TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Ilya Makarov 185282IACB

Touch-free Solution for Smart Elevator Operation with Dynamic QR code Generation

Bachelor's thesis

Supervisor: Uljana Reinsalu
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ilya Makarov 185282IACB

Puutevaba lahendus nutilifti opereerimiseks dünaamilise QR koodi genereerimisega

Bakalaureusetöö

Juhendaja: Uljana Reinsalu
PhD

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Ilya Makarov

17.03.2021

Abstract

Behind this work is a project for KONE elevators in TalTach buildings. The task of this thesis is to create a device that allows users to access the elevator control without physical buttons. In this thesis, only an embedded device that displays a QR code is considered. The QR code contains dynamic URL of the web application, in which users can call the elevator. The complete project plan includes a server and a web application, which are not part of this work. The dynamic part of the URL is a one-time password. The device is standalone, it does not have any connection with the server, so it is standalone. The physical device includes the main microcontroller and a screen on which the QR code is displayed. The result of the work is an assembled prototype, embedded software and libraries written for it. The TOTP algorithm is implemented to generate one-time password. Within TOTP, the HMAC algorithm and the SHA hash function are used. At the end of the work, remain open questions of synchronization of the device with the server, case, and testing in real conditions.

This thesis is written in English and is 42 pages long, including 7 chapters, 21 figures and 1 table.

Annotatsioon

Puutevaba lahendus nutilifti opereerimiseks dünaamilise QR koodi genereerimisega

Selle töö põhjaks on TalTechi liftide jaoks tehtud projekt KONE. Lõputöö eesmärgiks oli teha seade, mis võimaldab lifti kutsuda ja sisestada korruse numbri kontaktivabalt. Siin kontekstis kontaktivabalt tähendab – tavaliste liftiga suhtlemiseks tehtud nuppe kasutamata. Terve projekt sisaldab serverit, aga siin töös seda ei vaadelda. Selles projekti osas vaadeldakse ainult seadet, mis näitab ekraanile dünaamiliselt QR koodi. QR koodi kasutamisel tuleb URL, mis on mingi perioodi jooksul aktiivne. Veebilehele tuleb veebirakendus lifti juhtimiseks. URL genereeritakse sünkroniseeritult seadmel ja serveril, kasutades TOTP (Time-based One-Time Password) algoritmi. URL-i muudetavaks osaks tuleb ühekordne salasõna, mida saadakse TOTP abil, mis omaette määratleb veebilehe aadressi. TOTP algoritmis on kasutatud palju teisi algoritme, mis on kirjeldatud töös, näiteks räsifunktsioon. Eeldatakse, et mingi aja jooksul server paneb veebilehe vana salasõnaga kinni ja avab uut. Seadmel puudub igasugune side serveriga, ehk seade on iseseisev. Füüsiline seade sisaldab endas juhtivat mikrokontollerit ning ekraani, kuhu kuvatakse QR kood. Töö esimestes etappides ühendati ka ToF sensor, kuigi lõpuks tuli otsus sellest ära öelda. Töö tulemuseks on valmis prototüüp, prototüübi jaoks kirjutatud programm ja teegid. Avatuteks küsimusteks jääb: seadme ja serveri sünkroniseerimine, korpus ja testimine päris tingimustes.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 42 leheküljel, 7 peatükki, 21 joonist, 1 tabelit.

List of abbreviations and terms

API	Application Programming Interface
CCI	Character Count Indicator
ECL	Error Correction Level
eInk	Electronic Ink
HMAC	Keyed-hash Message Authentication Code
HOTP	HMAC-based One-time Password
I2C	Inter-Integrated Circuit
ID	Identifier
LCD	Liquid-crystal Display
MAC	Message Authentication Code
MCU	Microcontroller Unit
NFC	Near-field Communication
OLED	Organic Light-emitting Diode
OTP	One-time Password
PIN	Personal Identification Number
QR	Quick Response
RTC	Real-time Clock
RTTI	Run-time Type Information
SHA	Secure Hash Algorithm
SPI	Serial Peripheral Interface
SRAM	Static Random-access Memory
ToF	Time-of-Flight
TOTP	Time-based One-time Password
UART	Universal Asynchronous Receiver-transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
Wi-Fi	Family of wireless network protocols, based on the IEEE 802.11 family of standards

Table of contents

List of figures

List of tables

1 Introduction

As the technological advancements are brought into our everyday lives for wide range of products and services, the situation is no different for elevator systems, which facilitate transportation in our modern and tall buildings. More and more touch-free technologies and smart control solutions are spreading.

1.1 Background

In the consideration of all technological changes, this topic is focused on building and developing a touch-free usage system for KONE smart elevators that are specifically used in Tallinn University of Technology ICT building. The idea is to provide users an ability to call the elevator through their smartphones.

Final implementation of the whole system implies that users will be able to use a web application on a phone to operate the elevator without the need of touching its buttons and to receive information about it. Elevator calls will be done by scanning live QR (Quick Response) code, which is dynamically generated every given time interval and located next to the elevator. Scanning the QR code, users should be redirected to a web application to specify their destination as an input.

1.2 Problem definition

In this work, only a part of the system is created. A device on which dynamic QR code will be generated and displayed plus the software required for this will be created. The main tasks of the work can be divided into two parts: hardware and software. The creation of the hardware part implies the assembly of a prototype based on an affordable to an ordinary consumer microcontroller and the necessary peripherals, such as a screen, sensors and other modules. The part with the software includes the development of a program built into the microcontroller for its control, and the implementation of the algorithms necessary for solving the problem, for example, generation of a QR code.

The work covers such topics as the selection of suitable components, algorithms for generating hash sums and tokens to regularly create new URLs (Uniform Resource Locator), generation and displaying a QR code, communication of the device with a server, energy efficiency, as well as analysis of the restrictions imposed on the device and the system as a whole. At the same time, the possibility of using an energy efficient ToF (Time of Flight) sensor for detecting people and collecting statistics is being investigated.

One of the prerequisites for the device is battery power source, therefore high-energy efficiency of the device and long battery life are important criteria. Since the work will be carried out exclusively in the field of embedded systems, C/C++ will be used as a software development tool with appropriate restrictions, for example, without using the standard library.

1.3 Structure

The work consists of 7 chapters, including an introduction and conclusions, but logically the work can be divided into 3 parts. Chapters 3 and 4 will consider the algorithms used in the work and the software based on them. Chapter 5 describes the hardware, prototype assembly and firmware for the microcontroller, that is, the final program that uses the software from the previous chapters. The last logical part of the thesis consists of three chapters. Chapter 6, which describes the testing of the resulting prototype, Chapter 7, which analyzes the results obtained and the last chapter with a general conclusion.

2 Security token

Almost any system has security measures, be it protection against physical hacking, various integrity checks, condition monitoring, or authentication methods. The elevator control system is no exception. Access to the elevator control should be granted only to users who are in the immediate proximity of an elevator, on a specific floor. Since the system described in this work does not imply any authorization method, one of the simplest solutions is to constantly change the valid URL. After a fixed time interval, the server will invalidate the old link for controlling the elevator. At the same time, the device and the server will generate a new valid URL that will be displayed on the device screen. The device-specific URL itself will have two parts: static and dynamic. The static part includes protocol, hostname, and the ID (Identifier) parameter of the device. The dynamic part is the constantly updated parameter, which can be considered as a kind of one-time password. Below is an example URL.

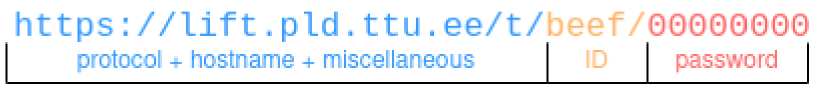

https://lift.pld.ttu.ee/t/beef/00000000
protocol + hostname + miscellaneous
ID
password

Figure 1. URL example

Since this work does not consider the server side, the static part of the link does not reflect the final form, including the names of the parameters and the domain. The possibility of transferring an active URL through any communication channels to a remote person is also not considered. In this case, the user's actions are considered as intentional compromise by transferring information to a third party.

To implement the above mechanism, the server and the device must somehow synchronously update their state using the same algorithm. There are technologies that have been created to solve this problem, namely algorithms for generating one-time passwords. The common name of the family of such algorithms is OTP (One-time Password) algorithms. They are widely used in the IT industry for authentication. These algorithms can be both proprietary and publicly available, and can be implemented both in hardware and in software. For example, the Google Authenticator app[1]. [1]

In the described above authentication mechanism, the device being developed is nothing more than a kind of security token. A security token, or simply token, is a device that allows accessing limited resources. Token can be based on several types of passwords [2]:

- Static password - a password is stored in token, does not change and is physically hidden from the user.

- Synchronous password - a new password is generated at a certain time interval based on the value of the counter (timer) and hidden information known to the token and the certifying device.

- Asynchronous password - a new password is generated based on hidden information and a cryptographic algorithm such as Vernon cipher or public key method.

To solve our problem, the option with a static password is not suitable, since it contradicts the idea of a dynamic one-time password. Since communication with the server should also be kept to a minimum in order to save energy, the most appropriate of the remaining options is a synchronous password based on a timer. This method is also successfully used in PIN calculators (Personal Identification Number) and other common security tokens, for example, in devices of RSA Security LLC - SecurID [3]. Further, we will consider specific algorithms for generating OTP, which are in the public domain. Other technologies used for their implementation will also be considered, for example, hash functions.

2.1 HOTP and TOTP algorithms

Algorithms for generating one-time passwords were originally invented to prevent replay attacks and were actively used for two-factor and multifactor authentication.

[1] https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&

One of the first algorithms of this class can be considered S/KEY One-Time Password System. It was designed to generate a unique password sequence to prevent eavesdropping and was used in Unix-like systems [4]. In the work of Dr. K. Mohan Kumar and G. BalaMurugan [5] comparison of some popular OTP algorithms happens. Among them are such algorithms as TOTP, HOTP, Lamport OTP, S/KEY, two-factor authentication algorithm and others. Many proprietary OTP algorithms are also just modified versions of the publicly available TOTP algorithm. In this work, the TOTP algorithm was chosen for use, since it is reliable and, most importantly, it avoids the need for communication between the device and the server. There are many materials and various open-source implementations of this algorithm. This confirms its prevalence in the security industry. Next, the HOTP and its modification TOTP, which introduces the time factor, will be described.

HOTP (HMAC-Based One-Time Password Algorithm) appeared in 2005 as a result of the work of the Open Authentication group, and is documented in the RFC 4226 standard [6]. The idea was to provide public with an open and free secure authentication algorithm, in the hope of an early adoption of two-factor authentication across the Internet. The three main elements of this algorithm are the secret key, counter, and hash-based message authentication code. A secret key is a key that is unique for each client, which is known in advance by the client and the certifying party, and which, for security reasons, has restrictions on the minimum length. Counter - an 8-byte counter that is incremented for each successful authentication by one, and the value of which must be synchronized between the client and the server. HMAC (Hash-based Message Authentication Code) is an algorithm used inside HOTP to reliably create unique values, which will be described in detail in the next chapter. Mathematically, the equation (1) describes the HOTP algorithm.

$$HOTP(K,C) = Truncation(HMAC_H(K,C)) \tag{1}$$

H – cryptographic hash function used in underlying HMAC algorithm.

