS/GA-2015-06.pdf>
- [15] M. D. Nils Quermann, Marian Harbach, “The state of user authentication in the wild,” 2018. [Online]. Available: <https://wayworkshop.org/2018/papers/way2018-quermann.pdf>
- [16] C. Shen, T. Yu, H. Xu, G. Yang, and X. Guan, “User practice in password security: An empirical study of real-life passwords in the wild,” *Computers & Security*, vol. 61, pp. 130 – 141, 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167404816300657>
- [17] D. McCarney, “Password managers: Comparative evaluation, design, implementation and empirical analysis,” 2013. [Online]. Available: <https://curve.carleton.ca/8e5c251c-54f8-43b9-a3c0-b847b47b989a>
- [18] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, “Password managers: Attacks and defenses,” in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 449–464. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver>
- [19] A. Bjørn-Hansen, T. A. Majchrzak, and T.-M. Grønli, “Progressive web apps: The possible web-native unifier for mobile development.” in *WEBIST*, 2017, pp. 344–351.
- [20] I. Malavolta, “Beyond native apps: Web technologies to the rescue! (keynote),” in *Proceedings of the 1st International Workshop on Mobile Development*, ser. Mobile! 2016. New York, NY, USA: ACM, 2016, pp. 1–2. [Online]. Available: <http://doi.acm.org/10.1145/3001854.3001863>
- [21] D. Sheppard and D. Sheppard, *Beginning Progressive Web App Development*. Springer, 2017.
- [22] “Wikipedia: Peer-to-peer,” <https://en.wikipedia.org/wiki/Peer-to-peer>, Accessed: 30.08.2019.
- [23] “Wikipedia: Secret sharing,” https://en.wikipedia.org/wiki/Secret_sharing, Accessed: 30.08.2019.
- [24] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: <http://doi.acm.org/10.1145/359168.359176>
- [25] “Mdn web docs: Navigator.geolocation,” <https://developer.mozilla.org/en-US/docs/Web/API/Navigator/geolocation>, Accessed: 04.09.2019.

- [26] “Mdn web docs: Rtcstatstype,”
<https://developer.mozilla.org/en-US/docs/Web/API/RTCStatsType>,
Accessed: 04.09.2019.
- [27] “Mdn web docs: Rtcicecandidatestats,”
<https://developer.mozilla.org/en-US/docs/Web/API/RTCIceCandidateStats>,
Accessed: 04.09.2019.
- [28] “Wikipedia: Private network,”
https://en.wikipedia.org/wiki/Private_network,
Accessed: 30.08.2019.
- [29] A. Keränen, C. Holmberg, and J. D. Rosenberg, “Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal,” *RFC*, vol. 8445, pp. 1–100, 2018.
- [30] P. Bontrager, A. Roy, J. Togelius, N. Memon, and A. Ross, “Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution*,” in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, Oct 2018, pp. 1–9.
- [31] “Dashlane,”
<https://www.dashlane.com>,
Accessed: 04.09.2019.
- [32] “Lastpass,”
<https://www.lastpass.com>,
Accessed: 04.09.2019.
- [33] “Safeincloud password manager,”
<https://safe-in-cloud.com>,
Accessed: 04.09.2019.
- [34] “awallet password manager,”
<http://www.awallet.org>,
Accessed: 04.09.2019.
- [35] “ewallet,”
<https://www.iliumsoft.com/ewallet>,
Accessed: 04.09.2019.
- [36] “Keeper,”
<https://keepersecurity.com>,
Accessed: 04.09.2019.
- [37] “1password,”
<https://1password.com>,
Accessed: 04.09.2019.
- [38] “Splashid 8,”
<https://splashid.com>,
Accessed: 04.09.2019.

