TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
IT System Development

Aleksandr Zahharjan 171913IADB

Developing an Open-Source Environment for Two-Factor Authentication

Bachelor's thesis

Juhendaja: Kaido Kikkas
PhD.

Tallinn 2022

Author’s declaration of originality

I hereby declare that I have compiled the thesis independently, and all works, important standpoints, and data by other authors have been properly referenced, and the same paper has not been previously presented for grading.

Aleksandr Zahharjan

……

(signature, date)

Student code: 171913IADB

Student e-mail address: zahharjan@gmail.com

Supervisor: Kaido Kikkas, Ph.D.:

The paper conforms to requirements in force

……

(signature, date)

Chairman of the Defence Committee:

Permitted to the defense

……..

(name, signature, date)

Abstract

In this thesis at hand, the author develops a virtual environment for automation of the Two-Factor Authentication process. The goal of the thesis is to provide an Open-Source Plug-and-Play solution that will decrease time spent on manual data input and keep users from unneeded distractions during authentication checks.

The scope of the practical part is limited to devices running the Android operating system and any desktop machines that could have Python installed. The main goal of the project is to build a core framework, which could be used as a stepping stone for future open-source development.

The final result consists of three parts: the Android app, the Python server component, and browser extension. Those services are supposed to communicate with each other and share data to provide a secure channel for Time-Based One-Time-Passwords sharing and import it to the user's clipboard.

Annotatsioon

Dünaamiline avatud lähtekoodiga rakendus kaheastmeliseks autentimiseks

Käesolevas lõputöös arendab autor virtuaalset keskkonda kahefaktorilise autentimise protsessi automatiseerimiseks. Lõputöö eesmärk on pakkuda avatud lähtekoodiga lahendust, mis vähendab käsitsi andmete sisestamisele kuluvat aega ja hoiab kasutajaid autentimise kontrollimise ajal tarbetute segajate eest.
Praktilise osa ulatus on piiratud Androidi operatsioonisüsteemi töötavate seadmetega ja mis tahes lauaarvutitega, millele võib olla installitud Python. Projekti põhieesmärk on ehitada tuumikraamistik, mida saaks kasutada tulevase avatud lähtekoodiga arenduse hüppelauana. Lõpptulemus koosneb kolmest osast: Androidi rakendusest, Pythoni serverikomponendist ja brauserilaiendist. Need teenused peaksid omavahel suhtlema ja andmeid jagama, et pakkuda turvalise kanali ajapõhiste ühekordsete paroolide jagamiseks ja importida need kasutaja lõikelauale.

List of abbreviations and terms

2FA	Two-Factor Authentication
ADB	Android Debug Bridge
API	Application Programming Interface
DAO	Data Access Object
DB	Database
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
MVP	Minimal Viable Product
NFC	Near Field Communication
OTP	One-Time-Password
QR-code	Quick Response code
SDK	Software Development Kit
SHA	Secure Hash Algorithm
TOTP	Time-Based One-Time-Password
U2F	Universal Two-Factor authentication
USB	Universal Serial Bus
UX	User experience

Table of Contents

List of Figures

List of Tables

1. Introduction

"It has been my observation that most people get ahead during the time that others waste." - Henry Ford [1]

In the twenty-first century, data about us has become our property and a product in need of privacy. And if in real life this principle can be more or less observed, then in the virtual space, quite often, there is an attempt to steal information about us. Unfortunately, gone are the days when people could only protect themselves with a strong password. With passwords algorithm progress goes the progress of the methods intruders and cybercriminals are using. The modern approach of stealing identity is picking up login and password from databases leaked to the network. Several big IT companies, such as GitHub, AdGuard, and Apple iCloud have already become victims[2]. And this is why providing a way to check the integrity of information transmitted over or stored in an unreliable medium is a prime necessity in the world of open computing and communications [3].

One of the most known methods of prooving and protecting user identity is a Two-Factor authentication check. The prime idea behind the second factor is to validate not only the password-login pair but also confirm if the user physically owns something, that was provided to him as an additional factor. It may be temporary generated codes, specific flash devices or even user’s biometry.

1.1 Problem and aim

The modern IT industry has generated several protocols of 2FA with different factors, security levels, and requirements for users. Unfortunately, they usually have some problems either from the user experience side of view, security, or the fact that users need to purchase some additional hardware.

This thesis aims to create a prototype of a 2FA application which would have a decent user experience as well as the complete functionality of existing analogues. The keystone in this project will be keeping the same level of security as 2FA APPs are providing. The author's role in this project is to perform an analysis of current methods and solutions, finding approaches with the biggest potential using their experience for creating an application with modifications which will fix the weakest sides of realization.

1.2 Relevance

According to secplicity.org , over 1 million passwords are stolen each week [4]. Combining it with information from webtribunal.net [5] which claims, that 67% of people are using the same password for all their accounts leads to the understanding of the huge threat of identity being stolen.

Nowadays cybercriminals are using multiple different ways of getting access to personal data. The most common of them are:

Keyloggers
Keyloggers are software programs that give cybercriminals access to personal data by recording all the keyboard keystrokes. The passwords and credit card numbers user types, the web pages user visits – all by logging keystrokes.

Social Engineering
This approach comes in a number of styles, all of which are rooted in the idea of deceiving or manipulating people into divulging their information or taking a certain action. Common social engineering methods used to steal passwords include phishing and using a trojan horse attack. A less common approach is shoulder surfing, in which the attacker simply watches a user type in his or her password.

Dictionary Attack
Cybercriminals try to guess a password by typing in a common list of words from a password "dictionary." More advanced password dictionaries include lists of the most commonly used words in passwords. This is a relatively simple method, but one that is effective in guessing less-complex passwords. In the other words, if the user used real words in any of his passwords, his credentials are at risk.

Brute Force Attack
While not as efficient as a dictionary attack, a brute force attack is more effective in eventually guessing a password. With this method, attackers use tools to repeatedly try every possible password combination of letters, numbers, and symbols until the password is cracked. A similar approach is a reverse brute force attack, in which an intruder tries one password against many usernames.

Credential Stuffing Attack
Since so many people use the same passwords or variations of passwords across accounts, cybercriminals found a way to automatically run database lists of breached username/password combinations against a target website login. According to Shape Security [6], 90% of login attempts at online retailers are from this type of attack and this method is effective for cybercriminals about 3% of the time.[4]

Concluding cyber attacks methods, the most common of them could be prevented if the user would have a 2FA set. As it is usually a physical factor, it cannot be leaked to the network or stolen via fishing as passwords are temporary and updated periodically. However, existing 2FA solutions usually have drawbacks in user experience and may be not-wanted because of the inconveniences they affect. Creating a solution that will cause less discomfort, maybe a crucial factor to convince more people to use 2FA.

1.3 Methodology

In the theoretical part of the thesis author is defining the scope and compares existing 2FA methods and their realisation. Doing the analysis of current implementation and highlighting their weak sides are to be resolved in the practical part.
The author is doing an analysis of suitable technologies that could be used during the practical part and comparing platforms and available software to be used.

The thesis provides information about common cyber threats and gives an overview of how 2FA is able to save users from them. Algorithms used for 2FA checks and OTP generation are provided and explained.

Finally, in the practical part author implements an environment involving 3 parts - Android APP, Python local server and browser extension. A description of each part is provided and the source code is attached in the appendix.

After the practical part author is providing an analysis of possible improvements and thesis results.