K – secret key, used also as key in HMAC.

C – counter, represents 8-byte “message” in underlying HMAC.

Truncation – function for dynamic truncation of HMAC result, described below.

First, the secret key and the counter value in the form of an 8-byte string are passed through the HMAC algorithm based on the selected hash function *H*. In the original document describing HOTP, SHA-1 (Secure Hash Algorithm 1) is used as *H*, so the size of the output string is equal to 20 bytes. Next, offset is calculated as least significant 4 bits of the last byte of the output line. That is, if the length of the HMAC result is 20 bytes, then offset will be taken from the 19th byte (when counting from zero). The final value will thus be in the range from 0 to 16, and in the C language this can be represented as: [6]

```
int offset = hmac_result[19] & 0xf;
```

Figure 2. HOTP offset calculation

This offset is then used as an index to select the start byte in the same array. A bitwise AND with 0x7f operation is performed on the byte selected by offset index in the array. It is shifted 24 positions to the left as the most significant byte of the resulting 32-bit code. Subsequent ascending 3 bytes in the array are stored in the resulting code, shifting in accordance with their position, but without applying the 0x7f mask. For example, if offset is 4, then the resulting code will be calculated as follows: [6]

```
uint32_t code = (hmac_result[4] & 0x7f) << 24 |
                 hmac_result[5]         << 16 |
                 hmac_result[6]         << 8  |
                 hmac_result[7];
```

Figure 3. HOTP code calculation

Since the maximum offset value of the index can be 15, the hash function used in the HMAC algorithm should yield an output of at least 19 bytes long, since elements can be taken from the 15th to the 18th index. After composing the resulting code, the modulus of number 10 to a power is taken from it. The authors of the algorithm propose to use degree 6 as a trade-off between security and ease of use. The end result will then be interpreted as an unsigned six-digit number, which is convenient to use as a one-time password or PIN code, as is done in bank PIN calculators [6]. Larger numbers can be used for modulo to improve security. This issue will be further discussed in the **Development** chapter during own implementation of this algorithm.

The authentication process itself is extremely simple. As soon as the client wants to perform the next operation that requires confirmation, he generates a new one-time password, increases the counter by 1 and sends the password to the server. The server accepts the password, authenticates and increments the counter by 1. In this process, the algorithm takes into account two more variables, which, however, concern only the server: T – throttling parameter and s – resynchronization parameter. Throttling parameter indicates the maximum number of failed authentication attempts after which the server will stop communicating. Resynchronization parameter - the number of iterations during which the server will receive a one-time password if the client's counter is out of sync. For example, if this parameter is equal to two, then the server will try to calculate not only the password with the active counter value, but also with two subsequent ones. These measures are necessary because in the naive version of this algorithm, the server counter is incremented only after successful authentication, and the client counter with each request. If, for any reason, the one-time password in the client's message is distorted, then the authentication will fail, and the client's counter will be ahead of the server's counter. Since this algorithm does not solve the issue of the reliability of the communication channel, the possibility of desynchronization is not considered as a problem. [6]

A further evolution of the HOTP algorithm is TOTP. Like the previous algorithm, it is a product of the Internet Engineering Task Force and is aimed at a wide range of authentication applications. There is only one change in this algorithm - the counter value is incremented not with each subsequent authentication, but at a fixed time interval. [7]

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

K – secret key.

T– timestamp counter, defined as (current time – starting time) / interval.

The original document suggests using a Unix epoch as a starting time reference point, and a 30 second interval. In practice, the starting point and interval can be selected optimally, taking into account safety requirements and other system parameters. Fixed interval means that the current one-time password will be valid for the entire period, but only once. The server is responsible for checking reuse. One of the problems with TOTP is that there is latency in the data network. Because of this, the one-time password used may be invalid, although it was generated in appropriate time window. To reduce the consequences of network delays, a wider window can be used during which the password is considered valid. In this case, the certifying party should also accept the passwords of all previous time steps within the window. However, increasing the window also weakens system security and gives attackers more time. The authors of the algorithm recommend a window size of one time step for sufficient security. [7]

Another problem that can arise with this authentication method is clock desynchronization. Due to the fact that even high-precision timers are not perfect, errors accumulate over time. Servers usually have access to the Internet and are able to synchronize theirs clock with the world standard. Tokens, on the other hand, usually do not have this opportunity. As well as with latency of data transmission, the validation window can reduce the effect of desynchronization. Necessary window can be set for both past and future time steps during which the server will accept the password. Over time, the clock will desynchronize so much that it will go beyond the window. Therefore, for systems designed for very long use without maintenance, an additional resynchronization mechanism is required. However, the authors of the algorithm did not consider this issue. [7]

2.2 HMAC

One of the common methods of protection against message falsification during transmission over a network is to add a special message authentication code. The MAC (Message Authentication Code) standard describes a method for exchanging data and verifying their integrity using a secret key. According to the standard, only parties who know the secret key can generate the same code for the original message, and selection of a suitable code is computationally impracticable [8]. There are a number of algorithms that conform to the MAC standard, each with its own advantages and disadvantages. The general scheme of MAC operation can be presented in the form of a diagram:

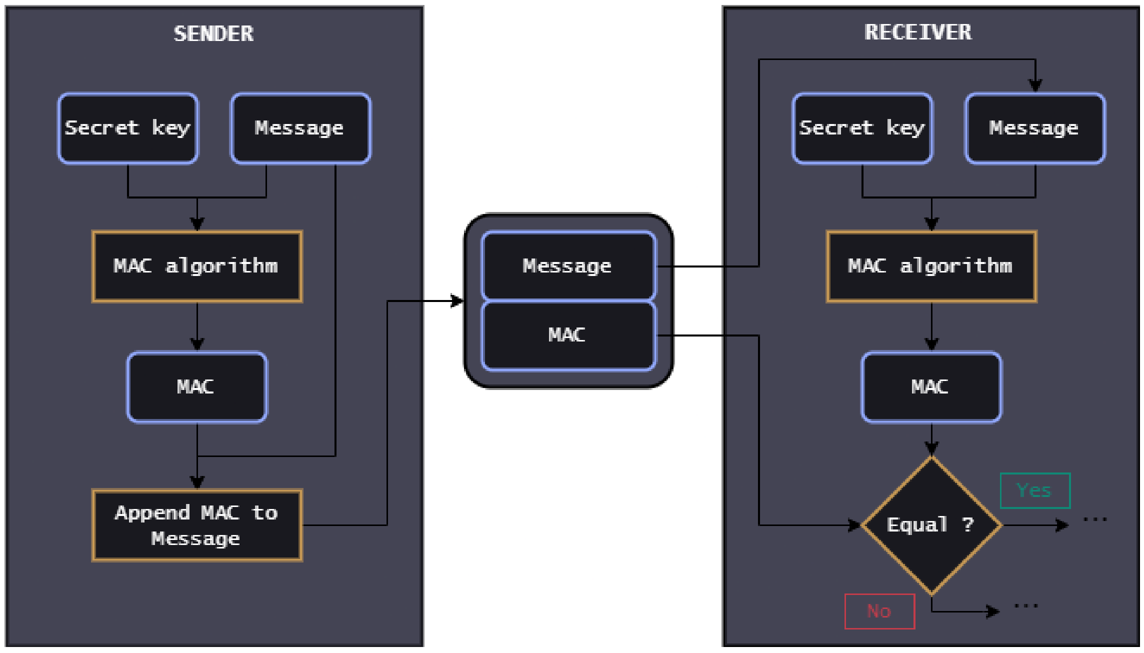


Figure 4. MAC algorithm diagram

If the computed MAC does not match the MAC received with the message, the recipient cannot determine if the message itself has been changed or if its source has been falsified. Since the original version of the HOTP algorithm [6] uses a message authentication code based on a hash function, that is, HMAC, only this particular algorithm will be considered in detail below.

The first types of algorithms to follow the MAC standard used symmetric block ciphers such as DES (Data Encryption Standard) [8]. After some time, the need arose to use already existing hash functions, which were computationally cheaper and were publicly available. Many of the hash functions common at that time could not accept two parameters at once, a key and a message, as required by the MAC. For these purposes, in 1996 H. Krawczyk, M. Bellare and R. Canetti created the HMAC algorithm, and in 1997 published RFC 2104 with a detailed description [9]. Any hash function can be used as the basis of this algorithm. This also simplifies the analysis of cryptographic strength, since it directly depends on the used hash function. Specific versions are usually denoted as HMAC-*, where * is the name of the hash function. For example, HMAC using MD5 is denoted as HMAC-MD5. HMAC is used in protocols such as IPsec (Internet Protocol Security), SSH (Secure Shell Protocol) and TLS (Transport Layer Security). Mathematically, the algorithm can be described by formula (2).

$$HMAC_H(K,m) = H(K \oplus opad, H(K \oplus ipad, m)) \tag{2}$$

H – used cryptographic hash function.

K – secret key, truncated and padded according to rules described below.

m – message.

opad – outer padding, array of same size as hash block filled with 0x5C.

ipad – inner padding, array of same size as hash block filled with 0x36.

Two parameters not specified in formula (2) also depend on the selected hash function: the length of the hash block in bytes, and the length of the output or the hash in bytes. Further, the block length will be denoted as *B*, and the hash length as *L*. Given these data, the algorithm can be described verbally through stages: [9]

1. If the *K* length is greater than *B*, then to obtain a new value, *K* is passed through *H*, that is, *K = H (K)*
2. If the length of *K* at this stage is less than *B*, then *K* is padded with zeros to size *B*
3. The array obtained in stage 2 is XORed with the *ipad* array
4. Message *m* is added to the array obtained at stage 3
5. Apply the hash function *H* to the array obtained in step 4
6. The array obtained in stage 2 is XORed with the *opad* array
7. To the array obtained in step 6, the array obtained in step 5 is added to the end
8. Apply the hash function *H* to the array obtained in step 7 and display the result

Or using a diagram:

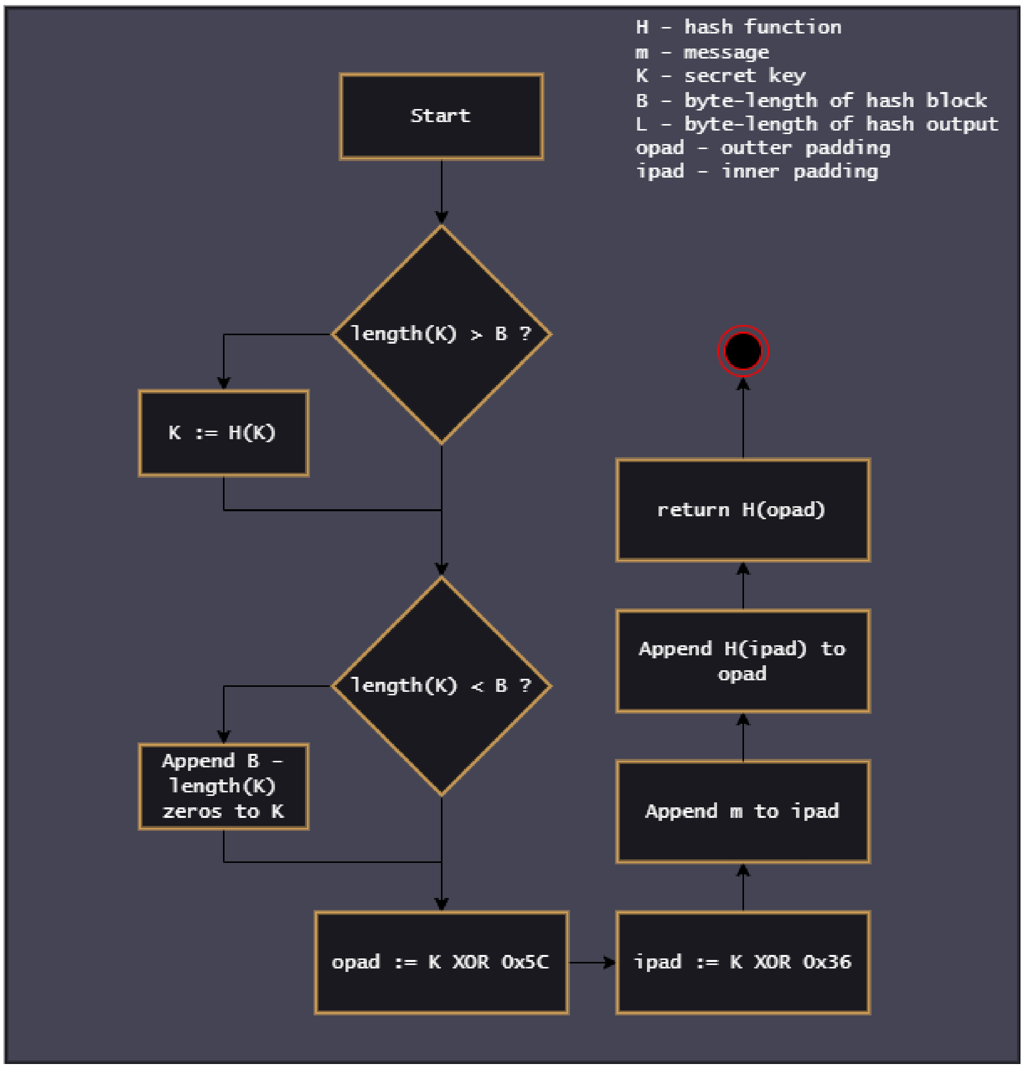


Figure 5. HMAC algorithm diagram

As with other MAC algorithms, all relying parties must know the secret key in advance. The length of the key, for reasons of cryptographic strength, must be no less than the length of the output of the used hash function. For example, when using the SHA-1 function, the length of the *L* output is 160 bits or 20 bytes. This means that the key length must be at least 20 bytes. However, according to the analysis of the original publication, exceeding the hash length *L* will not significantly increase the strength of the algorithm. The overall security of the HMAC directly depends on the hash function *H* used and its cryptographic strength. However, even in the case of a high collision level of *H*, HMAC is considered acceptable for use, since it plays a different role than a pure hash function. [9]. At the time of this writing, the MD5 hash function is considered completely insecure for use in encryption, but RFC 6151 [10] explains why an attack on HMAC-MD5 has no practical value. The original article on HMAC also explains why collision exposure does not pose a direct threat to the algorithm [9].

2.3 Hash functions

The main component of the entire password generation algorithm is a reliable cryptographic hash function. Hash function - a function that takes an array of data of arbitrary length and converts it to another array of data with a fixed length. The main criteria when choosing a particular function are computational complexity, cryptographic strength and size of the output value. To assess the quality and strength of the hash function, the collision metric is used, that is, when different arrays of input data are converted to the same value at the output [11]. A comparison of the most common hash functions will be made and the reasons why a particular function was chosen for use in work will be explained.

In A. Maetouq et al. work [12] 5 hashing algorithms common in 2018 are compared by size characteristics, the number of rounds and the complexity of various types of attacks on them. The list includes such algorithms as MD5, RIPEMD-160, SHA-1, SHA-2 and SHA-3. The first four algorithms are based on the Merkle-Damgård structure. The class of algorithms using this structure process a message of arbitrary length, breaking it into blocks of the same size. Blocks are processed in order, and the result of each subsequent compression depends on the previous one. Thus, an "avalanche effect" is obtained, when the slightest change in the original message causes large differences in the final output of the function. Despite the fact that the SHA-3 algorithm is called SHA, it has nothing to do with the previous ones and uses a different structure, namely sponge construction. The idea of the sponge function is to process a message of arbitrary length and create also an output of the arbitrary length, as required in a specific practical case.

Among the presented hash functions, all except SHA-3 were in practice susceptible to successful attacks. On the other hand, SHA-3 is more difficult to compute in software, so the question arises between efficiency and safety. Since additional methods such as HMAC and TOTP are used to generate one-time passwords, in the context of this work, even outdated hash algorithms, such as SHA-1, can be considered suitable. Based on this, the choice was made in favor of the efficient in software algorithms SHA-1 and SHA-2. MD-5 at the time of writing is considered obsolete, and its use is not recommended. RIPEMD-160 is slower than SHA-1 and not as reliable as SHA-2. Therefore, if security requirements are increased in the future, it would be wiser to go straight to SHA-2. For the prototype device, it was decided to use the SHA-1 hash function. This algorithm does not differ much from SHA-2 and will be considered further in more details. [12]

SHA-1 algorithm consists of 3 stages:

1) Message padding. SHA algorithm works with fixed-length data blocks. In SHA-1, the block length is 512 bits. In addition, the end of the message should contain information about its length. To meet these requirements, the message is aligned following certain rules. First, 1 bit equal to one is added to the message. Then zeros are added until the message length satisfies the equation: L mod 512 == 448, where L is the length. That is, the size of the last message block together with zero padding should be equal to 448 bits. The last 64 bits of the block are reserved for the size of the original message before padding was applied. [13]

2) Setting initial hash. The hash results are also an intermediate state that is used when hashing the subsequent message block. For SHA-1, the internal state is stored as five 32-bit words. Finally, these words can be interpreted as an array of 20 bytes. The initial values of these words are always the same and defined in the algorithm standard. At the beginning of a new hashing process, that is, upon initialization, these values are simply rewritten into words, that is, into the state of the hash object. [13]

3) Hash computation. To calculate the SHA-1 hash, 80 additional 32-bit variables are used, that is, words. Let us mark the array of these words as *W*. The first 16 words are initialized equal to the words of the message block being hashed at this stage: [13]

$$W_t = M_t$$

The rest from 16 to 80 are initialized with the formula (3).

$$W_t = ROL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \tag{3}$$

t – word index.

ROL – circular left shift operation.

Also, 5 working words are used: *a, b, c, d, e*, which are initialized to be equal to the state of the hash object, and the temporary variable *TMP*. Further, the hashing process is divided into rounds. SHA-1 consists of 4 rounds of 20 iterations. Depending on the round, the function *f* and the value of the key *K*, which are defined by the standard, change. At each iteration, the same sequence of actions is performed (4): [13]

$$TMP = ROL^5(a) + f_t(b, c, d) + e + K_t + W_t \tag{4}$$

$e = d$

$d = c$

$c = ROL^{30}(b)$

$b = a$

$a = TMP$

After completing all 80 iterations, the state of the hash object changes according to the formula (5).

$$H_0 = H_0 + a \tag{5}$$

$H_1 = H_1 + b$

$H_2 = H_2 + c$

$H_3 = H_3 + d$

$H_4 = H_4 + e$

H – 32-bit internal state word.

2.4 Development

It was decided to develop the software for the device in C++, trying to avoid pure C as much as possible. However, the field of embedded systems and microcontrollers has its own characteristics, so the choice towards C ++ also brings restrictions on the language capabilities used. Limited microcontroller resources mean that a small and fast library is needed that is convenient to use in a microcontroller development environment. When dealing with systems with limited resources, there are a number of restrictions widely accepted by the community: exceptions, RTTI (Run-time Type Information) and dynamic memory allocation [14]. Sometimes there are no developed C++ compilers for specific microcontrollers, no support for modern language standards or for separate parts of the standard library. One of the goals of the work is the independence of the security token related software from the microcontroller used. This means that a cryptographic library, in addition to containing all the necessary hash functions, HMAC and HOTP algorithms, should not have dependencies on parts of the standard library that use dynamic memory allocation, and use common standard among embedded systems. At the time of this writing, many platforms already support C++14 and partially C++17. Based on compiler support, it was decided to limit itself to the C ++ 11 standard, since this way the vast majority of modern microcontrollers will be covered. [15]

There are several popular C/C++ cryptographic libraries: OpenSSL, WolfSSl, Crypto++ and Botan. Perhaps the most popular is OpenSSL, but it is not designed for microcontrollers and is unnecessarily complex. Crypto++ and Botan use language features that are unsuitable for systems with limited resources (exceptions), so they are also an undesirable option. WolfSSL is as an analogue of OpenSSL for embedded systems, but it is written entirely in C, which is contrary to the original intent. Even if consider using this library, it also turns out to be unnecessarily complex. Because of this, it was decided to implement the minimum required functionality in the form of standalone files with SHA, HMAC and HOTP algorithms.

Since SHA-1 and SHA-2 are almost the same algorithm, except for some parts, they were written as one class using templates. The library implements all versions of the SHA algorithms described in the FIPS 180-4 standard: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 [13]. Thus, in the future it will be very easy to replace SHA-1 with any of the SHA-2 family. Methods init(), update() and final() and static members BLOCK_SIZE and HASH_SIZE can be considered as part of the API (Application Programming Interface) of the hash function class. They are used further in the HMAC function, and they need to be implemented if you add new hash classes, for example, SHA-3. HOTP and HMAC algorithms were invented as independent of the hash function used inside them, so it was also reasonable to implement them using templates. The only standard library dependencies in these implementations are <cstddef> and <cstdint> for size_t and fixed-width integers.

In the default version of the HOTP algorithm, the modulus operation of 10 to the power of 6 was applied to the resulting code. The output of the algorithm thus turned out to be a six-digit number, which is usually represented in decimal form for the convenience of users [6]. In our case, there is no user who needs to use this one-time password manually, so readability is not a priority. Therefore, as a "free" method of improving security, the

HOTP algorithm has been slightly modified. The module operation has been removed and the mask 0x7f is not applied in the most significant byte. The result of the function in our case is a 32-bit number, which will be stored in hex format in the URL, taking 8 characters. Differences from the original version of HOTP can be seen in the C++ code, where the changed parts are simply commented out:

```
uint32_t bin_code =
            (digest[offset]/*& 0x7f*/)  << 24 |
            (digest[offset + 1])        << 16 |
            (digest[offset + 2])        <<  8 |
            (digest[offset + 3]);
return bin_code; // % ipow(10, 6);
```

Figure 6. Modified HOTP

Thanks to templates and the absence of dependencies on the standard library, the implementation is very compact, simple, and can be easily used in any C++ project, be it a microcontroller or a desktop system. The SHA algorithm takes only 1 file, and the API of the class follows the popular standard init() => update() => final(). HOTP and HMAC algorithms, also thanks to templates, represented by only two functions. They can be used with the written SHA library and with any other hash function, if you implement the necessary dependencies. Since these algorithms are widely used in various fields, it was advisable to make them publicly available as separate libraries. The libraries code and examples of use can be found in the author's GitHub repositories[1][2].