- [39] “msecure,”
<https://www.msecure.com>,
Accessed: 04.09.2019.
- [40] “Enpass,”
<https://www.enpass.io>,
Accessed: 04.09.2019.
- [41] “Roboform,”
<https://www.roboform.com>,
Accessed: 04.09.2019.
- [42] “Bitwarde,”
<https://bitwarden.com/>,
Accessed: 04.09.2019.

Pairing Process

A.1 Notation

On creation a PolyPass device A generates the following objects

- ID of A: ID_A
- Asymmetric key pair for encrypting: $(K_{E_A}, K_{E_A}^{-1})$
- Asymmetric key pair for signing: $(K_{S_A}, K_{S_A}^{-1})$

K represents the public key and K^{-1} the associated private key. Sending an encrypted packet from A to B is represented as

$$A \rightarrow B : \{Data\}_{Key}$$

where the data is encrypted with the public key of A . Each device has shared data noted as *SharedData*, which contains all the ID's and keys of trusted devices as well as the encrypted passwords. The server is denoted as S . Upon entering the pairing process, a device generates a random ID, notated ID_R .

A.2 Pairing

Register at server with ID_R

$$A \rightarrow S : \{ID_R\}$$

B receives ID_R from the user

$$B \rightarrow S \rightarrow ID_R : \{ID_B\}$$

$$A \rightarrow S \rightarrow B : \{ID_A\}$$

A and B establish WebRTC connection

$$B \rightarrow A : \{ID_R\}$$

$$A \rightarrow B : \{ID_R\}$$

Exchange shared data containing the public keys

$$A \rightarrow B : \{ID_R, SharedData_A\}$$

$$B \rightarrow A : \{ID_R, SharedData_B\}$$

Merge shared data to $SharedData_{AB}$

$$A \rightarrow B : \{ID_R, SharedData_{AB}\}$$

$$B \rightarrow A : \{ID_R, SharedData_{BA}\}$$

A.3 Data Exchange

All messages from $A \rightarrow B$ are signed with the senders private signing key $K_{S_A}^{-1}$. The decryption with K_{S_A} ensures the origin of the data is A .

$$A \rightarrow B : \{message\}_{K_{E_A}^{-1}}$$

When sending shares the data is additionally encrypted with the receiver's public encryption key $K_{E_B}^{-1}$.

$$A \rightarrow B : \{\{share\}_{K_{E_B}}\}_{K_{S_A}^{-1}}$$

A.4 NoKey Communication

For the sake of completeness we also present the pairing and communication process of NoKey. The data exchange is the same as in PolyPass but instead sending the message directly from A to B it must be relayed over the server

$$A \rightarrow S \rightarrow B : \{Data\}_{Key}$$

Instead of the device, in NoKey the server generates tokens on pairing requests. A token generated for a request from A is denoted as S_A .

A.4.1 Pairing

Exchange token and IDs

$$A \rightarrow S : \{ID_A\}$$

$$S \rightarrow A : \{S_A\}$$

B receives S_A from the user

$$B \rightarrow S : \{S_A\}$$

$$S \rightarrow A : \{ID_B\}$$

$$S \rightarrow B : \{ID_A\}$$

Exchange shared data containing the public keys

$$A \rightarrow S \rightarrow B : \{S_A, SharedData_A\}$$

$$B \rightarrow S \rightarrow A : \{S_A, SharedData_B\}$$

Both peers merge shared data to $SharedData_{AB}$

$$A \rightarrow S \rightarrow B : \{S_A, SharedData_{AB}\}$$

$$B \rightarrow S \rightarrow A : \{S_A, SharedData_{BA}\}$$

Password Manager Analysis

We selected popular password managers from the Google Play Store, shown in Table B.1 and from the chrome web store, displayed in Table B.2. The applications from Google Play store were selected from the top 100 apps listed on appfigures.com under the category *Productivity*. The chrome web store apps were found by going through the search results for "*password manager*" on the [chrome web store](https://chrome.google.com/webstore) with country set to Switzerland and language to German. The applications were briefly analyzed in regards to their feature set and security model (see Table B.3). All of them use the same underlying security model, which we call the *local* approach. Encrypting the sensitive data with a user-defined master password. They differ in the cryptographical algorithms they use and in how often the algorithm is applied. All of them apply what they call a *Zero-Knowledge* principle, which means that data is only encrypted and decrypted at the local machine. Some also call it *No-Knowledge*, to not confuse it with the better known zero-knowledge proof.