2. Two-Factor Authentication

Two-factor authentication or 2FA is a method of identifying a user in a service (usually on the Internet) by requesting two different types of authentication data, which provides a two-layer, and, therefore, more effective account protection against unauthorized entry. In practice, it usually looks like this: the first frontier is a login and password, and the second is a unique code that comes from a specific application or SMS. Less commonly, the second "layer" of protection requests a unique USB key or user's biometric data. In general, the essence of the approach is very simple: to get somewhere, the user needs to double confirm the fact that he is he and with the help of two different keys stored in different places. One of which is the static password known by the user and the second is generated temporal “second factor”.

2.1 Comparison scope

The comparison of 2FA methods is going in 2 rounds. First - description of each approach, highlighting its weak and strong points. And the second step includes a rating from 1 to 10 different 2FA kinds in the following criteria:

- Security - how safe is the method and how hard to hack it
- Accessibility - how complicated to start using the method
- UX - user experience

2.2 Variants of 2FA

2.2.1 USB key

Physically, a USB security key (also called a U2F key) is a type of hardware security that resembles a USB drive and plugs into one of the computer’s USB ports. In practice, a security key is a physical security device with a totally unique identity. It houses a small chip with all of the security protocols and code that allows it to connect with servers and verify the user's identity. It’s used to ensure that owner of the U2F is the person actually accessing a site or service. It can be considered as a key for the house - keep it safe and never share it with anyone.

Some security keys even have NFC and/or Bluetooth built-in, making them perfect for use with newer Android and iOS smartphones. The keys work with browsers like Google

Chrome, along with web services like Gmail, Facebook, Dropbox, 1Password, Twitter, GitHub, Microsoft, and many others.[7]

Examples:

- YubiKey
- Fido

Strong sides:

- No manual data input.
- High level of security.

Drawbacks:

- Price - Hardware costs in the range of 25-90 EUR.
- If the key or user's phone does not have an NFC module, 2FA on mobile devices becomes impossible.
- Hard to recover in case of key lost.

2.2.2 SMS

One of the variants of 2FA verification is the code that the web service is sending to the phone via the SMS system. For doing so, the server generates a unique combination of symbols and applies it to the internet-telecom service, which is supposed to convert it into SMS and send it to the user's phone.

Strong sides:

- No additional hardware or software is needed
- Free to use

Drawbacks:

- Depending on the service used to send an SMS, it might be slow
- Manual data input
- SIM swap scam method can get access to users messages
- Expiring time is quite long

2.2.3 2FA Application

Authenticator mobile apps are generating a one-time code that can be used to confirm that it's legit person is logging in to a website or service. When a user set up an authenticator application with a website, that site generates a secret key - a random collection of numbers and symbols - which the user then save to the app. The site usually shows that key in the form of a QR code. When a user scans that with the app, the key is then saved to his phone. [8] After this step, the application starts to generate a unique OTP every 30 seconds. Then the user needs to login into the 2FA-secured page he takes the valid code from the application and put it in the required field.

Strong sides:

- No additional hardware needed
- High level of security.
- Free to use

Drawbacks:

- Manual data input

2.2.4 Biometric 2FA

Biometric authentication is a method of verifying a user's identity using personal user's data such as their fingerprint, facial features, hand shape, iris structure, voice, or typing behaviour (such as how strongly a user depresses keys on their keyboard).

These factors contain a large number of unique data points that require sophisticated technology to replicate, which most bad actors don't have access to.]

It is worth mentioning, that simple systems such as face scanning are not providing a suitable level of security. However, more complicated solutions which are more sensible and validate data from multiple factors have the right to be called the most secure solutions in the 2FA market.

Strong sides:

- Great user experience
- With proper implementation and tools provide high-level security

Drawbacks:

- May require additional hardware
- Price to use and develop
- Cheap or even free variants usually have poor security

2.2.5 Email

The idea behind email verification is similar to the SMS approach, the only difference, that instead of SMS-services serer is using emailing services to send a code to the mailbox. Despite public opinion, this method cannot be considered a second factor. As provided statistics show, users have a bad habit of reusing their credentials access to the email isn't “something user has” it is “something user knows”. Hence leaked passwords or password dictionaries may have keys to both email and external accounts

Strong sides:

- No additional hardware or software is needed
- Free to use

Drawbacks:

- Depending on emailing service used, might be slow
- Not secure / Not a second factor

2.3 2FA rating

Table 1: 2FA 1 to 10 rating

2FA kind	Security	Accessibility	User experience
USB key	10	4 (Not free)	9 (Need to keep the device with yourself)
SMS	8	10	8 (Manual data input)
2FA APPS	9	9	8 (Manual data input)
Biometric 2FA	4-10 (depends on type)	1-9 (depends on type)	10

Email	6	10	9 (Need to open the email and copy-paste password)

The ratings above are subjectical and represents author’s opinion based on research done.

3. Time-Based One-Time Passwords

Time-Based One-Time Passwords or TOTP is an algorithm for generating one-time passwords for secure authentication, which is an improvement on HOTP (HMAC-Based One-Time Password Algorithm). It is a one-way authentication algorithm - which means, that only one side is doing verification of the seconds. Usually, the server verifies the identity of the client.

3.1 Flow

For a better explanation of how Time-Based Passwords works and how it is applied to 2FA APPs, here is the user-story example:

1) A user wants to log into a TOTP 2FA-protected application or website. For the OTP authentication to run, the user and the server must initially share a static parameter (a secret key).

2) When the client logs into the protected website, they have to confirm they possess the secret key. So their TOTP application merges the seed and the current timestamp and generates a HASH value by running a predetermined HASH function. This value essentially is the OTP code the user sees on the screen.

3) Since the secret key, the HASH function, and the timestep are the same for both parties, the server makes the exact computation as the user's OTP generator.

4) The user enters the OTP, and if it is identical to the server's value, the access is granted. If the results of the calculations aren't identical, the access is, naturally, denied.