[1] https://github.com/nth-eye/SHA-cpp

[2] https://github.com/nth-eye/HOTP-cpp

3 QR code

The device under development should provide user the ability to control the elevator in some convenient way. The simplest and most convenient option is a web application, because in this case the user will not have to install any additional software or connect in another specific way. The web application will be a simple web page with a unique address that is generated synchronously on the device and the server according to the rules described in the **Security token** chapter. The question arises how to pass the URL to the user. There is a technology that is adapted just for the fast and convenient transmission of information in a graphical form - QR code. Once every 30 seconds, a graphic image will be generated for the active URL and displayed on the device screen. All a user will have to do is to scan the code and follow the link saved in it.

3.1 Technology description

The QR code was developed in DensoWave in 1994 as a replacement for the barcodes that were used to identify automotive parts in the factory. The main goals in the development of the new format were reading speed and information capacity. In addition to the large information capacity, an important property of the new format was its noise resistance. The developer of the code, Masahiro Hara, set himself the task of creating not only a capacious and easy-to-read format, but also resistant to damage, dirt and oil spots, which are usual thing in this kind of production. The result of the work was the QR code technology, which they decided to make publicly available [16]. Later, the official ISO standard (International Organization for Standardization) was issued [17]. Today QR code is a common format for URL encoding, product labeling and two-factor authentication.

QR code is a two-dimensional image consisting of black and white modules. These modules are a graphical representation of a binary code that stores encoded information. Black module corresponds to one. There are five types of QR codes: [18]

1) Regular QR – first Model1 version and its improved Model2.

2) Micro QR – smaller size and reliability but greater storage efficiency.

3) iQR – allows to create rectangular shaped codes and has higher data capacity, but currently is not free licensed.

4) SQRC – similar to Model2, but part of data is encrypted with public key and can be read only by reader who possess private key.

5) FrameQR – also similar to Model2, but allows to integrate another content, like image, in area selected by user.

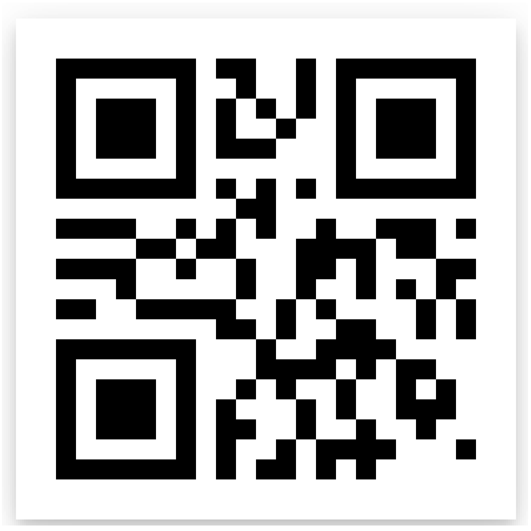


Figure 7. QR code example

In this work, only QR Code Model2 will be considered further. There are several dozen versions of the QR code, from 1 to 40. The version determines its size and capacity. The size of a side of a QR code is determined by the formula 17 + Version * 4. For example, version 12 will look like a 65 by 65 modules square. Besides the version, the adjustable code parameter is ECL (Error Correction Level). There are 4 levels of redundancy - L, M, Q and H, which can restore 7, 15, 25 and 30% of the data, respectively. The capacity of the QR code also depends on the type of data being encoded and is determined from the table. For example, if the data type is numeric only, and ECL is L, then the maximum version of the QR code allows you to save 7089 characters. There are 4 types of data in the regular version of QR code: [18]

1) Numeric – numbers only

2) Alphanumeric – digits, upper-case english alphabet letters, space, ’$’, ’%’, ’*’, ’+’, ’-’, ’.’, ’/’ and ’:’

3) Byte – any data as series of bytes

4) Kanji - Shift JIS character set

3.2 Algorithm

The algorithm for creating a QR code can be divided into 7 stages:

1) Data encoding. First, the data to be encoded is parsed and the required encoding mode is selected. The user pre-selects the ECL he needs. Data encoding into a bit string always starts with adding a data mode indicator: Numeric - 0001, Alphanumeric - 0010, Byte - 0100, Kanji - 1000 Then the so-called CCI (Character Count Indicator) is added. CCI is a bit string that indicates the number of characters to be encoded. The CCI length depends on the version of the QR code and the data mode and is taken from the table. Based on the mode and user selected ECL, the input data is encoded and appended to the bit string. Padding with zeros is applied if necessary, so that the length of the entire string is a multiple of 8. If after the above operations, the length of the bit string is less than defined length in the table for this version and ECL, another padding with 0xec and 0x11 bytes is applied alternately. [19]

2) Error correction encoding. The bit string of data obtained at the previous stage is split into bytes, or the so-called codewords, and divided into blocks in accordance with the Error Correction Table[1]. Error Correction bytes are generated for each data block using the Reed-Solomon method. Next, data bytes and ECC blocks are interleaved. The first byte of data is taken from the data block, then from the second, etc. Then the second byte from the second block, etc. When the data blocks run out, the same procedure is done with error correction blocks. After mixing, padding bits are added if necessary to get the required data length according to the table. [19]

3) Functional patterns and version placement. The necessary functional patterns are then written into the reserved sections of the QR code. The four elements that are present on any QR code version are finder patterns, separators, timing patterns and dark module. Their location and pattern do not change. If the QR code version is greater than or equal

[1] https://www.thonky.com/qr-code-tutorial/error-correction-table

to 2, then alignment patterns are also added. After all functional patterns are placed, information about the QR code version is added. In addition, sections are reserved for the format, that is, an index of the mask used in step 6. All detailed information about patterns can be found in the QR code standard. [19]

4) Data placement. Data is added along with error correction blocks obtained in step 2. Starting from the lower right corner, moving in a zigzag in two rows, a bit string of data is written. If a module is already used for the functional pattern or is reserved, it is simply skipped. Having reached the top, the data continues to be added also in a zigzag in two rows, but from top to bottom. Thus, the entire QR code field is traversed, where the only exception is a column with a vertical timing pattern, it is skipped entirely. [19]

5) Selecting mask and format placement. To improve the readability of the code, so-called masks are used. The QR code standard defines 8 masks that change the color of the modules depending on their location. Masks are designed to counteract repetition of patterns used for functional purposes or excessive uniformity of the QR code. All masks are one by one applied to areas with data and evaluated according to four criteria. The first criterion evaluates the number of consecutive 5 or more modules of the same color in a row and a column. The second evaluates the number of 2x2 squares of the same color. The third evaluates the number of patterns similar to finder patterns. The last criterion evaluates the balance of white and black modules in the entire QR code. After all the masks have been evaluated, the mask with the lowest score is selected, and information about the selected mask is written to the reserved format area. The mask can also be manually selected, although this practice is not recommended by the standard. [19]

6) Applying mask. The mask selected in step 5 is applied to the finished QR code. As in step 5, the mask is applied only to data and error correction blocks, ignoring functional patterns, sections with version and format information. [19]

3.3 Development

As well as with the algorithm for generating one-time passwords, the microcontroller needs a lightweight and fast library for generating QR code in C++. Two open-source libraries were found [1][2]. The first was written in C, used not the safest elements of this language and was partly a copy of the second library. The second, was in the C version and in C++. However, the C ++ version used items requiring dynamic memory allocation. Analysis of ready-made open-source solutions led to the conclusion that it is worth writing own effective solution in C ++ using object-oriented programming methods. The ultimate goal was a minimalistic library that can be easily used on both desktop systems and systems with limited resources. When developing own solution, the same restrictions were imposed on the use of the standard library and on the language standard as in the development of SHA, HMAC and HOTP libraries. The second open-source QR code library was taken as a reference.

Unlike the C libraries, in the developed from scratch the QR code generator was implemented as a class, and the QR code version is selected as a template parameter at compile time. All the necessary data is encapsulated inside the class, and the sizes of the arrays are predefined, that is, the variable-length array is not used. Also, implemented library has a check for the size of the input data, in case it does not fit into the given version of the QR code. The library user needs to manually select the version, ECL and either leave the automatic mask selection, or select it manually. The final solution for generating QR code is very compact, and the API is simple. The library consists of 3 files, which are divided into: tables, miscellaneous functions and QR class, where the main functionality is located. There have also been some small speed tests for comparison with other libraries.

Table 1. Comparison of QR code libraries

Library	Language	Average time with automatic mask [clock_t]	Average time with manual mask 0 [clock_t]
ricmoo/QRCode	C	165	Unavailable

[1] https://github.com/ricmoo/QRCode

[2] https://github.com/nayuki/QR-Code-generator

nayuki/QR-Code-generator	C	192	12
nayuki/QR-Code-generator	C++	242	Unavailable
nth-eye/QR-cpp	C++	160	11

As the unit of measurement, clock_t was used which is defined by the standard as a platform-specific type representing the time used by a processor. The measurements were taken on a laptop running Manjaro Linux operating system with an Intel Core i5-8300H processor. Time was calculated as the average between 10,000 function calls. For the calculation, the templated function was used, presented below.

```
template<size_t N = 1, class Fn, class ...Args>
clock_t measure_time_fn(Fn &&fn, Args &&...args)
{
    clock_t begin = clock();
    for (size_t i = 0; i < N; ++i)
        fn(args...);
    clock_t end = clock();

    return (end - begin) / N;
}
```

Figure 8. Function for time measurement

Although the difference is not very large, the author's library showed the best result. Only two libraries had the ability to manually select a mask. However, automatic mask selection, according to tests, takes approximately 95% of the computational time. From this it follows that for systems with limited resources it makes sense to select the mask manually in advance. If the displayed data is known in advance and does not change, this also does not imply any compromises. The author's library can be found on GitHub [1].

The resulting library was used in the project to generate a QR code containing the URL described at the beginning of the **Security token** chapter. 4 hex characters have been allocated under the elevator ID, which allow addressing 65536 devices. For a one-time

[1] https://github.com/nth-eye/QR-cpp

password, 8 hex characters were allocated, that is, exactly as much as the resulting code of the modified HOTP algorithm takes - 32 bits. Although the entire URL does not reflect the final appearance, it allows to roughly estimate the length of the link. Thanks to this, we can set minimum and maximum restrictions for the version and size of the QR code. The resulting URL for the prototype is 39 characters long. Since the Byte mode will always be used to encode the URL string, the required version can be determined from the table in the Binary / Byte column. This URL is placed in the 3rd version of the QR code, the side of which is 29 modules. When using L or M error correction level, 42 or 53 characters can be stored respectively. When using the L level, there are 14 free characters left, which seems like enough headroom in case future need to make edits in the URL format. Based on these data, the minimum and maximum version of the QR code was determined to be 3. All these parameters will be further taken into account when choosing a screen with the resolution required for readability.

4 Hardware

The main part of the project is directly assembling the physical prototype and developing the embedded software. To create a working prototype, we need to select the appropriate components. The choice of components is based not only on their technical characteristics, but also on the price, availability, prevalence of information, tools for work and manufacturer support. In this work, the hardware that was considered for use will be described. The selected components and their characteristics will be described in more detail. At the end, the creation process and features of the firmware for the final version of the prototype will also be described. The problems that have arisen during the development of the program and assembly of the device will be partially touched upon.