Since users dislike entering complex passwords, most of the applications also support biometric and 2-factor authentication. All password managers provide some form of data synchronization. but often the encrypted data is stored locally and in the cloud [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42].

Rank	Name	Price	Distributor
26	LastPass PM	Free	LogMeIn, Inc.
99	Dashlane PM	Free	Dashlane

(a) Ranking Play Store: Free

Rank	Name	Price	Distributor
8	PM SafeInCloud Pro	\$4.99	SafeInCloud
29	aWallet Cloud PM	\$4.49	Synpet
74	eWallet - PM	\$9.99	Ilium Software

(b) Ranking Play Store: Paid

Rank	Name	Price	Distributor
6	PM - Keeper	Free	Keeper Security, Inc.
13	Dashlane PM	Free	Dashlane
16	LastPass PM	Free	LogMeIn, Inc.
28	1Password - PM and Secure Wallet	Free	AgileBits
58	SplashID Safe PM	Free	SplashData, Inc
67	mSecure - PM	Free	mSeven Software LLC
72	Enpass PM	Free	Sinew Software Systems Pvt. Ltd.
91	PM SafeInCloud Pro	\$4.99	SafeInCloud
92	RoboForm PM	Free	Siber Systems Inc

(c) Ranking Play Store: Grossing

Table B.1: Password manager Play Store ranking within Top 100 of category *Productivity*. Evaluated on 03. Sept. 2019

Name	Reviews
Keeper	5.981
LastPass	28.700
Dashlane	1.364
RoboForm	1.224
Bitwarden	1.170

Table B.2: Password managers in chrome web store with more than 1000 reviews and average rating of 4+/5 Stars. Evaluated on 04. Sept. 2019

Name	Web-Based	Cloud-Supported	No-Knowledge	Approach	2-FA	Biometrics
Dashlane	x	x	x	Local		
LastPass	x	x	x	Local	x	x
SafeInCloud PM		x	x	Local		x
aWallet PM		x	x	Local		x
eWallet		x	x	Local		
Keeper	x	x	x	Local	x	x
1Password	x	x	x	Local	x	x
SplashID 8	x	x	probably	Local		
mSecure		x	x	Local		x
Empass		x	x	Local		x
RoboForm	x	x	x	Local	x	
Bitwarden	x	x	x	Local	x	x

Table B.3: Password manager features: Evaluated from official website on 04. Sept. 2019

Elm

Elm is a pure and functional programming language. The architecture of Elm is centered around data structures and a program written in Elm is very reminiscent of a state machine. Therefore Elm is well suited to write the control logic of any web application. Any application programmed in Elm must be compiled. The compiler produces highly optimized JavaScript code that can be integrated into web applications exactly like JavaScript code. Elm provides ports to interact with parts of the application written in JavaScript. Data flowing through these Elm ports must be well defined. This prevents data with the wrong type declarations from being sent to the control logic and creating run-time errors.

C.1 Data Types

Elm is a statically typed language. Every variable needs to have a data type assigned. Additionally, to the basic types, Elm offers to create *custom types*. A custom type must be well defined and all possible values this type can have must be listed in its definition. Elm also supports *type aliases*, which is a well-defined group of data objects. We can combine custom types and type aliasing to build complex data structures, that are easy to read. In our example below, the type PeerStatus can be one of the values Online, Offline or Connected. The data type Peer has two properties, peerId which is type String and status which is type PeerStatus. The value of type Sender is either NotAuthorized or Authorized with an object of type Peer attached.

```
type PeerStatus
  = Online
  | Offline
  | Connected

type alias Peer =
  { peerId : String
    , status : PeerStatus }

type Sender
  = NotAuthorized
  | Authorized Peer
```