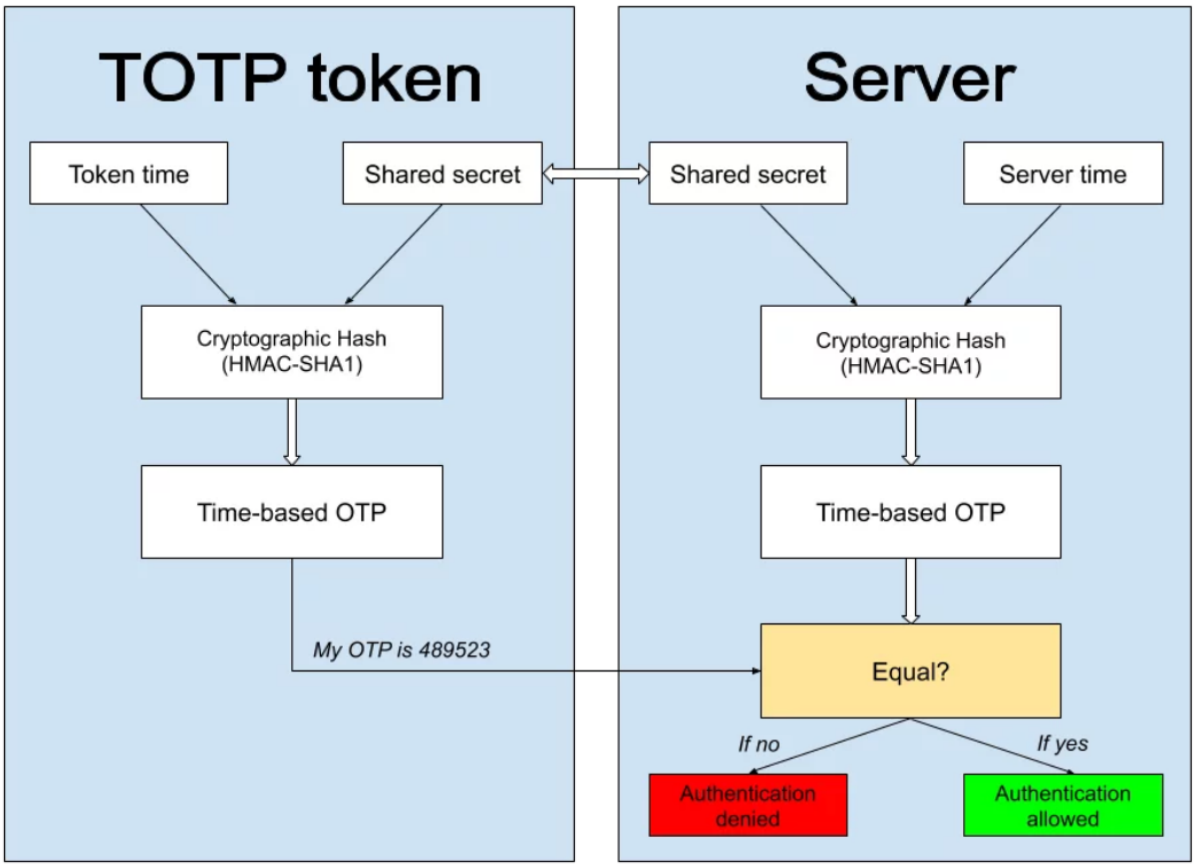

Figure 1: One-Time Password authentication diagram[9]

To clarify the above example, note that the seed mentioned is a string of random characters, typically 16-32 characters long. "Sharing" a key usually involves scanning a QR code that contains a server-generated seed.

The time step is calculated using UNIX time, which is the number of seconds passed since January 1, 1970, UTC. The time steps must be 30 or 60 seconds, so the time value used for TOTP is the number of seconds since 00:00 on January 1, 1970, divided by 30 or 60. Finally, the mentioned HASH function is a cryptographic mathematical function. which generates the same result for the same input, if any input arguments differ (secret key or current timestamp) result will be different. The result is usually 6-8 characters long.[9]

3.2 HMAC

Mechanisms that provide integrity checks based on a secret key are usually called "message authentication codes" (MAC). Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. MAC mechanisms based on the HASH functions are called HMAC or Hash-based

Message Authentication Code. HMAC can be used in combination with any iterated cryptographic hash function. MD5 and SHA-1 are examples of such hash functions. HMAC also uses a secret key for the calculation and verification of the message authentication values. The main goals behind this construction are:

- To use, without modifications, available hash functions. In particular, hash that performs well in software, and for which code is freely and widely available.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the underlying hash function.
- To allow for easy replaceability of the underlying hash function in case faster or more secure hash functions are found or required.[3]

3.3 HOTP algorithm

HMAC-Based One-Time Password Algorithm or HOTP is an algorithm for generating one-time passwords based on HMAC(SHA-1) technology. The main idea is to create unique passwords with every algorithm iteration. The algorithm is event-based. System running authentification checks with HOTP technology is keeping an internal counter of algorithm iterations and uses it as one of the parameters during OTP generation. [10]

The formula presentation of the HOTP algorithm will look like that:

$$HOTP(K, C) = Truncate(HMAC - SHA - 1(K, C))$$

where

K - Secret key shared between user and server

C - Current value of the 8-bit counter

3.4 TOTP algorithm

TOTP algorithm is another way of generating One-Time Passwords. Follows the same algorithm of using the HASH function on a secret key. This variant of the HOTP algorithm specifies the calculation of a one-time password value, based on a representation of the counter as a time factor.[11]

The TOTP algorithm can be defined with the following formula:

$$TOTP = HOTP(K, T)$$

where T is an integer and represents the number of time steps between the initial counter time T0 and the current Unix time. More specifically:

$$T = (Current\ Unix\ time - T0) / X$$

where

X - represents the time step in seconds (default value X =30 seconds) and is a system parameter.

T0 - the Unix time to start counting time steps (default value is 0, i.e., the Unix epoch) and is also a system parameter.

The computation is requiring the default floor function to be used. For example, with T0 = 0 and Time Step X = 30, T = 1 if the current Unix time is 59 seconds, and T = 2 if the current Unix time is 60 seconds.

4. Minimal prototype requirements and technology tool selection

The first thing to be done is to define the minimal requirements of the project. Knowing what expected functionality is, will lead to the understanding of what technologies and solutions should be used during implementation.

4.1 Prototype requirements

The thesis practical part should be able to provide users with the same service as other 2FA mobile APPs.

Apart from basics, it is supposed to keep users from touching the phone during the authentication process and completely solve manual data input issue.

In the most probable scenario, a user is using multiple services which require 2FA. Hence application should be able to identify which one of the passwords should be given to the user and when. To get this data, there should be a browser extension that collects information about the current webpage and use it as an identificator. Also, it would be a perfect point to trigger the whole process.

Concluding this scope, the following functionality should be implemented

- Scan QR codes, parse them, and convert into salt for Time-Based Password generation
- Securely keep salts in phone memory
- Every 30 seconds generate TOTP for each salt record
- Delete, modify records
- Not dependent on the Network. Works offline
- A tunnel between mobile and PC to pass TOTPs
- The user is able to request a TOTP via the browser extension

- The Browser extension collects information about users' opened web page and passes it as a password identity.

4.2 Technology stack

The modern IT sector is offering lots and lots of different types of approaches, solutions and technologies. Every single problem could have several solutions, and each would be relevant. In projects like this thesis, where infrastructure consists of multiple parts, the proper planning could be crucial. The developer should consider each part's efficiency and their ability to be integrated with each other.

4.2.1 Phone platform

According to the web portal “gs.statcounter.com”, the most spread mobile platforms are Android and IOS. In May 2022, their combined share is about 99.3% of the mobile market. where Android has 71.7% and IOS 27.57% [12]

