Ain Shams University
Faculty of Computer and Information Sciences
Computer Systems Department

جامعة عين شمس
كلية الحاسبات والمعلومات
قسم نظم الحاسبات

TwoKey — Password Manager

with a Physical Access Token

By:

Ahmed Wael Elmayyah

Danya Rizk Hamouda

Omar Hesham Fathy

Nouran Khaled Ibrahim

Ahmed Yasser Abdellah

Under Supervision of:

Dr. Karim A. Emara,

Assistant Professor, Computer Systems Department,

Faculty of Computer and Information Sciences,

Ain Shams University.

TA. Abdulrahman Ali,

Teaching Assistant, Computer Systems Department,

Faculty of Computer and Information Sciences,

Ain Shams University.

Acknowledgment

It has been a great opportunity to gain experience and work on a real project, utilising all of the knowledge gained in four years of studying at the Faculty of Computer and Information Science, Ain Shams University.

We thank all of the FCIS staff and management for all of the knowledge and experience they granted us across our years of study.

Special thanks to our graduation project supervisor Dr. Karim Emara, who has guided us with great advice, reviewing, encouragement, and support with a deep technical perspective during the entire year.

Special thanks to TA. Abdulrahman Ali, who has helped us throughout the semester in coordination and reviewing our progress constantly.

A huge thanks to our friend and FCIS-ASU graduate Adam Zahran who gave us many useful tips for designing the architecture of the project.

Thanks to anyone else who helped us throughout the entire project, that includes connections and people who helped us in the survey before implementing the project. A special thank you to all of the open source developers and organisations who open source their libraries and software such as Davy Landman, rxi, Qt, Crypto++, OpenSSL and many other contributors that help developers develop better software.

Abstract

The use of online services is increasing day after day and these services have become necessary to our day-to-day lives. Most, if not all of the services, require an account for each user, and most services have independent accounts. This makes it hard to keep track of all the different passwords for different accounts. This may lead to the user choosing insecure passwords which are easy to memorise or storing their passwords in an insecure way such as a plaintext file, an unencrypted online note or writing them on a piece of paper or in a notebook. Even worse, a user may use a single password for all accounts, making all of his accounts vulnerable in the case that the single password was leaked or cracked.

Password managers began to gain popularity as a solution to the aforementioned problems. They encrypt and store the users' different passwords for different services, allowing the user to access them using a single strong key or password, often referred to as the “Master Password”. However, this solution can sometimes be vulnerable due to poor security practices and the use of weak 2FA methods like SMS-based authentication.

TwoKey is a password manager which uses a unique physical token as a 2FA method for every user. Alongside the master password which is used for encrypting the passwords and more cybersecurity practices such as memory scrubbing, encryption, hashing, challenge-response based authentication, and secure communication, TwoKey is a secure password manager aimed at users which care more about security than they care about convenience.

Table of Content

List of Figures

Charts

Chart 1.1

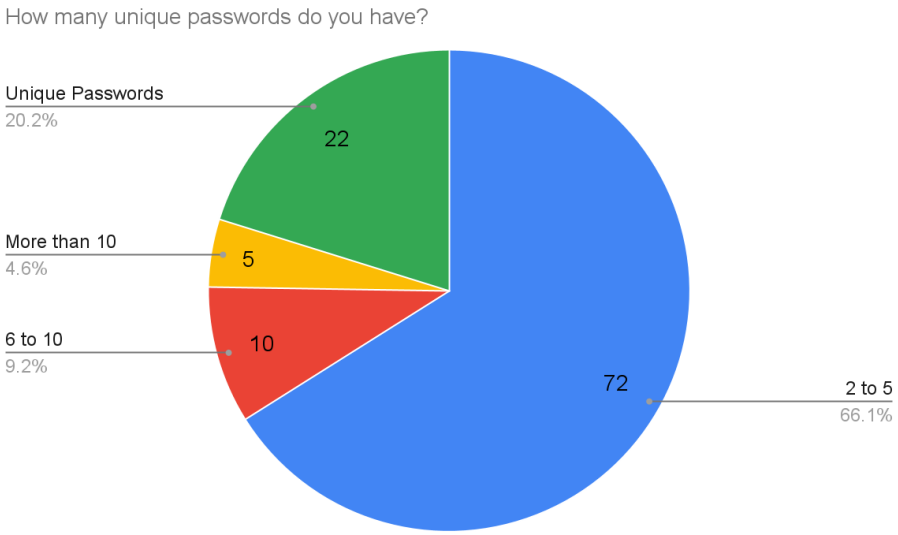


Chart 1.2

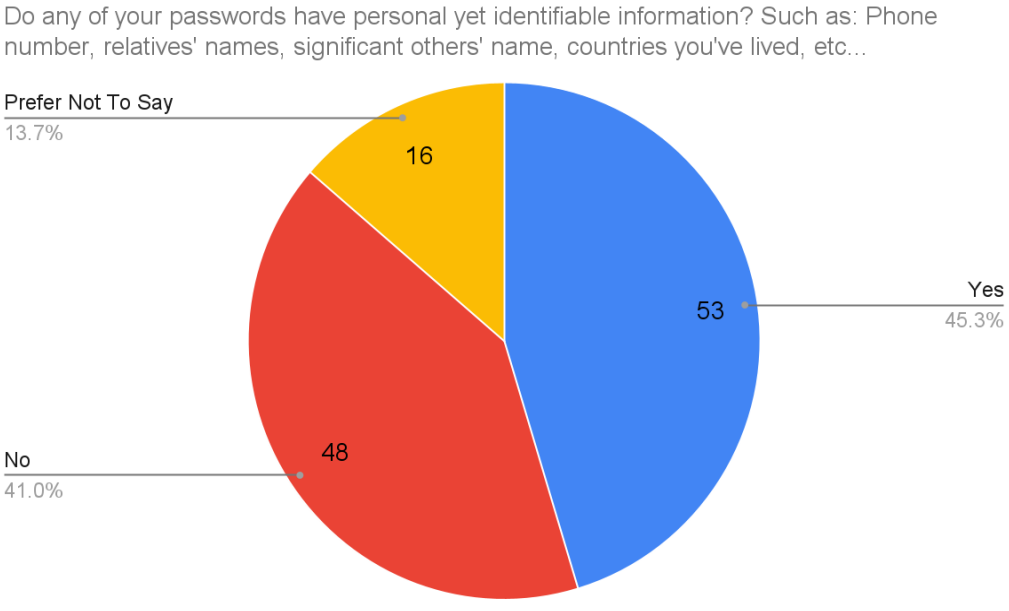


Chart 1.3

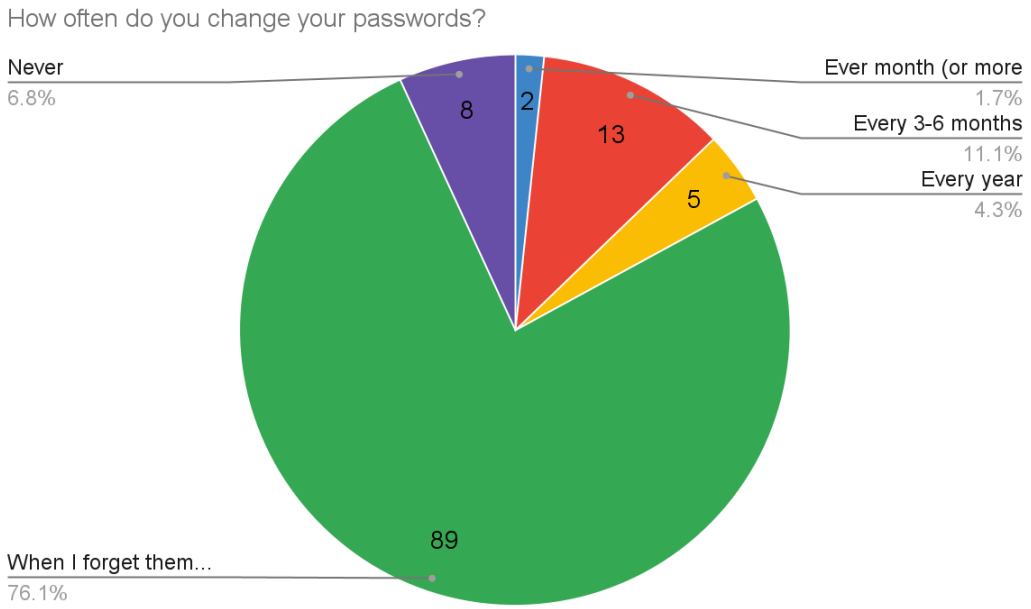
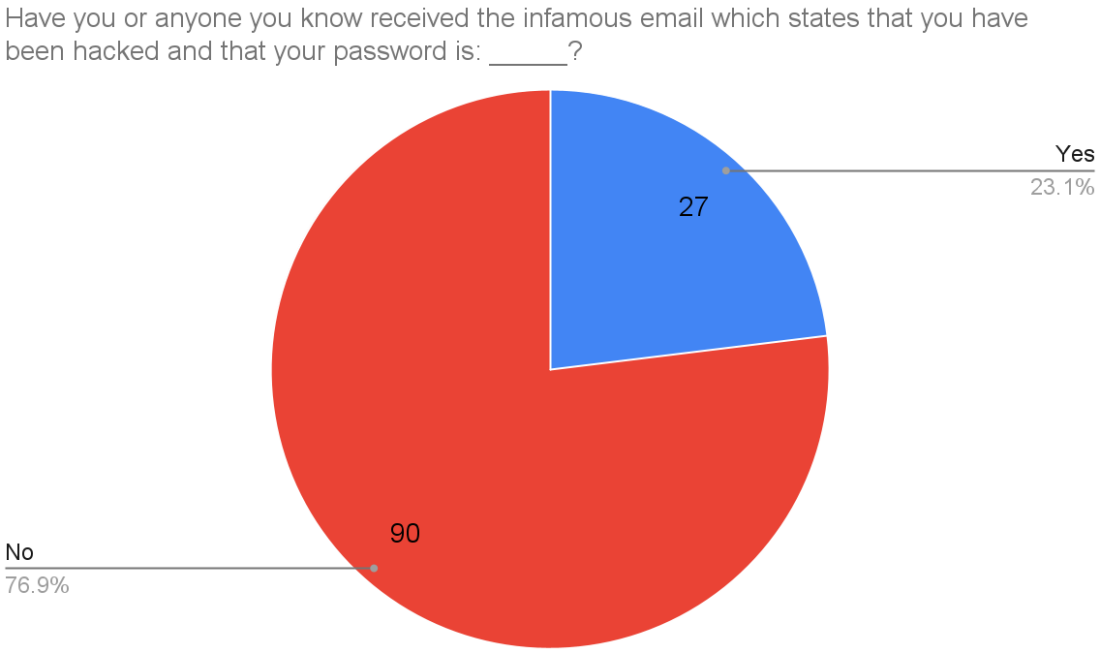


Chart 1.4



Figures

Figure 1.1

Hello!
I'm a member of an international hacker group.

As you could probably have guessed, your account [redacted] was hacked, because I sent message you from it.

Now I have access to you accounts!
For example, your password for [redacted] is c0yhu48e3q96

Within a period from July 17, 2018 to October 3, 2018, you were infected by the virus we've created, through an [redacted] you've visited.
So far, we have access to your messages, social media accounts, and messengers.
Moreover, we've gotten full damps of these data.

Transfer $800 to our Bitcoin wallet: 14bXUoPwruptLamUfKTuMW39Qy1q4ohX9w
If you don't know about Bitcoin please input in Google "buy BTC". It's really easy.

I guarantee that after that, we'll erase all your "data" :)

A timer will start once you read this message. You have 48 hours to pay the above-mentioned amount.

Your data will be erased once the money are transferred.
If they are not, all your messages and videos recorded will be automatically sent to all your contacts found on your devices at the moment of infection.

You should always think about your security.
We hope this case will teach you to keep secrets.
Take care of yourself.