We can use custom types and type aliases to simplify complex control logic. The Elm compiler makes sure that every possible case is covered by the control logic. The pseudo-code

below *does nothing* if the sender is not authenticated or offline. If the sender is connected a message is sent, else a connection is established.

```

case sender of
  NotAuthorized ->
    "Do nothing"
  Authorized trustedPeer ->
    case trustedPeer.status of
      Online ->
        "Establish Connection with trustedPeer.peerId"
      Offline ->
        "Do nothing"
      Connected ->
        "Send message to trustedPeer.peerId"

```

C.2 Programming Pattern

Any Elm program is split into three logically separated parts. *Model*, *Update* and *View*, seen in Figure C.1. The Model represents the programs data structure. The Model can be any data structure but in any program with reasonable complexity, it will be a type alias. The content of the model is changed via the Update function. The View function represents the state as HTML code.

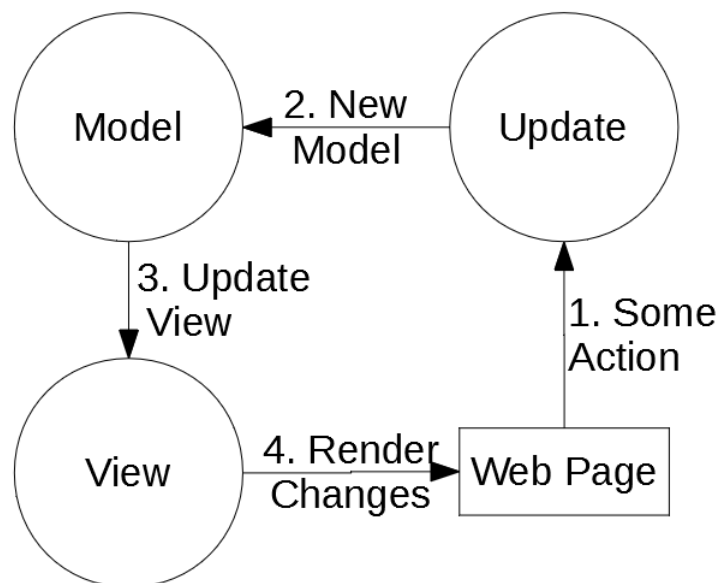


Figure C.1: The Elm Architecture and application life cycle.

In more complex applications, a selection of each part can be split into a sub-model, depicted in Figure C.2. This might look more complicated in this high-level structure but it functionally partitions the program and reduces code clutter immensely.

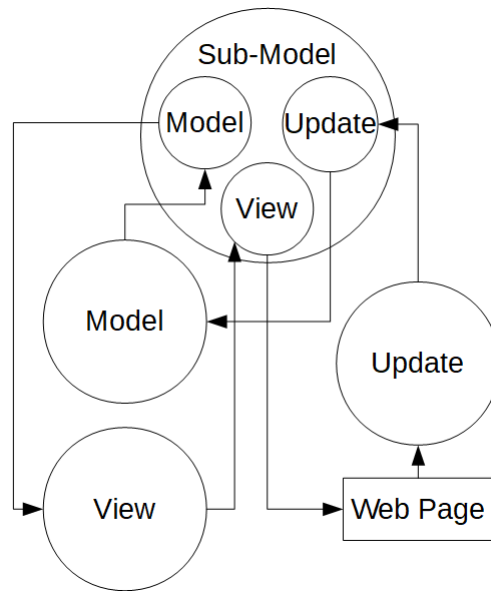


Figure C.2: The Elm Architecture and application life cycle.