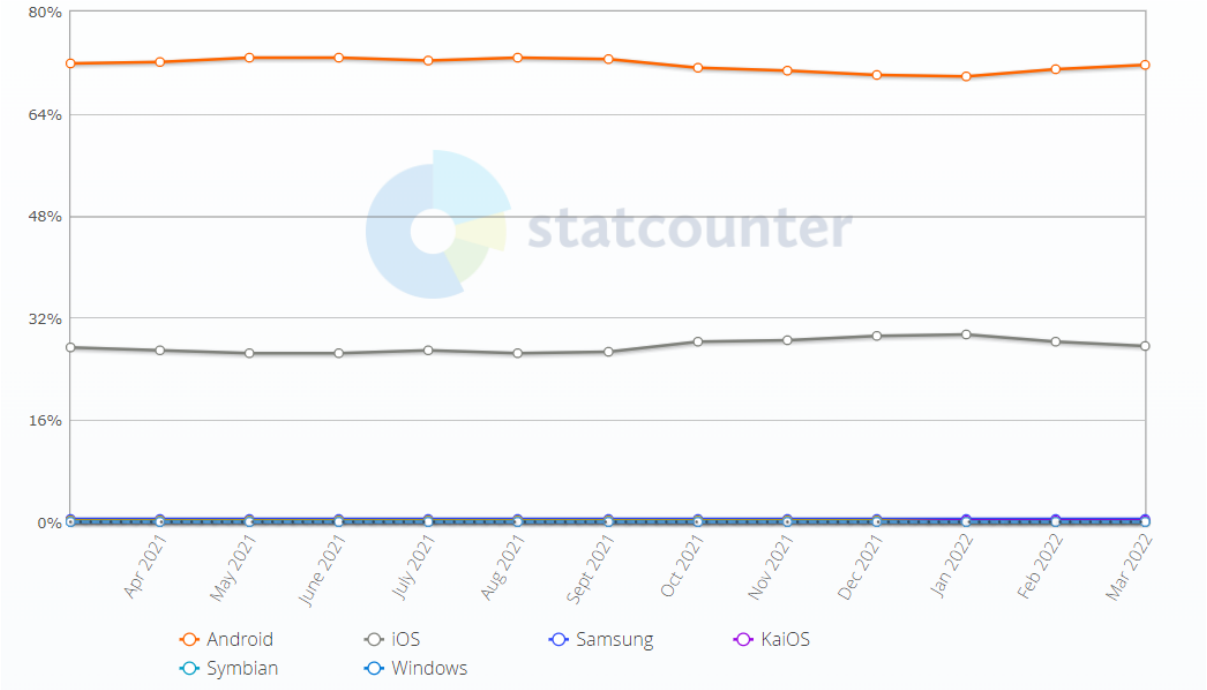

Figure 2: Mobile operating system market share [12]

It would be logical to use any Cross-Platform languages such as Flutter or Xamarin, to cover both most popular OS. However, running any code on IOS requires an "Apple Developer"

certificate which costs money and takes months to obtain. so it was decided to stick to Android only.

When it comes to Android development, there are multiple options of supported languages such are Java, Kotlin, C#, Lua, and Basic. Java has the biggest community, and it is the most widely spread solution for Android development. Kotlin, on the other hand, took all the best from Java[13] and became its inheritor. As a result, it has more syntactic sugar, functional programming support, and full compatibility with Java, which allows the use of Java libraries.

The fundamental factor why Kotlin was selected for this project is the fact that Kotlin has a feature called “Coroutines”. It is similar to the Java “Threads”, but it is more flexible and efficient. Moreover, it runs concurrently, which means it will never block other processes running on the system. As it was claimed before, the application is supposed to generate a TOTP every 30 seconds, and this is why it is critical to use an efficient multi-processing architecture.

4.2.2 Desktop solution

As one of the critical points of this project is making the environment easy to install, cross-platform languages will be perfectly suitable. The choice was made into Python because it has a vast community, which means there are several open-source libraries that can be easily added to the project with Python’s package manager called PIP. Also, Python has integration with Operating Systems and can approach its functions from scripts. And last but not least, Python allows developer to run Bash commands and manage their result.

Apart from excellent developer experience, Python provides a powerful compiler that can build the Python part of the project into lightweight files.

4.2.3 Browser extension

There are not so many ways to build browser extensions nowadays. The only decision that developers should make is either use `manifest.json` v2 or v3. The second version is quite old already, and some browsers such as Google Chrome are going to deprecate it in the future totally. On the other hand, the third version has the complete functionality of the previous

standard and a list of improvements on board. The only drawbacks are a small number of educational materials and possible compatibility issues with browsers like Microsoft Edge.

According to “gs.statcounter.com”, the most popular browsers at the current moment are Google Chrome and Safari [14], which both have the full support of the latest manifest version, so it was decided to use `manifest.json` v3 to avoid deprecations in the near future.

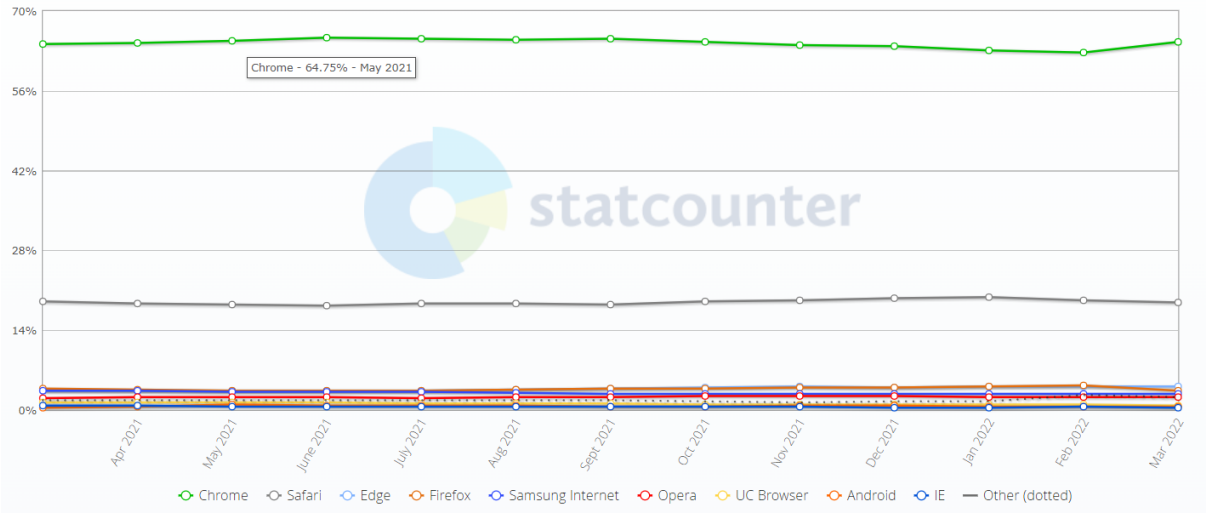

Figure 3: Browser market share [14]

5. Data tunnel

Defining how to transfer data from phone to browser extension is one of the most complex parts of this research. There are multiple parameters that should be taken into consideration and may affect productivity and user experience. Solutions that are too complex to set up will not suit most users. The same goes for the design that uses too many computer or phone resources.

The crucial factors of this decision are the following:

- As lightweight as possible
- Do not require any maintenances on daily bases
- Do not block or affect any other processes going in the systems
- Initial installation should be easy for the user

5.1 API on Local network

Keeping data track on the local networks is probably one of the most obvious solutions, but it has a lot of drawbacks. Unfortunately, just keeping API running on the local machine does not provide other devices access to it, so the android phone will simply not “see” an API. There is a possibility of running it on a router if the device allows it. However, it is extra complicated for regular users, plus it probably will not be an option in the office environment. Another way to do so is to rent a separate IP address from the Internet provider and run your own server. [15] It also would work, but it is an “overkill” and is too complex for the initial setup.

5.2 API on public Network

Looking back on research done, it could be said, that this approach has one of the most considerable potentials. However, it requires the most complicated solution running on a remote server.

It has to have an authorization system by itself, to separate one user from another. And some kind of security level to be a proper second factor, because just transferring passwords to the browser is ruining the whole idea of 2FA,

Other drawbacks of such an approach would be - the complicity of development and the fact that it needs the host and server to run, which aren't free, and it also means that it could not be open-source as opened code will show all the weak points to everyone. Hence such a system will probably be insecure. However, this infrastructure may be used by some commercial companies to solve the same issues that this thesis is trying to solve.

5.3 Bluetooth to PC

Using Bluetooth is one of the possible solutions. However, it requires having a Bluetooth adapter on the PC and being able to identify the phone and connect to it. Unfortunately, Windows machines have a specification on how they are supposed to work with Bluetooth-connected mobile devices. And this specification is not working in a way where the phone can set a raw data stream to the computer. Nevertheless, it is still possible to set up a specific COM port for a data stream with programs like PuTTY, and then the stream must be unencrypted in a readable way.