Figure 2.1

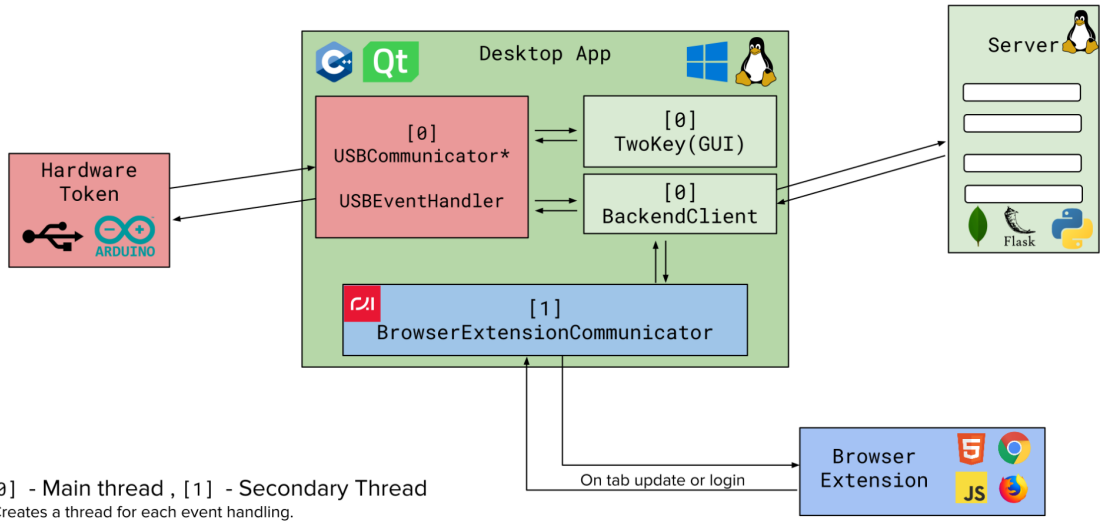


Figure 4.1

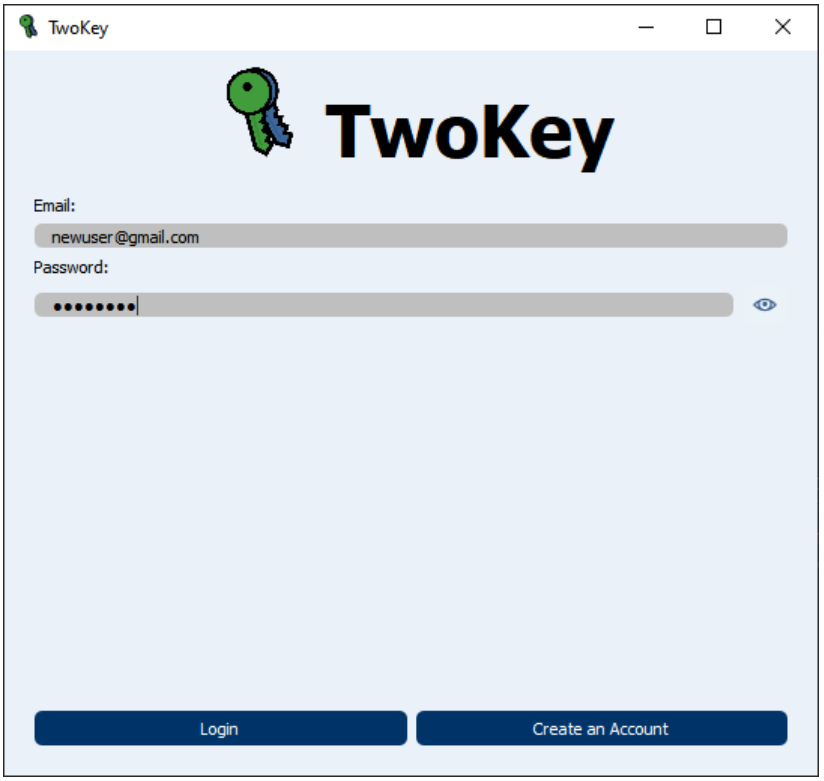


Figure 4.2

Figure 4.3

Figure 4.4

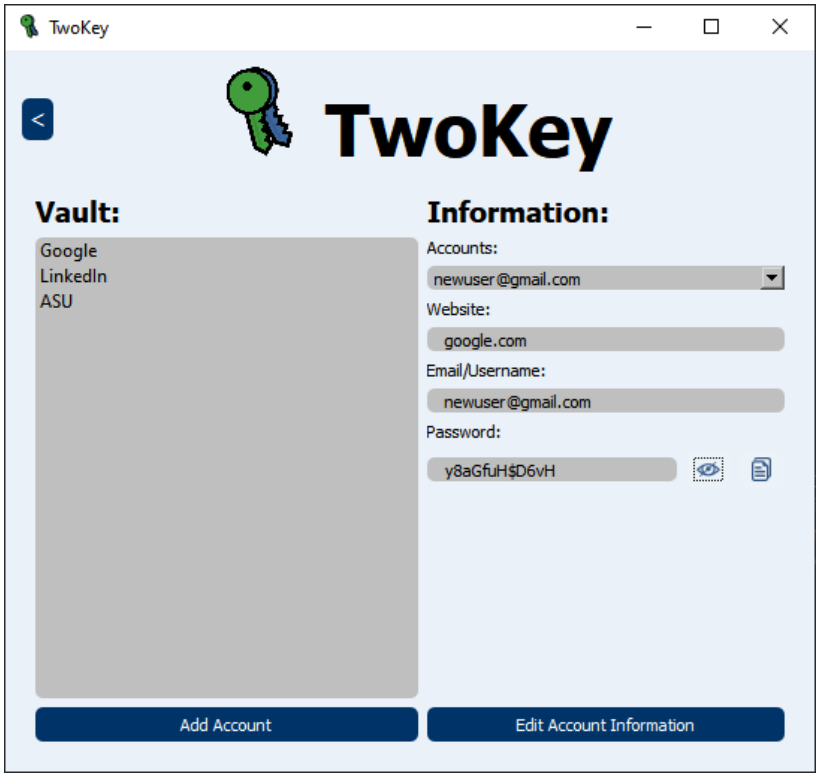


Figure 4.5

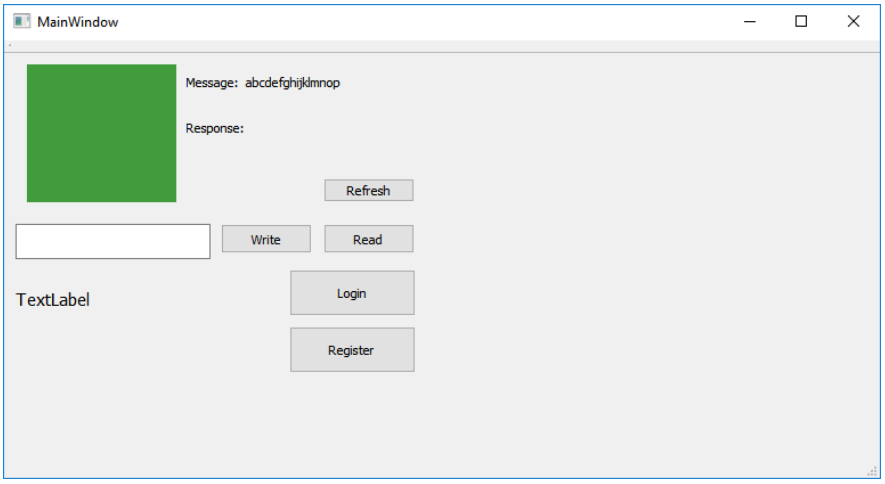


Figure 4.6

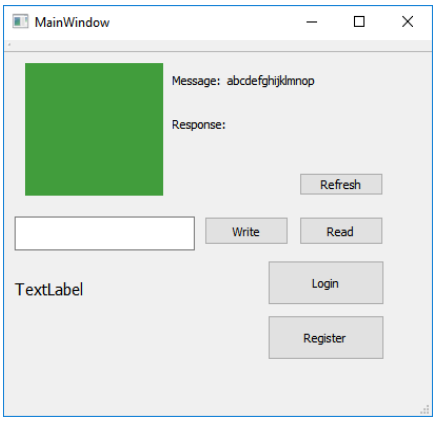


Figure 4.7

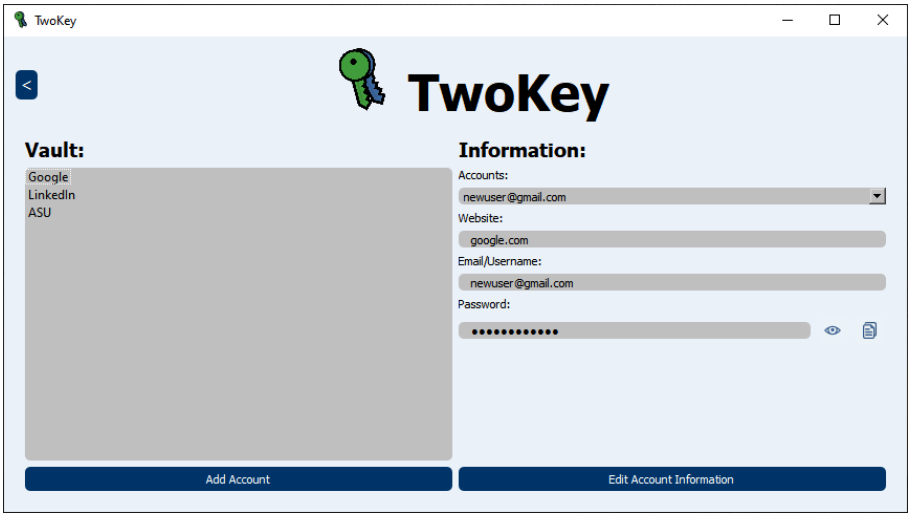


Figure 4.8

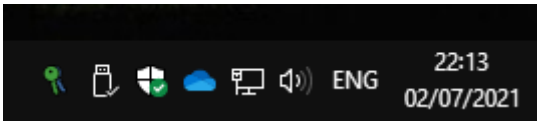


Figure 4.9

Figure 4.10

Figure 4.11

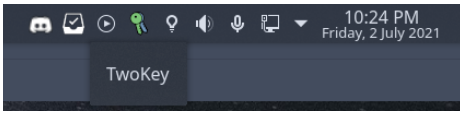


Figure 4.12

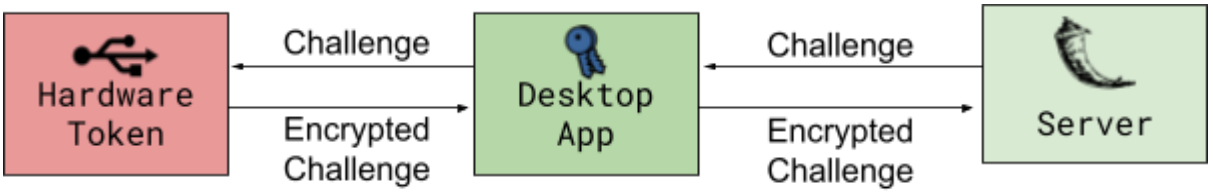


Figure 4.13

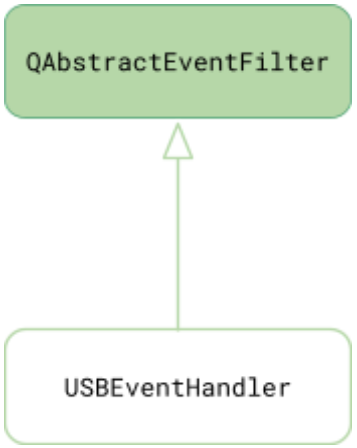


Figure 4.14

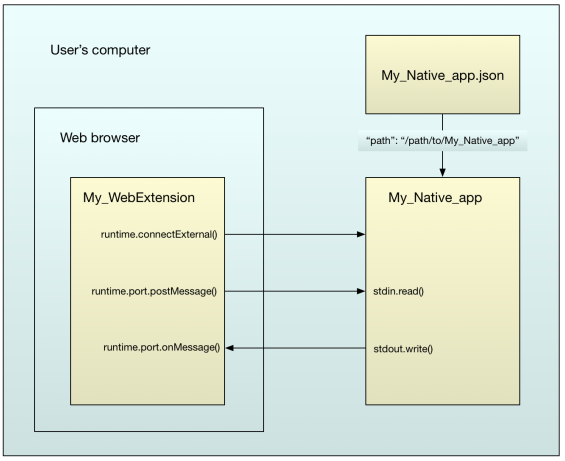


Figure 4.15

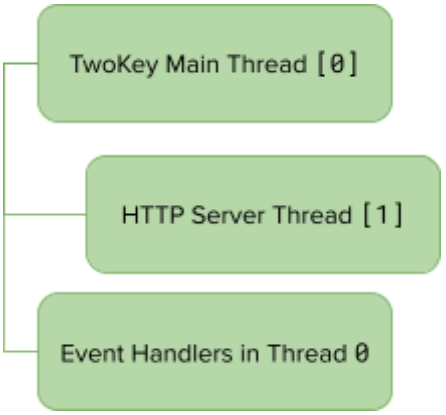


Figure 4.16

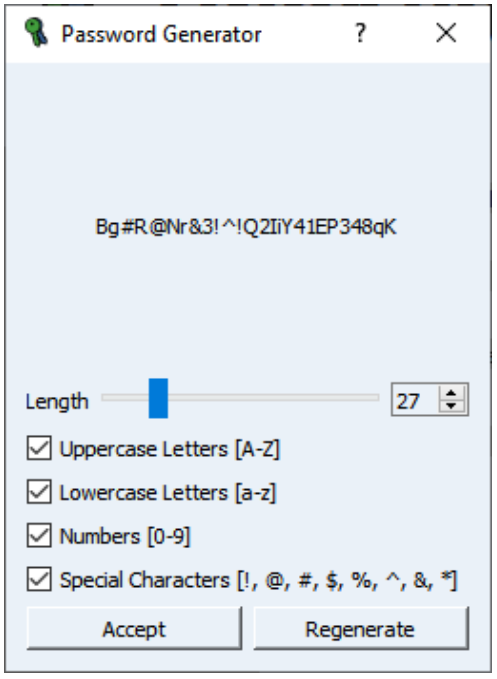


Figure 4.17

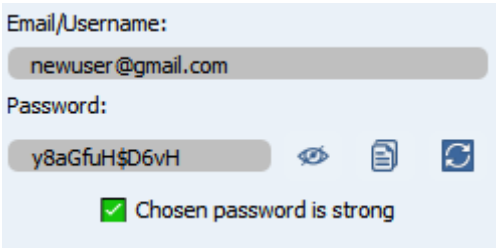


Figure 4.18

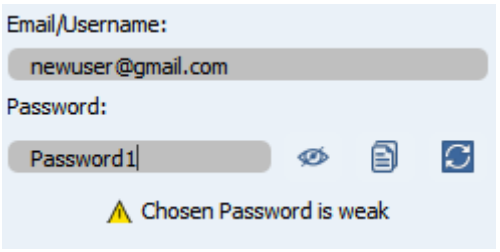


Figure 4.19

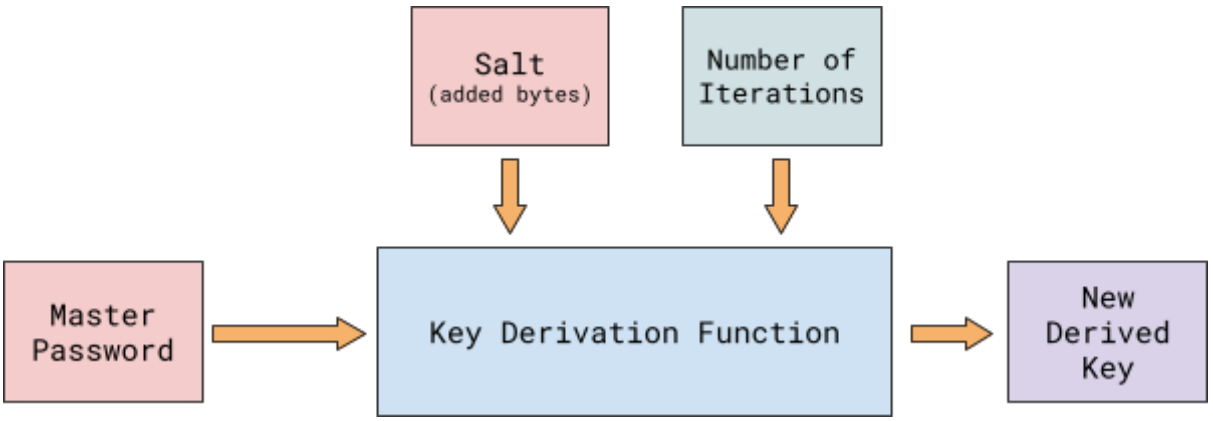


Figure 5.1

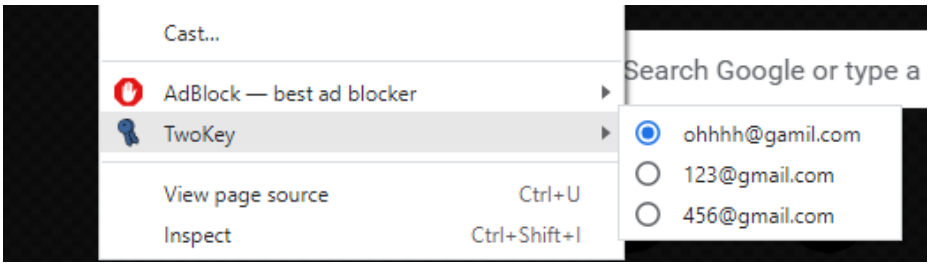


Figure 5.2

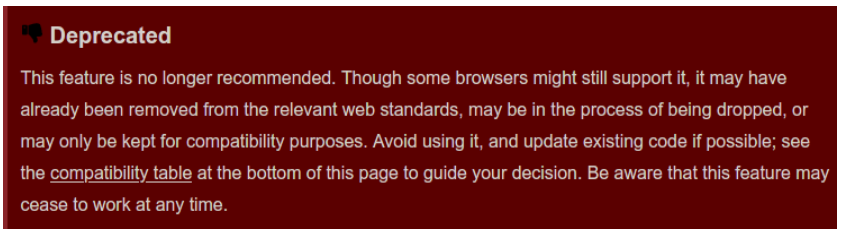


Figure 6.1

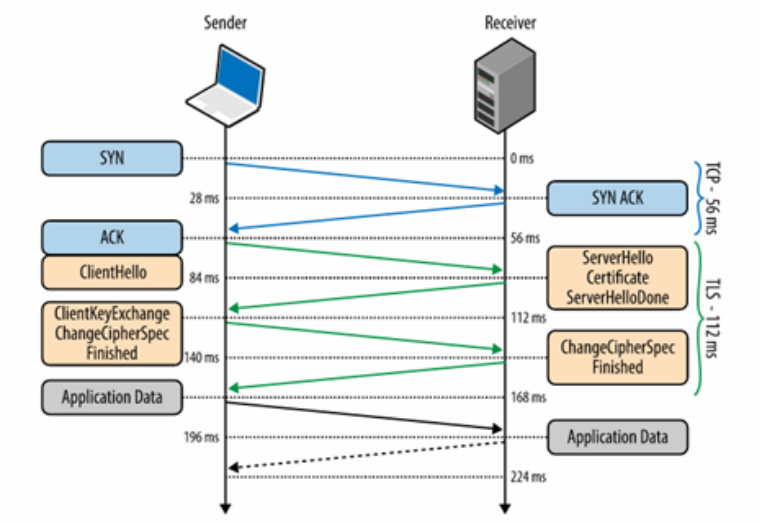


Figure 6.2

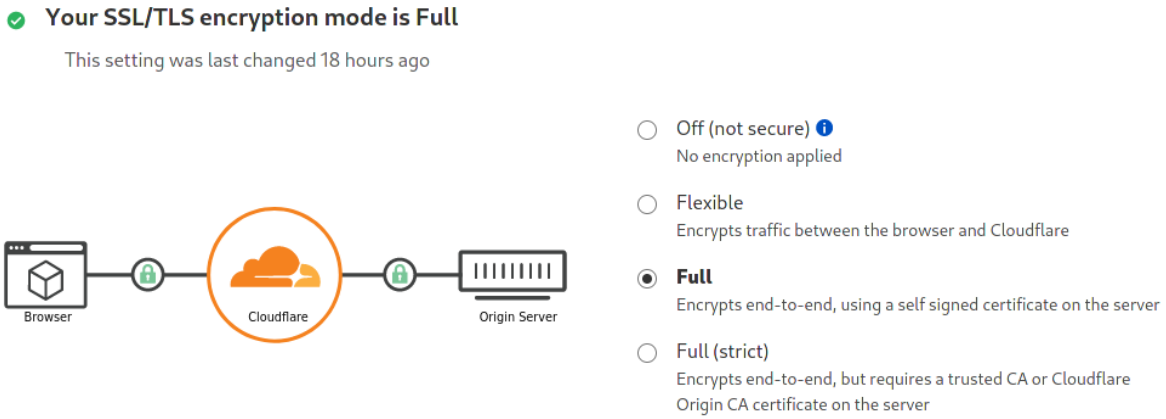


Figure 6.3

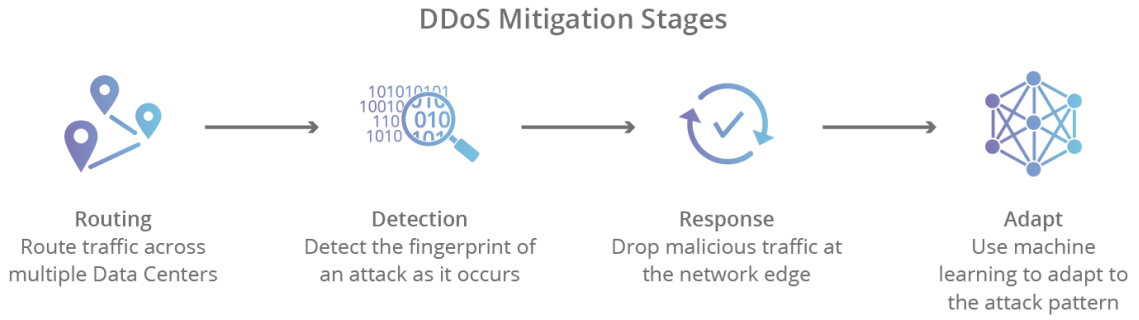


Figure 6.4

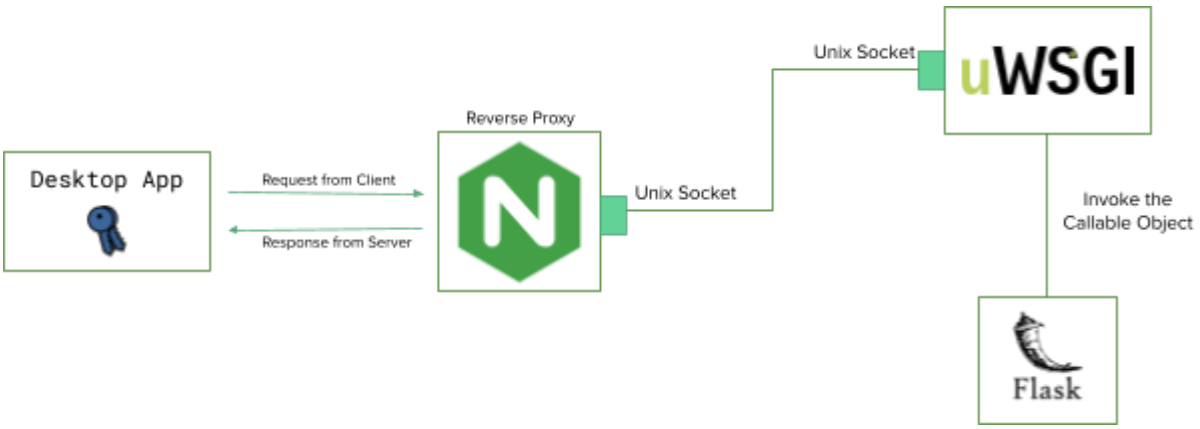


Figure 6.5

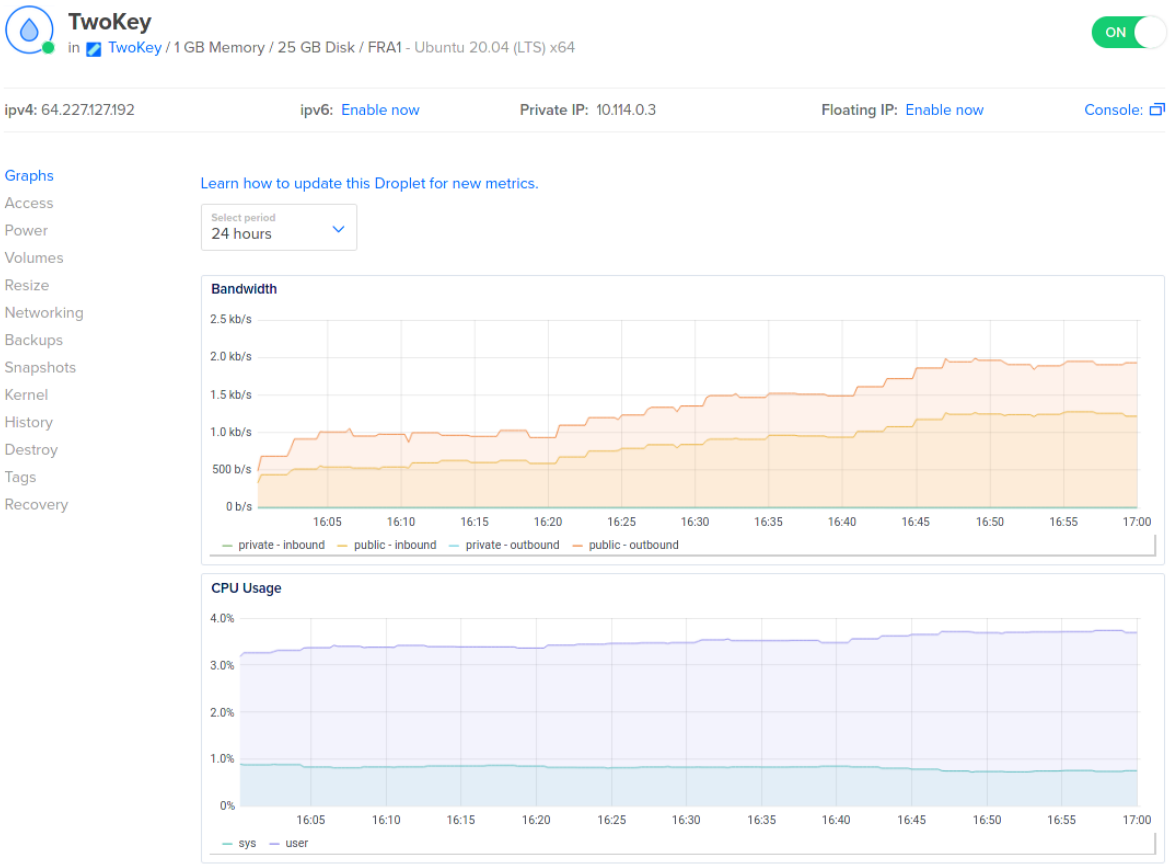


Figure 6.6

Figure 6.7

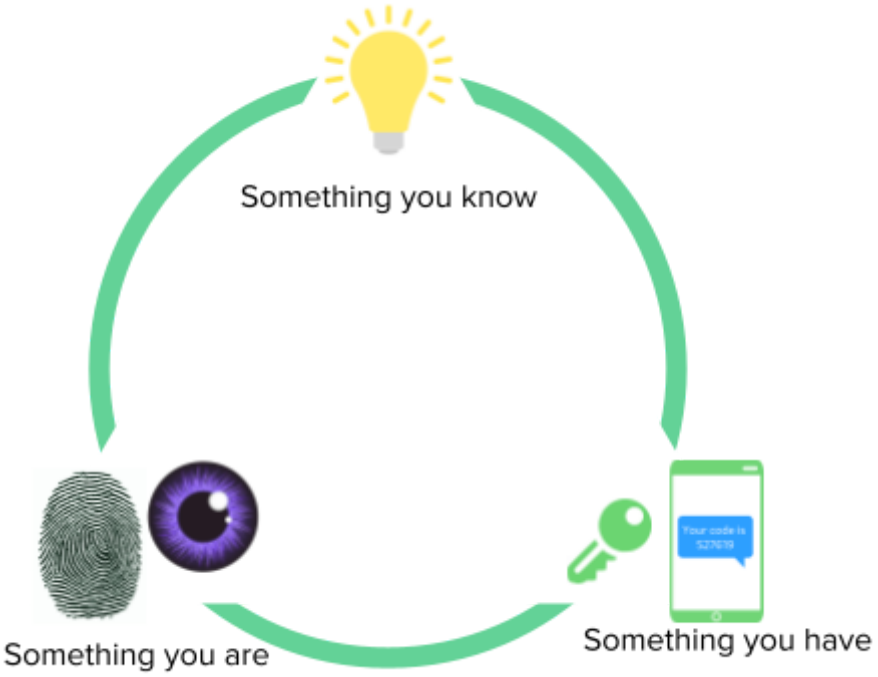


Figure 6.8

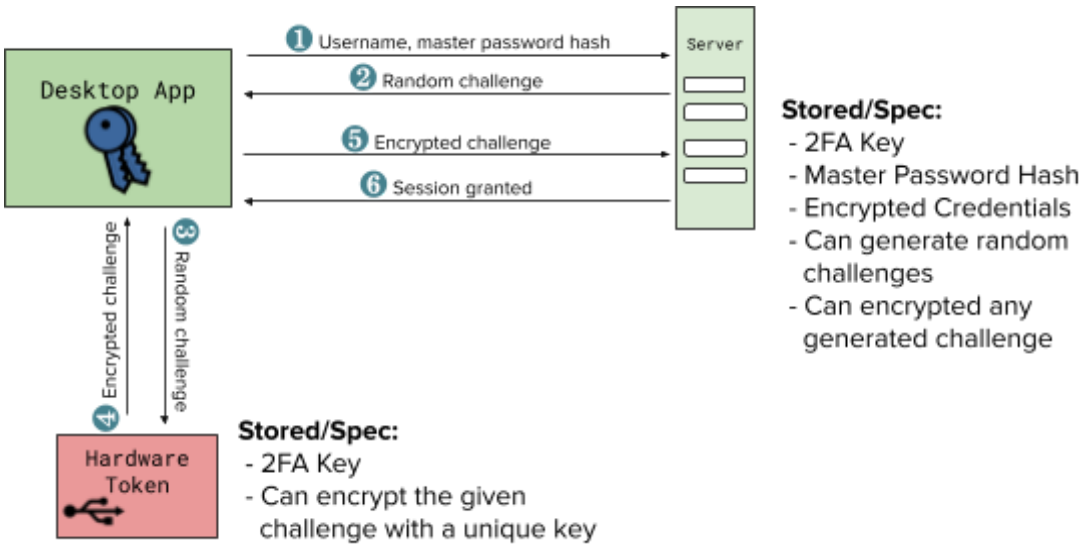


Figure 6.9

```
"Physical_id" : "Lg8V2YcuOsM3Kq2r", "Serial" : "0757312d-31e7-4a80-b430-adbeac9b7acb" }
"Physical_id" : "sl5Vb3RsGOQtwAHM", "Serial" : "8wUy6" }
"Physical_id" : "rskDSkocuB6YuLxO", "Serial" : "lEE6g" }
"Physical_id" : "LDQAOcbqwBiYDk36", "Serial" : "zCcNp" }
"Physical_id" : "R1q55nhUK9SijkZx", "Serial" : "vxXHh" }
```