4.1 Main MCU

First of all, let us consider the conditions in which the device is supposed to be used. The first thing to notice is that the device must work for a long time without human service. Due to the specificities of the security token technology based on TOTP, the device clock must be synchronized with the server clock in advance before installation. During the development of the prototype, it was not planned to use any means of communication with the server after installation. All this means that if the power supply stops, then device will have to be dismantled and synchronized again. If power is supplied to the device in the field, then the synchronization and configuration process will be carried out in the field. All this is not very comfortable for developers and staff who, in theory, will install these devices. The rational solution was to use a small portable power supply stored with the device in the same case. Since the prototype under development will be powered by a battery, a very energy efficient microcontroller is needed. When choosing a microcontroller, it was also necessary to take into account the need for a minimum peripheral for communication with the screen and other modules: I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) and UART (Universal Asynchronous receiver-transmitter).

Originally, before a clear work plan, it was assumed that the device would need to communicate with the server via a wireless protocol. An ESP32[1] board with built-in Wi-Fi and Bluetooth would work well for this. After some time, the idea of communication with the server was put aside, and with it this board. It became clear that an affordable standalone microcontroller was needed for operation without unnecessary modules that were not planned to be used. There are many large manufacturers of affordable microcontrollers: STM, Nordic Semiconductor, NXP, Atmel, Texas Instruments, and others. Among all the others, one can single out STM, which has a wide range of affordable and energy-efficient microcontrollers, and a developed set of tools for working with them. Since author also had experience with microcontrollers from this manufacturer, the available options were studied in detail. In total, STM has 4 classes of modern 32-bit microcontrollers, one of them is ultra-low power microcontrollers [2].

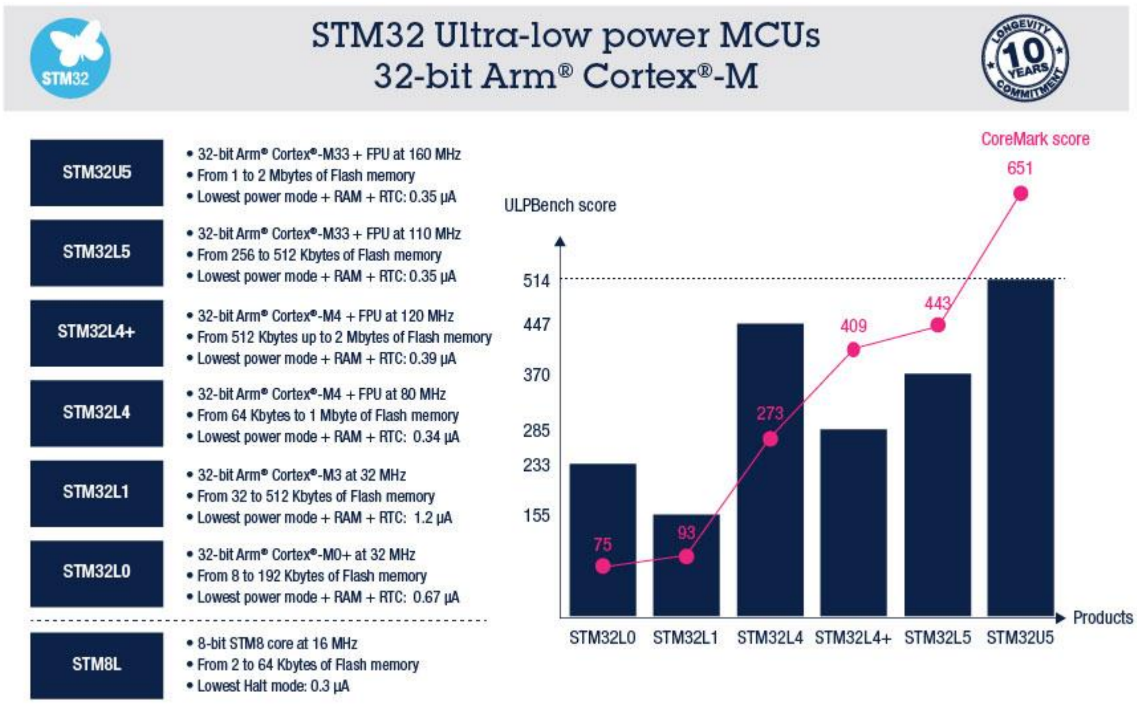


Figure 9. Comparison of STM32 MCU's

Within this class, there are 5 microcontrollers of different levels of performance and savings. The STM32L4 microcontroller was chosen as the balance between computational power and price. However, to create a prototype, not just a microcontroller,

[1] https://www.mouser.ee/ProductDetail/Espressif-Systems/ESP32-DevKitC-32UE

[2] https://www.st.com/en/microcontrollers-microprocessor

but a full-fledged development board is needed. STM also has out-of-the-box development platforms - NUCLEO series. In order to save money, the board of the smallest format was chosen - NUCLEO-L432KC[1].

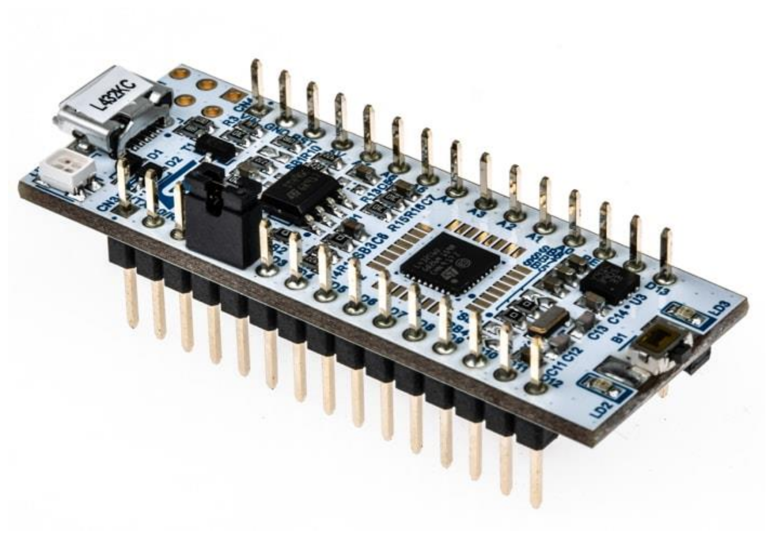


Figure 10. NUCLEO-L432KC board[2]

This board has all the necessary peripherals, 256kb flash memory and 64kb SRAM (Static Random-access Memory), which is more than enough for the task solved in this work. Also, the board is made with support for Arduino Nano connectivity, which makes it very convenient to use when working with the Arduino framework. One of the interesting points of this class of microcontrollers is a rich selection of operating modes. Specifically, this microcontroller has as many as 10 different power modes, which can be fine-tuned for a variety of conditions. [20]

4.2 Display

For maximum energy savings, all selected components must be energy efficient. The main component of the device, in addition to the microcontroller, is the screen on which the QR code is displayed. Originally were considered LCD (Liquid-crystal Display) and OLED (Organic Light-emitting Diode) screens. LCD display is the most common solution that can be found due to its low price. OLED, on the other hand, is more

[1] https://www.mouser.ee/ProductDetail/STMicroelectronics/NUCLEO-L432KC

[2] https://sa.rsdelivers.com/product/stmicroelectronics/nucleo-l432kc/stmicroelectronics-stm32-nucleo-32-mcu/1438574

expensive, not so common in the microcontroller environment. OLED technology is fundamentally different from LCD and does not require an additional back panel for backlighting. This makes OLED displays much more energy efficient. Overall, OLED also outperforms LCD in many other aspects [21]. At first, the SSD1306-based Adafruit[1] display with a resolution of 128x64 pixels seemed to be a good option. However, calculations of the required size of the QR code showed that its size may not be enough. In addition, the above-described displays actively consume current, and in our case, the image on the screen is static most of the time. After a little market analysis, another solution was found - eInk (Electronic Ink) technology.

The main advantage of eInk technology is battery life because energy is consumed only when the displayed information changes, while others consume energy constantly. One of the disadvantages of the technology is the long updating of information, which does not allow displaying complex graphic elements on the screen. For example, for an interactive interface, but in this work there is no need for such functionality. Also, the disadvantage can be considered not such a rich choice and the relative high cost of even small displays. Nevertheless, a more or less inexpensive Adafruit[2] display with a resolution of 200x200 pixels was found, which would be enough to display a QR code of version 3 with a granularity of up to 6 pixels per module.

[1] https://www.adafruit.com/product/326

[2] https://www.adafruit.com/product/4196

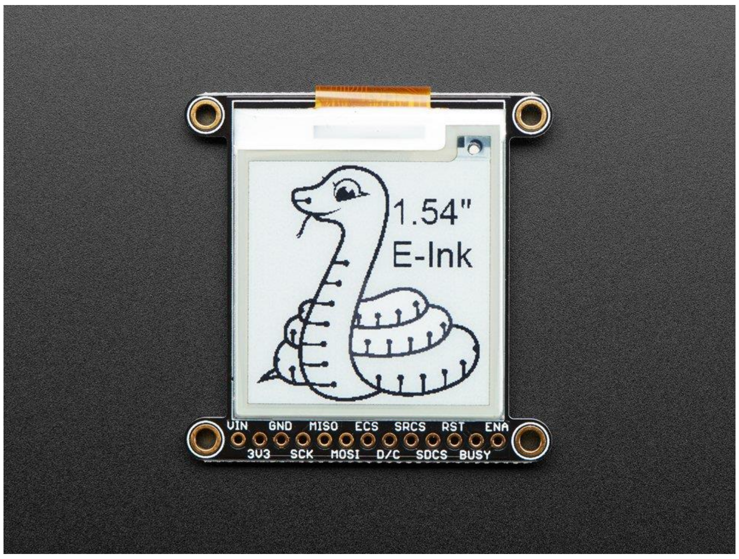


Figure 11. Adafruit 1.54" Monochrome eInk display

The Adafruit 1.54" SSD1608 is a monochrome eInk display that uses SPI to communicate with the microcontroller. Adafruit also has tri-color displays of the same series, but the image refresh on them takes about 15 seconds. The black and white version takes 2 seconds to update, so it was chosen. According to the specification, the consumed operating current is 2 mA, and the sleep current is approximately 30 µA [22]. This display has quite a lot of pins, but such a number is justified by its rich capabilities. Besides the SPI pins, it has separate lines for the onboard SRAM and for the SD (Secure Digital) card. That is, when using this display, it is not necessary to waste the memory of the main microcontroller, you can directly record the displayed material into an SRAM or SD card, although an SD card is not included. There is also a BUSY pin to effectively determine the end of the operation. BUSY pin may not be used, but then you have to use the polling method for a certain number of seconds. Also, were considered the display of the manufacturer Cypress[1], which was suitable for the price and has a resolution of 264x176 pixels. The advantage of this display was the built-in motion sensors, temperature sensors and a microphone. These functions could be useful in the further development of the project for collecting statistics or connecting voice commands. In the end, it was decided to purchase both of these displays for testing and then select the most suitable option for the prototype.