Because of the extremely high complexity of the setup, this approach was declined. However, this limitation exists only on Windows and only for Android phones. Furthermore, research has shown that any microcontrollers such as Arduino may send data streams to Windows-driven machines without any workarounds with pots.

5.4 Bluetooth to browser extension

A relatively new technology[16] that allows to access Bluetooth devices straight from the browser extension. The Windows-based machine has the same issues.

It was not tested during this research, but potentially approaching this technology to Linux and IOS machines will allow skipping one step of infrastructure as data will be passed directly between phone and computer.

5.5 USB tethering / Hotspot

Both HotSpot and USB tethering are solutions to deliver the Internet connection to the computer from the phone. The difference is that hotspot simulates a WI-Fi network, and USB tethering creates a wired connection via a USB port.

Both approaches may be used to share localhost between devices and use a local API server as a data tunnel. The problem is that this connection isn't consistent and requires extra interactions. To be more specific, every time when this network is created IP addresses may be changed, so the user will have to check the current computer IP address and put it into the phone every time he disconnects the cable or turns the hotspot off.

Apart from mentioned issues, this way maybe not be suitable for people working in the office environment as it often has its own Wi-Fi network, which may be needed for work routine. It also wastes the user's internet traffic and may affect internet speed as USB tethering has a speed limitation of 1 Mb/s.

5.6 Android Debug Bridge

Android Debug Bridge or ADB is a command-line utility included with Google's Android SDK. ADB can control mobile devices over USB from a computer, copy files back and forth, install and uninstall apps, run shell commands, and more[17].

Using ADB, users are able to access localhost services running on the PC, as the phone and computer are in a single space. This is why having a simple API on the computer seems to be a solution, as browser extensions also may easily access it to request the current password value.

It works perfectly fine on paper, as does on a phone emulator, but going this way reviled some inconveniences. To use ADB, the user should have android SDK installed on his machine. Moreover, the phone should have developer mode enabled. Thanks to Google, it may be done in a couple of clicks. And last but not least, the port in use for running the server on the localhost should be opened for ADB.

5.7 USB / Bluetooth file modification

During the planning stage, this approach was declined because modifying a text file on the computer seemed to be a workaround. However, after several hours spent on ADB implementation and polishing the user experience of setting it up, it may be said that file modification could be the best solution for the data transfer problem.

It is super easy to set up because it does not require any interaction from the user side to make his phone able to modify files. It works “from the box”.

The implementation would be similar to the ADB way, but instead of sending passwords from phone to PC via HTTP request, the phone would write passwords and timestamps to a txt file in a specific location. Then local server just needs to parse it and save passwords in memory. Then, when it will be needed, the browser will request this data and check the timestamp to know if passwords are fresh. The last step is vital because after the USB cable would be plugged out, data saved in a text file will remain the same, which may bring false-positive results.

6. Architecture

The practical part of this thesis is built on Android Debug Bridge basis. The ADB provides functionality to access localhost-running services on the specific ports that can be opened or closed per device.

The computer has a simple Python HTTP server running on the specific port. This port is exposed to the Android device, so the Android application is able to send REST requests to the localhost, and the computer will receive them.

As the user needs to receive his passwords while he is using the Internet, the entry point should be the browser. The extension will take the domain name of the website user has opened and send it as a parameter in the HTTP request to the Python API. A domain name is used to identify which password the user is expecting to receive, but unfortunately, it isn't possible to send a request from PC to Android. Only vice versa. So the following solution was taken: every 30 seconds, when passwords are updating Android Application sends a PUT request with a complete list of current passwords and their domains to the server. The server saves it to its memory and keeps it until the next update or when it receives a POST request with a domain name from the browser. When a request is received server checks if the password for the provided domain is in place. If a value is found, it responds with a valid password value.

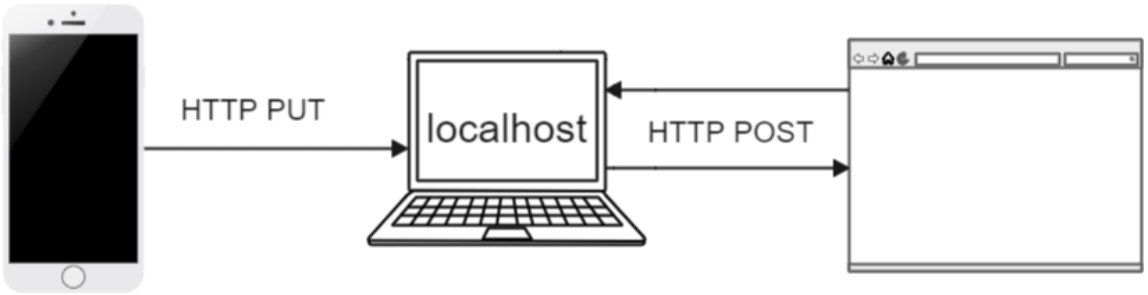

Figure 4: Project infrastructure schema

7. Android application development

7.1 Design

Before making an actual APP, it was decided to build a maquette first. For that purpose, the web service called UxPin was used. The archetypal design was inspired by Google Authenticator APP. It contained all the same details and a similar colour scheme. As the initial plan was to use Bluetooth to connect devices, it has a Bluetooth icon on the corner, which was supposed to work as a button to set up the connection.

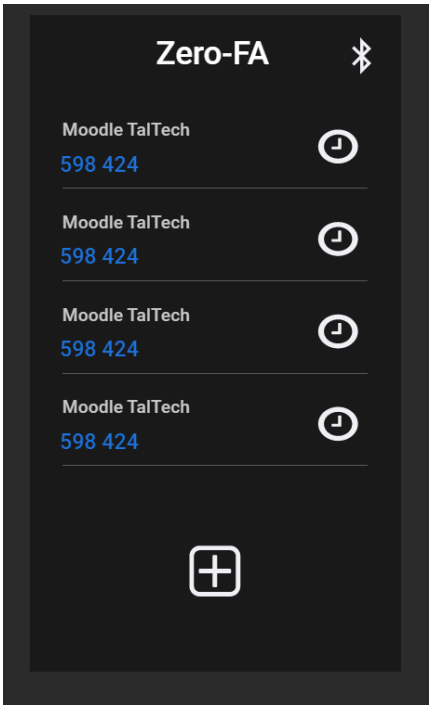

Figure 5: Android application design created in UxPin service

After finishing the maquette, it was used to recreate the same layout into the Android Studio program using the internal constructor with some XML modifications. The main structure consists of three main parts. `Appbar` with the application name and “Network” button, `Recycler View` which is used to render password records, and “Plus” button which turns on the camera to scan QR code and add a new record to the database.