Figure 6.10

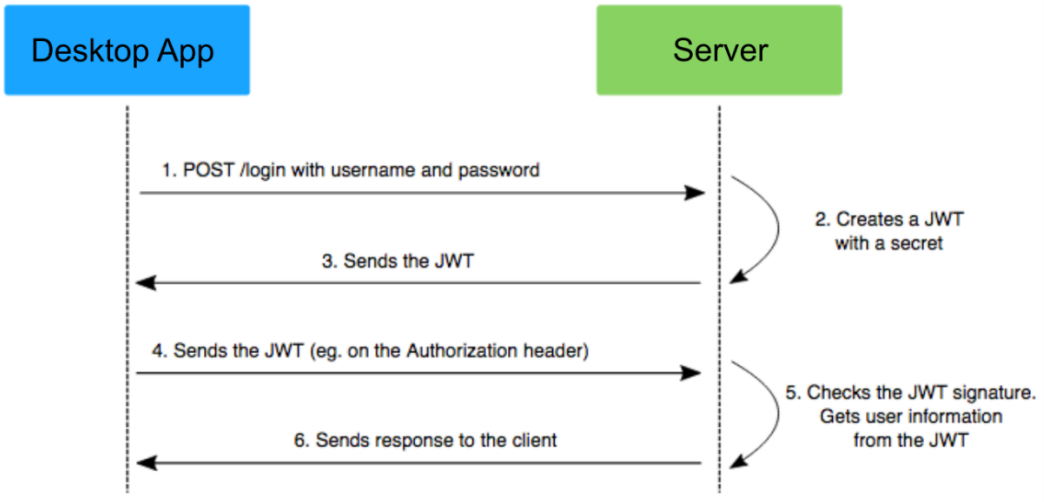


Figure 7.1

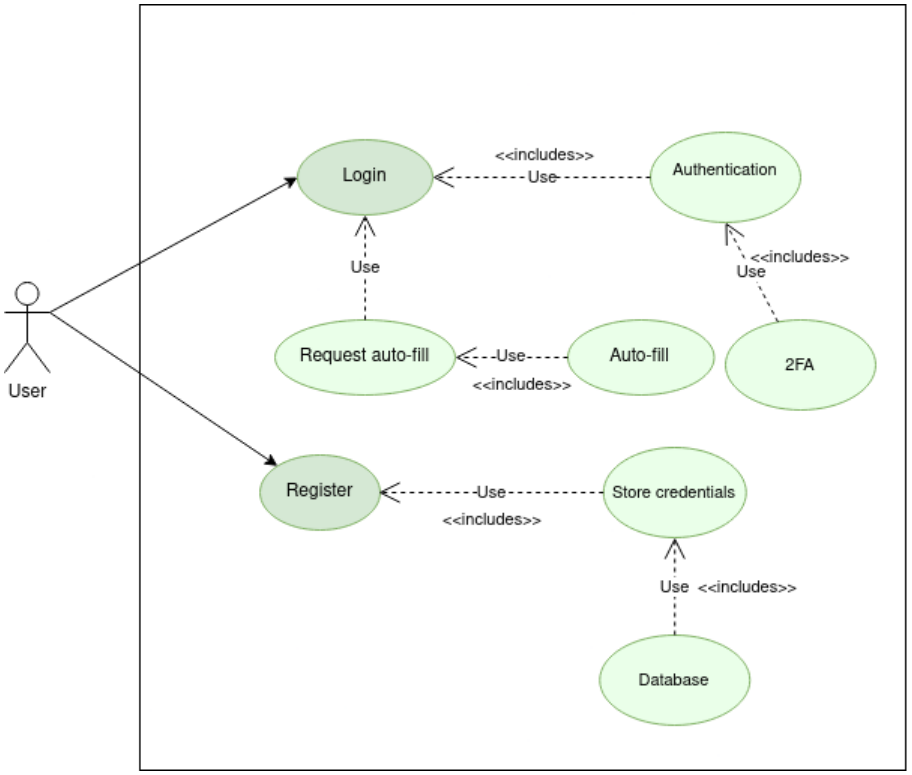


Figure 7.2

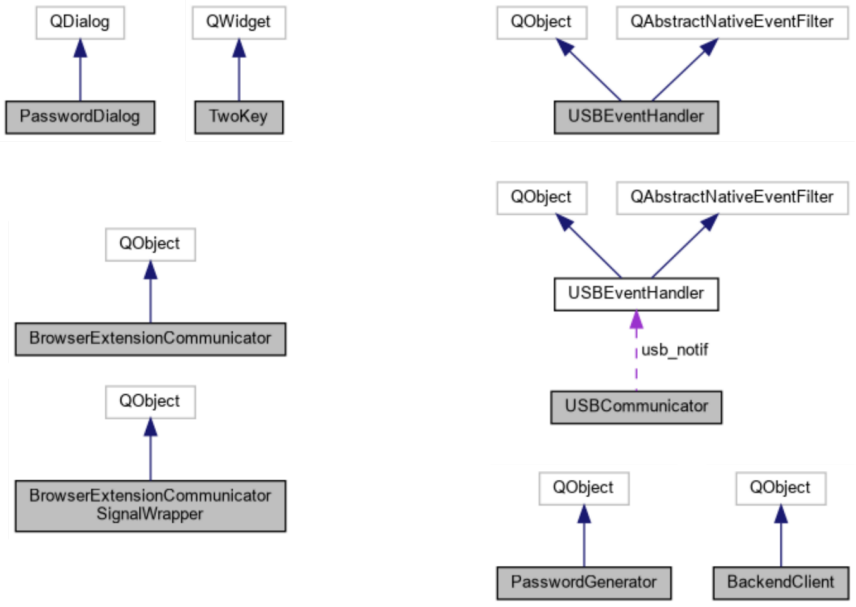


Figure 7.3

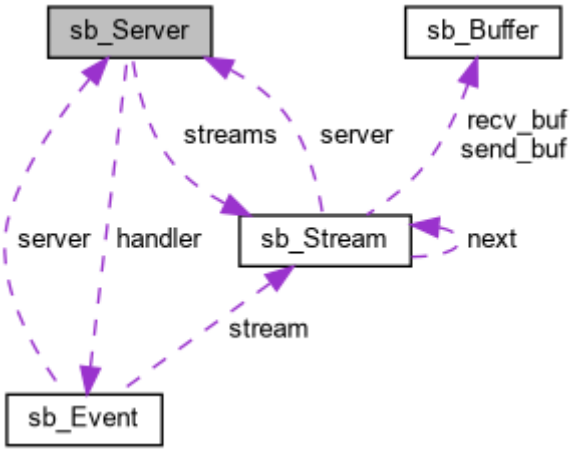


Figure 7.4

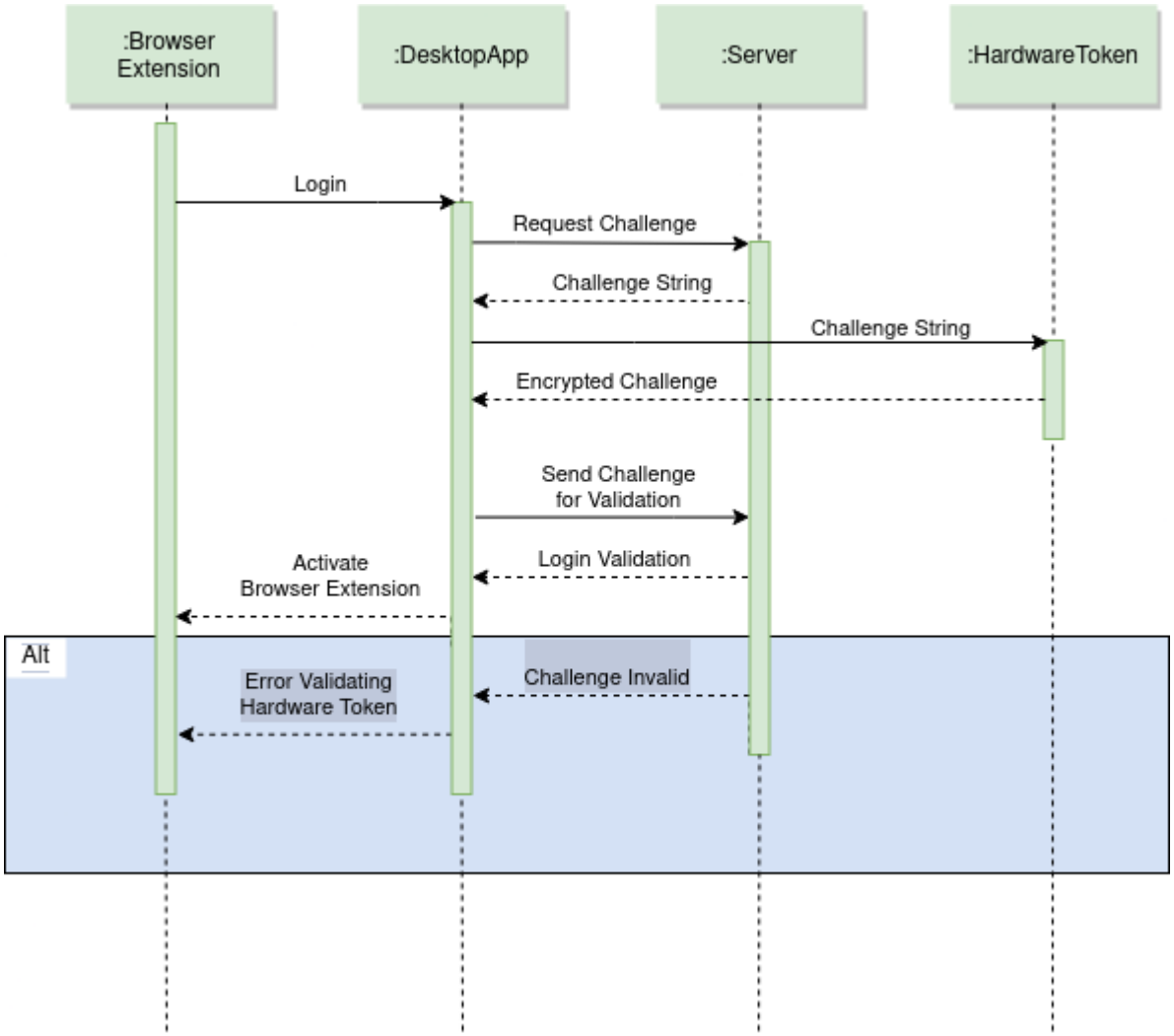


Figure 7.5

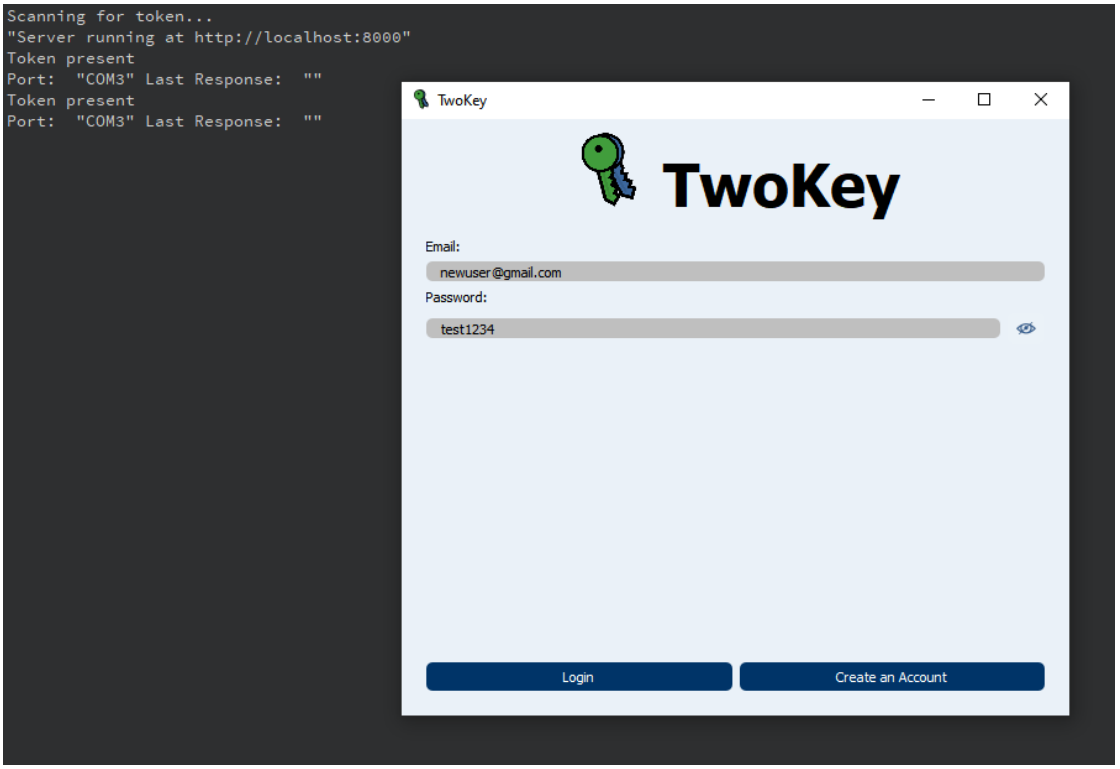


Figure 7.6

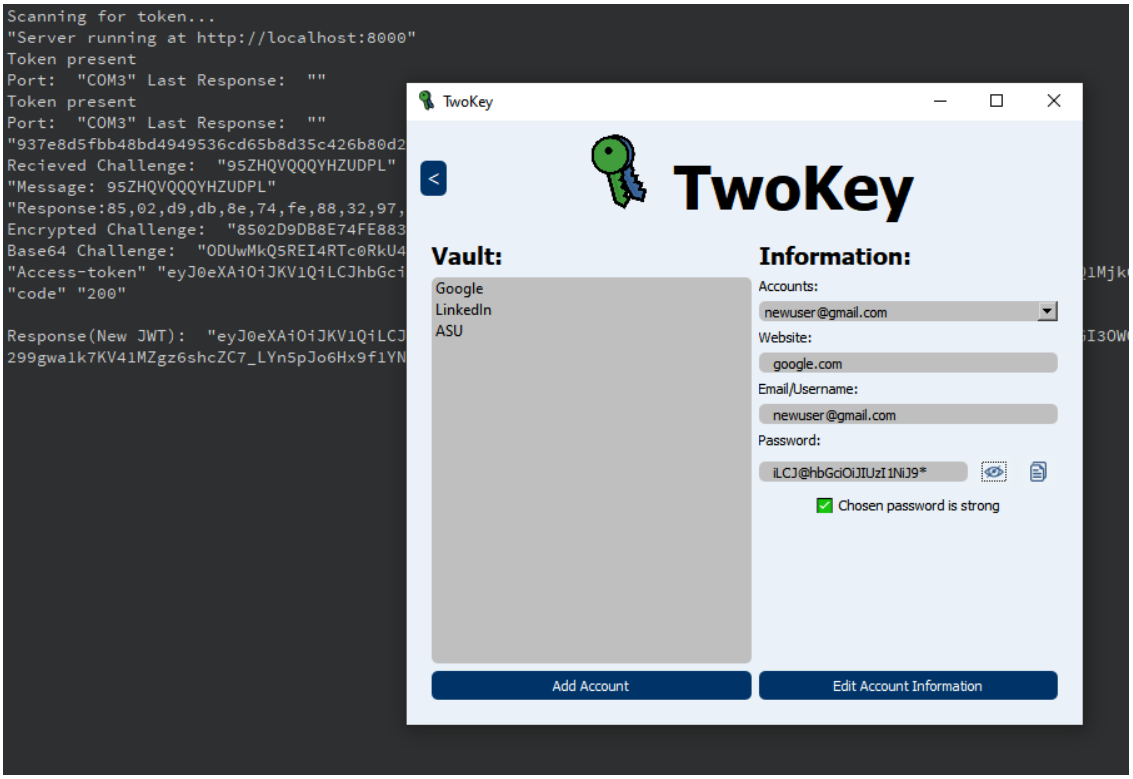


Figure 7.7

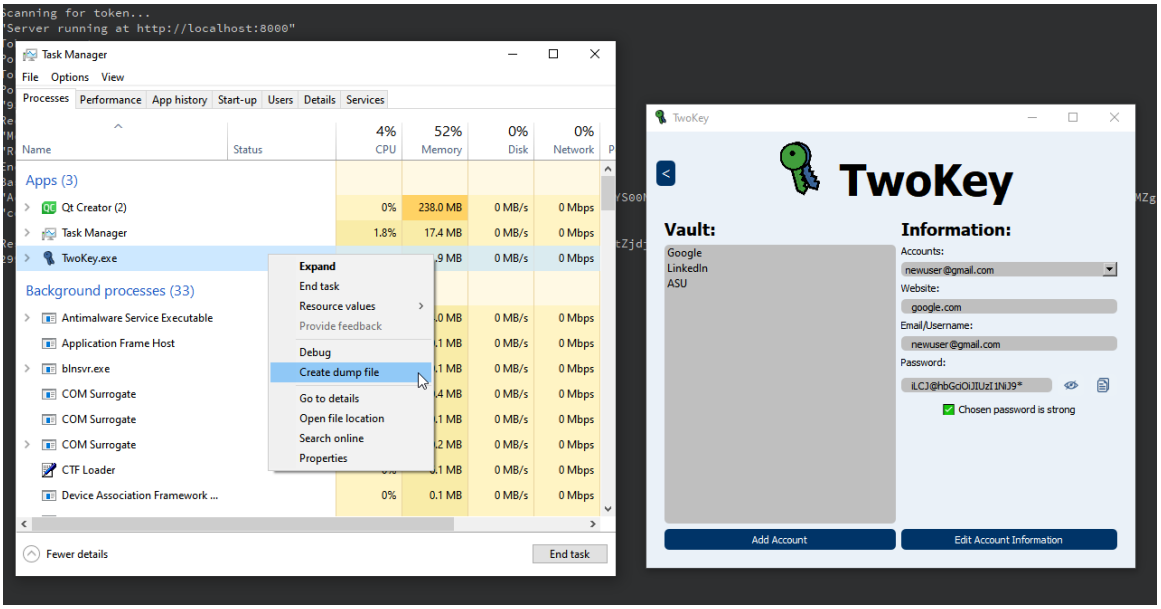


Figure 7.8

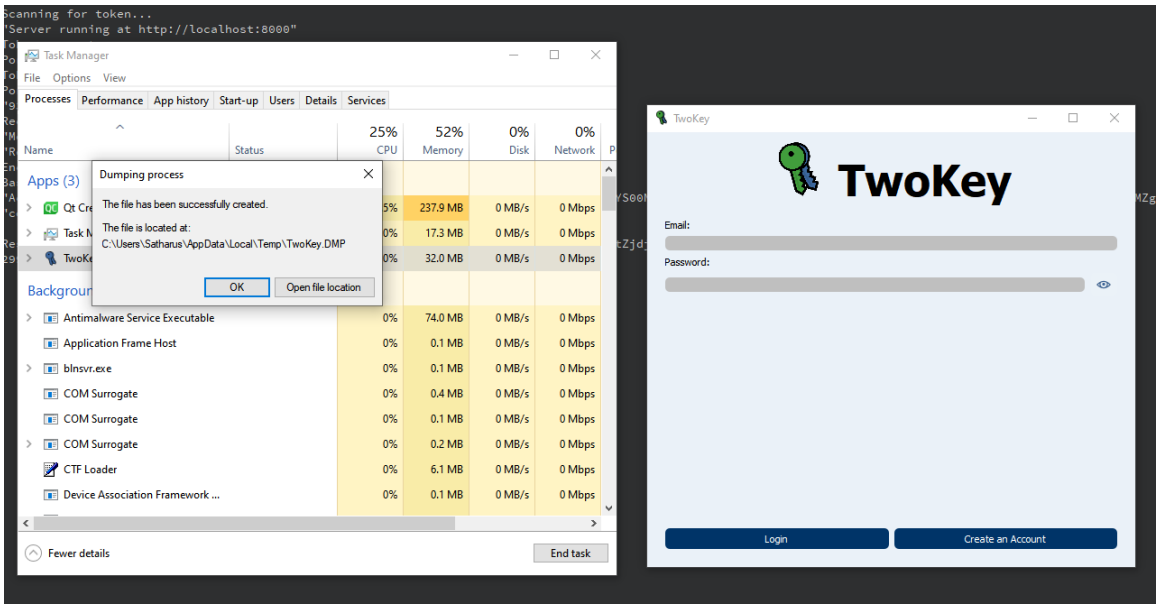


Figure 7.9

```
[Hotline:Desktop]$ [[ -z `strings TwoKey.DMP  | grep "iLCJ@hbGciOiJIUzI1NiJ9"` ]] && echo "No credentials found"
No credentials found
[Hotline:Desktop]$ [[ -z `strings TwoKey.DMP  | grep "test1234"` ]] && echo "No credentials found"
No credentials found
[Hotline:Desktop]$ strings TwoKey.DMP  | grep "iLCJ@hbGciOiJIUzI1NiJ9"
[Hotline:Desktop]$ strings TwoKey.DMP  | grep "test1234"
[Hotline:Desktop]$
```

Figure 7.10

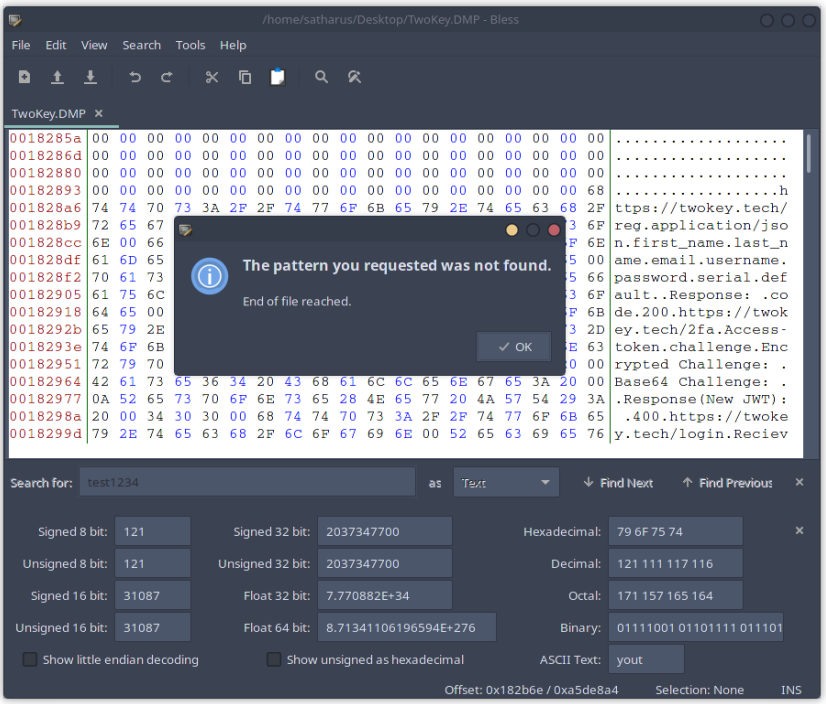


Figure 7.11

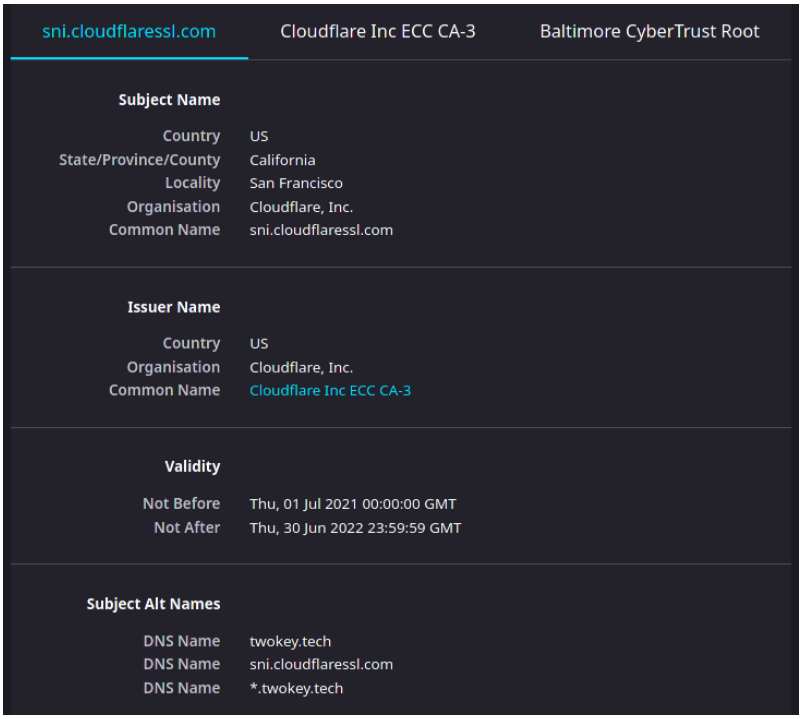
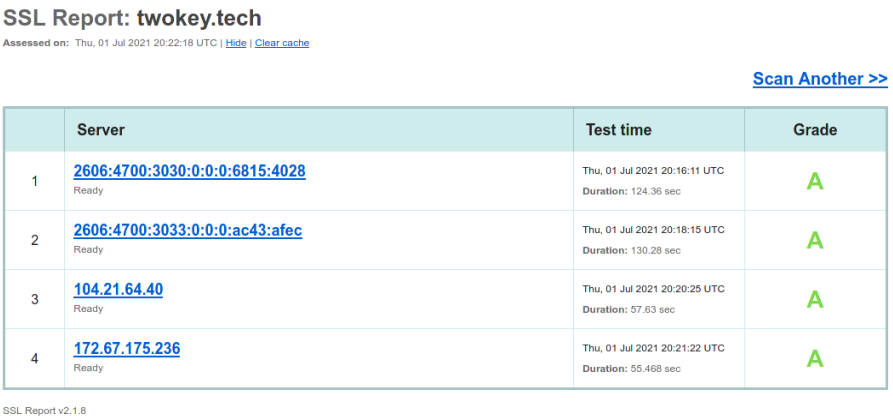


Figure 7.12



SSL Report: twokey.tech

Assessed on: Thu, 01 Jul 2021 20:22:18 UTC | Hide | Clear cache

Scan Another >>

	Server	Test time	Grade
1	2606:4700:3030:0:0:0:6815:4028 Ready	Thu, 01 Jul 2021 20:16:11 UTC Duration: 124.36 sec	A
2	2606:4700:3033:0:0:0:ac43:afec Ready	Thu, 01 Jul 2021 20:18:15 UTC Duration: 130.28 sec	A
3	104.21.64.40 Ready	Thu, 01 Jul 2021 20:20:25 UTC Duration: 57.63 sec	A
4	172.67.175.236 Ready	Thu, 01 Jul 2021 20:21:22 UTC Duration: 55.468 sec	A

SSL Report v2.1.8

Figure 7.13

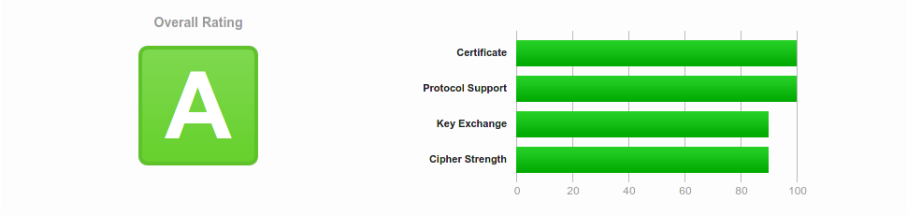


Figure 7.14

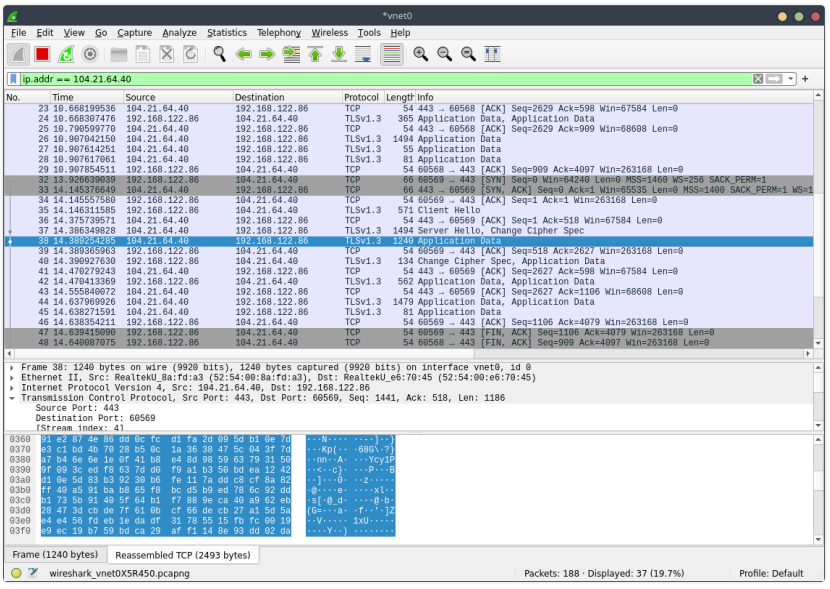


Figure 7.15

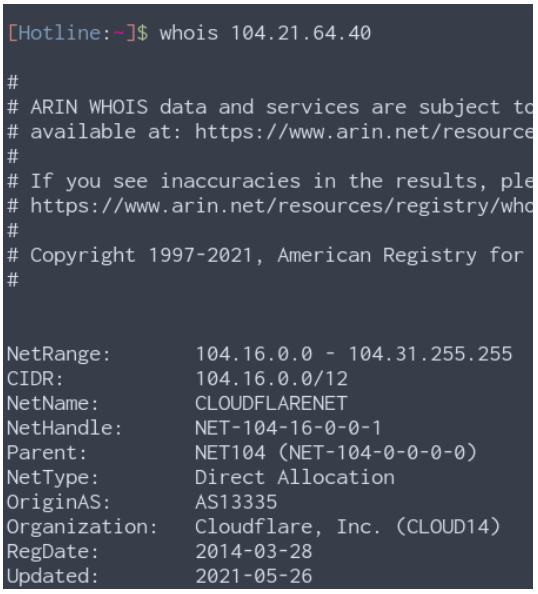


Figure 7.16

Figure 7.17

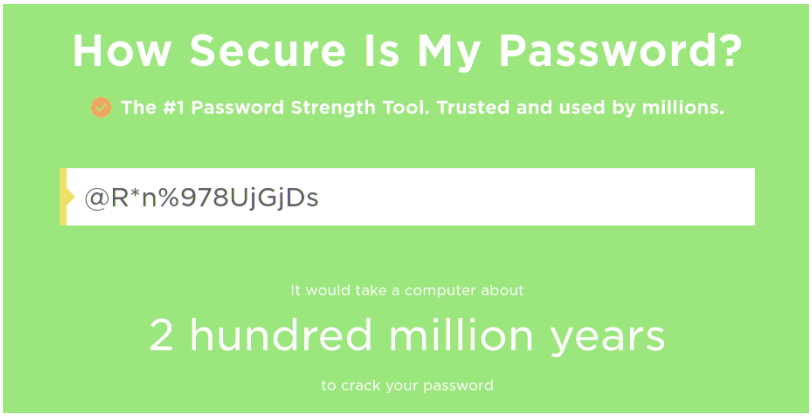


Figure 7.18

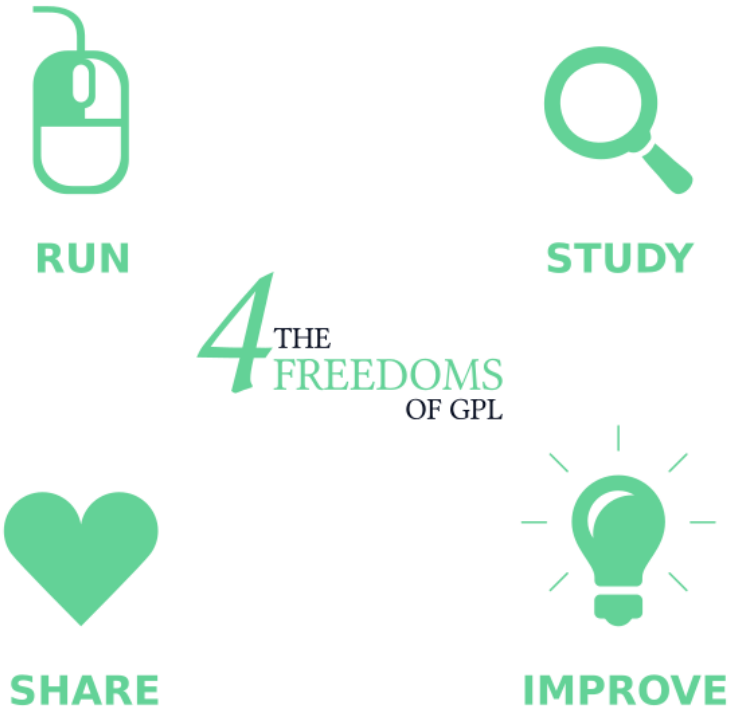


Figure 8.1

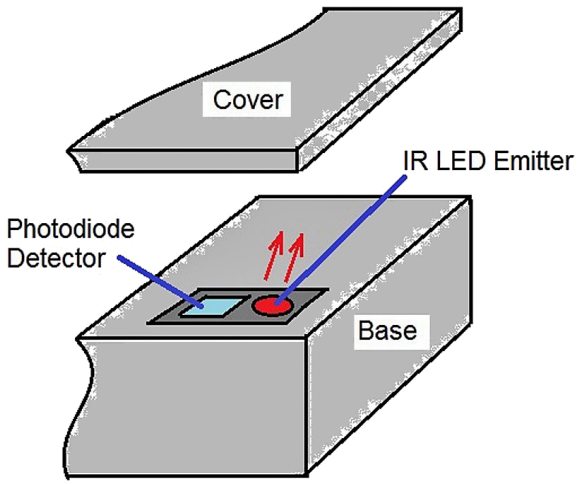
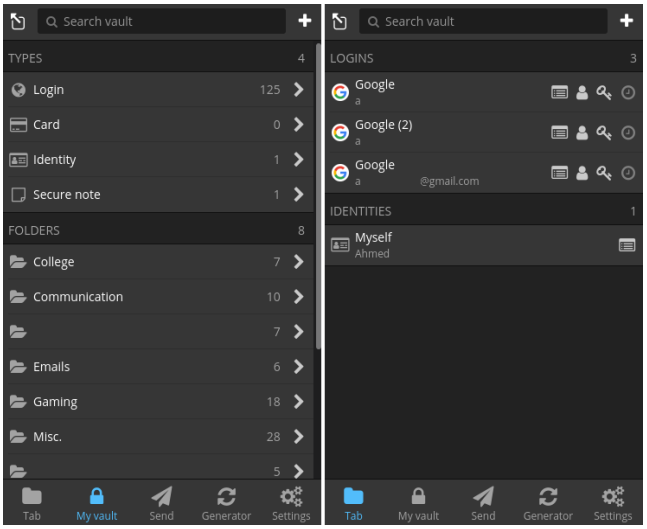


Figure 8.2


Search vault
TYPES 4
Login 125
Card 0
Identity 1
Secure note 1
FOLDERS 8
College 7
Communication 10
7
Emails 6
Gaming 18
Misc. 28
5
Tab
My vault
Send
Generator
Settings
Search vault
LOGINS 3
Google
a
Google (2)
a
Google
a
@gmail.com
IDENTITIES 1
Myself
Ahmed
Tab
My vault
Send
Generator
Settings

Chapter 1: Introduction

1.1 Background

Online services have become inseparable from our day to day life. In a world where almost every online service needs an account, it is important for a user to practice good password hygiene in order to keep their accounts secure.

A survey was conducted to check the average level of security practices between a relatively sizable sample of people.

The goal of the survey was to assess the following points:

- How many unique passwords do they have?
- Do any of these passwords contain personal identifiable information such as names of relatives, phone numbers, countries they have lived in, etc..?
- How often they change their passwords.
- Have ever been compromised due to a weak password (i.e received a fake email containing their password or any similar phishing attempt)

The results were as follows:

- 66.1% of people (72/109) use only 2-5 unique passwords across all of their accounts. That's approximately 4 accounts sharing the same password at the best case, if that user uses only 20 services.

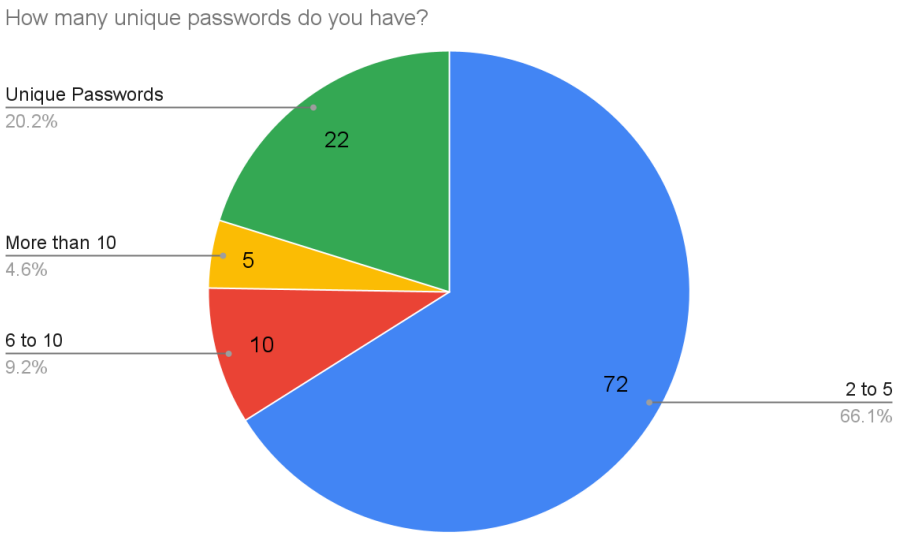


Chart 1.1

- 45.3% of people (53/117) have personal information in their passwords.

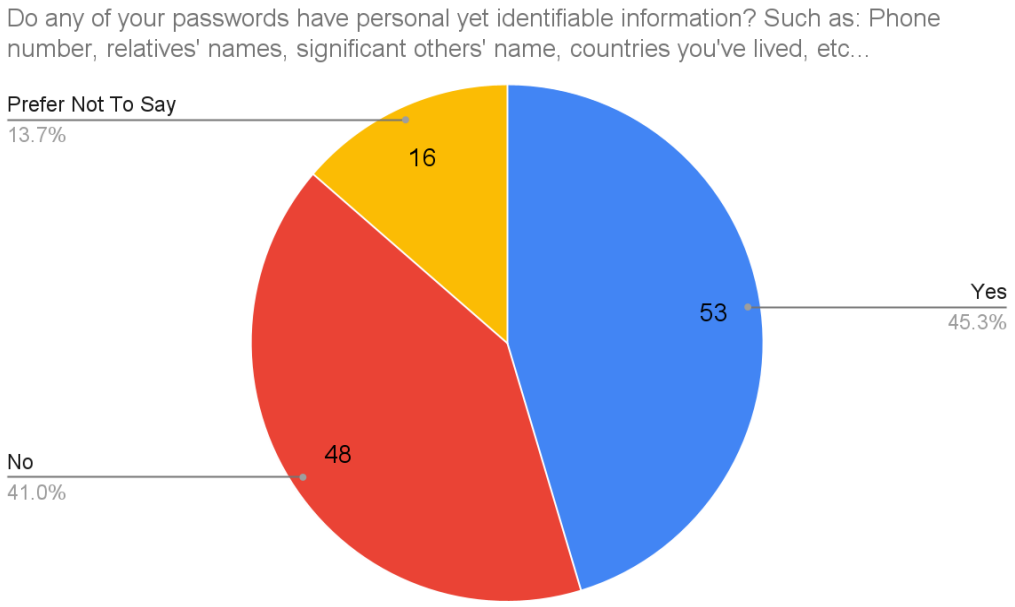


Chart 1.2

- 76.1% of people (89/117) only change their passwords whenever they forget them or *feel* like someone has accessed their accounts.

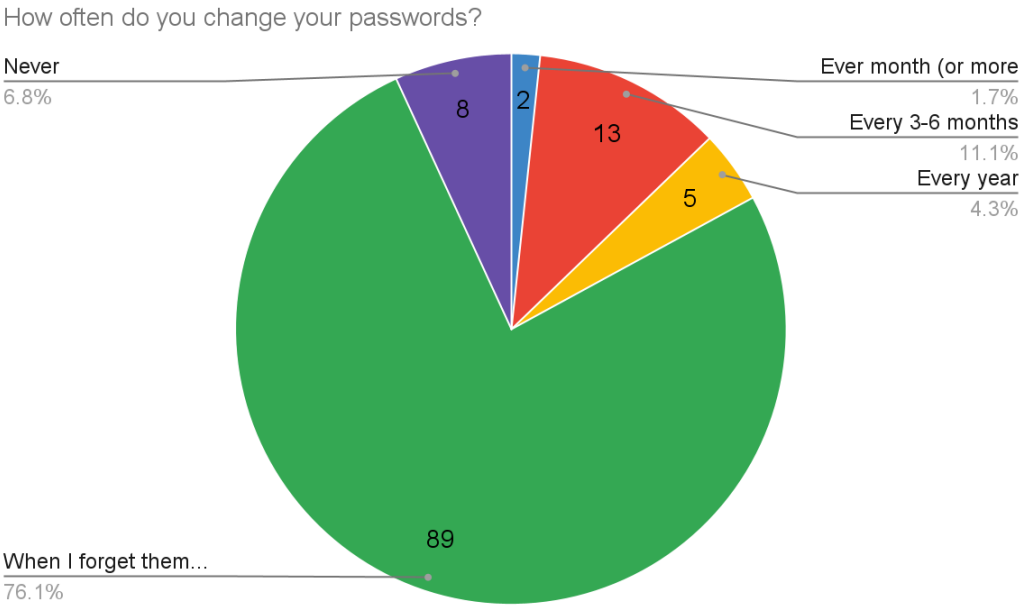


Chart 1.3

- 23.1% of people (27/117) have been compromised or received the infamous phishing email (Figure 1.1) which contains their leaked password and threatens them to leak their information if they don't pay a ransom.