---

[1] https://www.cypress.com/documentation/development-kitsboards/e-ink-display-shield-board-cy8ckit-028-epd

4.3 ToF sensor

At the stage of the initial selection of components, when LCD and OLED displays were considered, the idea was to use an energy efficient sensor for the presence of people to turn on the screen only when it was really needed. A sensor based on Time-of-Flight technology - VL53L1X, or rather its variant in the VL53L1X-SATEL[1].breakout board format was chosen very quickly. ToF technology is a very accurate and efficient method for estimating short distances by calculating the difference between the moment a signal is emitted and the moment it receives a reflection from an object. The selected sensor is capable of detecting a distance of up to 4 meters with a viewing angle of 27 degrees when using long range mode. The manufacturer supplies two low-level driver sets for this sensor. One is a usual one with an extended and convenient API for a variety of use cases, the second is an ultra light driver for maximum efficiency and minimum size. As a communication protocol, the sensor uses I2C up to 400 KHz, so it is easy to use with any microcontroller in a system of any complexity. Average current consumption in inactive mode is about 10 μA, and at the moment of measurement is about 16 mA. [23]

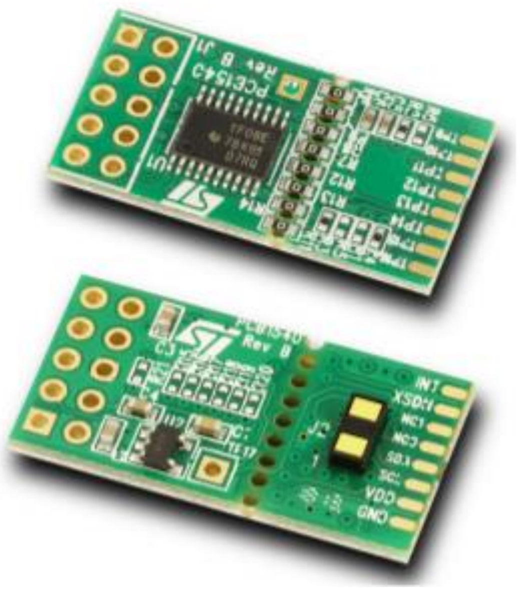


Figure 12. VL53L1X-SATEL ToF sensor

The author of this work has previously worked with this sensor, so the integration process into the project would not create difficulties. The sensor itself was already available from

[1] https://www.st.com/en/evaluation-tools/vl53l1x-satel.html

the supervisor of the thesis. Initially, the idea was that if the sensor does not detect a person for a certain period of time, then the screen will be turned off in order to save energy. By detection we mean a change in the measured distance. When detecting a person, the screen would turn on and the QR code would be updated. However, after analysis of the current consumption, it turned out that this sensor alone consumes more than the eInk display. For this reason, the ToF idea was set aside.

Despite the fact that in the end in this work it was decided to abandon the ToF sensor, this sensor can be used during further development. For example, to collect statistics or as a motion detector to control other things. Therefore, it was decided to test the sensor. The platform-specific part of the driver was written and tested together with the microcontroller. The result was successful. The written driver layer for the STM32L4 microcontroller allows to easily connect a sensor to the project and will be described in more detail in the **Embedded Software** chapter.

4.4 Assembly and schematics

To assemble the prototype, it was necessary to connect the microcontroller to the screen. Not all pins of the screen were used, but only the necessary. SPI pins for communication with the screen itself and built-in SRAM and BUSY pin for detecting the completion of the operation. In the early stages of the project, a ToF sensor was also tested, which uses the I2C protocol for communication. The screen and sensor were connected to the NUCLEO-L432KC according to this scheme:

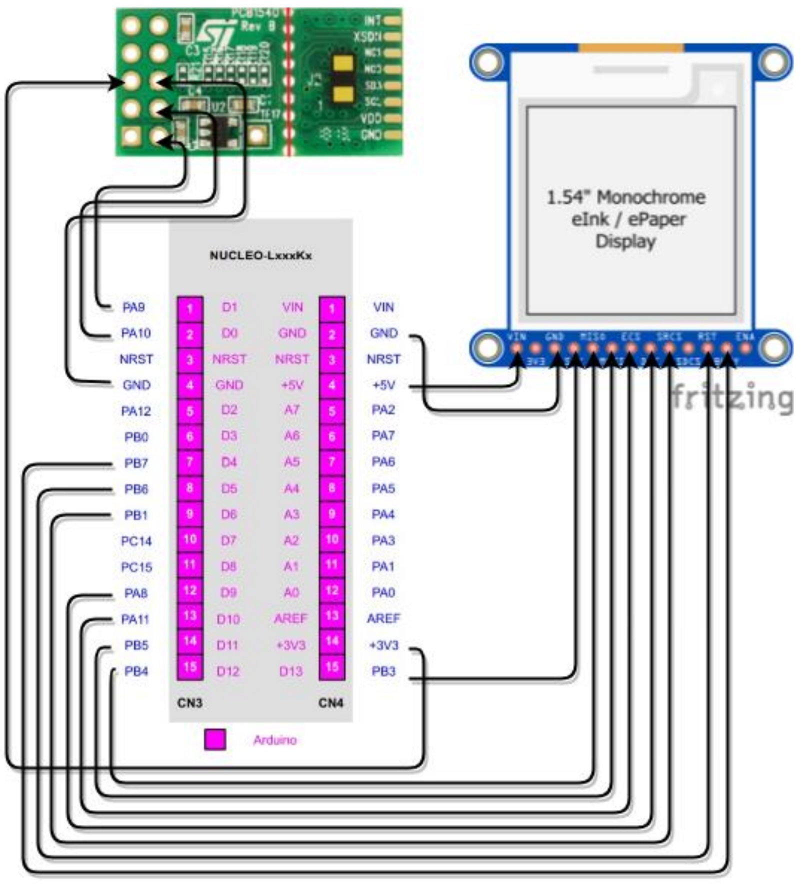


Figure 13. Wiring schematic

2D screen model was taken from the Fritzing[1] program from the Adafruit[2] library. Pictures of the microcontroller circuit and ToF sensors were taken from the documentation from the manufacturer STMicroelectronics[3][4]. The schema was connected in the draw.io[5] web app.

[1] https://github.com/fritzing/fritzing-app

[2] https://github.com/adafruit/Fritzing-Library

[3] https://www.st.com/resource/en/data_brief/vl53l1x-satel.pdf

[4] https://www.st.com/resource/en/user_manual/dm00231744-stm32-nucleo32-boards-mb1180-stmicroelectronics.pdf

[5] https://app.diagrams.net/

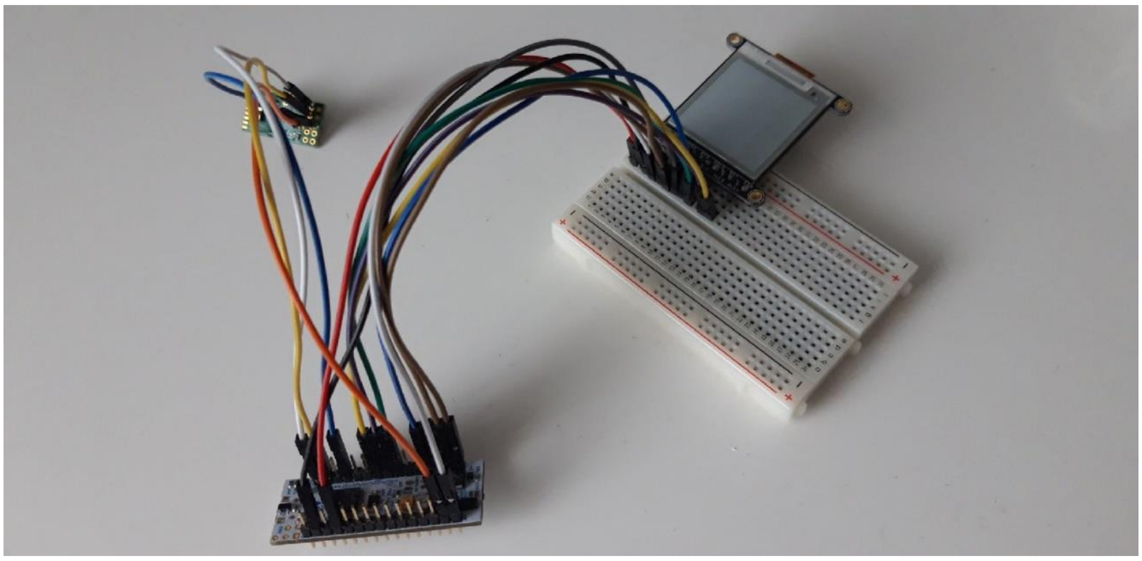


Figure 14. Assembled prototype

This is what the prototype looked like with a connected screen and a ToF sensor. In the final version the prototype, the ToF sensor was not used, and the microcontroller was only connected with the screen.

4.5 Embedded software

The embedded software is usually extremely platform-specific. Therefore, in this part, the work was to correctly initialize hardware, add libraries, configure the operating mode and create the main program flow. The QR code, HOTP and SHA libraries written for this project were created taking into account the use on a large number of microcontrollers and with a minimum number of dependencies. For this reason, using them with the STM32L4 did not require any changes. The link to the GitHub repository with the project is attached in Appendix 2. This is how the flow of the main program looks like in pseudocode:

```
initialize_hardware()
initialize_variables()
WHILE TRUE:
    code = hotp(sha_1, key, counter++)
    url = create_url(code)
    encode_qr_code(url, ecc)
    display_qr_code()
    go_to_sleep()
```

Figure 15. Main program pseudocode

Setting up the microcontroller and initializing peripherals is usually a routine process. The ST manufacturer has created a special program to automatically generate this necessary code - STM32CubeMX[1]. In it, the user can create a template for the project using the graphical interface. Using this program, such microcontroller elements as UART, I2C, SPI and RTC (Real-time Clock) were configured. During development, the VSCode[2] editor was also used along with the stm32-for-vscode[3]and Cortex-Debug[4] extensions. To download the program to the board, we used a utility from the STMicroelectronics[5] manufacturer.