Performance Measurements

We evaluated the performance of the auto unlocking by unlocking several password groups. The devices were within the same local network, at the same GPS location and in a known location. Therefore, all proximity parameter should evaluate to true and each auto group request should be fulfilled without any user interaction. The results were captured using the telemetry tool build for the user study (see Section 4.6). The success rate of the proximity detection was 49 out of 51. The raw data is listed in Table D.2 and Table D.3. The average duration to unlock each group is present in Table 5.1. We observed that the duration of unlocking is not higher than unlocking a group manually and the more restrictive the security group is, the longer is the unlocking duration. This statement has to be taken with the caveat that consecutive request for auto unlocking profit from cached GPS data and the duration of the manual group is heavily dependent on the reaction time of the user. Also some of the auto groups took significantly longer, up to 30 seconds (see Table D.1). The data for the duration measurements are stored in Table D.4, D.5 and D.6.

Timestamp	Type	Group	Peer	Duration
1568044258246	ReceivedShare	(3,Strong)	Peer B	32174
1568044275197	ReceivedShare	(3,Strong)	Peer B	10273
1568044279044	ReceivedShare	(3,Strong)	Peer B	13827

Table D.1: Request where the auto unlocking took significantly more time.

Timestamp [ms]	Result	Group	Auto Unlock
1568043806396	GrantAutomatically	(2,Weak)	True
1568043808215	GrantRequestManually	(2,Manual)	True
1568043836099	GrantAutomatically	(3,Average)	True
1568043842109	GrantAutomatically	(3,Average)	True
1568043846708	GrantAutomatically	(3,Average)	True
1568043856535	GrantAutomatically	(3,Average)	True
1568043859249	GrantAutomatically	(3,Average)	True
1568043881171	GrantAutomatically	(2,Weak)	True
1568043883635	GrantAutomatically	(2,Weak)	True
1568043886160	GrantAutomatically	(2,Weak)	True
1568043888658	GrantAutomatically	(2,Weak)	True
1568043890780	GrantAutomatically	(2,Weak)	True
1568043893228	GrantAutomatically	(2,Weak)	True
1568043894263	GrantAutomatically	(2,Weak)	True
1568043910609	GrantRequestManually	(3,Strong)	
1568044231989	GrantAutomatically	(3,Strong)	True
1568044257441	GrantAutomatically	(3,Strong)	True
1568044264401	GrantAutomatically	(3,Strong)	True
1568044265573	GrantAutomatically	(3,Strong)	True
1568044274507	GrantRequestManually	(3,Strong)	
1568044279219	GrantAutomatically	(3,Strong)	True
1568044280888	GrantAutomatically	(3,Strong)	True
1568044280943	GrantAutomatically	(3,Strong)	True
1568044284243	GrantAutomatically	(3,Strong)	True
1568044284443	GrantAutomatically	(3,Strong)	True
1568044286893	GrantAutomatically	(3,Strong)	True
1568044287053	GrantAutomatically	(3,Strong)	True
1568044289783	GrantAutomatically	(3,Strong)	True
1568044290123	GrantAutomatically	(3,Strong)	True
1568044292665	GrantAutomatically	(3,Strong)	True
1568044292999	GrantAutomatically	(3,Strong)	True
1568044299823	GrantAutomatically	(2,Weak)	True
1568044299823	GrantAutomatically	(2,Weak)	True
1568044301489	GrantAutomatically	(3,Strong)	True
1568044301915	GrantAutomatically	(3,Strong)	True
1568044310908	GrantRequestManually	(2,Manual)	True
1568044312344	GrantRequestManually	(2,Manual)	True
1568044324216	GrantAutomatically	(3,Strong)	True
1568044324220	GrantAutomatically	(3,Strong)	True
1568044326583	GrantAutomatically	(2,Weak)	True
1568044326914	GrantAutomatically	(2,Weak)	True

Table D.2: Part 1: Success rate of unlocking request. All proximity parameters should evaluate to true. 49/51 evaluated correctly