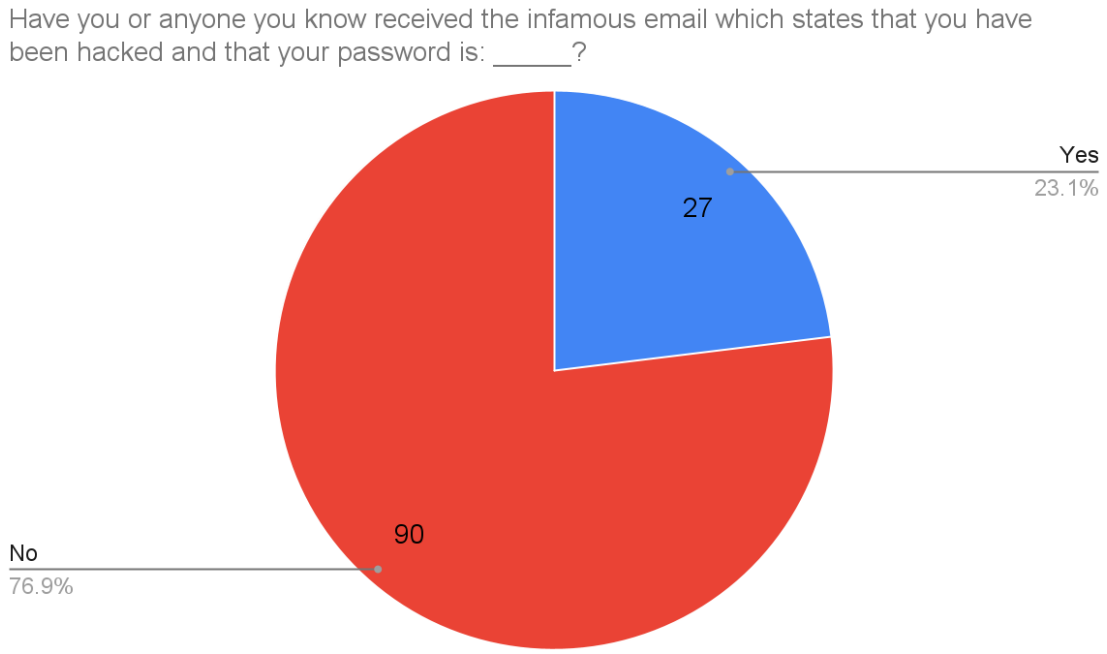


Chart 1.4

Hello!
I'm a member of an international hacker group.

As you could probably have guessed, your account [redacted] was hacked, because I sent message you from it.

Now I have access to you accounts!
For example, your password for [redacted] is c0yhu48e3q96

Within a period from July 17, 2018 to October 3, 2018, you were infected by the virus we've created, through an [redacted] you've visited.
So far, we have access to your messages, social media accounts, and messengers.
Moreover, we've gotten full damps of these data.

Transfer $800 to our Bitcoin wallet: 14bXUoPwruptLamUfKTuMW39Qy1q4ohX9w
If you don't know about Bitcoin please input in Google "buy BTC". It's really easy.

I guarantee that after that, we'll erase all your "data" :)

A timer will start once you read this message. You have 48 hours to pay the above-mentioned amount.

Your data will be erased once the money are transferred.
If they are not, all your messages and videos recorded will be automatically sent to all your contacts found on your devices at the moment of infection.

You should always think about your security.
We hope this case will teach you to keep secrets.
Take care of yourself.

Figure 1.1

All of the following results show that security awareness when it comes to passwords is an issue. That's why a lot of users have resorted to using password managers. Some of which have issues.

1.2 Problem Definition

Passwords are often mishandled by the majority of internet users. Users often choose insecure passwords which are easy to memorise, or some may even store their passwords in an insecure way such as a plaintext file or an unencrypted online note. Even worse, a user may use a single password for all accounts, making all of his accounts vulnerable in the case that the single password was leaked or cracked.

A good solution to this problem is using a password manager, which a lot of people have started resorting to recently. A password manager is software which stores and encrypts your passwords using a single master password which allows you to decrypt and access your passwords when logging into services.

There are a lot of existent password managers, many of which store data online. According to research by ISE[2], a lot of password managers have issues securing their secrets/passwords. Which in turn lowers the security of the user as they already have all of their passwords stored using the password manager.

Issues with existing password managers are:

- Weak 2FA or lack thereof
- Leaving plaintext passwords in memory even after the password manager has been closed
- Weak encryption or lack thereof (Especially when it comes to communication)

Hence the idea of TwoKey was created.

1.3 Objective

The idea of TwoKey is creating a password manager which has the following features:

- **Secure Programming**
  - Doesn't leave any unencrypted secrets behind in memory.
  - Protected against memory leaks, ROP Chain Attacks, Buffer Overflows, etc...
- **Secure Communication**
  - No transmission of any unencrypted data.
  - Cryptographic keys never leave any of the system's modules.
- **Strong Passwords**
  - The software forces the user to use strong passwords which are randomly generated by TwoKey itself.
  - The software warns the user if any of their passwords are weak.
  - The passwords are never sent or stored in plaintext. Even if a breach were to happen, the passwords are still unreadable to the attacker as the passwords are encrypted by the master password, which only the user knows.
- **Strong Second Factor of Authentication**
  - Each user has a unique physical USB token which they use as their second factor.
  - The user can't log in or use the software without their token.
  - Users can't interchange their tokens. Each user has a single token which can only be used by them.
- **Trustworthy**
  - Using software such as a password manager requires trust from the end user, as they are putting the security of their online presence into the hands of the password manager itself.
  - That's why TwoKey is free open source software, which is licensed under the GPL license. In which anyone can run, study, share, and improve the software. These are the 4 freedoms of software as depicted by the Free Software Foundation[3]
  - This allows any user/auditor to verify the claims of TwoKey, and allows any other party which uses the software to modify it to their liking and deploy it in their own environment, making the responsibility for security and uptime theirs.

Project modules are described in Chapter 2: System Overview.

1.4 Document Organization

The following chapters of this document are organized as follows:

Chapter 2: System Overview

Chapter 2 discusses the modules of the project and an overview of how they integrate with each other.

Chapter 3: Hardware Token

Chapter 3 discusses the hardware token's function, its mechanism for second factor authentication, and HSMs.

Chapter 4: Desktop Application

Chapter 4 discusses the desktop application's functions, the UI/UX design, the design choices, and the features.

Chapter 5: Browser Extension/Plugin

Chapter 5 discusses the browser extension's functions, how it communicates with the desktop application, and the various challenges met.

Chapter 6: Server Backend and Deployment

Chapter 6 discusses the server backend, the secured communication, DDoS protection and other features which are provided by this module.

Chapter 7: Implementation and Testing

Chapter 7 discusses the system analysis and design, the final implementations, and the testing methodology and results.

Chapter 8: Conclusion and Future Work

Chapter 8 discusses the conclusion of the project, a link to the demo video, and recommendations/possible improvements for future work.

Chapter 2: System Overview

2.1 Project Modules

TwoKey consists of 4 modules, all of which implement security practices which protect against known attacks which target password managers.

Attack	Module	Security Practice
Brute Force	Desktop App, Server-Side	Strong and different password for each account.
Dictionary/Rainbow Table Attack	Desktop App	PBKDF is used for key stretching and to make password cracking difficult.
Phishing	Browser Extension	Fill password when the service domain is verified to be the legitimate domain. i.e the browser extension won't autofill gogle.com mistaking it for google.com.
Less Common Interfaces	Desktop App, Browser Extension	Hardware tokens are always checked for verification.
GSM Sniffing	Hardware Token	No SMS Security Codes are used as a 2FA. The only 2FA is the USB token.
MITM Attack	Desktop App, Browser Extension, Server-Side	Credentials are encrypted and transferred securely between modules and memory is sanitized.
	Hardware Token	Authentication token and encryption keys never leave the chip on the USB device.
Replay/Playback Attack	Hardware Token, Server-Side	A random challenge is always sent each time for authentication.
	Desktop App, Server-Side	Random session keys/tokens.
Insecure Programming Attacks	The Entire Project	Secure programming to avoid common vulnerabilities such as BOF, ROP/Ret2LibC, Format String Attacks, etc...

2.2 General View of System Architecture

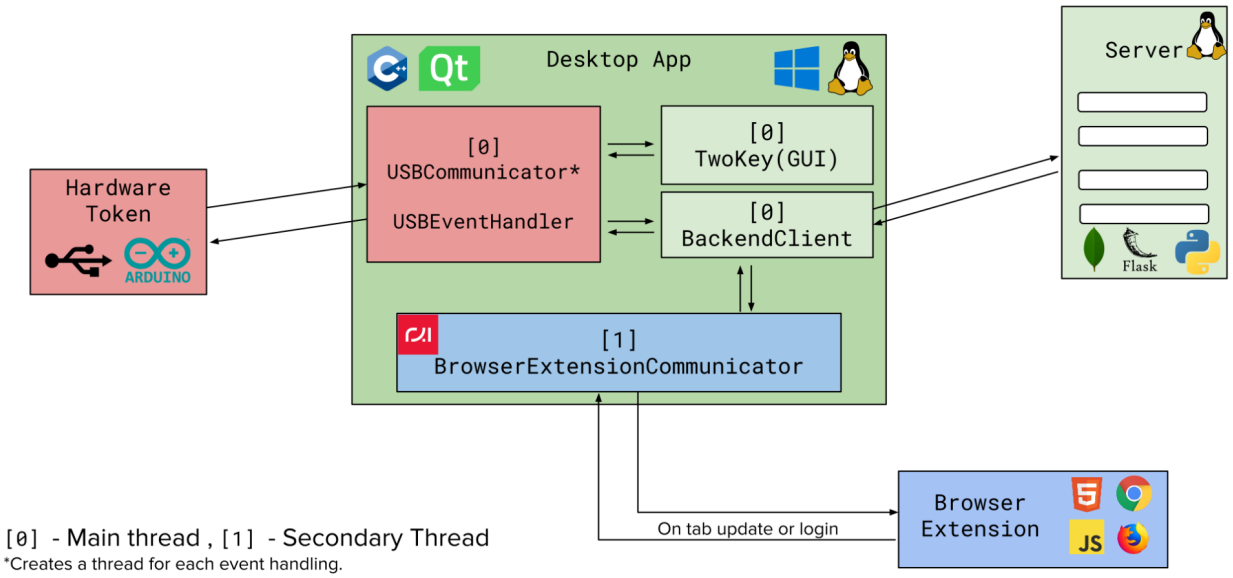


Figure 2.1

The project's four modules all revolve around the desktop app, which is the central component connecting all of the modules together.

- **The Desktop App** is mainly written in C++ with the aid of the Qt Application Framework, it runs an embeddable HTTP server by rxi called SandBird[4]. It is cross platform and works on both Windows and Linux.
- **The Browser Extension** is written mainly in JS and HTML. It relies heavily on Browser Extension APIs and Javascript WebExtension APIs.
- **The Hardware Token** can be implemented on most of the common commercially-available AVR. For this version of the project, an Arduino Nano, and an ATtiny85 custom board. Both of which are valid POCs for different-sized USB tokens.
- **The Backend Server** is mainly written in Python with the aid of Flask, pyaes, and a couple of other python packages in order to help with data formatting and organisation. The server also utilises MongoDB as a database to store user and hardware information

2.2.1 The Desktop App

The desktop app itself has 2 main threads:

- The main thread, which contains:
    - The GUI
    - The USBEventHandler which communicates with the hardware token and detects plug/unplug events.
    - The backend client which communicates with the backend server, to perform functions like login, register, and retrieve, update, and remove credentials.
- The secondary thread, which contains the Browser Extension Communicator which is an embed of SandBird by rxi. The communicator receives requests from the browser extension to allow them to communicate using a client-server architecture where the extension is the client and this thread is the server.

2.2.2 The Browser Extension

- Communicates with the desktop app to retrieve credentials from the server.
- Autofills service credentials based on the domain name of the website and the added credentials.

2.2.3 The Hardware Token

- Responsible for 2FA with the server through the desktop app.
- The application refuses to log in or perform any functions without it. If the token is removed, the desktop app logs the user out and deletes the session.

2.2.4 The Server

- Responsible for data management, credential storing and retrieval, user authentication, and user registration.
- Employs security features such as HTTPS/SSL encryption and DDoS protection

Chapter 3: Hardware Token

3.1 Security

The authentication process is done using a security challenge-response such that a sequence of bytes is sent to the hardware token from the desktop application to be encrypted using 128bit AES encryption and then received by the desktop app to be forwarded to the desktop app for validation.

```
#include <AESLib.h>

void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(3000);
}

uint8_t message[17];
uint8_t key[] = "rskDSkocuB6YuLxO";
uint8_t iv[]  = "su4djDtAWNuvyjVg";

void loop()
{
    if (Serial.available() > 0)
    {
      Serial.readBytes(message, 16);
      aes128_cbc_enc(key, iv, message, 16);
      for (int i = 0; i < 16; i++)
      {
        if (message[i] < 16) Serial.print("0");
        Serial.print(message[i], HEX);
      }
      Serial.flush();
    }
}
```

Code Snippet 3.1

As seen in Code Snippet 3.1, the USB token waites for any incoming bytes, after it reads the bytes it encrypts them using a key that is constant for every device, and a constant initialisation vector (in order to have the same encryption output as the server). After the bytes are encrypted, they are sent back via the serial port to the desktop app.

3.2 Hardware Technology

The chosen technology for the hardware token device is an AVR microcontroller due to its small size which is suitable for carrying, as well as having enough processing capability to perform 128-bit AES encryption which is sufficient for the authentication process.

3.3 Hardware Security Model

Using the challenge-response authentication allows the implementation of a security model such that the key will never leave the device.

Since the token is only used as a second factor of authentication, even if it were to be lost or stolen, the user's data wouldn't be accessible by someone else as the attacker would need to know the user's password itself.

Chapter 4: Desktop Application

4.1 UI/UX

The frontend of the desktop app was created using C++ and the Qt Application Framework.

The desktop application has multiple forms for the essential functions of the application, `QStackedWidget` allows the existence of multiple widgets stacked on top of each other for the user to navigate between the pages they need.

The application also consists of many widgets (Text boxes, Buttons and Labels) and these objects must be consistent and have a unified theme. Using Qt Style Sheets, which are similar to HTML Cascading Style Sheets (CSS) a style was added for each widget to make a custom harmonious theme. Also adding a different look for the multiple states of the widgets like hovering, pressing, and selecting a certain style was added to each state specific to each widget.

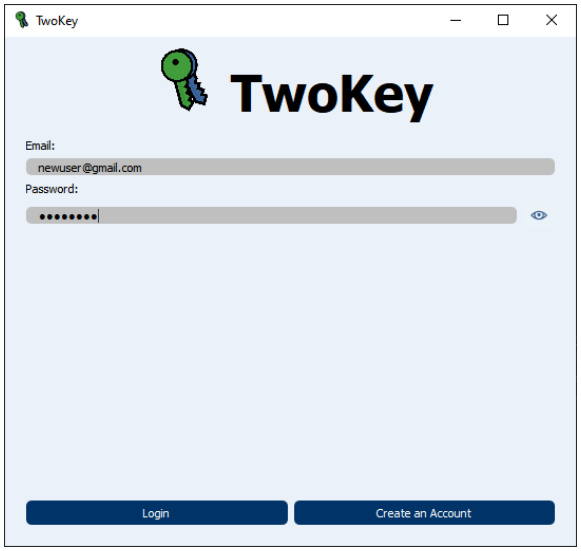


Figure 4.1 TwoKey Login Screen

Figure 4.2 TwoKey's UI when token is removed

Figure 4.3 TwoKey when a password is hidden

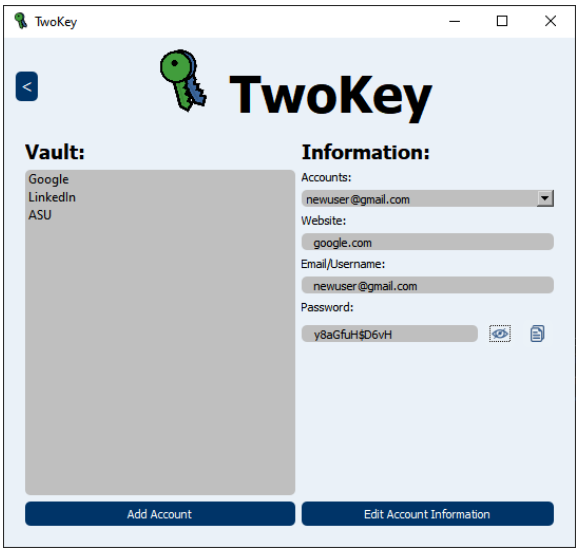
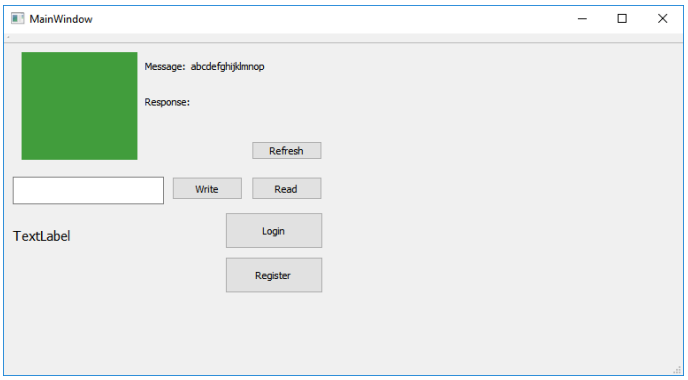
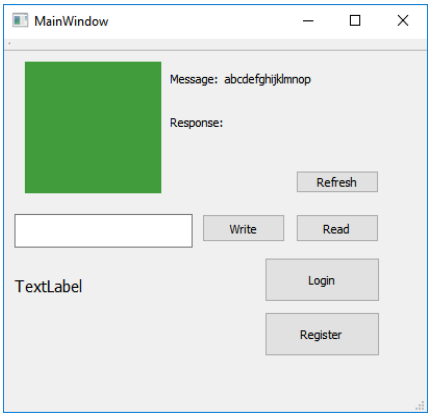


Figure 4.4 TwoKey when a password is shown

The UI was designed to be dynamic (Figure 4.7) and scalable to any size the user prefers by using QLayout which also allowed the widgets to be consistent.





Figures 4.5, 4.6 : Early non-scalable prototype of the Desktop App

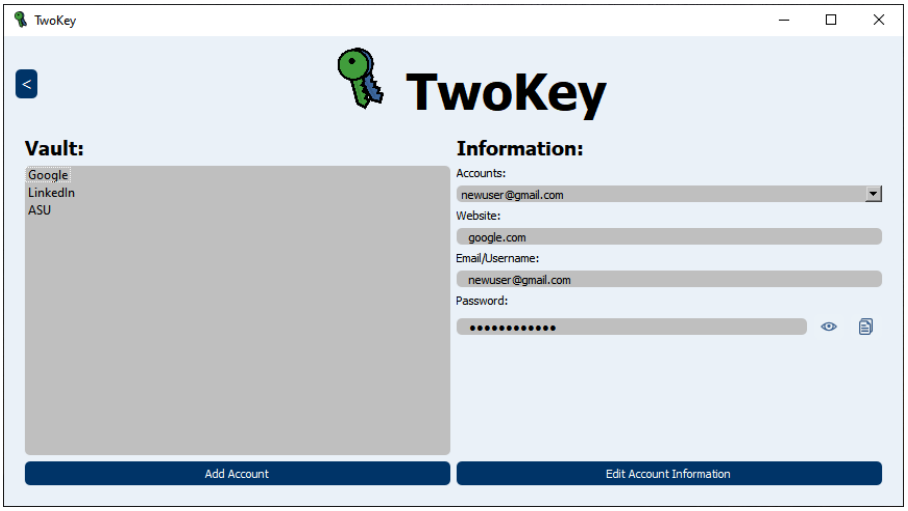


Figure 4.7: TwoKey utilising the dynamic UI

The desktop app does not halt when an external event occurs, such as the plugging of the token which is indicated in the application's UI and the tray icon (Figure 4.8) which is added for a more intuitive and convenient UX. The user is allowed to keep TwoKey running without having to keep the window itself open and taking up space, with an option to close the program if needed (Figure 4.9).

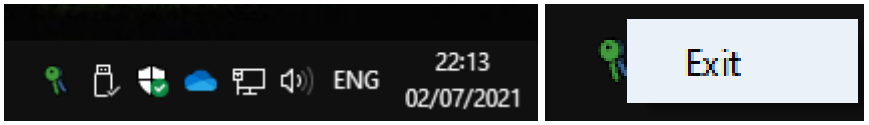


Figure 4.8, 4.9: The taskbar tray icon

Using a cross-platform framework like Qt allowed the UI to look consistent on all platforms (Figure 4.10) and operate with native speed and capabilities which focuses on the required functionalities that the users may need. Ensuring that the interface elements are easy to access and facilitate the functionalities to the user. Using such a framework makes TwoKey easy for contributors to join, as the project is open source.

Figure 4.10: TwoKey running on KDE Plasma 5.12.5

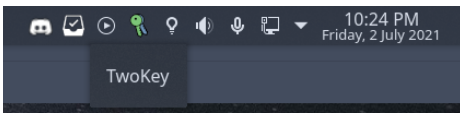


Figure 4.11, TwoKey tray icon on KDE

4.2 Hardware Token Handler

4.2.1 Token Communication

The communication between the desktop application and the hardware token is confined to sending and receiving a sequence of bytes for authentication and validation. This happens through serial communication between the AVR device and the desktop app. Qt provides classes to interface with serial devices such as `QSerialPortInfo` and `QSerialPort`.

Communication happens between them to authenticate the user when it comes to the second factor, which is the challenge-response authentication from the server. The server sends the desktop app a challenge, which the desktop app then forwards to the hardware token. The desktop app then forwards the token's response to the server to authenticate the user (Figure 4.12).

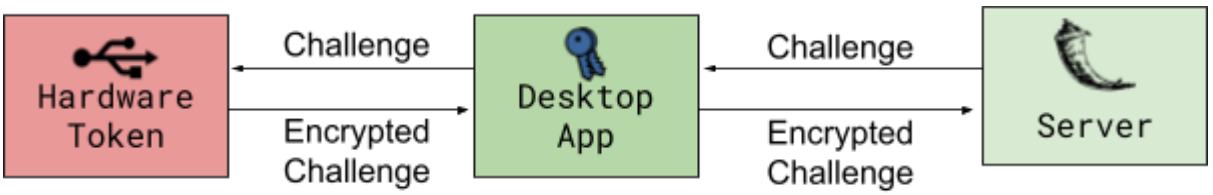


Figure 4.12

The desktop application has the ability to auto detect if the hardware token was plugged in or out and change the UI accordingly, as stated in figures 4.1 and 4.2.

When the token is plugged out, the user is automatically logged off and the session is deleted from the machine.

4.2.2 Token Auto-Detection

The initial method of auto detecting the usb token was by creating a thread which constantly polled the device in an infinite loop while the program was running (Code Snippet 4.1). That worked, but was very unstable and had problems with thread safety due to sharing pointers. On further research, Windows Device Notification APIs were found to be a much more solid and stable alternative.

```
QSerialPort *token = new QSerialPort();
bool tokenIsAvailable = false;
QString tokenPortName;
static void checkForToken(int *running)
{
    while (*running)
    {
        foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts())
        {
            if (serialPortInfo.hasVendorIdentifier() &&
                    serialPortInfo.hasProductIdentifier() &&
                    serialPortInfo.vendorIdentifier() == TOKEN_VID &&
                    serialPortInfo.productIdentifier() == TOKEN_PID)
            {
                tokenPortName = serialPortInfo.portName();
                tokenIsAvailable = true;
            }
        }

        if (tokenIsAvailable)
        {
            if (!token->isOpen())
            {
                token->setPortName(nanoPortName);
                token->setBaudRate(QSerialPort::Baud9600);
                token->setDataBits(QSerialPort::Data8);
                token->setParity(QSerialPort::NoParity);
                token->setStopBits(QSerialPort::OneStop);
                if (!token->open(QIODevice::ReadWrite))
                {
                  //Log error, device is likely being used
                }
            }
        }
        else //Log failure
    }
}
```

Code Snippet 4.1

4.2.2.1 Windows Systems

On Windows systems, an API is provided which allows a GUI window to register for a device notification[5]. That is, if any device is inserted or removed while the window is open, it is notified by the OS. The program can then implement a handler function which executes every time the notification is fired. Qt also provides a class `QAbstractEventFilter`[6], this class can be implemented in order to use it as an event handler class for any event (Code Snippet 4.2).

Every time a device is connected, a separate thread is created by TwoKey to check if that device is the token or not based on its USB vendor ID and product ID. If it is the token then it adjusts the UI accordingly, the user can then try to login. Note that the user will only login if the token is his own token and the master password matches his account.

```
class USBEventHandler : public QObject, public QAbstractNativeEventFilter
{
    Q_OBJECT
public:
    //Constructor and Desctructor
    explicit USBEventHandler(QObject *parent = nullptr);
    //Implementing the original function from QAbstractNativeEventFilter
    virtual bool nativeEventFilter(const QByteArray &eventType, void *message, long *result);

signals:
    //Signals for the states of USB devices
    void SerialDeviceInserted();
    void SerialDeviceRemoved();
};

USBEventHandler::USBEventHandler(QObject *parent) : QObject(parent)
{
    //Check for the token every time a device is inserted or removed
    connect(this, SIGNAL(SerialDeviceInserted()), SLOT(checkDeviceID()));
    connect(this, SIGNAL(SerialDeviceRemoved()), SLOT(checkDeviceID()));
}

bool USBEventHandler::nativeEventFilter(const QByteArray &eventType, void *_message, long *result)
{
#ifdef Q_OS_WIN
    //Get the message and interpret it
    MSG *message = static_cast<MSG*>(_message);

    //Get the device structure
    PDEV_BROADCAST_HDR device = reinterpret_cast<PDEV_BROADCAST_HDR>(message->lParam);

    //If device is inserted
    if (message->message == WM_DEVICECHANGE && message->wParam == DBT_DEVICEARRIVAL)
    {
        if (device->dbch_devicetype == 0x3) emit SerialDeviceInserted();

    }
    else if (message->message == WM_DEVICECHANGE && message->wParam == DBT_DEVICEREMOVECOMPLETE)
    {
        if (device->dbch_devicetype == 0x3) emit SerialDeviceRemoved();
    }
#endif
    return false;
}
```

Code Snippet 4.2

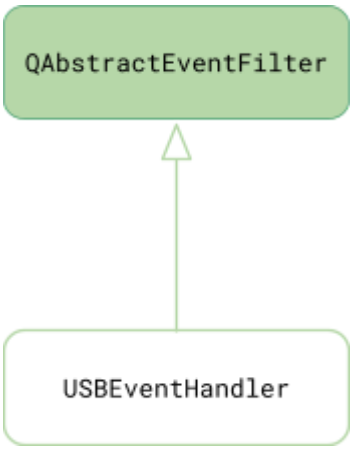


Figure 4.13

4.2.2.2 GNU/Linux Systems

On GNU/Linux systems, there isn't an option to register for device notifications via the OS itself. The only option was to use libudev[7] and libusb[8] and create a timer which polls the device every *X* milliseconds.

```
USBEventHandler::USBEventHandler(QObject *parent) : QObject(parent)
{
#ifdef Q_OS_LINUX
    timer = new QTimer(this);

    udev = udev_new();
    if (!udev)
        qDebug() << "Failed to allocate new udev";

    //Initialise the timeout for Udevs pooling
    udevTimeout.tv_sec = 1;
    udevTimeout.tv_usec = 0;
    //Initialise the udev monitor in order to be able to pool the devices
    mon = udev_monitor_new_from_netlink(udev, "udev");
    //Add a filter to get only USB devices
    udev_monitor_filter_add_match_subsystem_devtype(mon, "usb", NULL);
    //Enable receiving for the monitor
    udev_monitor_enable_receiving(mon);

    //Connect the timer's timeout with the tick function, to check every X ms
    connect(timer, SIGNAL(timeout()), this, SLOT(tick()));
    timer->start(250/*X*/);
#endif
    //Check for the token every time a device is inserted or removed
    connect(this, SIGNAL(SerialDeviceInserted()), SLOT(checkDeviceID()));
    connect(this, SIGNAL(SerialDeviceRemoved()), SLOT(checkDeviceID()));
}

#ifdef Q_OS_LINUX
void USBEventHandler::tick()
{
    //Get file descriptor for events from udev
    deviceFD = udev_monitor_get_fd(mon);
    //Create a file descriptor set
    fd_set fds;
    //Initialise to zero
    FD_ZERO(&fds);
    //Populate the set
    FD_SET(deviceFD, &fds);

    //Retrieve the events
    int ret = select(deviceFD+1, &fds, NULL, NULL, &udevTimeout);
    if (ret <= 0)
        return;

    //Check if the descriptor is in the set
    if (FD_ISSET(deviceFD, &fds))
    {
        //Get the device from the monitor
        struct udev_device* dev = udev_monitor_receive_device(mon);
        if (dev)
        {
            //Get the event from the device
            if (udev_device_get_devnode(dev))
            {
                const char* action = udev_device_get_action(dev);
                //Emit the signals accordingly
                if (strcmp(action, "add") == 0) emit SerialDeviceInserted();
                else if (strcmp(action, "remove") == 0) emit SerialDeviceRemoved();
            }
        }
    }
}
#endif
```

Code Snippet 4.3

As seen in snippet 4.3, a `QTimer` is created which has its `timeout()` signal connected to the `tick()` slot which checks if the device is connected or not.