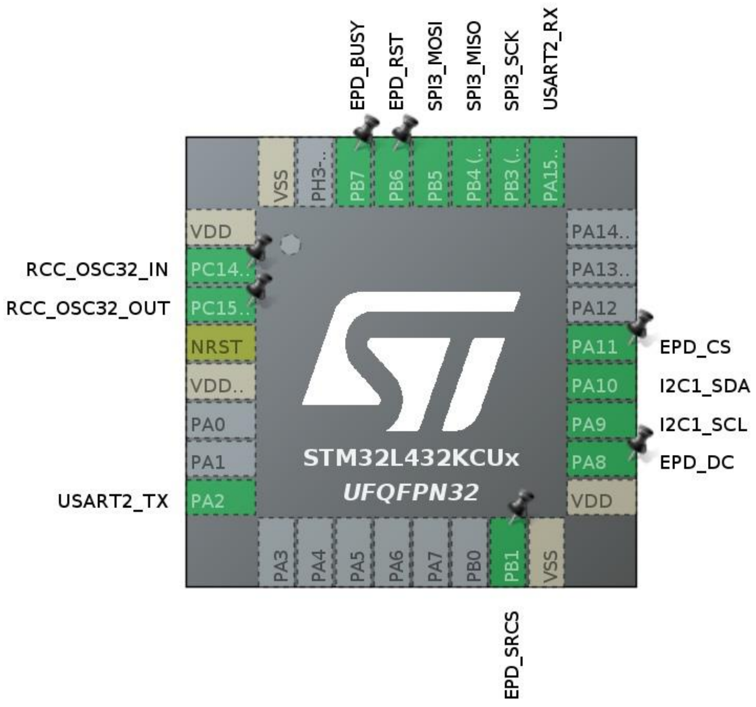


Figure 16. STM32CubeMX used periphery

This microcontroller has no default input-output methods. Therefore, for communication and testing, Virtual Com Port was configured via the built-in USB (Universal Serial Bus)

[1] https://www.st.com/en/development-tools/stm32cubemx.html

[2] https://code.visualstudio.com/

[3] https://marketplace.visualstudio.com/items?itemName=bmd.stm32-for-vscode

[4] https://marketplace.visualstudio.com/items?itemName=marus25.cortex-debug

[5] https://www.st.com/en/development-tools/st-link-v2.html

UART2 interface. The settings were also set in the CubeMX program. To simplify the debugging process, the functions necessary for printf were implemented.

```
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)

extern "C" PUTCHAR_PROTOTYPE
{
    HAL_UART_Transmit(&huart2, (uint8_t*) &ch, 1, 0xFFFF);
    return ch;
}
extern "C" int _write(int file, char *ptr, int len)
{
    for (int idx = 0; idx < len; idx++)
        fputc(*ptr++, (FILE*) file);
    return len;
}
```

Figure 17. Functions necessary for debugging

An ultra light driver[1] from the manufacturer was used to connect the VL53L1X sensor. To communicate with the sensor via I2C, it was necessary to implement basic functions that are well isolated from the rest of the driver The STM32L4 has two I2C interfaces located on different pins. The project used I2C1 and was configured at a speed of 100 kHz in the CubeMX program when generating the project.

There is an Arduino-based library[2] for the Adafruit eInk screens. At the first time when the screen was connected to a microcontroller using Arduino, it did not work. At that time, it was not known if there was any error in the connection itself or inside the library. After that it was decided to try the second eInk display CY8CKIT-028-EPD. Unfortunately, the drivers[3] for it were platform-specific and were intended to work with microcontrollers from the same company. An attempt was made to rewrite the driver for STM32. Results were negative and second screen did not work. Subsequently, work continued on the Adafruit library. During the experiments, it turned out that there was an error in the SPI configuration inside the library. The screen refused to work if a non-default SPI3 interface of the STM32L4 microcontroller was used for communication. Since it was planned to use the STM32 native code generated with CubeMX for

[1] https://www.st.com/en/embedded-software/stsw-img009.html

[2] https://github.com/adafruit/Adafruit_EPD

[3] https://sdkdocs.cypress.com/html/psoc6-with-anycloud/en/latest/api/psoc-middleware/kits-support/CY8CKIT-028-EPD/index.html

programming, the task was to rewrite the Adafruit library. The goal was to isolate the platform-specific functions of the library (for communicating via SPI) so that it can be used with any microcontroller and without dependence on the Arduino framework. A side goal was to figure out the SPI setup, so that, if desired, use not only the default interface in Arduino. In the end, the library was successfully rewritten. Initial tests showed that the library[1] is working, but due to the large amount of code, the full functionality has not been verified.

Next, there was work with timers. HOTP and TOTP algorithms are practically the same, the only difference is in the source of the counter value. For the TOTP algorithm, there must be a timer and starting point for timing. The STM32L4 microcontroller has a built-in timer for this purpose - RTC. A low-speed external oscillator with a frequency of 32.768 kHz was chosen as the most accurate source. The timer was also set to an interval of 30 seconds using the CubeMX program.

[1] https://github.com/nth-eye/AdafruitEPD-cpp

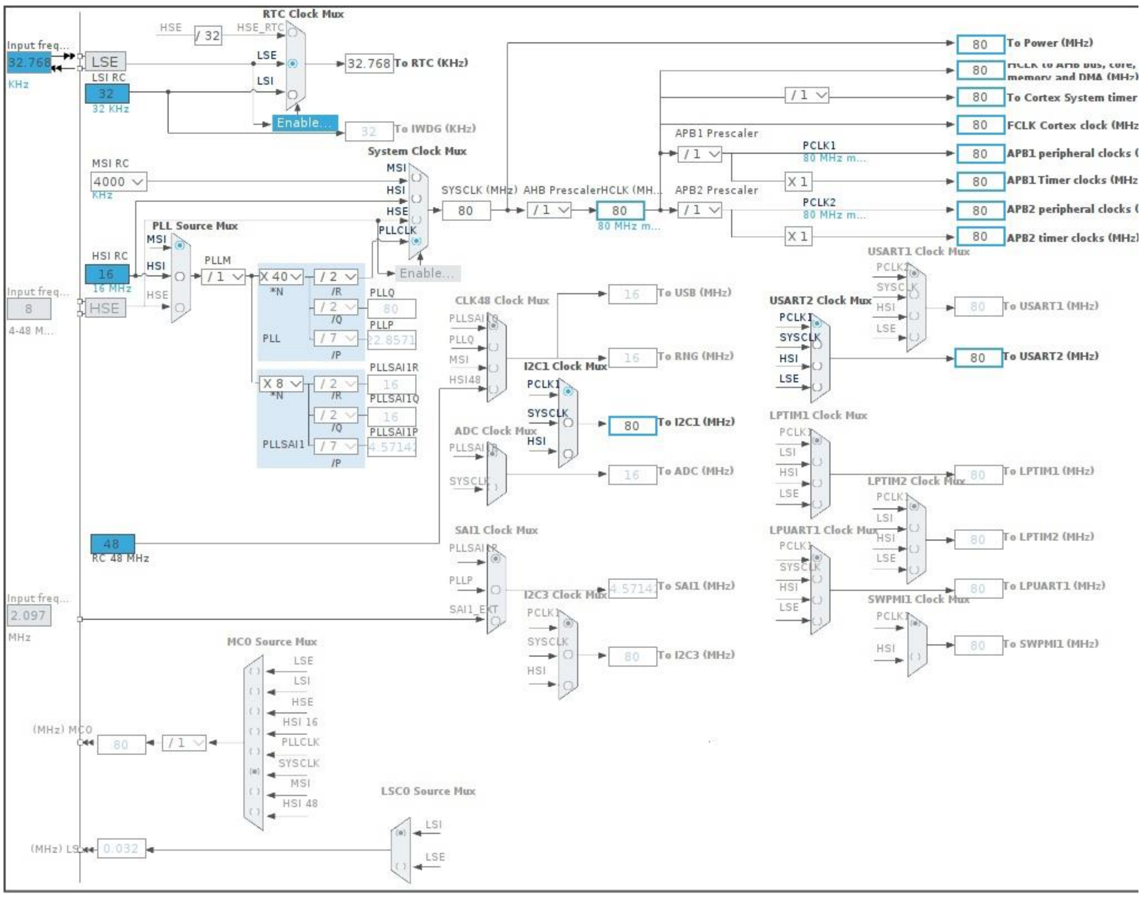
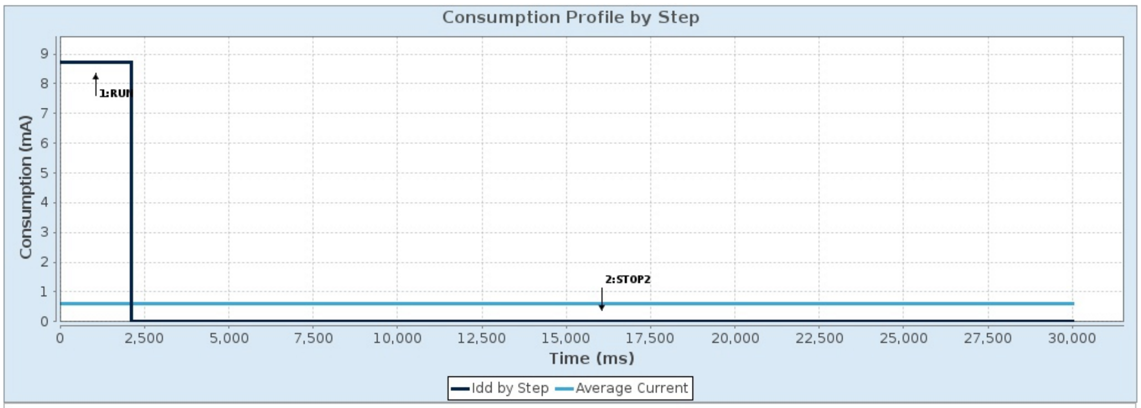


Figure 18. STM32CubeMX clocks configuration

To fix the starting point of reference, it was decided to record the preprocessor time at the start of the program. There was no other way to record a more or less accurate moment in time without specialized software. The download time of the program to the microcontroller is within a few seconds, so this should not greatly affect the accuracy. To extract the date and time of compilation, the corresponding preprocessor macros __DATE__ and __TIME__ were used. The values of these macros are parsed at the beginning of the program, the data is written to the necessary structures, and the timer value is overwritten.



Sequence Time / Ta Max	30 s / 103.98 °C	Average Consumption	610.69 µA
Battery Life Estimation	7 months, 17 days, 15 hours	Average DMIPS	100 DMIPS

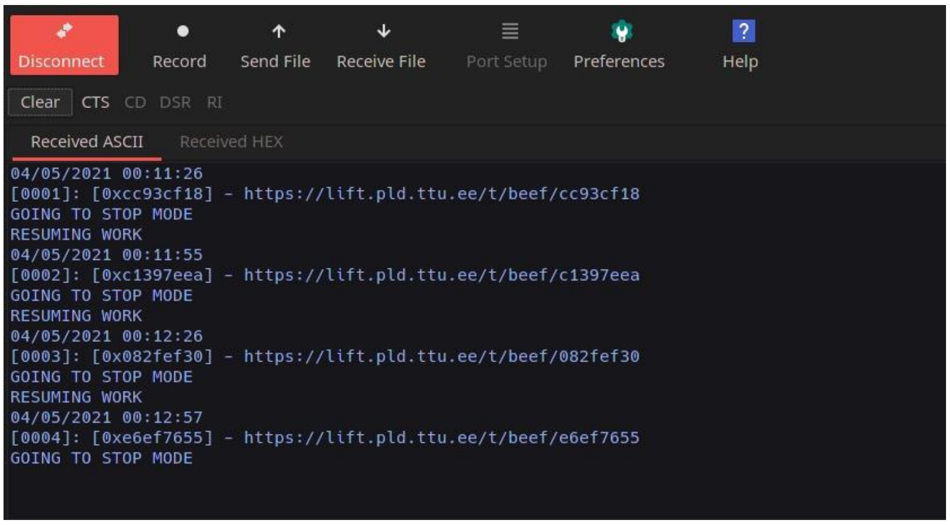
Figure 19. STM32CubeMX current consumption

In the end, theoretical calculations of current consumption were carried out. Using the STM32CubeMx program, an approximate current consumption was calculated, taking into account that the microcontroller will be in active mode for about 2.5 seconds (refresh the screen), and 28 seconds in one of the energy-saving STOP modes. At the same time, the program calculated the operating time of the device, assuming power supply from Li-SOCL2 (A3400) with a capacity of 3400 mAh. The expected operating time was 7 months and 17 days. In real life, there will certainly be additional losses, but we can say that the expected operating time with this battery is at least 7 months.