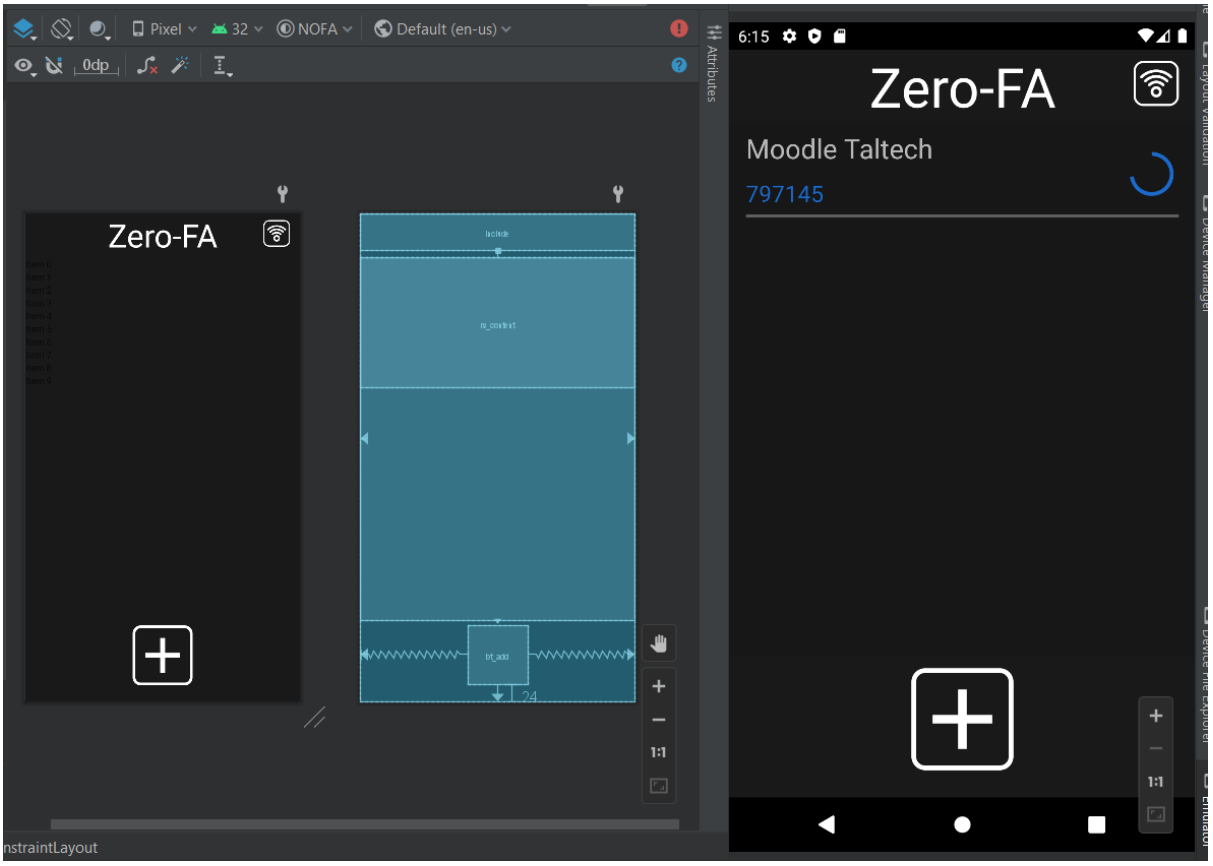

Figure 6: Android application design created in Android Studio

7.2 Database implementation

According to official Android documentation, the best practice for implementing databases is using a “Room” dependency inside the project. Room is a library that is working as an abstractor between the code and SQLite database running on the phone.

“Room” consists of three major components:

- The database class that holds the database and serves as the main access point for the underlying connection to APP's persisted data.
- Data entities that represent tables in the APP's database.
- Data access objects (DAOs) provide methods that applications can use to query, update, insert, and delete data in the database [18].

All three components have to be realised in code to start using Rooms functionality and maintain data in the database.

This project does not require any complicated DB schemas. The only table called “Accounts” represents records of the secret key and domain name to which it belongs.

Data classes implementation can be found in Appendix 2

7.3 OTP generation

When “Accounts” are received from the database, they can be turned into OTPs. For doing that and for faster development, a library called “Kotlin-onetimepasswords” was added to the project. The library itself is pretty powerful and contains several different ways of OTP generation. As a result, the developer saves a lot of time, as he only needs to convert the secret key into a byte array, take the current time, and pass parameters to the function.

7.4 Recycler View adapter

As was claimed before, one of the parts of the main layout is a recycler view which contains as many lines as mush password the user is keeping in the application.

Figure 7: Recycler View row design

The primary purpose of the `Recycler View` adapter is to bind data and XML objects. Every time a new record is added, the adapter is supposed to make an XML injection and render a new line in the recycler view. Moreover, every 30 seconds, when passwords are expired, the adapter changes all values to relevant ones.

Also, each line in the Recycler View has its own `Progress Bar,` which shows how much time the user has until the password expires. The progression is counted with the following formula:

$$Progress = ceil(T * 100 / X)$$

where

T - time left before password expiration

X - interval presenting how long the password is valid

7.5 QR code scanner

The library called "Zbar" was added to the project dependencies to have the functionality of QR code scanning. The library by itself is able to parse almost any QR code and does not have any limitations of usage.

The format basic format of OTP data kept in QR code is following [19]:

```
otpauth://totp/some_label?secret=secret_code&issuer=TalTech&algorithm=SHA1&digits=6&period=30
```

Mandatory parameters are:

- Secret - the salt
- Issuer - who provides this code. Usually domain name

Optional parameters:

- algorithm - shows what HASH function should be used to get OTP
- digits - how many symbols are expected in the OTP
- period - password expiring time

The application parses provided sting into a set of parameters. From the mentioned list, it takes "issuer" and "secret" to save them as an "Account" entity in the database. After that, the valid value of the password will be immediately counted and displayed on Recycler View.

7.6 API

For the proper setup of ADB-based API, the port used for running a local server has to be opened for mobile devices [20]. This procedure requires a specific console command to be executed in the terminal. After that step, physical devices will be able to send HTTP requests to `http://localhost:port`. However, for the application running on the emulator destination should be changed to `https://10.0.2.2:port`.

The current implementation of API is sending PUT requests with a complete list of valid passwords and their domains. The Content-Type is application/JSON, and the format of data is an array of “TOTP” entity records converted to JSON.

For performing HTTP requests, the application uses the "Retrofit" library and object mapping the "GsonConverterFactory" module from the same space.

It is worth mentioning that Android has strict limitations regarding any networking - it has to be separated from the main thread to another parallel process, so the startup application is creating a “Job” which executes every 30 seconds to perform API requests in the Courutine scope.

7.7 Permissions

To properly run its processes, the application will ask the user for permissions to some system functions, such as access to the camera and access to the Internet and the Internet state. All required permissions are written down in the “AndroidManifest.xml” file - the root of the project source set [21].

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figure 8: Android application required permissions

7.8 Security analysis

The architecture of the application is built in a way, that it will avoid the majority of possible abuses. As it was claimed before, the port with localhost should be opened to the phone and it happens per device basis. Before the application will start sending OTPs user have to unlock the phone and allow it. SQLite database is secured and only can be accessed if the attacker gets root access on the phone. However, if this happens, all the data and applications will be compromised. Concluding these points, it can be said that security at least on the regular 2FA application level.

8. Python API development

Using Python as a language for the desktop side was one of the most excellent decisions. The Python part of the project is elementary to install, as all used libraries can be downloaded via the PIP file manager. Implementation of the server that can serve three different types of requests and keep data in memory took less than 100 lines of code. Despite being that simple, the webserver works perfectly fine and does its job - keeping track of password updates and providing them by request.

Modules that are used in this part of the project are:

- HTTP.server - starts the server, handles requests and responses
- JSON- parses JSON objects from requests to python data formats (Arrays or dictionaries)
- BytesIO from IO module - used to convert strings into byte arrays and then send them as a response.

The server can be started only on HTTP space. HTTPS will not be working. However, one of the adjustments users can be made is to select the port to run the server. It may be beneficial if the user already has something running on his machine.