4.3 Browser Extension Communication

4.3.1 Native Messaging API Test

The desktop app needs to serve the browser extension with any data it requires such as credentials or if the user wants to login from the browser extension. All of this has to happen without halting the UI or slowing it down.

The first try was to use the Native Messaging[9]API for browser extensions, it allows extensions to communicate with desktop apps. It also enables extensions to access resources that are not accessible through WebExtension APIs.

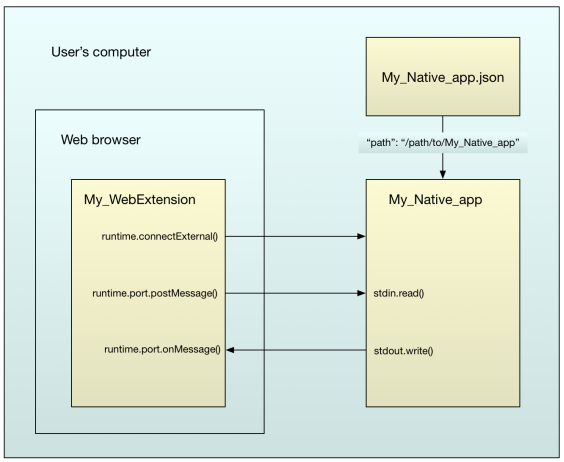


Figure 4.14

The code for testing this API in the desktop app was an implementation of `QAbstractEventFilter`[10] which ran a slot `readFromStdin()` every time something arrived at `stdin` and the signal was fired (Code Snippet 4.4). However, due to the API being poorly documented at the time of development and the API using plaintext to communicate with native desktop apps, it was excluded after multiple tries of getting it to work.

```
MainWindow::MainWindow(QWidget *parent, USB_communicator *usb_comm) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ...
    this->stdinNotifier = new QSocketNotifier(fileno(stdin), QSocketNotifier::Type::Read, this);
    ...
    connect(this->stdinNotifier, SIGNAL(activated(int)), this, SLOT(readFromStdin()));
    ...
}

QString MainWindow::readFromStdin()
{
    std::string stdinMessage;
    std::cin >> stdinMessage;
    QString retMessage = QString::fromStdString(stdinMessage);


    this->ui->browserExtensionLabel->setText("Extension Message: " + retMessage);

    return retMessage;
}
```

Code Snippet 4.4

4.3.2 Embedded Web Server (HTTP Communication)

The solution was to use a small embeddable HTTP server -SandBird by rxi- in the desktop application in a separate thread using QThread[11]. The HTTP server waits for any request coming from the browser extension.

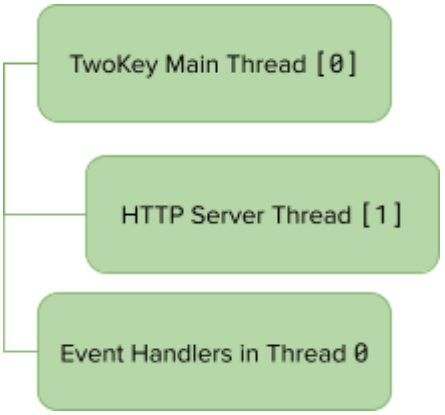


Figure 4.15

Using Diffie-Hellman key exchange, the desktop app and the browser extension agree on an encryption key to use per session which allows them to encrypt the traffic between them.

```
int compute(int a, int m, int n)
{
    int r;
    int y = 1;

    while (m > 0)
    {
        r = m % 2;
        if (r == 1)
            y = (y*a) % n;

        a = a*a % n;
        m = m / 2;
    }

    return y;
}

int generate_key(int public_B)
{
    srand(time(0));
    int p = 997;
    int g = 5;
    int secret_alice = rand();

    int public_A = compute(g, secret_alice, p);

    printf("Desktop App's Public Key: %d", public_A);

    int key_A = compute(public_B, secret_alice, p);
    printf("Secret Key: %d", key_A);
}
```

Code Snippet 4.5

After the keys are exchanged successfully, the browser extension is able to communicate with the desktop app securely. An example for a login scenario from the browser extension can be seen in code snippet 4.6.

```
void BrowserExtensionCommunicator::startServer()
{
    running = true;
    qDebug() << "Server running at http://localhost:" + QString(opt.port);

    while(true)
    {
      sb_poll_server(server, 1000);
    }
}

int BrowserExtensionCommunicator::event_handler(sb_Event *e)
{
    BrowserExtensionCommunicatorSignalWrapper* signalWrapper = (BrowserExtensionCommunicatorSignalWrapper*)e->udata;
    if (e->type == SB_EV_REQUEST)
    {
        char *masterpasswd = reinterpret_cast<char*>(malloc(1000));
        sb_get_var(e->stream, "masterpasswd", masterpasswd, 1000);

        char *email = reinterpret_cast<char*>(malloc(1000));
        sb_get_var(e->stream, "email", email, 1000);

        char *dec_masterpasswd = decrypt(masterpasswd);
        char *dec_email = decrypt(email);
        ...
        if (strlen(dec_email) && strlen(dec_masterpasswd) &&
            signalWrapper->getBackendClient()->login(QString(email), QString(masterpasswd)) ==
            BackendClient::loginStatus::SUCCESS)
        {
            sb_send_status(e->stream, 200, "OK");
            sb_send_header(e->stream, "Content-Type", "text/plain");
            signalWrapper->emitSuccessfulLogin();
            sb_writef(e->stream, signalWrapper->getBackendClient()->getJwt().toStdString().c_str());
        }
        else if (!strlen(url))
        {
            sb_send_status(e->stream, 401, "Unauthorised");
            sb_send_header(e->stream, "Content-Type", "text/plain");
            sb_writef(e->stream, "Unauthorised login, wrong password.");
        }

    }
    ...
}
```

Code Snippet 4.6

4.4 Server Backend Communication

Using `QNetworkRequest`[12] and `QNetworkAccessManager`[13] an encrypted secure connection is established for traffic transfer between the desktop app and the server.

The API functions for TwoKey are available at https://twokey.tech/<function>, the function could be register, login, 2fa, etc...

The desktop app can invoke web requests with JSON content containing information such as the email, password hash, requested URL, etc...

All of the communication is encrypted over an HTTPS connection, which is managed by Cloudflare, inc. (i.e. not self signed). Thus creates no SSL errors.

```
int BackendClient::login(QString email, QString password)
{
    QNetworkRequest request(QUrl("https://twokey.tech/login"));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
    QJsonObject json;

    QCryptographicHash *sha256sum =  new QCryptographicHash(QCryptographicHash::Algorithm::Sha256);
    QByteArray password_hash = sha256sum->hash(password.toUtf8(), QCryptographicHash::Algorithm::Sha256);

    json.insert("email", email);
    json.insert("password_hash", QString(password_hash.toHex()));

    QNetworkAccessManager nam;
    QNetworkReply *reply = nam.post(request, QJsonDocument(json).toJson());

    while (!reply->isFinished())
    {
        qApp->processEvents();
    }

    QByteArray response_data = reply->readAll();
    QJsonDocument jsonResponse = QJsonDocument::fromJson(response_data);

    QString challenge = jsonResponse.object()["challenge"].toString();
    this->jwt = jsonResponse.object()["Access-token"].toString();

    //Send the challenge to the usb token and receive it
    usbComm->writeToToken(challenge.toStdString().c_str());
    tokenChallengeResponse = usbComm->readFromToken();
    //Begin 2FA process
    int loginStatus = _2fa();
    delete reply;
    delete sha256sum;
    return  loginStatus;
}
```

Code Snippet 4.7

Code snippet 4.7 demonstrates the login function from the backend client in the desktop application. It sends a request containing the password hash and email to the server, and then starts the 2FA process by sending the challenge received in the response to the token and then sending the result back to the server.

4.5 Credentials Generation

The desktop app generates customisable random strong passwords with uppercase and lowercase characters, numbers, and symbols with custom length which follow all the password strength guidelines and recommended tips to prevent the user from using guessable or easily-crackable passwords and the common dictionary passwords and it warns the user if a weak password (previously picked) is added or replaces an existing password.

Password strength is assessed based on 4 factors:

- Password length has to be at least 11 characters long.
- Contains a mix of lowercase [a-z] and uppercase [A-Z] characters
- Contains at least one special character
    - Special characters are defined as !@#$%^&*
- Contains at least one numeric character [0-9]

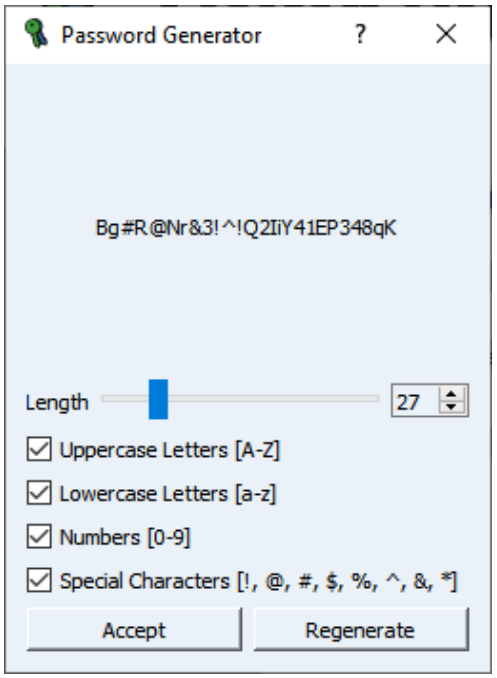


Figure 4.16: Password generation dialog

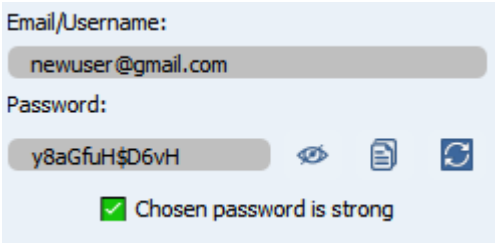


Figure 4.17: A strong password is chosen

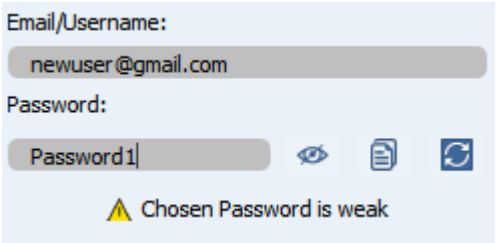


Figure 4.18: A weak password is chosen

4.6 Key Derivation

Using Crypto++[14] a key derivation function (Figure 4.19) was implemented for key stretching using the user's master password and a salt value and a number of iterations for repeating the process as many times as possible.

Unlike the regular hashing which is vulnerable to brute force attack in which the attacker tries as many combinations of passwords as possible till a matching hash is found.

Another type of attacks that the common password hashing techniques are vulnerable to is the dictionary/rainbow table attack in which the attacker uses a huge (multiple gigabytes) precomputed dictionary of passwords and the corresponding hashes and takes the password hash and look it up in the dictionary and check if it exists the attacker can take the password.

The key derivation function makes the password cracking more difficult taking a grueling amount of time to attack and as the number of iterations and the amount of repetitions of the function increases the amount of time required for the password to be cracked increases making it harder and harder to attack.

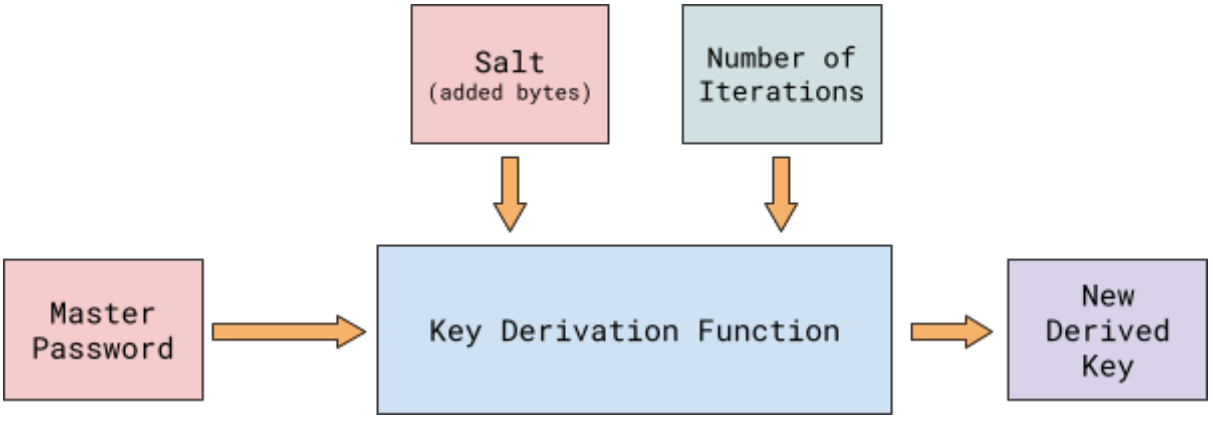


Figure 4.19

Chapter 5: Browser Extension

5.1 Module Definition

A browser extensions are plugins for web browsers that add and extend functions and features to browsers. Extensions can modify the user interface or add web service functionality to the browser. This makes the browser extension an important module in the project as it makes it easier for the user to use TwoKey without having to return to the application to retrieve the password every time.

The passwords are stored in TwoKey's database and are automatically retrieved by the browser extension if the domain name is saved in the database. This gives the ability for the browser extension to auto-fill all of the user's saved passwords, making the user experience more seamless and convenient. This must happen without sacrificing security. This was discussed in chapter 4, section 3: Embedded Web Server (HTTP Communication). The extension also protects the user from phishing links as it verifies the domain name, e.g. it won't mistake gogle.com for google.com like a human might.

Browser Extension has a session which is in sync with the desktop application. If the desktop application isn't logged in, the browser extension doesn't log in.

The Browser Extension communicates with the server only through the desktop app and doesn't communicate directly with TwoKey's server. All the communication between the desktop app and the browser extension is encrypted (as discussed in chapter 4, section 3). The role of this module is critical as it ensures the convenience of the user, as simply clicking on the account the user wants to log in makes it autofill its credentials.

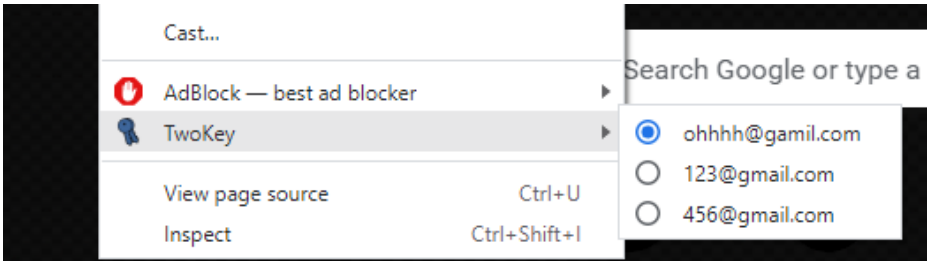


Figure 5.1: The different accounts selection

5.2 Browser Extensions APIs

```
chrome.tabs.onUpdated.addListener(getTab)
function getTab()
{
  chrome.tabs.query({active: true, currentWindow: true}, tabs => {
          let link = tabs[0].url;

           var http=new XMLHttpRequest();
           const url="http://localhost:8000/?url="+link;
           http.open("GET",url,true);
           http.send("");
           http.onreadystatechange = (e) => {
           retrievedCredentials = http.responseText;
           if (retrievedCredentials.length != 0)
              console.log(retrievedCredentials);
           }
           chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
      });
}
```

Code Snippet 5.1: Chrome Tabs API

A lot of browser extension APIs were utilised such as the tabs API[15] which returns tab information, such as Tab ID, Current Browser Window, Current Browser Window ID, Currently opened URL. The main use of this API is retrieving credentials for the current web page being browsed through its URL. The URL is sent to the desktop app, which then retrieves the credentials for the specified account if the user is authenticated.

The Runtime API[16] is also used extensively for sending messages between the background script and the content script of the web page. Changes such as font changes, colour, text highlighting, are handled through this API. As well as retrieving information from webpages.

Menus or contextMenu API[17] is critical to the system as it plays a very important role in the autofill feature. Many users have multiple accounts for the same service. This API allows the user to have a context menu which allows them to select the account they want to log in to (Figure 5.1). The list of accounts are retrieved from the server by the desktop app when it receives the URL from the browser extension.

```
for (int i = 0; i < 3; i++)
{
  chrome.contextMenus.create({id:i.toString()}, type: 'radio', title:x[i]});
  chrome.contextMenus.onClicked.addListener((info, tab) => {
    alert("Item " + info.menuItemId + " clicked " + "in tab + tab.id");
    //Send account information to the desktop app
  })
}
```

Code Snippet 5.2: Chrome contextMenu API

5.3 API Problems

As mentioned before, the browser extension relies heavily on the APIs like the tabs API. The problem with the tabs API is that the `onActivated()` listener only listens to tab change events but not tab update events. i.e. it only updates when the tab is switched, not if the url is changed. The workaround for this was to use two event handlers (Figures 5.3 and 5.4), one for the current tab if the user changes the webpage and another for when the user switches between open tabs. This ensures that all the browser extension is always in the loop with what the user is browsing, in order to be able to retrieve the credentials for the service being browsed.

5.3.1 Cross-Compatibility

Browser Extension APIs create a uniform landscape for the development of browser extensions. However, among the browsers that use the extensions API (the major ones being Chrome, Firefox, Opera, and Edge), there are differences in both the implementation of the API and the scope of coverage. Not just that, Safari uses its own proprietary Safari Extensions JS.

There are two API namespaces, which are `browser.*`, the proposed standard for the extensions API, used by Firefox and Edge. The other being `chrome.*` which is used by Chrome and Opera. While the namespace is called chrome, some of its functions are usable with Firefox, Edge, and other browsers too.

5.3.2 Deprecated APIs and Poor Documentation

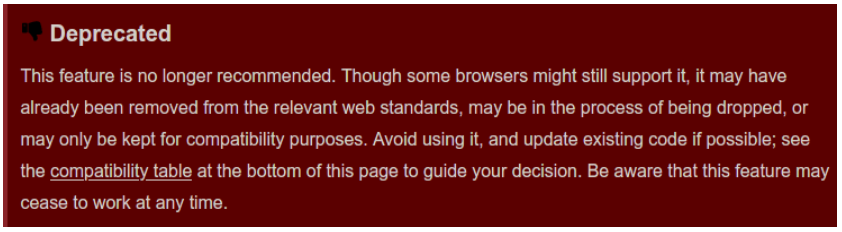


Figure 5.2: A deprecated API message from the documentation

Another problem related to the extensions API was the lack of resources and usable documentation. A lot of the resources and documentation are outdated. A lot of time was spent experimenting with different APIs to find out which work for which use cases. Even then, a lot of APIs had unpatched bugs which needed manual handling.

5.3.3 Native Messaging API

Native Messaging API is a protocol which is recommended by Chrome to communicate with a local application. However, it is riddled with bugs such as occasionally refusing to open executables whether on Linux and Windows.

Sometimes, the API would also log random errors which prevented it from functioning. This, alongside the poor documentation, led to this API not being a usable option. It is worth noting that the Native Messaging API's documentation was updated after the project was finished (implementation-wise) on May 24, 2021, by MDN contributors.

The API also had a problem which was that it sends data in plain text, and relies on STDIN/STDOUT to communicate with desktop apps. This makes it very easy for any program running in the userspace to read the traffic between the desktop app and the browser extension.

That made it not suitable for TwoKey as it is a security-oriented project. The final implementation for the communication between the browser extension and the desktop app which ensures the security of the credentials and the info of the users has already been discussed in Chapter 4, Section 3: Browser Extension Communication.

5.4 Solutions for the Browser Extension Problems

As discussed previously the APIs had a lot of problems which needed manual handling instead of relying on different APIs.

For example, the problem of the tabs API mentioned previously was handled by adding the two listeners mentioned previously.

```
chrome.tabs.onUpdated.addListener(getTab)
function getTab()
{
        chrome.tabs.query({active: true, currentWindow: true}, tabs => {
                let link = tabs[0].url;
                console.log(link);
                 var http=new XMLHttpRequest();
                 const url="http://localhost:8000/?url="+link;
                 http.open("GET",url,true);
                 http.send("");
                 http.onreadystatechange = (e) => {
                   retrievedCredentials = http.responseText;
                   if (retrievedCredentials.length != 0)
                      console.log(retrievedCredentials);
                 }
                 chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
            });
}
```

Code Snippet 5.3

```
chrome.tabs.onActivated.addListener(getTab)
function getTab()
{
        chrome.tabs.query({active: true, currentWindow: true}, tabs => {
                let link = tabs[0].url;
                console.log(link);
                 var http=new XMLHttpRequest();
                 const url="http://localhost:8000/?url="+link;
                 http.open("GET",url,true);
                 http.send("");
                 http.onreadystatechange = (e) => {
                   retrievedCredentials = http.responseText;
                   if (retrievedCredentials.length != 0)
                      console.log(retrievedCredentials);
                 }
                 chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
            });
}

var sending = chrome.runtime.sendMessage(retrievedCredentials);
```

Code Snippet 5.4

The advantages of this workaround is that it ensures that the desktop app always has the correct URL in order to be able to return the credentials for it. The disadvantage is that it makes the code more verbose and sometimes sends the url twice. A filter on the desktop app is needed in order to fulfil the request only once.

The second problem for autofill is that if the user has multiple accounts on the same website and wants to switch between them, it won’t be possible if TwoKey only autofills one account. This was solved by using the contextMenu API which shows the user all of the available accounts for the current website being browsed.

5.5 Communication Problems and Solutions

As discussed previously, the native messaging API wasn't suitable for the communication between the desktop app and the browser extension.

The main issues of the API is that it sends data unencrypted to the desktop application and that isn't secure as it could leak user secrets if there was a third party on the system.

This problem was solved by using a local embedded HTTP server in the desktop app (refer to Chapter 4, Section 3: Browser Extension Communication) this didn't affect the performance or the speed of the extension as a local web request containing few bytes doesn't take much overhead at all. Even though the communication is over HTTP, it is still secure and encrypted, with diffie-hellman being the method of key exchange between the modules in order to not leak secrets and communicate securely and efficiently.

The advantages of using a local minimal server is that it is far more secure than using native messaging, without compromising a lot on speed.

The disadvantage is that an extra thread was added to the desktop app in order to handle the requests, and while this isn't an issue on most modern systems, it may be an issue on less capable machines.

Chapter 6: Server Backend and Deployment

6.1 Module Definition

The backend is the technology required to process incoming requests and respond to the client appropriately with the requested data or action.

The backend of TwoKey is responsible for 5 major parts:

1. Secure Communication with Desktop App
2. Server Management: The hosting device which receives the incoming requests
3. User Authentication and Login
4. Registration
5. Session Management
6. Securing and Managing the User Data.

Each point will be discussed in the next sections.

6.2 Secure Communication

6.2.1 Using HTTP with TLS/SSL

Communication between the server and the desktop is the only way that the user can access their data. So, securing the communication between them is critical to the privacy and the security of the user to prevent any interception of any data being transmitted from the server to any client by a third party.

HTTP is the standard protocol for the wide majority of web apps. Why can't it be used for communication in TwoKey?

It is not effective because it transfers the data in the clear without any kind of encryption. So, the solution is to use HTTPS.

HTTPS is HTTP plus another layer of security called TLS/SSL.

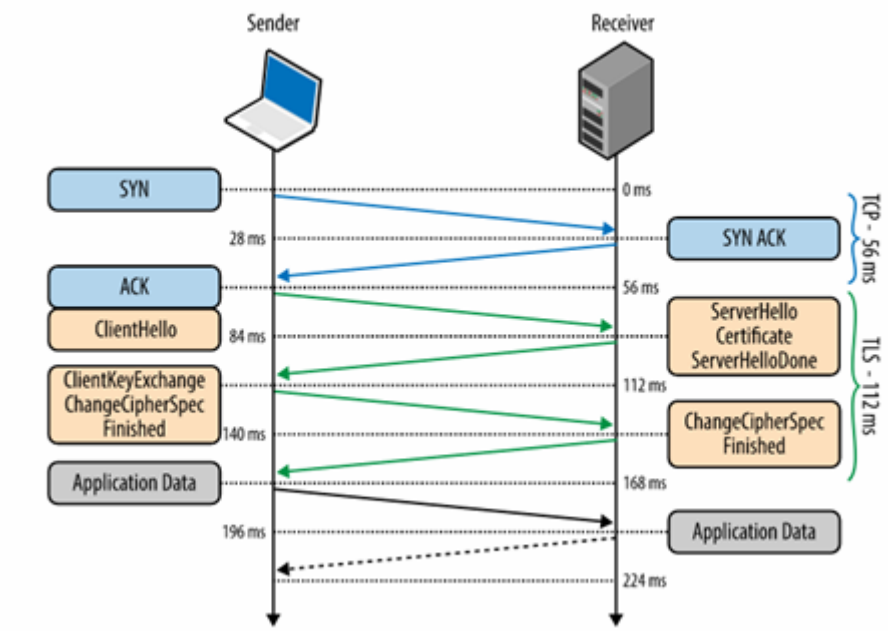


Figure 6.1

It typically uses an asymmetric encryption technique which is usually RSA to make sure no one can view the content of any transmitted data.

Implementing the first edition of the self signed HTTPS is discussed in the next section, but a problem with self signed SSL certificates is that they are vulnerable to impersonation. That's why TwoKey's certificate had to be certified by a certificate authority in order to avoid using a self-signed certificate. So, the best solution was to use Cloudflare so that we can use cloud flare to do the job.

With Cloudflare, traffic going to the end user will always be encrypted. There are several options when implementing Cloudflare:

1. To not secure any data transmission. (Use plain HTTP)
2. Only encrypts the traffic between Cloudflare and the end user
    a. That is, the desktop app in the case of TwoKey.
3. To use a self-signed certificate with the internal communication on the server itself, and use an SSL certificate which encrypts the data end to end from the desktop app to the server.
4. Use the previous setting, but replace the self-signed certificate on the server with a trust CA certificate.

The first two options aren't valid to TwoKey's design philosophy of not transmitting any unencrypted data.

The fourth option (getting a trust CA certificate) isn't a valid option as TwoKey isn't a registered company.

That leaves only one option which is using Full mode, explained in figure 6.2.