Timestamp [ms]	Result	Group	Auto Unlock
1568044330079	GrantRequestManually	(2,Manual)	True
1568044336519	GrantAutomatically	(2,Weak)	True
1568044336659	GrantAutomatically	(2,Weak)	True
1568044347374	GrantAutomatically	(3,Strong)	True
1568044347375	GrantAutomatically	(3,Strong)	True
1568044348863	GrantAutomatically	(2,Weak)	True
1568044348963	GrantAutomatically	(2,Weak)	True
1568044354361	GrantRequestManually	(2,Manual)	True
1568044356970	GrantAutomatically	(3,Strong)	True
1568044357096	GrantAutomatically	(3,Strong)	True

Table D.3: Part 2: Success rate of unlocking request. All proximity parameters should evaluate to true. 49/51 evaluated correctly.

Timestamp	Type	Group	Peer	Duration
1568043831101	ShareRequest	(2,Average)		
1568043835765	ReceivedShare	(3,Average)	Peer B	4664
1568043836687	ReceivedShare	(3,Average)	Peer A	5586
1568043841332	ShareRequest	(2,Average)		
1568043842883	ReceivedShare	(3,Average)	Peer B	1551
1568043842883	ReceivedShare	(3,Average)	Peer A	1551
1568043846052	ShareRequest	(2,Average)		
1568043846370	ReceivedShare	(3,Average)	Peer A	318
1568043847495	ReceivedShare	(3,Average)	Peer B	1443
1568043851602	ShareRequest	(2,Average)		
1568043856056	ReceivedShare	(3,Average)	Peer B	4454
1568043857073	ReceivedShare	(3,Average)	Peer A	5471
1568043858613	ShareRequest	(2,Average)		
1568043858943	ReceivedShare	(3,Average)	Peer A	330
1568043860283	ReceivedShare	(3,Average)	Peer B	1670
1568043805571	ShareRequest	(2,Manual)		
1568043808445	ReceivedShare	(2,Manual)	Peer A	2874
1568044304235	ShareRequest	(2,Manual)		
1568044311177	ReceivedShare	(2,Manual)	Peer B	6942
1568044312164	ReceivedShare	(2,Manual)	Peer A	7929
1568044328118	ShareRequest	(2,Manual)		
1568044330377	ReceivedShare	(2,Manual)	Peer A	2259
1568044352209	ShareRequest	(2,Manual)		
1568044354597	ReceivedShare	(2,Manual)	Peer A	2388
1568043801436	ShareRequest	(2,Weak)		
1568043806507	ReceivedShare	(2,Weak)	Peer A	5071
1568043807583	ReceivedShare	(2,Weak)	Peer B	6147
1568043876232	ShareRequest	(2,Weak)		
1568043881307	ReceivedShare	(2,Weak)	Peer B	5075
1568043881307	ReceivedShare	(2,Weak)	Peer A	5075
1568043882964	ShareRequest	(2,Weak)		
1568043883816	ReceivedShare	(2,Weak)	Peer A	852
1568043883988	ReceivedShare	(2,Weak)	Peer B	1024
1568043885474	ShareRequest	(2,Weak)		
1568043886450	ReceivedShare	(2,Weak)	Peer A	976
1568043886589	ReceivedShare	(2,Weak)	Peer B	1115

Table D.4: Part 1: Duration of unlocking request. All proximity parameters should evaluate to true.