The code of the Python API implementation can be found in Appendix - 2

9. Browser extension development

The initial idea was to run the extension in the background and insert a button into `textfield` for OTP input. However, after fast research, it turned out that there is no specific format for how people on different sites would mark such textfields. Hence the idea was changed to create a button for the extension, which will show a popup with its status.

When a user clicks on the button to open this popup, the extension sends a POST request to the local server providing the hostname of the current page as an argument.

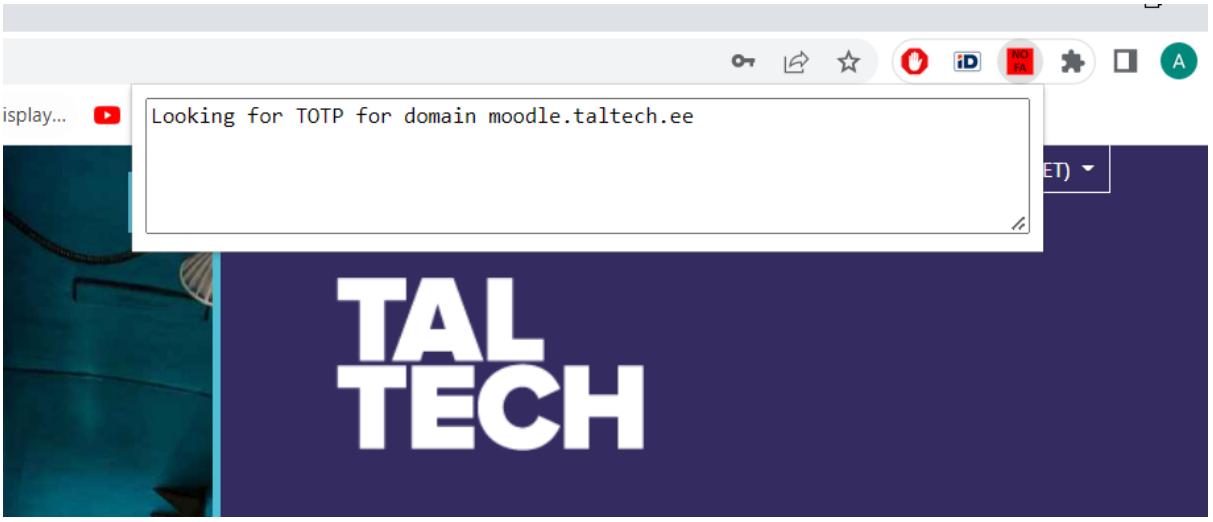

Figure 9: Browser extension popup

In case of success, a popup will disappear, and the password will be saved into the user's clipboard. However, if something goes wrong, it will show the error message.

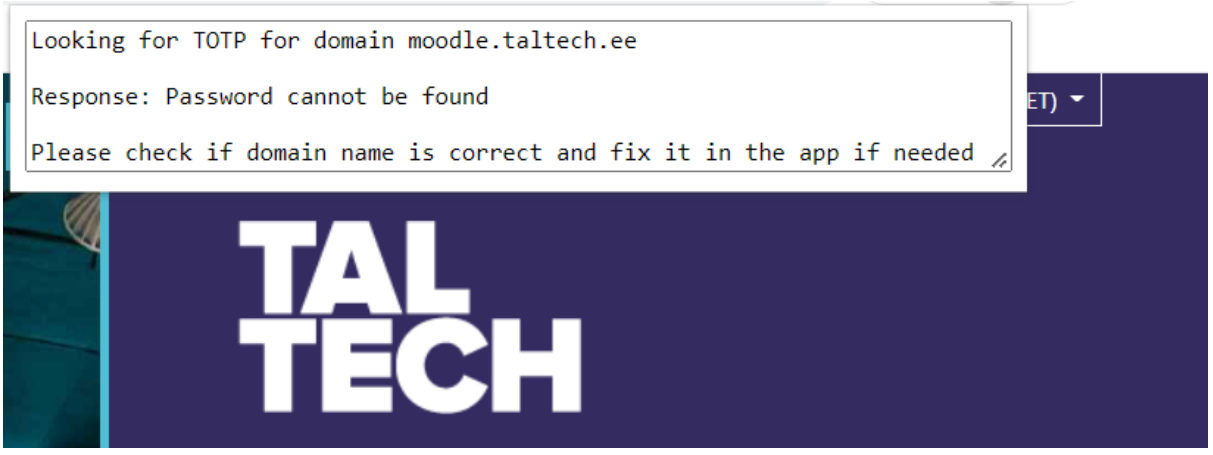

Figure 10: Browser extension error message

10. Application performance analysis

Using the common authenticator application takes 8 seconds on average to unlock the phone, open the application, find the password, and put it into the web form. It is also worth mentioning that mobile authenticator applications make users turn their focus away from the screen and distract them from the task they are performing. Unfortunately, there was no way to measure this distraction, but it is clearly seen that this factor has to be completely removed.

The application created during this thesis only takes 1-2 seconds to fill the password, so it is 6 seconds average profit. However, the new solution takes more time to set up, so it would be more profitable for people using 2fa regularly. It may be said that in the long perspective, users not only get a better experience, but he also saves his time spent on unneeded manual process.

11. Possible improvements and future steps

As was mentioned before, the practical part of this thesis is just an MVP or proof of concept, so there are many possible features that can be added to the final versions. For example, support of TOTP with an expiration time different from 30 seconds. The code is pretty structural, so it will not be a problem adding more functionality and polishing existing, especially because the code base is going to be public.

The level of security can be improved even more. It is possible to make data secure even if intruders received root access. For example, encrypt the database using SQLite Cipher and ask the user for the pin input on the application startup and use this pin to decrypt it back.

From the browser perspective - UI may be revisited to provide a better user experience. The current version seems to be too raw.

The most significant change that may be taken into consideration is a switch from ADB to fail modification flow. As it will make the initial setup extremely less complicated, and the program won't depend on Android SDK anymore, which also makes the weight of the environment less.

The future steps that are already decided are to make the code public and prepare the first version of the build, including all necessary files and scripts for opening ports, downloading Python and its dependencies, adding a Python server component into startup programs, etc.

12. Summary

The bachelor thesis aimed to create the prototype of a next-level environment for Two-factor authentication verification. Analysis of the existing platform shows the weak sides of current solutions and their possible modifications. As a result, the automated way of passing 2FA was invented. The manual data input factor was completely removed as well as needed to be distracted from the desktop during the verification. However, Security was kept on the same level as existing analogues. In the theoretical part of this paper, different approaches and technologies were compared and systemised that giving a lot of valuable data for future development of this product. MVP was reached, and functionality is the same as declared at the thesis's beginning. However final solution hardly can be used by the masses. Currently, it isn’t simply to install. Future steps are described as well as the pattern of solving the most significant issue of this project - complexity.

In conclusion, it may be said that the concept is proven and the prerequisites for the further development of the project have been created.

Bibliography

[1] Henry Ford, ”Today and Tomorrow” 1926

[2] gimpanews.com, “Two-factor authentication can save you from hackers”. 08 February 2019 [Online].
Available: https://app.gimpanews.com/two-factor-authentication-can-save-you-from-hackers-ericitguy/ [Accessed 14 April 2022].