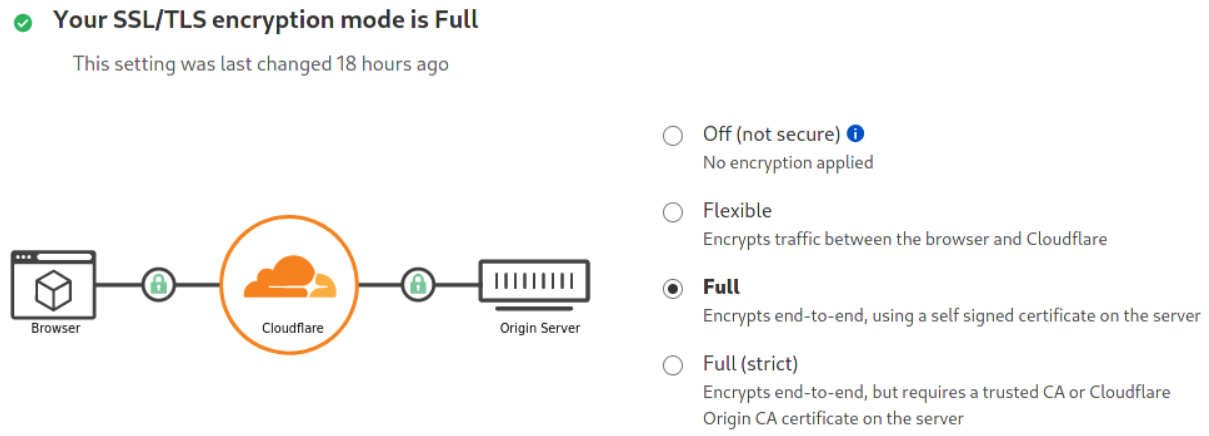


Figure 6.2

6.2.2 DDoS Protection

Cloudflare can also successfully protect servers from distributed denial of service (DDoS) attacks.

The four stages of mitigation which Cloudflare takes are:

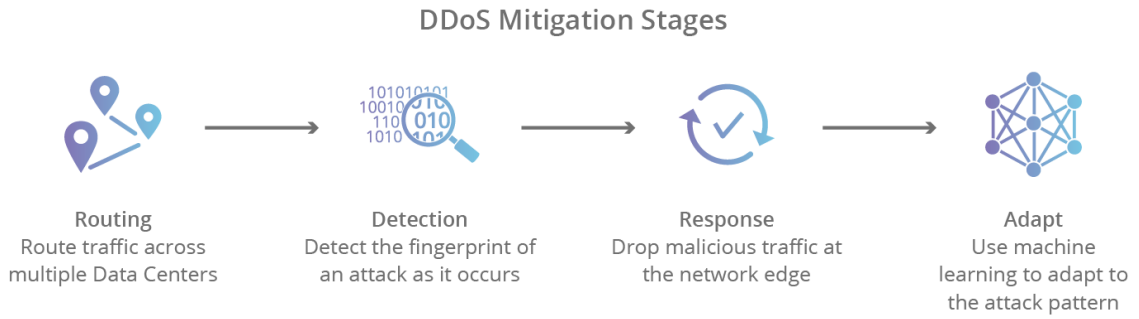


Figure 6.3 The DDoS Mitigation Stages by CloudFlare

1. **Routing** - By intelligently routing traffic, an effective DDoS mitigation solution will break traffic into manageable chunks over multiple data centers to make it possible to detect denial of service attacks.
2. **Detection** - It should be able to distinguish an attack from the intensity of normal traffic with correlation to the IP reputation of the originating connections, and know the common attack patterns.
3. **Response** - It should respond to an incoming identified threat by intelligently dropping malicious bot traffic, and letting the rest of the traffic pass through.
4. **Adaptation** - It analyzes traffic for patterns such as repeating offending IP blocks. By adapting to attack patterns, a protection service can harden itself against future attacks.

6.3 The Hosting Server

6.2.1 Nginx and uWSGI servers

The Backend in our project is implemented using Flask (a python-based framework), Flask has a built-in web server (uWSGI) but it has a problem in scalability, meaning it handles requests sequentially in turns, one at a time.

So, to solve this problem NGINX was used. Which acts as a proxy to handle multiple requests at a time and a reverse proxy to communicate with Flask API through uWSGI. Because NGINX cannot directly communicate with Flask.

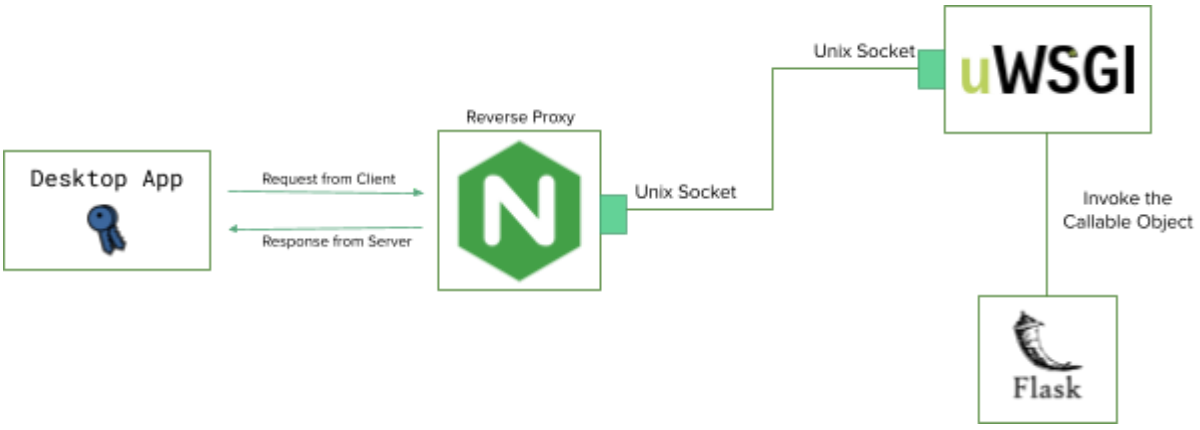


Figure 6.4: The integration of the backend components

6.2.2 Online Hosting (DigitalOcean)

TwoKey needs a 24/7 uptime and constant availability. DigitalOcean is a cloud infrastructure platform which allows for easy scalability. Hence, it was chosen.

TwoKey's backend is hosted on a DigitalOcean droplet running Ubuntu 20.04 LTS which allows for 24/7 uptime and easy management on the go without worrying about the underlying infrastructure.

It is worth noting that TwoKey is open source and can be self-hosted per organisation on the server of their choice.

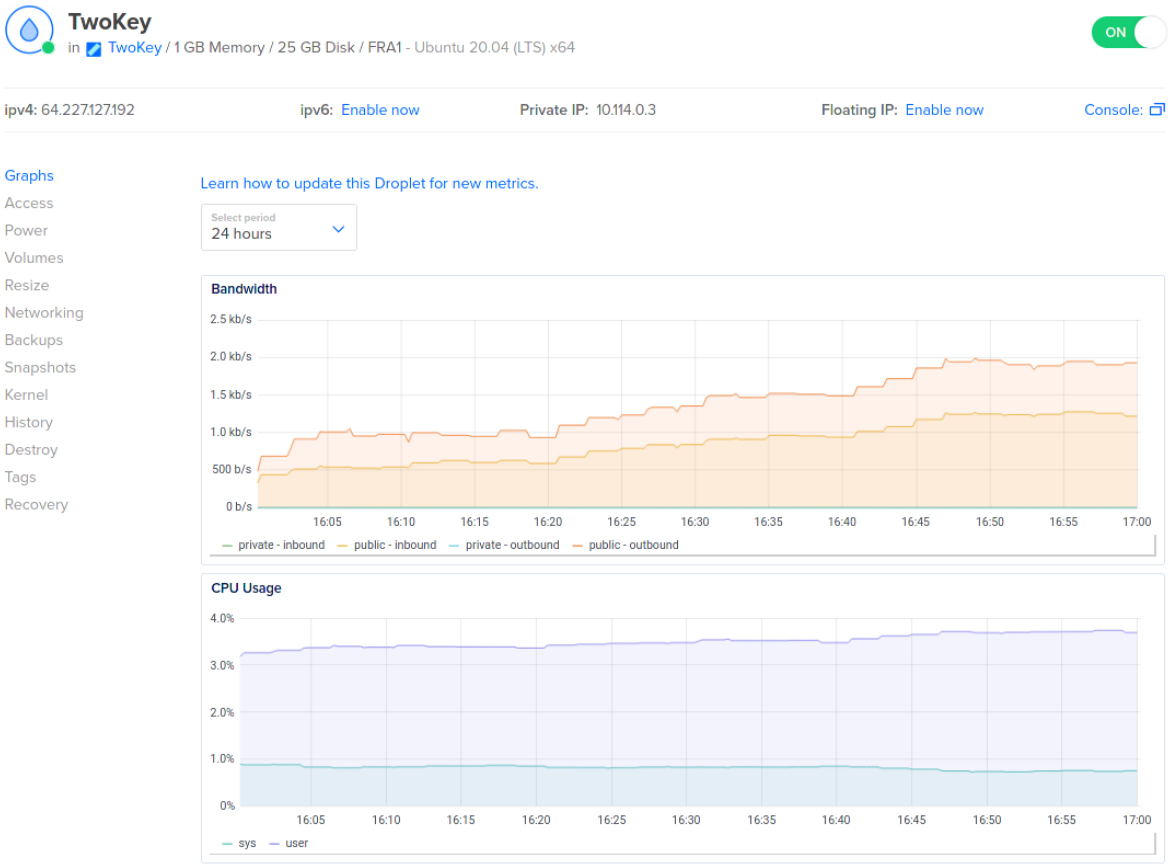


Figure 6.5: TwoKey's dashboard on DigitalOcean, the cloud hosting platform

6.4 User Authentication

6.4.1 Authentication

First, let's discuss how authentication works in the first place,

1. The user enters their email and password to verify themselves, this is the first factor of authentication.
2. The server checks if the email and password are correct.
   2.1. Are they correct? The server opens a session for the user (discussed later)
   2.2. Not correct? User is not authenticated and no session is created.

This is typically how the authentication process works[1]. But, this is not secure enough to verify that the user is who he claims to be. As any adversary could have gotten their hands on the user's email and password. So, two-factor authentication is applied.

6.4.1.1 Authentication Maturity Model

Figure 6.6: The Consumer Authentication Strength Maturity Model (CASMM) v5 by Daniel Miessler[18]

6.4.2 The Factors of Authentication

The factors of authentication are mainly five:

- Something you know (Password, passphrase, etc...)
- Something you are (Fingerprint, eyescan, voice, etc...)
- Something you have (Credit Card, Mobile Phone, Key, etc...)
- Somewhere you are (A location)
    - Not commonly used as it is easily spoofable
- Something you do (Such as a construction worker)
    - This factor isn't used as people can impersonate job titles quite easily enough to pass for authentication

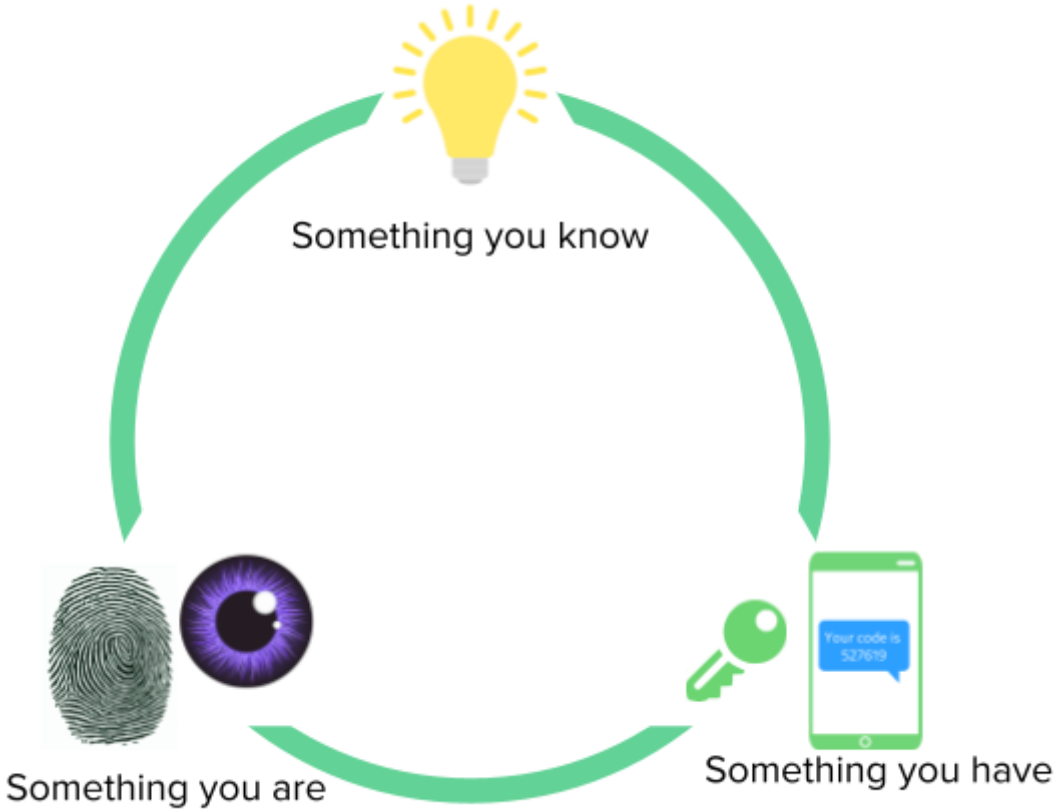


Figure 6.7: The Factors of Authentication

6.4.3 Applying 2FA to TwoKey

In the case of TwoKey, as mentioned previously, a physical hardware USB token is used as a second factor of authentication.

The full authentication scenario is as follows:

1. User enters their email and master password.
2. The server responds with a challenge to be encrypted using the user's physical token.
3. The desktop app sends the encrypted challenge to the server after it has been encrypted by the token.
    3.1. Does the challenge match the one on the server, and the user's email and master password are valid and match the token? A session is opened for the user, and a JWT is sent to the user.
    3.2. No? A session is not created for the user.

Figure 6.8 illustrates the 2FA process as a whole.

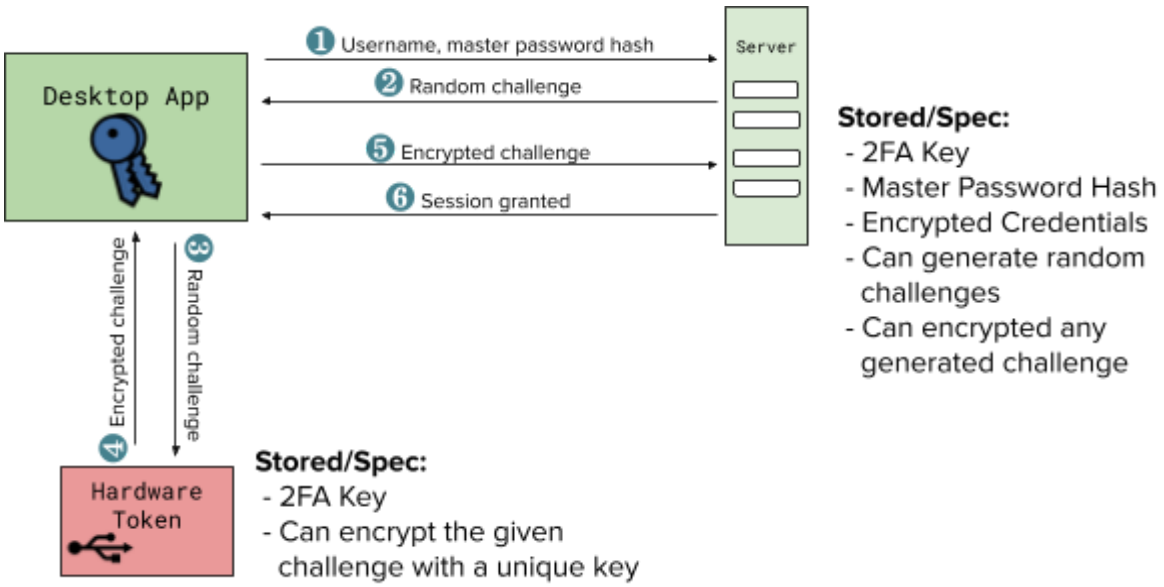


Figure 6.8: The 2FA process

6.5 Registration

When users buy the token in order to use it as their 2FA, they have to link the device with their user ID in order to bind the encryption key on the token with their account. This ensures that each user has a device which is linked to their account and can't be changed. This decreases the attack surface as the chance of impersonation greatly decreases.

To solve this problem, each device is accompanied with a unique serial number on it. In the registration phase, the user enters their registration data (first name, last name, email, master password, etc...) and the serial number of the device.

After the registration info is processed by the backend, the backend generates a random unique user identifier (UUID) for every user which links that device with their account. At that point, the serial is expired and can't be reused by any other user.

Figure 6.8 demonstrates the device encryption keys and their serial numbers and how they reside in the database.


```
"Physical_id" : "Lg8V2YcuOsM3Kq2r", "Serial" : "0757312d-31e7-4a80-b430-adbeac9b7acb" }
"Physical_id" : "sl5Vb3RsGOQtwAHM", "Serial" : "8wUy6" }
"Physical_id" : "rskDSkocuB6YuLxO", "Serial" : "lEE6g" }
"Physical_id" : "LDQAOcbqwBiYDk36", "Serial" : "zCcNp" }
"Physical_id" : "R1q55nhUK9SijkZx", "Serial" : "vxXHh" }
```


Figure 6.9

6.6 Session Management

6.6.1 Addressing the Statelessness of HTTP

HTTP is a stateless protocol. Which means that the connection between the server and the desktop app is lost once the transaction ends. Meaning that with each request, the users have to authenticate themselves from the start. That is very inefficient, and creates an overhead with each request the user sends.

So, from here came the idea of opening a session and closing it only when there are no more requests required or it times out after a certain time due to security concerns and resource management.

Basically, after authentication the server is supposed to give each user a unique token or ID which lasts as long as the session is open. Normally, each server has its own session ID per user. Session IDs stored on that server to verify each request in that session.

6.6.2 Issues when Load Balancing

However, this raises an issue when there are multiple servers for load balancing. If there are multiple requests on multiple servers at the same time, the first server which opens the user's session has to be available to handle other servers' requests as the session tokens are stored on it. The user needs to be able to send requests to any server without having to reauthenticate.

Which means that a way is needed to make the state in the token itself not on the server. And that is why the JWT[19] was chosen, as the state (info of the user) is stored in the token itself.

6.6.3 Protection Against User Enumeration Attacks

As discussed before, the user enters their credentials first. The server sends a temporary JWT with the 2FA challenge. A temporary session is opened using the temporary JWT with the received credentials which the user uses to reply with the encrypted challenge. This is done to prevent username enumeration attacks.

Username enumeration attack is a common vulnerability when an attacker can guess or know if a username is valid or not. Which makes it easy to brute force the password if the attacker already knows that the username is correct.

So, the temporary JWT and the random challenge sent to the user are stored in the database temporarily. After the final authentication, the temporary JWT is removed and a new JWT with a longer expiration time will be generated and sent to the user to be sent with each request after that.

Figure 6.9 illustrates how a JWT works

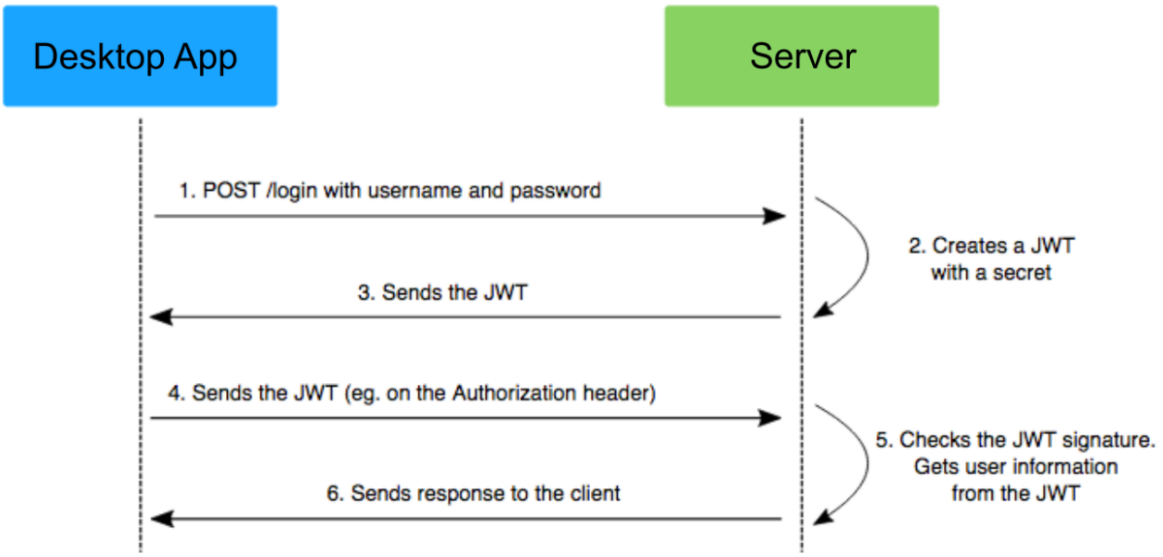


Figure 6.10

6.7 Database

6.7.1 Choosing the Database

There were multiple choices of what databases could be used. There's MongoDB, Firebase, and MySQL for example.

MySQL creates a strict schema-template. It employs the concept of storing data in rows and tables.

MongoDB has no restrictions on schema design. It stores the data in collections of JSON-formatted objects.

Firebase isn't open source, so it wasn't an option for this project as it aims to be 100% trusted free software.

So, MongoDB was used to have flexibility to add the user's data without any restrictions of how many accounts he has or how many accounts for the same service, without worrying too much about database schemas and the restrictions behind SQL. Especially since there aren't too many different types of data being stored, so it makes sense to use the simpler option.

6.7.2 Securing the Database

As discussed before, each user enters their master password which is then hashed at the client side and sent to the server to be stored in the database. The password itself is never transmitted, only the hash which is a SHA256 hash. This makes it extremely difficult for any attacker to get the user's master password as hashes are irreversible, and are usually cracked by dictionary or rainbow table attacks.

All of the passwords in the database are encrypted using a key derived from the master password (as discussed in Chapter 4, Section 6: Key Derivation) and are only decryptable at the client side. They are retrieved as they are stored, encrypted, and then sent to the client side where the desktop app decrypts them in memory and then removes them right after the user is finished with using them.

Chapter 7: Implementation and Testing

7.1 Use Case Diagram

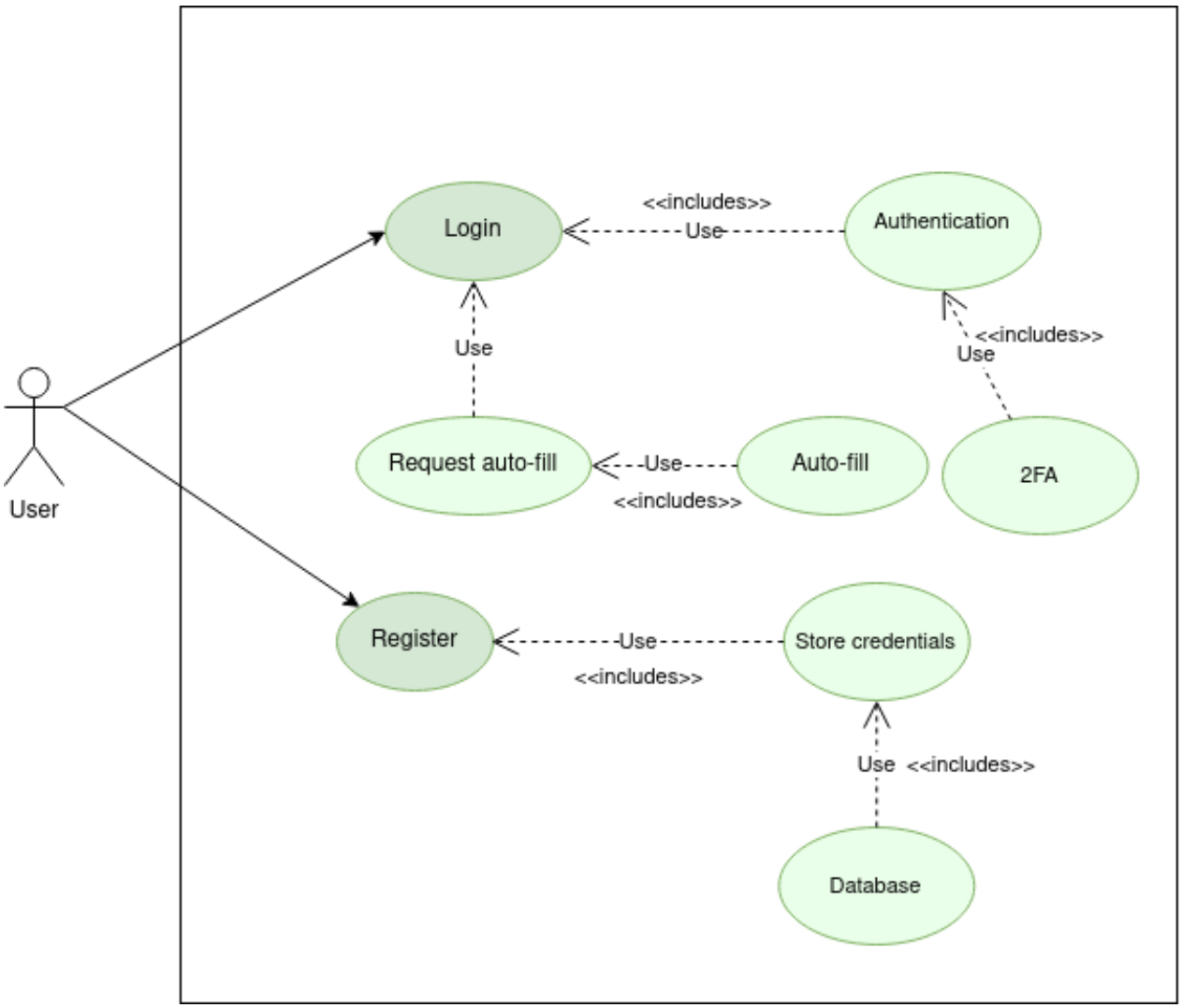


Figure 7.1 The use case diagram for TwoKey

7.2 Class Diagram

The following is the class diagram of the classes in the Desktop App which is the main component and the center of the whole project.