Timestamp	Type	Group	Peer	Duration
1568043887914	ShareRequest	(2,Weak)		
1568043888838	ReceivedShare	(2,Weak)	Peer A	924
1568043888987	ReceivedShare	(2,Weak)	Peer B	1073
1568043890124	ShareRequest	(2,Weak)		
1568043891039	ReceivedShare	(2,Weak)	Peer A	915
1568043891169	ReceivedShare	(2,Weak)	Peer B	1045
1568043892365	ShareRequest	(2,Weak)		
1568043893052	ReceivedShare	(2,Weak)	Peer A	687
1568043893097	ReceivedShare	(2,Weak)	Peer B	732
1568043893489	ShareRequest	(2,Weak)		
1568043894524	ReceivedShare	(2,Weak)	Peer A	1035
1568043894654	ReceivedShare	(2,Weak)	Peer B	1165
1568044294903	ShareRequest	(2,Weak)		
1568044299920	ReceivedShare	(2,Weak)	Peer B	5017
1568044300022	ReceivedShare	(2,Weak)	Peer A	5119
1568044325994	ShareRequest	(2,Weak)		
1568044326375	ReceivedShare	(2,Weak)	Peer A	381
1568044326752	ReceivedShare	(2,Weak)	Peer B	758
1568044335510	ShareRequest	(2,Weak)		
1568044336872	ReceivedShare	(2,Weak)	Peer B	1362
1568044336929	ReceivedShare	(2,Weak)	Peer A	1419
1568044348271	ShareRequest	(2,Weak)		
1568044349003	ReceivedShare	(2,Weak)	Peer A	732
1568044349164	ReceivedShare	(2,Weak)	Peer B	893
1568043900524	ShareRequest	(3,Strong)		
1568043910479	ReceivedShare	(3,Strong)	Peer A	9955
1568044226072	ShareRequest	(3,Strong)		
1568044231786	ReceivedShare	(3,Strong)	Peer A	5714
1568044258246	ReceivedShare	(3,Strong)	Peer B	32174

Table D.5: Part 2: Duration of unlocking request. All proximity parameters should evaluate to true.

Timestamp	Type	Group	Peer	Duration
1568044263624	ShareRequest	(3,Strong)		
1568044264371	ReceivedShare	(3,Strong)	Peer A	747
1568044264924	ShareRequest	(3,Strong)		
1568044265217	ReceivedShare	(3,Strong)	Peer A	293
1568044275197	ReceivedShare	(3,Strong)	Peer B	10273
1568044279044	ReceivedShare	(3,Strong)	Peer B	13827
1568044280028	ShareRequest	(3,Strong)		
1568044280692	ReceivedShare	(3,Strong)	Peer A	664
1568044281659	ReceivedShare	(3,Strong)	Peer B	1631
1568044283610	ShareRequest	(3,Strong)		
1568044283946	ReceivedShare	(3,Strong)	Peer A	336
1568044284891	ReceivedShare	(3,Strong)	Peer B	1281
1568044286284	ShareRequest	(3,Strong)		
1568044286654	ReceivedShare	(3,Strong)	Peer A	370
1568044287627	ReceivedShare	(3,Strong)	Peer B	1343
1568044289124	ShareRequest	(3,Strong)		
1568044289553	ReceivedShare	(3,Strong)	Peer A	429
1568044290838	ReceivedShare	(3,Strong)	Peer B	1714
1568044292066	ShareRequest	(3,Strong)		
1568044292594	ReceivedShare	(3,Strong)	Peer A	528
1568044293525	ReceivedShare	(3,Strong)	Peer B	1459
1568044300897	ShareRequest	(3,Strong)		
1568044301144	ReceivedShare	(3,Strong)	Peer A	247
1568044302724	ReceivedShare	(3,Strong)	Peer B	1827
1568044319369	ShareRequest	(3,Strong)		
1568044324784	ReceivedShare	(3,Strong)	Peer B	5415
1568044324784	ReceivedShare	(3,Strong)	Peer A	5415
1568044342368	ShareRequest	(3,Strong)		
1568044347039	ReceivedShare	(3,Strong)	Peer A	4671
1568044347039	ReceivedShare	(3,Strong)	Peer B	4671
1568044356171	ShareRequest	(3,Strong)		
1568044356912	ReceivedShare	(3,Strong)	Peer A	741
1568044357868	ReceivedShare	(3,Strong)	Peer B	1697

Table D.6: Part 3: Duration of unlocking request. All proximity parameters should evaluate to true.