[3] Internet Engineering Task Force, “ HMAC: Keyed-Hashing for Message Authentication”. February 1997 [Online]
Available: https://datatracker.ietf.org/doc/html/rfc2104 [Accessed 12 April 2022]

[4] Sam Manjarres, “2021 World Password Day: How Many Will Be Stolen This Year?” 4 May 2021 [Online]
Available: https://www.secplicity.org/2021/05/04/2021-world-password-day-how-many-will-be-stolen-this-year/ [Accessed 14 April 2022]

[5] Branko Klark, “Impressive Password Statistics to Know in 2022” 6 April 2022 [Online]
Available: https://webtribunal.net/blog/password-stats/ [Accessed 23 March 2022]

[6] Shape Security, “Credential Stuffing Attacks” 6 July 2015 [Online]
Available: https://www.f5.com/solutions/credential-stuffing [Accessed 1 April 2022]

[7] Suzanne Humphries, “What is a USB Security Key, and Should You Use One?” 8 December 2022[Online]
Available: https://www.reviewgeek.com/63448/what-is-a-usb-security-key-and-should-you-use-one/ [Accesed 19 April 2022]

[8] computing.which.co.uk, “How to set up an authenticator app for two-factor authentication” 31 October 2018 [Online]
Available: https://computing.which.co.uk/hc/en-gb/articles/360006153539-How-to-set-up-an-authenticator-app-for-two-factor-authentication [Accesed 20 April 2022]

[9] Maxim Oliynyk, “TOTP algorithm explained”. 24 June 2020 [Online].
Available: https://www.protectimus.com/blog/totp-algorithm-explained/ [Accessed 14 April 2022]

[10] Internet Engineering Task Force, “HOTP: An HMAC-Based One-Time Password Algorithm”. December 2005 [Online]
Available: https://datatracker.ietf.org/doc/html/rfc4226 [Accessed 14 April 2022]

[11] Internet Engineering Task Force, “TOTP: Time-Based One-Time Password Algorithm”. May 2011 [Online]
Available: https://datatracker.ietf.org/doc/html/rfc6238#section-4 [Accessed 14 April 2022]

[12] gs.statcounter.com, “Mobile Operating System Market Share Worldwide”. March 2022 [Online]
Available: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed 15 April 2022]

[13] kotlin.org, “Frequently asked questions”. 11 January 2022 [Online].
Available: https://kotlinlang.org/docs/faq.html [Accessed 15 April 2022]

[14] gs.statcounter.com, “Browser Market Share Worldwide”. March 2022 [Online]
Available: https://gs.statcounter.com/browser-market-share [Accessed 15 April 2022]

[15] itstillworks.com, “How to Set Up a Private Network”. 21 July 2017 [Online]
Available:
https://computing.which.co.uk/hc/en-gb/articles/360006153539-How-to-set-up-an-authenticator-app-for-two-factor-authentication [Accessed 21 April 2022]

[16] François Beaufort, “Communicating with Bluetooth devices over JavaScript”. 1 October 2021 [Online]
Available: https://web.dev/i18n/en/bluetooth/ [Accessed 16 April 2022]

[17] Chris Hoffman, “How to Install and Use ADB, the Android Debug Bridge Utility”. 28 October 2021 [Online]
Available:
https://www.howtogeek.com/125769/how-to-install-and-use-abd-the-android-debug-bridge-utility/ [Accessed 16 April 2022]

[18] developer.android.com, “Save data in a local database using Room” 27 October 2021 [Online]
Available: https://developer.android.com/training/data-storage/room [Accessed 5 April 2022]

[19] Thomas Habets, Google LLC, “Google-authenticator Key Uri Format” 26 November 2016 [Online]
Available: https://github.com/google/google-authenticator/wiki/Key-Uri-Format [Accessed 11 April 2022]

[20] developer.android.com, “Android Debug Bridge (ADB)” 30 March 2022 [Online]
Available: https://developer.android.com/studio/command-line/adb [Accessed 5 April prill 2022]

[21] developer.android.com, “App Manifest Overview” 7 April 2022 [Online]
Available: https://developer.android.com/guide/topics/manifest/manifest-intro [Accessed 14 April 2022]

Appendix 1 - Non-exclusive license for reproduction and publication of a graduation thesis[1]

I Aleksandr Zahharjan

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Developing an Open-Source Environment for Two-Factor Authentication” supervised by Kaido Kikkas

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

16 May 2022

[1] The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Database implementation code

```
package com.example.no_fa.modules

import android.util.Log
import androidx.room.Entity
import androidx.room.PrimaryKey
import dev.turingcomplete.kotlinonetimepassword.GoogleAuthenticator
import java.sql.Date

@Entity(tableName = "accounts")
data class Account(

    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val domain: String,
    val salt: String

) {
    fun convertToTotp(): Totp {
        val baseSalt = salt.toByteArray()
        val code = GoogleAuthenticator(baseSalt).generate(Date(System.currentTimeMillis()))
        Log.d( tag: "code generator", code)
        return Totp(domain, code)
    }

}
```

Figure 11: Android application. “Accounts” entity code

```kotlin
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import com.example.no_fa.dao.PasswordDao
import com.example.no_fa.modules.Account

@Database(entities = [Account::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun totpDao(): PasswordDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            val tempInstance = INSTANCE
            if (tempInstance != null) {
                return tempInstance
            }
            synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                     name: "totp_database"
                ).allowMainThreadQueries().build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

Figure 12: Android application database class implementation code

```kotlin
package com.example.no_fa.dao

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.Query
import com.example.no_fa.modules.Account

@Dao
interface PasswordDao {
    @Query( value: "SELECT * FROM totps")
    fun getAll(): List<Account>

    @Query( value: "SELECT * FROM totps WHERE domain LIKE :domain LIMIT 1")
    fun findByName(domain: String): Account

    @Insert
    fun insert(account: Account)

    @Delete
    fun delete(account: Account)

    @Query( value: "DELETE from totps")
    fun deleteAll()
}
```

Figure 13: Android application DAO class code

Appendix 3 - Python API code

```python
import json
from http.server import BaseHTTPRequestHandler, HTTPServer
from io import BytesIO

hostName = "localhost"
serverPort = 2104
passwords = {}


class MyServer(BaseHTTPRequestHandler):...


if __name__ == "__main__":
    webServer = HTTPServer((hostName, serverPort), MyServer)
    print("Server started http://%s:%s" % (hostName, serverPort))

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

    webServer.server_close()
    print("Server stopped.")
```

Figure 14: Python API entry point code

```python
class MyServer(BaseHTTPRequestHandler):

    # health check
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        response = BytesIO()
        response.write("{\"status\": \"connected\"}".encode())
        self.wfile.write(response.getvalue())

    # get passwords
    def do_POST(self):
        global passwords
        content_length = int(self.headers['Content-Length'])
        body = self.rfile.read(content_length).decode()

        self.send_response(200)
        self.end_headers()
        response = BytesIO()

        value = passwords.get(body)

        if value:
            response.write(str(value).encode())
        else:
            response.write("Password cannot be found".encode())

        self.wfile.write(response.getvalue())
```

Figure 15: Python API GET and POST requests code

```python
# update passwords
def do_PUT(self):
    global passwords
    content_length = int(self.headers['Content-Length'])
    body = self.rfile.read(content_length)
    body_string = body.decode()
    self.parsePasswords(json.loads(body_string))

    self.send_response(200)
    self.send_header("Content-type", "application/json")
    self.end_headers()
    response = BytesIO()
    response.write("{\"status\": \"OK\"}".encode())
    self.wfile.write(response.getvalue())

def parsePasswords(passwords_array):
    global passwords
    passwords = {}
    for elem in passwords_array:
        passwords[elem.get("title")] = elem.get("value")
```

Figure 16: Python API PUT request code