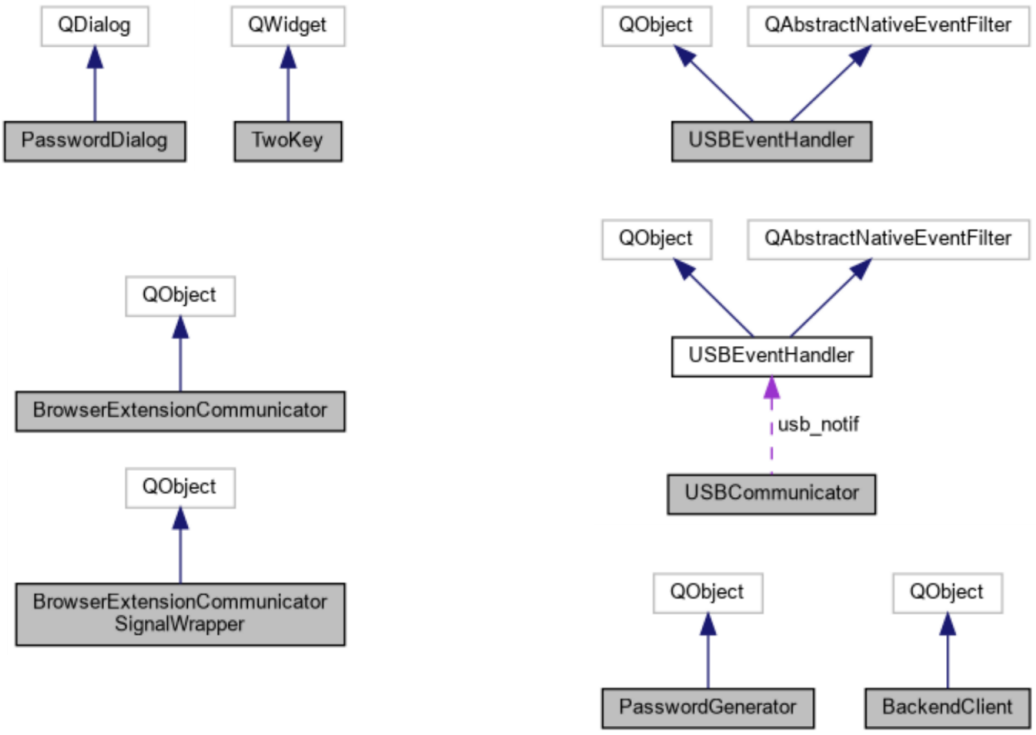


Figure 7.2 The class diagram for TwoKey

7.3 Communication Diagram of SandBird

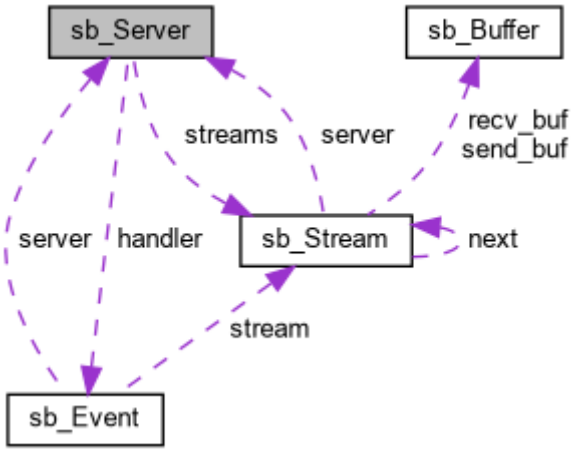


Figure 7.3 The communication diagram for SandBird

7.4 Sequence Diagram

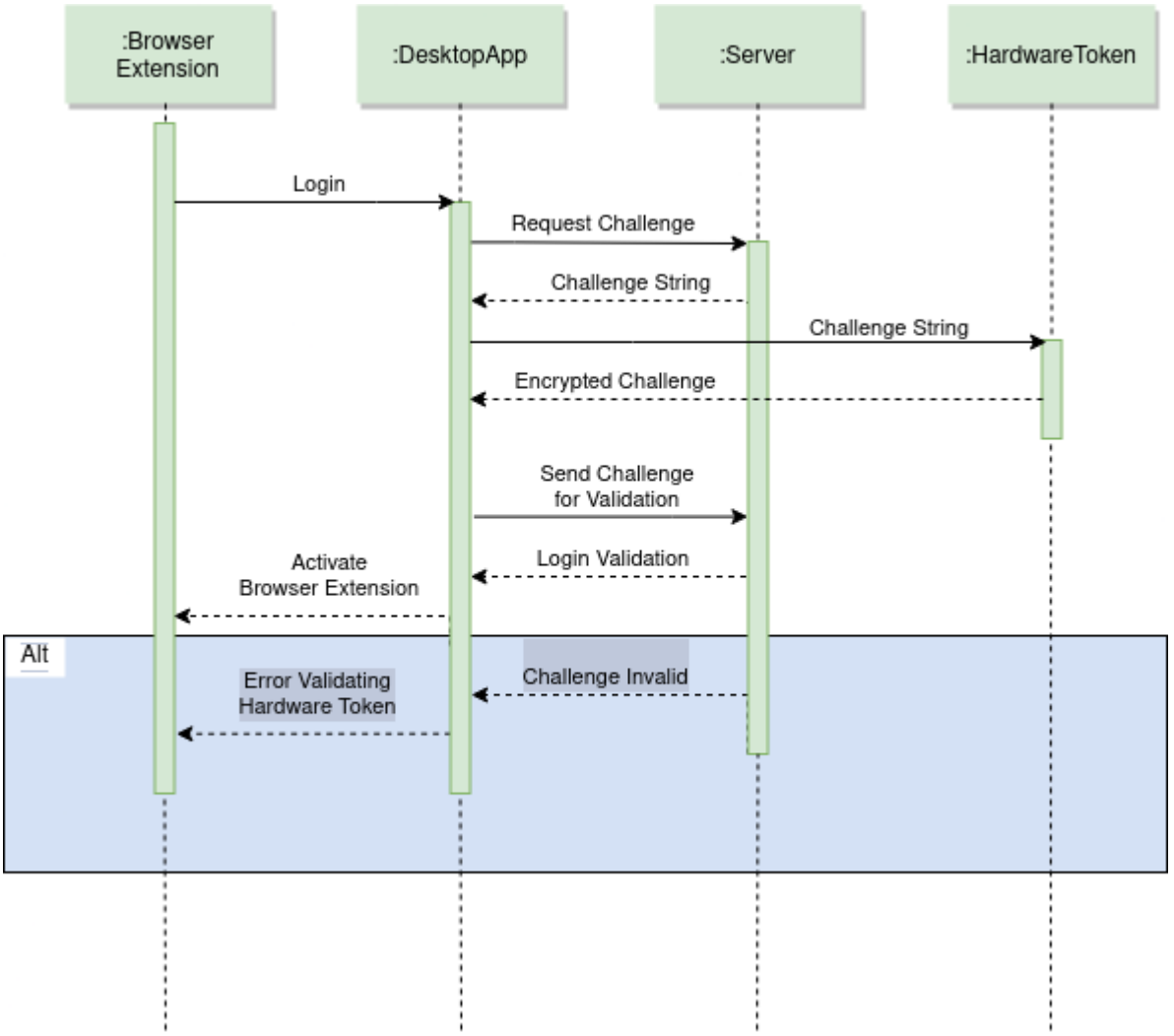


Figure 7.4 The sequence diagram for TwoKey

7.5 Testing and Results

7.5.1 Secure Programming

As discussed previously, TwoKey's desktop application is programmed with security in mind to avoid memory artifacts and memory leaks which may compromise the security of the user or slow the application down.

```
for (int i = 0; i < password->length(); i++)
{
    (*password)[i] = static_cast<char>(0);
}
delete password;
ui->manager_password->clear();
```

Code Snippet 7.1: An example on memory scrubbing in C++

To verify the security of the programming, the process was dumped in its locked/logged-out state and then analysed and checked against the passwords to see if they still reside in memory.

First step is logging in, verbose debug output is enabled in the background for demonstration purposes only. The master password for the test account is *test1234*. The password for the google account is *iLCJ@hbGciOiJIUzI1NiJ9**. As shown in figures 7.5 and 7.6, we can see that the token is collected (since the key is green).

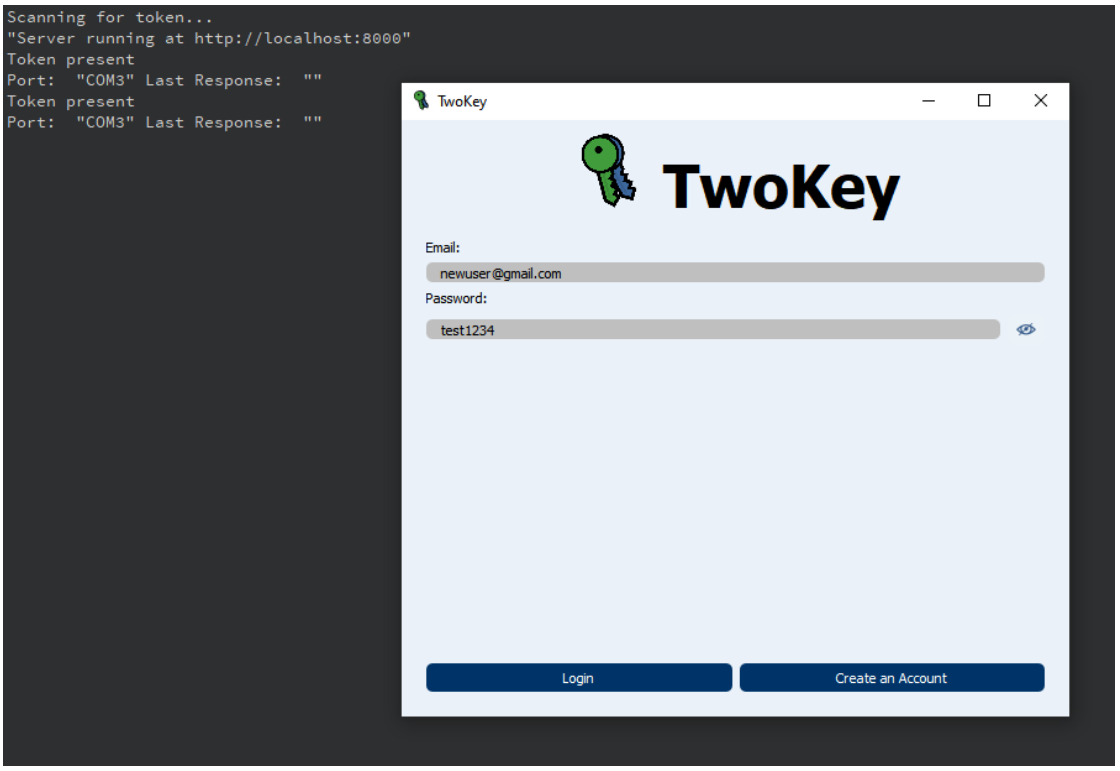


Figure 7.5

After the login succeeds, the user's vault can be accessed.

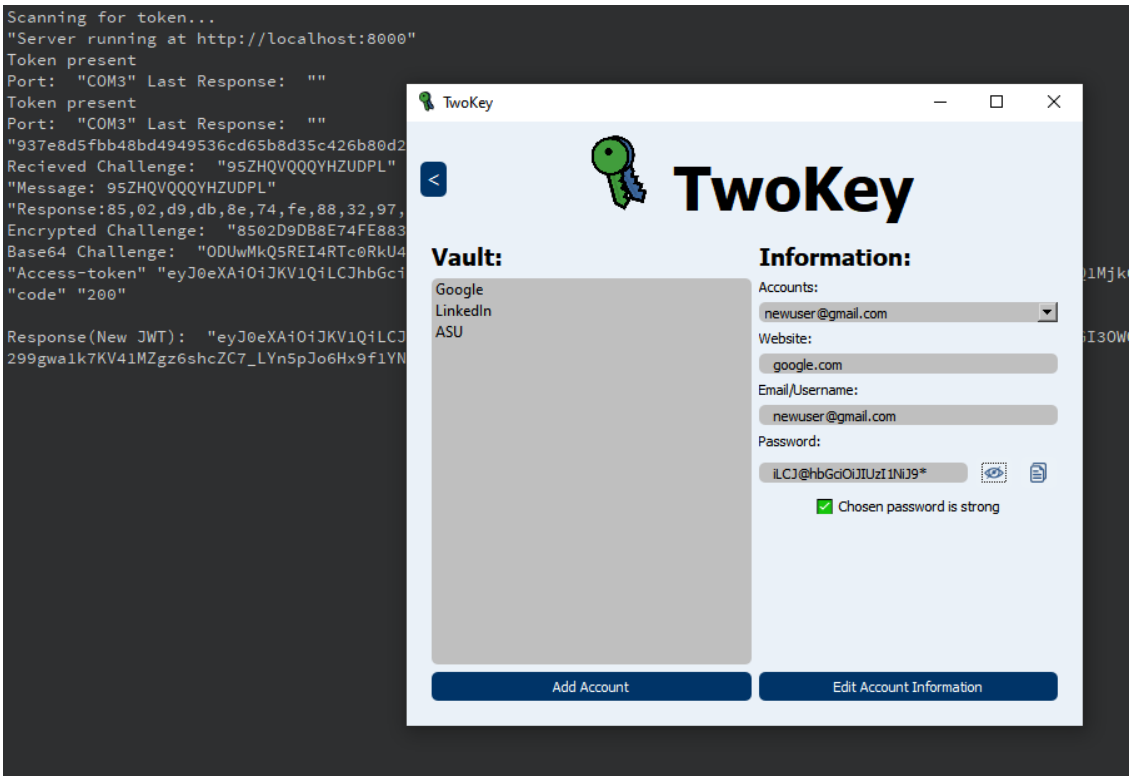


Figure 7.6

From now, like any process in Windows, the memory can be dumped by going to the task manager, and selecting “Create dump file”.

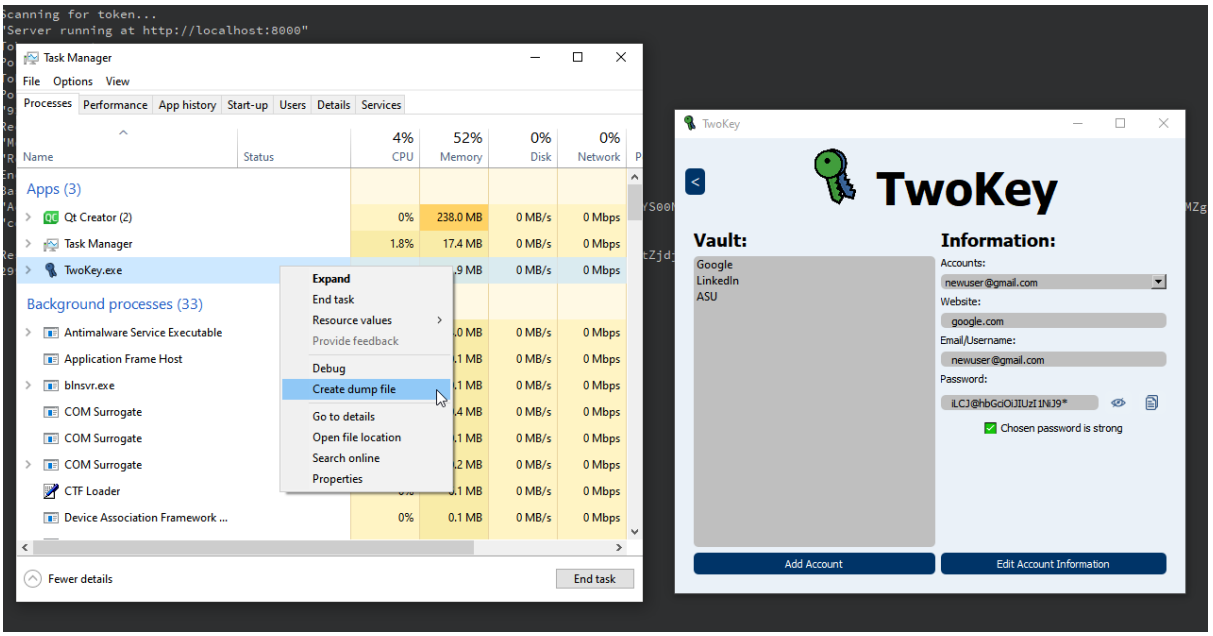


Figure 7.7: Dumping a process on Windows

After signing out, the process can be dumped to check if any artifacts or previous password are still resident in memory.

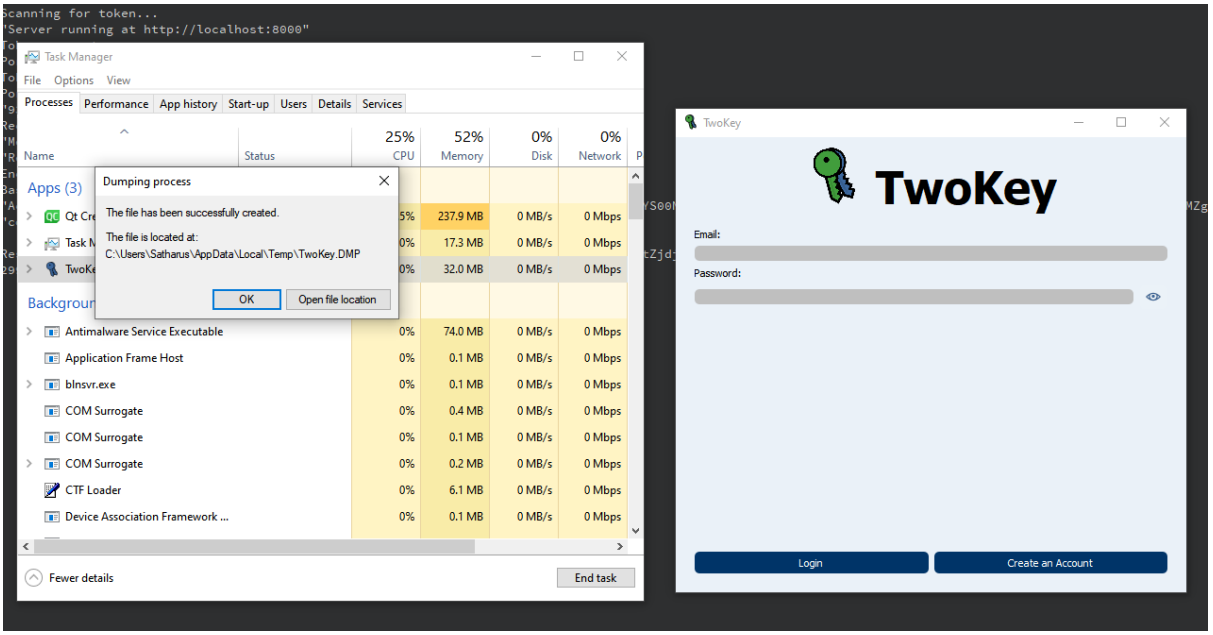


Figure 7.8: Dumping TwoKey after singing out

After acquiring the dump, it can be analysed using many tools. A simple way of checking whether certain ASCII bytes (in this case passwords) reside within a binary file is by running a tool like GNU Strings[20] which extracts all consecutive ASCII characters from a binary file which are at least of length 4. After extracting all of the ASCII strings from the memory dump, they can be searched through using a tool like GNU Grep[21].

The following figure (7.9) demonstrates that the passwords were not found in memory. The first two commands check if the output of grep is empty (i.e. no matches were found) and then print “No credentials found” if that’s the case.

The second two commands are there for verification, as grep returns no output.

```
[Hotline:Desktop]$ [[ -z `strings TwoKey.DMP  | grep "iLCJ@hbGciOiJIUzI1NiJ9"` ]] && echo "No credentials found"
No credentials found
[Hotline:Desktop]$ [[ -z `strings TwoKey.DMP  | grep "test1234"` ]] && echo "No credentials found"
No credentials found
[Hotline:Desktop]$ strings TwoKey.DMP  | grep "iLCJ@hbGciOiJIUzI1NiJ9"
[Hotline:Desktop]$ strings TwoKey.DMP  | grep "test1234"
[Hotline:Desktop]$
```

Figure 7.9: Analysing the memory dump

Another way of analysing is also looking with a hex editor and looking for the needed bytes. Which, too, yielded no results.

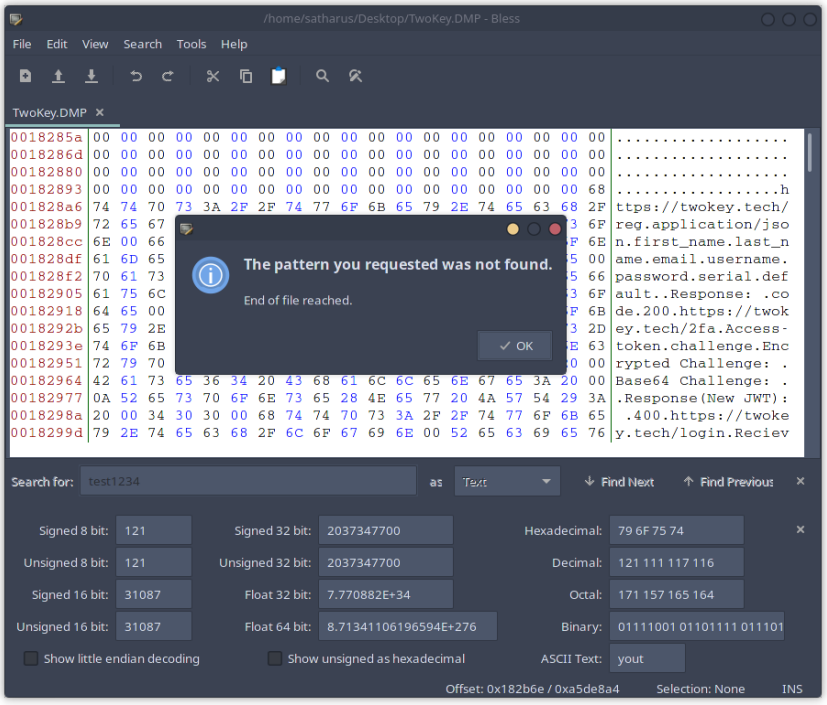


Figure 7.10: Looking through the dump with a hex editor

7.5.2 Secure Communication

7.5.2.1 SSL Certificate Validation

The SSL certificate for twokey.tech is valid. Figure 7.11 demonstrates so.

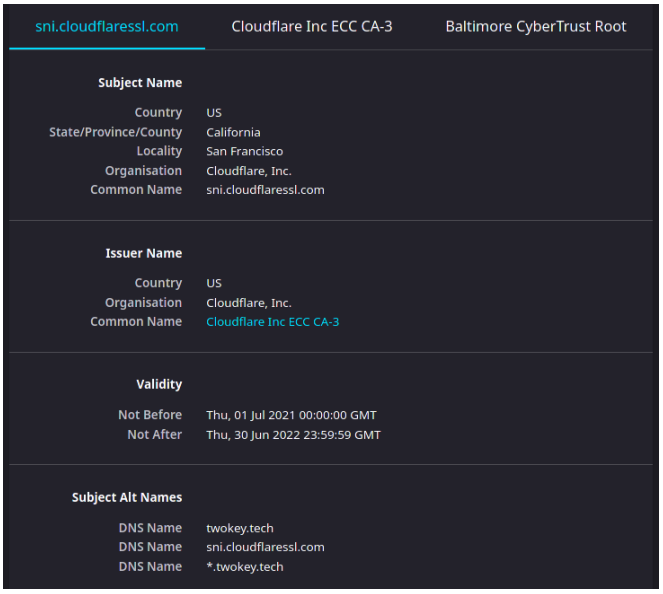
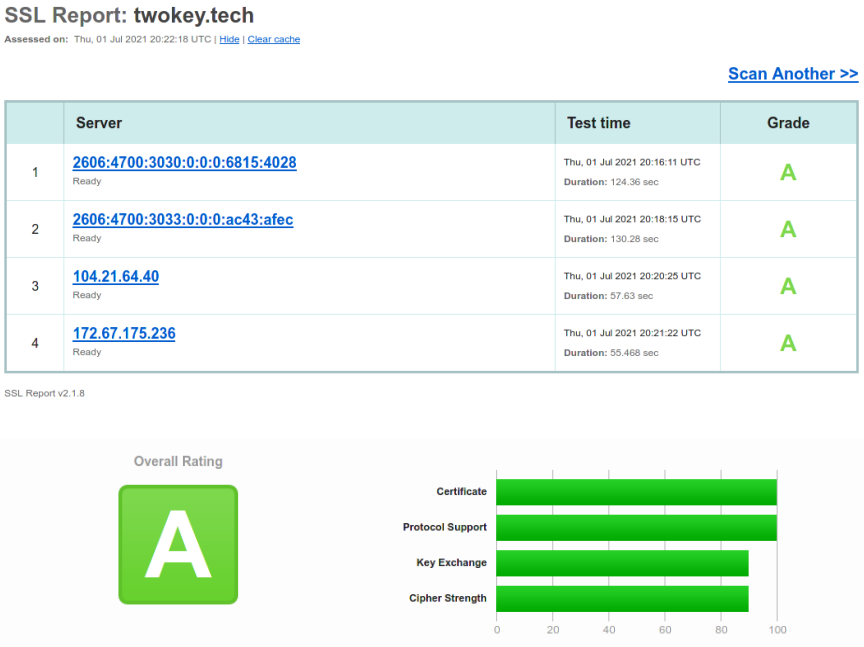


Figure 7.11

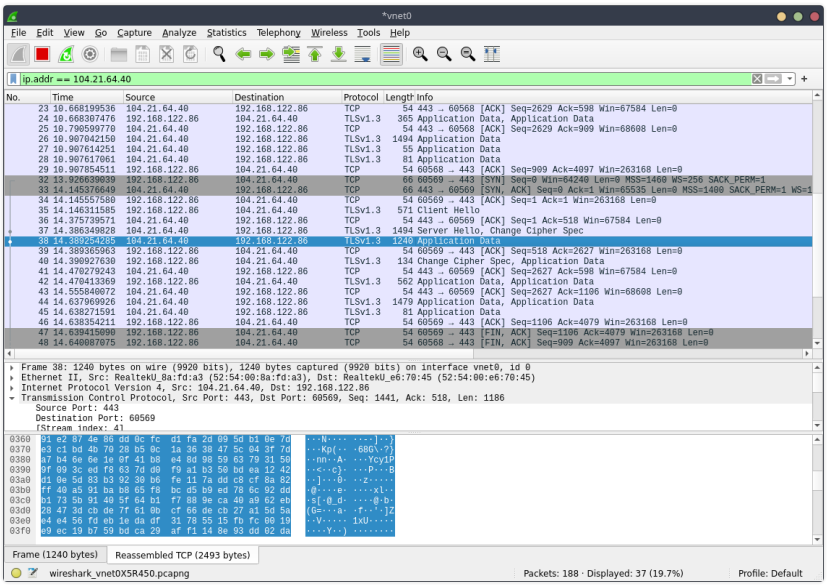
According to a test performed by Qualys SSL Labs[22], twokey.tech has level A SSL based on its certificate and SSL/TLS configuration as shown in Figures 7.12&7.13.



Figures 7.12 and 7.13

7.5.2.2 Desktop App and Backend Server COmmunication

To verify that communication is secure, a packet analyser such as Wireshark[23] can be used. Open inspecting the packets, it is shown that not only is the data encrypted (Figure 7.14), but the desktop app is also not communicating directly with the TwoKey server, but instead is being proxied to the public IP 104.21.64.40, which belongs to CloudFlare as shown in Figure 7.15.



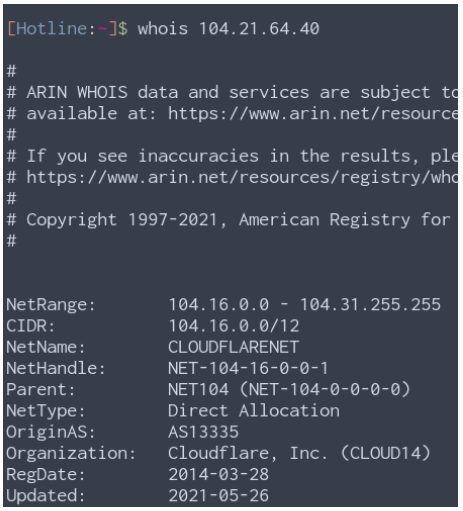


Figure 7.14

Figure 7.15

7.5.3 Strong Passwords

Passwords are generated with a strong criteria as stated before, as verification generated passwords were tested on security.org/how-secure-is-my-password[24] and all of the passwords which were deemed strong by TwoKey were also verified as strong by the website. Figures 7.16 and 7.17 demonstrate the tests.

Figure 7.16: A password of length 11, the minimum that is considered secure by TwoKey

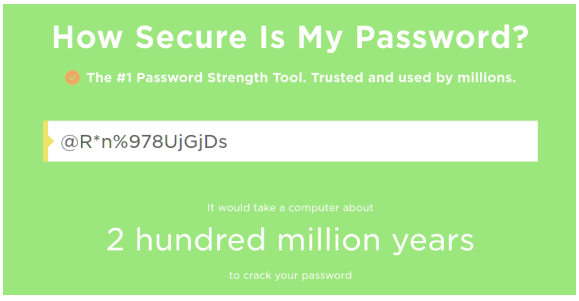


Figure 7.17: A password of length 14, a fair average that a user may use

7.5.4 Strong Second Factor of Authentication

Using traditional methods like security questions or phone call/text message verification have been criticised, as they are not secure options for users.

As discussed before, security questions can be guessed and are heavily vulnerable to social engineering, e.g. someone who knows the victim may guess the security questions very easily. Phone calls and SMS verification are vulnerable to GSM sniffing and a few services have started to remove them as 2FA options. They're also heavily vulnerable to phishing attacks.

A few alternatives started to surface but they heavily affected the balance between the security and convenience for the users like using an alternative email or phone number specifically for 2FA. Another alternative is using a 2FA app and keeping backup security codes on the side. The alternative email option requires logging in onto the email service and checking for the verification mail, and then using the sent code or link to authenticate. The user may not be logged onto the mailing service on the same device, which may result in them resorting to logging in from another device which makes copying the code/link between devices grueling.

Using a backup security code may be difficult for some users as they are required to save the code or print it to keep it safe. They also require to have constant access to it as it will be the only code to login with in the case of not being able to access the 2FA app. The backup codes differ between all of the different services too.

With TwoKey's hardware methods the following points apply:

- The 2FA method is not vulnerable to sniffing or phishing attacks as it doesn't transmit any keys. It also only communicates through USB with the desktop app. Unless the system has malware on it, it isn't possible for a remote system to sniff the USB packets.
- The challenge-response based 2FA can't be replicated by another device as the key is embedded on the device itself. It also uses 128bit AES which makes it impossible to brute force it due to a large number of possibilities.
- It is one single token that allows the user to access all of their passwords, if and only if they have the correct username and master password.

7.5.5 Trustworthy

TwoKey is fully open source with a GPLv3.0 license, any claims made by the TwoKey team can be verified at the official GitHub page at:

- github.com/Satharus/TwoKey.

Any other party that decides to fork TwoKey and improve it is required to share the changes made to their code, according to the GPLv3.0 license.

This makes TwoKey 100% trustworthy as all of its modules are open source and completely available for everyone to run, share, study, and improve.

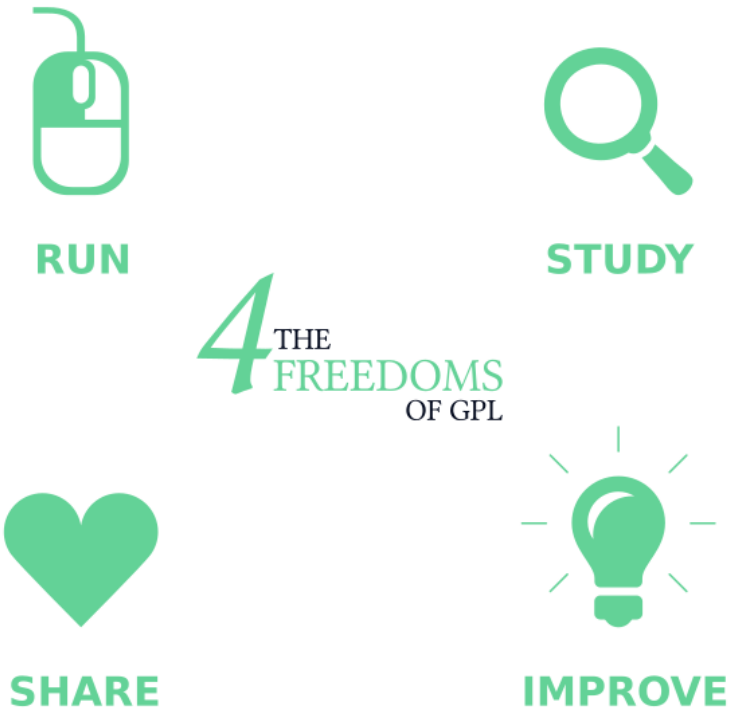


Figure 7.18: The Four Freedoms of the GPL License

Chapter 8: Conclusion and Future Work

8.1 Conclusion

This document discusses TwoKey, the secure password manager with a physical access token. TwoKey consists of four modules, the desktop application which is the middle-man of the whole system and connects all of the other modules together, the hardware token which is used as a second factor of authentication, the backend server which manages user authentication and data, and finally the browser extension which is responsible for auto filling the user credentials instead of the user having to copy/paste them everytime.

TwoKey utilises secure programming, secure communication, strong randomly generated passwords, encryption, hashing, and a solid second factor of authentication in order to maintain a high level of user security.

While TwoKey may not be convenient, it is well known that convenience and security don’t go hand in hand together. The more secure something is, the more likely that it will be less convenient to use.

TwoKey balances between security and convenience with very low compromises regarding security, performance, and privacy.

8.2 Future Work

Like any project, there are many improvements which could be implemented in TwoKey. These points are discussed in the following subsections, on a per-module basis.

8.1 Desktop Application

- Auto Filling Credentials in Native apps
    - TwoKey's desktop applications can be updated so that it can autofill all apps, not only web services through the browser extension.
    - This will provide a better UX for the user as they wouldn't have to go back to TwoKey's desktop app and copy/paste passwords every time.
- A feature could be added to remind the user to change their passwords every while, such as every month or so.
- There could be a possible integration with a service like havibeenpwned.com which notifies the user when one of their accounts is found in a breach. This could be a good prompt for the user to change their password.
- Employ the use of hardware-based features such as SGX[25] which help in securing application data and binaries.

8.2 Hardware Token

8.2.1 Physical Design of The Hardware Token

- Adding an elegant cover to the token itself to make it look more modern.

- Adding a suitable casing with special design to support physical tamper protection, such as a light sensor which prompts a wipe of the chips on the token itself, whenever it senses direct light or removal of the cover.

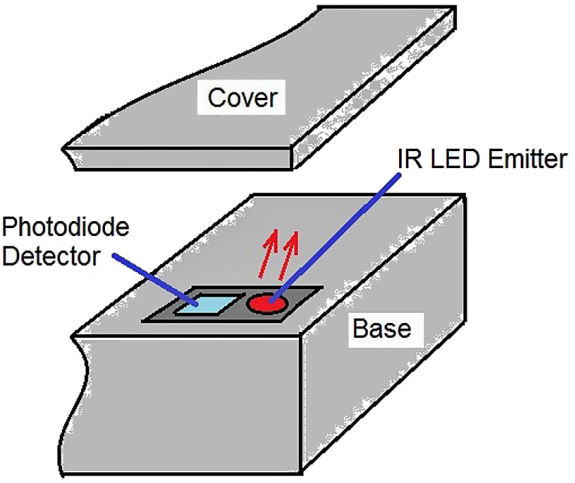


Figure 8.1: Example of Tamper Protection

8.2.2 Hardware Token Security

- Increasing the security of the challenge-response authentication by using 256bit AES instead of 128bit AES.

8.2.3 Hardware Security Module

- Integrating a physical HSM to increase the resistance to more complex attacks such as firmware dumping and reverse engineering.

8.2.4 Mobile Devices Support

- NFC could be added to the token in order to support mobile phones and potentially use TwoKey as a mobile app.
- This needs careful studying before it is implemented as it may increase the attack surface, where the token may be vulnerable to NFC sniffing/skimming attacks.

8.3 Browser extension

- UI Enhancement
    - The browser extension could have a management UI which allows the user to manage their accounts and credentials without having to use the desktop app directly. (Like in Figure 8.2)
- Website Certificate Check
    - The browser extension could check the legitimacy of the website using its certificate before filling in the credentials, not just based on the domain name. This is to prevent phishing.
- Unlock with Biometrics
    - Biometrics could be used as a 3rd factor of authentication, or could replace the password and be the first factor of authentication.
- Launch a website from history
    - TwoKey's browser extension will remember a history of websites the user logged into, from which they can load a site they had open.

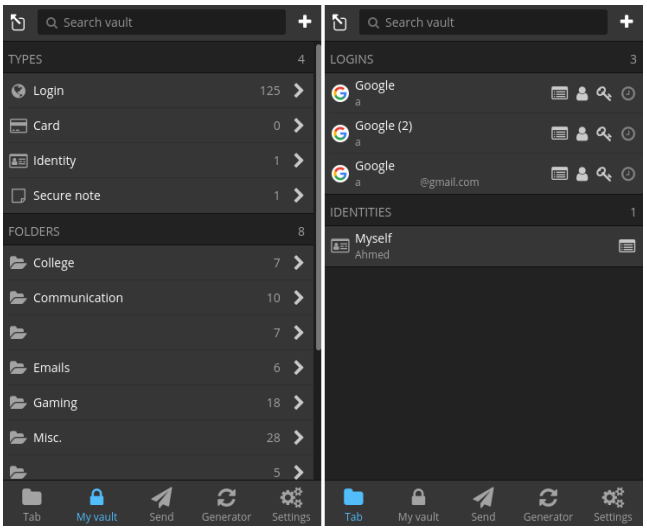


Figure 8.2: Browser Extension UI from Bitwarden, an open source password manager.

8.4 Server Side and Infrastructure

- Adding multiple servers for load balancing and backup in case any server fails.
- Applying system administration to the project such as regular backups incase of a data failure.

8.5 Additional Modules

- Creating an API for service vendors to implement into their codebase in order to enable TwoKey's hardware token as an option of 2FA without having to use TwoKey's software itself.

Appendix 1: User Manual

1.1 Registration

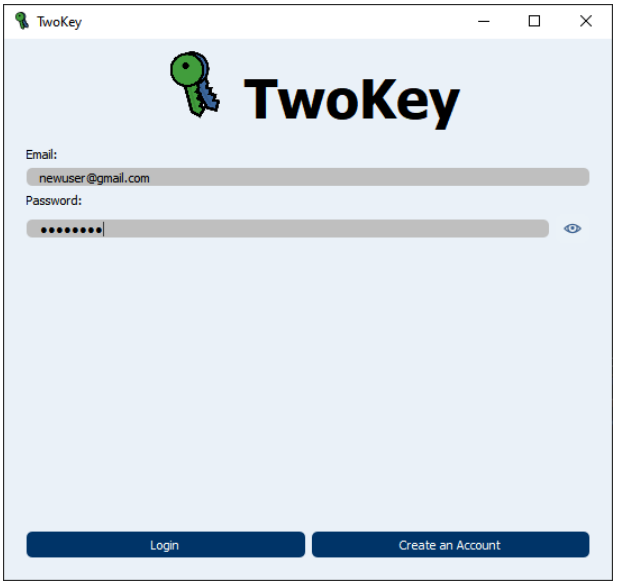
The user must enter:

1. First name
2. Last name
3. Email
4. Strong Master Password
5. Serial number which came with the device to link their account with it.

1.2 Login

1.2.1 Login Screen

First, the user enters their email and password to login.



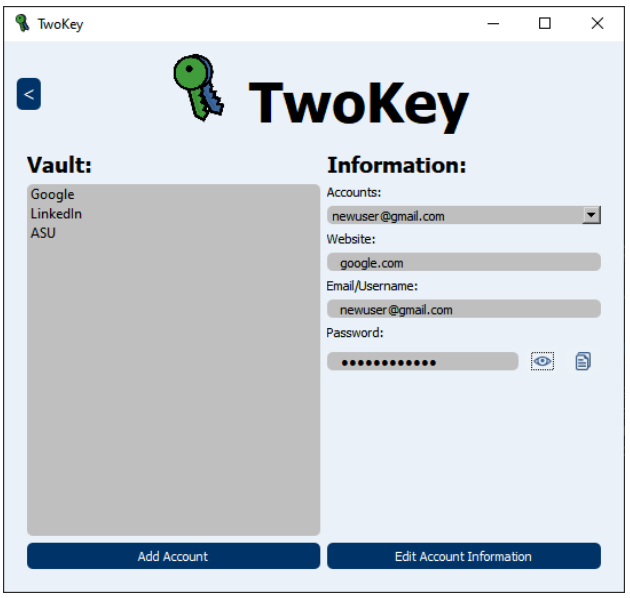
1.2.2 Check Token Validation

Second, if the device is not connected or the physical ID on it is not correct. Then, the system pops up with an error message.

And after the authentication, it goes to the manager's main page.

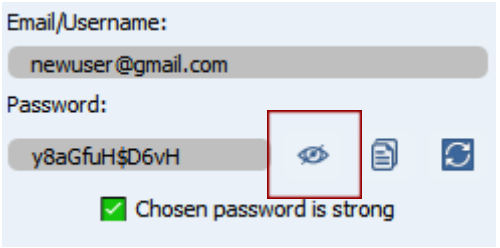
1.3 Manager Page

Where the user can view his vault with all the accounts.



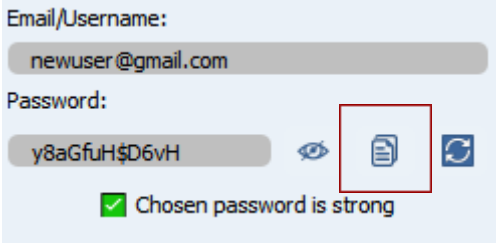
1.3.1 Show Password

Clicking on the eye icon, the user is able to see the password. Clicking on it again hides the password like it was previously.



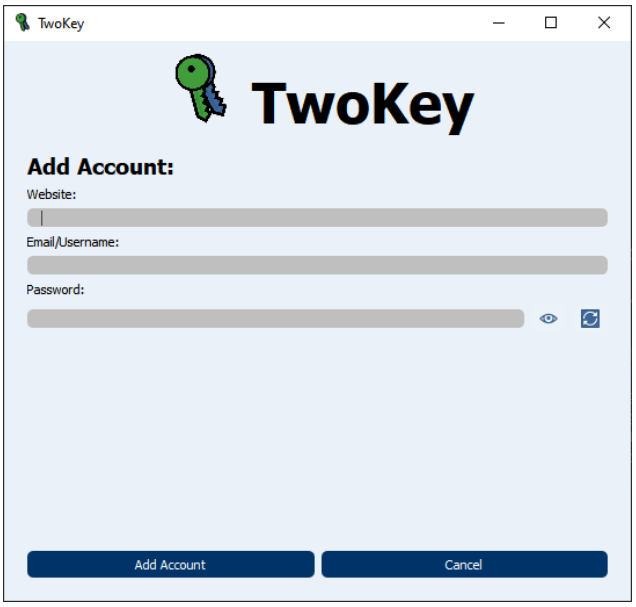
1.3.1 Copy to Clipboard

Clicking on the clipboard icon to copy the password to the system's clipboard.



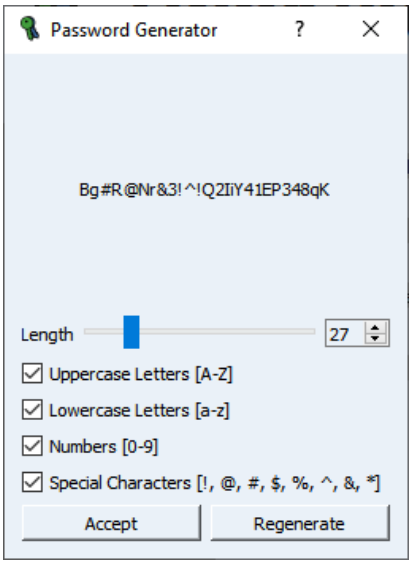
1.4 Add New Account

1.4.1 Add credentials



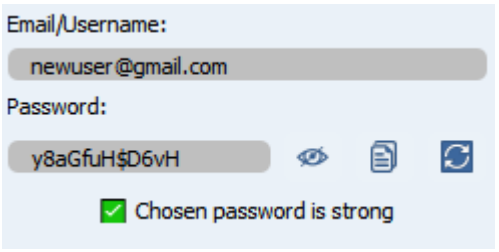
1.4.2 Generate Random Strong Password

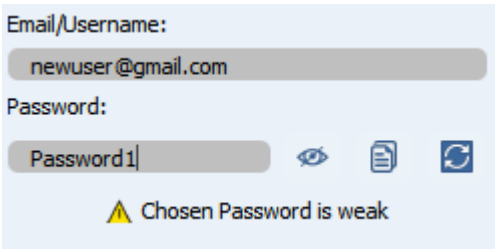
The user can choose the length of the password, whether the password contains uppercase and/or lowercase letters, numbers, and special characters.



1.4.3 Password Strength Test

TwoKey checks if the password is strong enough or not.



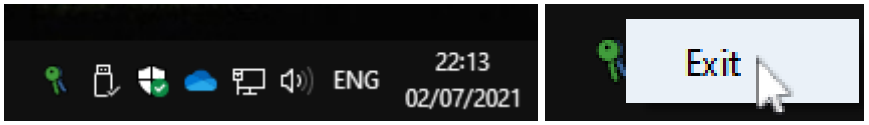


1.5 Edit an Existing Account

On the manager's main page, a user can choose an account from the vault and edit the data and by clicking on the “Edit Account Information” button.

1.6 Exiting TwoKey

When closing TwoKey’s main window, it is minimised to a tray icon. The user must right click that icon and click on “Exit” to close TwoKey.



Appendix 2: Code Snippets

Code Snippet 3.1

```
#include <AESLib.h>

void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(3000);
}

uint8_t message[17];
uint8_t key[] = "rskDSkocuB6YuLxO";
uint8_t iv[]  = "su4djDtAWNuvyjVg";

void loop()
{
    if (Serial.available() > 0)
    {
      Serial.readBytes(message, 16);
      aes128_cbc_enc(key, iv, message, 16);
      for (int i = 0; i < 16; i++)
      {
        if (message[i] < 16) Serial.print("0");
        Serial.print(message[i], HEX);
      }
      Serial.flush();
    }
}
```

Code Snippet 4.1

```
QSerialPort *token = new QSerialPort();
bool tokenIsAvailable = false;
QString tokenPortName;
static void checkForToken(int *running)
{
    while (*running)
    {
        foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts())
        {
            if (serialPortInfo.hasVendorIdentifier() &&
                    serialPortInfo.hasProductIdentifier() &&
                    serialPortInfo.vendorIdentifier() == TOKEN_VID &&
                    serialPortInfo.productIdentifier() == TOKEN_PID)
            {
                tokenPortName = serialPortInfo.portName();
                tokenIsAvailable = true;
            }
        }

        if (tokenIsAvailable)
        {
            if (!token->isOpen())
            {
                token->setPortName(nanoPortName);
                token->setBaudRate(QSerialPort::Baud9600);
                token->setDataBits(QSerialPort::Data8);
                token->setParity(QSerialPort::NoParity);
                token->setStopBits(QSerialPort::OneStop);
                if (!token->open(QIODevice::ReadWrite))
                {
                  //Log error, device is likely being used
                }
            }
        }
        else //Log failure
    }
}
```

Code Snippet 4.2

```
class USBEventHandler : public QObject, public QAbstractNativeEventFilter
{
    Q_OBJECT
public:
    //Constructor and Desctructor
    explicit USBEventHandler(QObject *parent = nullptr);
    //Implementing the original function from QAbstractNativeEventFilter
    virtual bool nativeEventFilter(const QByteArray &eventType, void *message, long *result);

signals:
    //Signals for the states of USB devices
    void SerialDeviceInserted();
    void SerialDeviceRemoved();
};

USBEventHandler::USBEventHandler(QObject *parent) : QObject(parent)
{
    //Check for the token every time a device is inserted or removed
    connect(this, SIGNAL(SerialDeviceInserted()), SLOT(checkDeviceID()));
    connect(this, SIGNAL(SerialDeviceRemoved()), SLOT(checkDeviceID()));
}

bool USBEventHandler::nativeEventFilter(const QByteArray &eventType, void *_message, long *result)
{
#ifdef Q_OS_WIN
    //Get the message and interpret it
    MSG *message = static_cast<MSG*>(_message);

    //Get the device structure
    PDEV_BROADCAST_HDR device = reinterpret_cast<PDEV_BROADCAST_HDR>(message->lParam);

    //If device is inserted
    if (message->message == WM_DEVICECHANGE && message->wParam == DBT_DEVICEARRIVAL)
    {
        if (device->dbch_devicetype == 0x3) emit SerialDeviceInserted();

    }
    else if (message->message == WM_DEVICECHANGE && message->wParam == DBT_DEVICEREMOVECOMPLETE)
    {
        if (device->dbch_devicetype == 0x3) emit SerialDeviceRemoved();
    }
#endif
    return false;
}
```

Code Snippet 4.3

```
USBEventHandler::USBEventHandler(QObject *parent) : QObject(parent)
{
#ifdef Q_OS_LINUX
    timer = new QTimer(this);

    udev = udev_new();
    if (!udev)
        qDebug() << "Failed to allocate new udev";

    //Initialise the timeout for Udevs pooling
    udevTimeout.tv_sec = 1;
    udevTimeout.tv_usec = 0;
    //Initialise the udev monitor in order to be able to pool the devices
    mon = udev_monitor_new_from_netlink(udev, "udev");
    //Add a filter to get only USB devices
    udev_monitor_filter_add_match_subsystem_devtype(mon, "usb", NULL);
    //Enable receiving for the monitor
    udev_monitor_enable_receiving(mon);

    //Connect the timer's timeout with the tick function, to check every X ms
    connect(timer, SIGNAL(timeout()), this, SLOT(tick()));
    timer->start(250/*X*/);
#endif
    //Check for the token every time a device is inserted or removed
    connect(this, SIGNAL(SerialDeviceInserted()), SLOT(checkDeviceID()));
    connect(this, SIGNAL(SerialDeviceRemoved()), SLOT(checkDeviceID()));
}

#ifdef Q_OS_LINUX
void USBEventHandler::tick()
{
    //Get file descriptor for events from udev
    deviceFD = udev_monitor_get_fd(mon);
    //Create a file descriptor set
    fd_set fds;
    //Initialise to zero
    FD_ZERO(&fds);
    //Populate the set
    FD_SET(deviceFD, &fds);

    //Retrieve the events
    int ret = select(deviceFD+1, &fds, NULL, NULL, &udevTimeout);
    if (ret <= 0)
        return;

    //Check if the descriptor is in the set
    if (FD_ISSET(deviceFD, &fds))
    {
        //Get the device from the monitor
        struct udev_device* dev = udev_monitor_receive_device(mon);
        if (dev)
        {
            //Get the event from the device
            if (udev_device_get_devnode(dev))
            {
                const char* action = udev_device_get_action(dev);
                //Emit the signals accordingly
                if (strcmp(action, "add") == 0) emit SerialDeviceInserted();
                else if (strcmp(action, "remove") == 0) emit SerialDeviceRemoved();
            }
        }
    }
}
#endif
```

Code Snippet 4.4

```
MainWindow::MainWindow(QWidget *parent, USB_communicator *usb_comm) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ...
    this->stdinNotifier = new QSocketNotifier(fileno(stdin), QSocketNotifier::Type::Read, this);
    ...
    connect(this->stdinNotifier, SIGNAL(activated(int)), this, SLOT(readFromStdin()));
    ...
}

QString MainWindow::readFromStdin()
{
    std::string stdinMessage;
    std::cin >> stdinMessage;
    QString retMessage = QString::fromStdString(stdinMessage);


    this->ui->browserExtensionLabel->setText("Extension Message: " + retMessage);

    return retMessage;
}
```

Code Snippet 4.5

```
int compute(int a, int m, int n)
{
    int r;
    int y = 1;

    while (m > 0)
    {
        r = m % 2;
        if (r == 1)
            y = (y*a) % n;

        a = a*a % n;
        m = m / 2;
    }

    return y;
}

int generate_key(int public_B)
{
    srand(time(0));
    int p = 997;
    int g = 5;
    int secret_alice = rand();

    int public_A = compute(g, secret_alice, p);

    printf("Desktop App's Public Key: %d", public_A);

    int key_A = compute(public_B, secret_alice, p);
    printf("Secret Key: %d", key_A);
}
```

Code Snippet 4.6

```
void BrowserExtensionCommunicator::startServer()
{
    running = true;
    qDebug() << "Server running at http://localhost:" + QString(opt.port);

    while(true)
    {
      sb_poll_server(server, 1000);
    }
}

int BrowserExtensionCommunicator::event_handler(sb_Event *e)
{
    BrowserExtensionCommunicatorSignalWrapper* signalWrapper = (BrowserExtensionCommunicatorSignalWrapper*)e->udata;
    if (e->type == SB_EV_REQUEST)
    {
        char *masterpasswd = reinterpret_cast<char*>(malloc(1000));
        sb_get_var(e->stream, "masterpasswd", masterpasswd, 1000);

        char *email = reinterpret_cast<char*>(malloc(1000));
        sb_get_var(e->stream, "email", email, 1000);

        char *dec_masterpasswd = decrypt(masterpasswd);
        char *dec_email = decrypt(email);
        ...
        if (strlen(dec_email) && strlen(dec_masterpasswd) &&
            signalWrapper->getBackendClient()->login(QString(email), QString(masterpasswd)) ==
            BackendClient::loginStatus::SUCCESS)
        {
            sb_send_status(e->stream, 200, "OK");
            sb_send_header(e->stream, "Content-Type", "text/plain");
            signalWrapper->emitSuccessfulLogin();
            sb_writef(e->stream, signalWrapper->getBackendClient()->getJwt().toStdString().c_str());
        }
        else if (!strlen(url))
        {
            sb_send_status(e->stream, 401, "Unauthorised");
            sb_send_header(e->stream, "Content-Type", "text/plain");
            sb_writef(e->stream, "Unauthorised login, wrong password.");
        }

    }
    ...
}
```

Code Snippet 4.7

```
int BackendClient::login(QString email, QString password)
{
    QNetworkRequest request(QUrl("https://twokey.tech/login"));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
    QJsonObject json;

    QCryptographicHash *sha256sum =  new QCryptographicHash(QCryptographicHash::Algorithm::Sha256);
    QByteArray password_hash = sha256sum->hash(password.toUtf8(), QCryptographicHash::Algorithm::Sha256);

    json.insert("email", email);
    json.insert("password_hash", QString(password_hash.toHex()));

    QNetworkAccessManager nam;
    QNetworkReply *reply = nam.post(request, QJsonDocument(json).toJson());

    while (!reply->isFinished())
    {
        qApp->processEvents();
    }

    QByteArray response_data = reply->readAll();
    QJsonDocument jsonResponse = QJsonDocument::fromJson(response_data);

    QString challenge = jsonResponse.object()["challenge"].toString();
    this->jwt = jsonResponse.object()["Access-token"].toString();

    //Send the challenge to the usb token and receive it
    usbComm->writeToToken(challenge.toStdString().c_str());
    tokenChallengeResponse = usbComm->readFromToken();
    //Begin 2FA process
    int loginStatus = _2fa();
    delete reply;
    delete sha256sum;
    return  loginStatus;
}
```

Code Snippet 5.1

```
chrome.tabs.onUpdated.addListener(getTab)
function getTab()
{
  chrome.tabs.query({active: true, currentWindow: true}, tabs => {
          let link = tabs[0].url;

           var http=new XMLHttpRequest();
           const url="http://localhost:8000/?url="+link;
           http.open("GET",url,true);
           http.send("");
           http.onreadystatechange = (e) => {
           retrievedCredentials = http.responseText;
           if (retrievedCredentials.length != 0)
              console.log(retrievedCredentials);
           }
           chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
      });
}
```

Code Snippet 5.2

```
for (int i = 0; i < 3; i++)
{
  chrome.contextMenus.create({id:i.toString()}, type: 'radio', title:x[i]});
  chrome.contextMenus.onClicked.addListener((info, tab) => {
    alert("Item " + info.menuItemId + " clicked " + "in tab + tab.id");
    //Send account information to the desktop app
  })
}
```

Code Snippet 5.3

```
chrome.tabs.onUpdated.addListener(getTab)
function getTab()
{
        chrome.tabs.query({active: true, currentWindow: true}, tabs => {
                let link = tabs[0].url;
                console.log(link);
                 var http=new XMLHttpRequest();
                 const url="http://localhost:8000/?url="+link;
                 http.open("GET",url,true);
                 http.send("");
                 http.onreadystatechange = (e) => {
                   retrievedCredentials = http.responseText;
                   if (retrievedCredentials.length != 0)
                      console.log(retrievedCredentials);
                 }
                 chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
            });
}
```

Code Snippet 5.4

```
chrome.tabs.onActivated.addListener(getTab)
function getTab()
{
        chrome.tabs.query({active: true, currentWindow: true}, tabs => {
                let link = tabs[0].url;
                console.log(link);
                 var http=new XMLHttpRequest();
                 const url="http://localhost:8000/?url="+link;
                 http.open("GET",url,true);
                 http.send("");
                 http.onreadystatechange = (e) => {
                   retrievedCredentials = http.responseText;
                   if (retrievedCredentials.length != 0)
                      console.log(retrievedCredentials);
                 }
                 chrome.tabs.sendMessage(tabs[0].id, retrievedCredentials);
            });
}

var sending = chrome.runtime.sendMessage(retrievedCredentials);
```

Code Snippet 7.1

```
for (int i = 0; i < password->length(); i++)
{
    (*password)[i] = static_cast<char>(0);
}
delete password;
ui->manager_password->clear();
```

Appendix 3: References

[1] William Stallings, Lawrie Brown. Computer Security: Principles and Practice.

[2] ise.io/casestudies/password-manager-hacking

[3] fsfe.org/freesoftware

[4] github.com/rxi/sandbird

[5] docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-registerdevicenotificationw

[6] doc.qt.io/qt-5/qabstractnativeeventfilter.html

[7] freedesktop.org/software/systemd/man/libudev.html

[8] libusb.info

[9] developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging

[10] doc.qt.io/qt-5/qabstractnativeeventfilter.html

[11] doc.qt.io/qt-5/qthread.html

[12] doc.qt.io/qt-5/qnetworkrequest.html

[13] doc.qt.io/qt-5/qnetworkaccessmanager.html

[14] cryptopp.com

[15] developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs

[16] developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/runtime

[17] developer.chrome.com/docs/extensions/reference/contextMenus

[18] danielmiessler.com/blog/casmm-consumer-authentication-security-maturity-model

[19] jwt.io/introduction

[20] sourceware.org/binutils/docs/binutils/strings.html

[21] gnu.org/software/grep

[22] ssllabs.com/ssltest/analyze.html?d=twokey.tech&latest

[23] wireshark.org

[24] security.org/how-secure-is-my-password

[25] software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html