5 Testing

All algorithms used in the project have their own official standards, which describe in detail all the rules, possible corner cases, and usually test vectors are also provided. The libraries written for the project were created following these standards. For example, when implementing HMAC and HOTP from RFC 4226 [6], vectors were taken for both algorithms with the SHA-1 hash function. For algorithms of the SHA family, test vectors were taken from the specified link[1]. All these small tests are stored in the main.cpp files in the author's repositories. All of them were passed, and there were no deviations during the work.

Testing of the final version of the program, that is, the built-in software together with the libraries, was carried out on a physical sample. When the prototype was assembled, the device was configured to communicate with the computer via the UART protocol. In the STM32L4 microcontroller, you can configure the so-called Virtual Com Port[2]via the built-in USB UART functionality, so no additional connections were needed.



```
04/05/2021 00:11:26
[0001]: [0xcc93cf18] - https://lift.pld.ttu.ee/t/beef/cc93cf18
GOING TO STOP MODE
RESUMING WORK
04/05/2021 00:11:55
[0002]: [0xc1397eea] - https://lift.pld.ttu.ee/t/beef/c1397eea
GOING TO STOP MODE
RESUMING WORK
04/05/2021 00:12:26
[0003]: [0x082fef30] - https://lift.pld.ttu.ee/t/beef/082fef30
GOING TO STOP MODE
RESUMING WORK
04/05/2021 00:12:57
[0004]: [0xe6ef7655] - https://lift.pld.ttu.ee/t/beef/e6ef7655
GOING TO STOP MODE
```

Figure 20. Test with serial terminal

[1] https://www.di-mgt.com.au/sha_testvectors.html

[2] https://www.st.com/resource/en/user_manual/dm00231744-stm32-nucleo32-boards-mb1180-stmicroelectronics.pdf

Development and testing took place on a computer with the Manjaro operating system based on Arch Linux. On the computer side, a moserial terminal[1] was used to communicate with the device. The 30 second timer and the transition to sleep mode were tested. Information appeared on the terminal every 30 seconds, and the QR code was updated on the device screen.

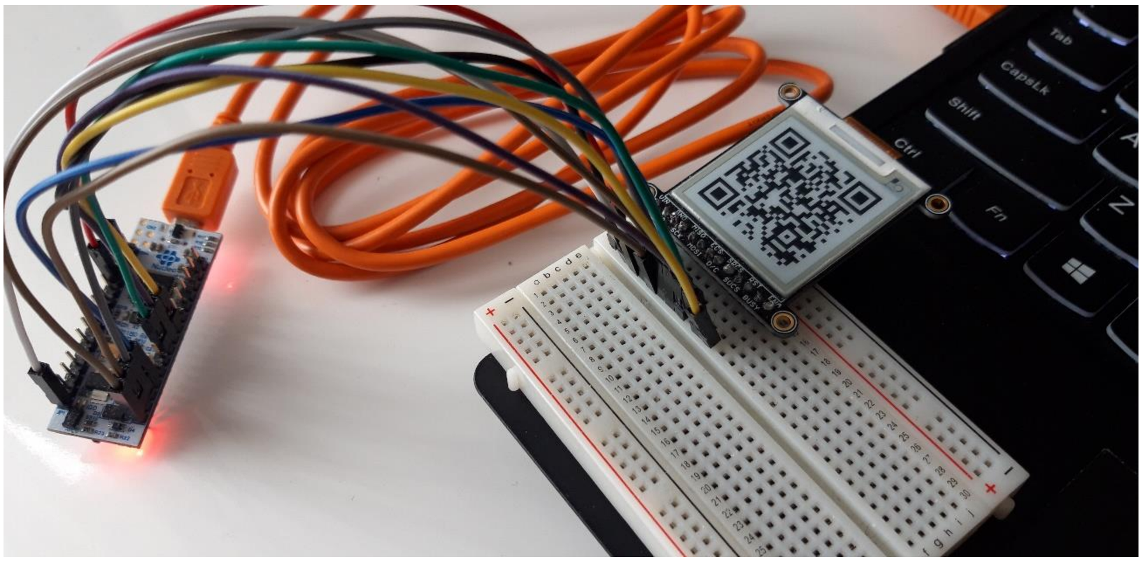


Figure 21. Working prototype

To check synchronous operation of the device and to simulate real conditions, a server was needed. The supervisor of the thesis set up a simple server where the open source TOTP library[2] was used. During the tests, it turned out that the generated one-time passwords between the device and the server do not match. A detailed study of the third-party TOTP library showed that Base64 was used internally to encode the message. This functionality is not provided by the standards describing the HOTP and TOTP algorithms, which means that the error was not in the library used on the device. Necessary changes were made on the server side, after which the one-time passwords began to match. At the time of this writing, tests in real conditions were not carried out, but the results with a demo server suggest that there should be no further problems.

[1] https://wiki.gnome.org/action/show/Apps/Moserial?action=show&redirect=moserial

[2] https://github.com/samdjstevens/java-totp

6 Analysis

In this work, many different topics were considered: security, QR code, many algorithms, hardware, and software. The initial goal was to implement a workable prototype, which can serve as the basis for further development of the project. Despite the working end result, during the work on the project there were many problems that could have been prevented by a more thorough analysis of the components before their final selection. Since this work is just a prototype, there is a number of things that can be added with further development. This is especially true of parts of the project that were not touched upon in this work: server, specialized software, case, design, and research of the opinions of potential users.

6.1 Problems encountered

Many of arisen problems were minor and eventually resolved. During testing of the TOTP library, discrepancies with open-source Java library used by server application were identified. Although the error in the implementation was in a third-party library, it was important to solve this problem by overwriting original class with inheritance.

The second serious problem was a questionable performance of the Adafruit library for the eInk display. This problem could have been avoided with more careful analysis during selection of components. It was necessary to check not only the screens themselves and their characteristics, but also the drivers and libraries supplied to them from manufacturers.

A serious unresolved issue was the initial synchronization of the device and the server, which is also described in more details in the chapter **Embedded Software**. Not much is known about how PIN calculators and other security token devices work. Since too little has been found about how they are initially programmed and synchronized with the server, no solution has been found in such a short time.

6.2 Possible improvements

A lot of software has been written for this project. Some code is platform specific, some is more general, such as for example QR code library. First of all, the rewritten library for managing the eInk display can be improved. Currently only the basic functionality for a specific display has been transferred from the Adafruit library, which was used in this work. When the rest of the graphics library would be ported, it will be possible to display on the screen non-trivial graphic elements: text, informative symbols, icons, or a trademark. Secondly, not all software has been thoroughly tested. Creating own unit tests or integrating ready-made libraries, for example, GoogleTest[1], would increase confidence in the reliability of the software.

Small improvements can be made to the firmware too. Tracking the battery charge level could be added to the firmware of the microcontroller. If the charge level would be below a certain level, the device would indicate this. One idea is to send battery level as additional 2 symbols in a URL. This would facilitate the maintenance of the device.

6.3 Further development

This work did not consider the server, which is an integral part of the project as a whole. Although a demo server was created for the purpose of testing by the supervisor, for practical use a web application for the user is also required. Further development of the project means, first of all, the development of the server and the creation of a database where secret keys and device IDs will be stored. The topic of security and server architecture should be discussed together with experts who deal with these issues. The next topic that requires research and competent implementation is device and server synchronization. The process of initial synchronization of the device and server clocks has not been studied due to the lack of the necessary time. It is assumed that this requires a specialized program that simultaneously updates the firmware of the device and saves the necessary data to the server database. In this way, the randomly generated secret key and timing reference for synchronization will be automatically and securely stored. In this work, the possibility of desynchronizing the device's clock some time after the initial synchronization was also mentioned. It is necessary to decide whether it is rational to add the connection with the server for regular synchronization, for example, via Wi-Fi

[1] https://github.com/google/googletest

once a week. It is possible that the economical aspect and energy costs will be too high, and it is better to synchronize the device manually after the battery is depleted.

The device itself, in addition to the basic functionality in the form of a display, can be further supplemented with other capabilities. The first possible addition is an NFC (Near-field Communication) tag, which will play the same role as a QR code. In this case, the user can choose what is easier to use. The second addition that will possibly change the elevator control cardinally is voice commands. By integrating a microphone into the device and creating the necessary software for recognizing simple voice commands, it would be even easier to control the elevator. Other sensors can be added to collect statistics, for example, the same ToF, or equip the device with a video camera to conduct security activities at the elevator area. However, in this case, power consumption and power supply should be seriously revised, and a high-speed communication method will be required to transfer video to the server.

Currently, the prototype also does not have any special case, which means that it is not suitable for use in real conditions. The form factor and the required level of protection for the electronics should be taken into account in order to create a complete device that is optimal in price and quality. Aesthetics is also important in case appearance, which means design is another thing to consider. The author of the work does not have the skills of 3D modeling and is not a professional designer, therefore he is not able to assess the complexity of creation and the cost of these parts. If consider this project not as a prototype, but a full-fledged product, then perhaps the next step is to create custom circuit for a printed circuit board. NUCLEO development board is well suited for developing prototypes that are convenient to experiment with, but the final product should be as cost effective as possible. With this, the issues of case design become even more urgent.

7 Summary

The goal of this work was to create an energy efficient device that generates a temporarily active URL and displays it as a QR code on the screen. Additionally, the possibility of using a ToF sensor was considered. The main stages of the work were the selection of components, their connection, and the creation of the program. In the course of the work, several algorithms were implemented to generate a one-time password contained in the URL. TOTP was chosen as the algorithm on which the whole idea of the dynamically generated link is built. The libraries written during the work were integrated into the program for the STM32L4 microcontroller. The result of the work is a device that generates a new URL every 30 seconds, displays it on the screen in the form of a QR code, and goes into sleep mode to save energy.

The goals set for this work were achieved. The issue of how to synchronize the device with the server has not been resolved due to limited time.

References

[1] "Wikipedia: OTP," 7 May 2021. [Online]. Available: https://en.wikipedia.org/wiki/One-time_password. [Accessed 7 May 2021].

[2] "Wikipedia: Security token," 19 April 2021. [Online]. Available: https://en.wikipedia.org/wiki/Security_token. [Accessed 7 May 2021].

[3] M. Patel, "Understanding the Security Features of RSA Tokens," 19 June 2018. [Online]. Available: https://thefintechtimes.com/understanding-the-security-features-of-rsa-tokens/. [Accessed 7 May 2021].

[4] N. Haller, "The S/KEY One-Time Password System," February 1995. [Online]. Available: https://tools.ietf.org/html/rfc1760. [Accessed 7 May 2021].

[5] G. B. Dr. K. Mohan Kumar, "Comparative Study on One-Time Password Algorithms," August 2018. [Online]. Available: https://ijcsmc.com/docs/papers/August2018/V7I8201811.pdf. [Accessed 7 May 2021].

[6] M. B. F. H. D. N. O. R. D. M'Raihi, "HOTP: An HMAC-Based One-Time Password Algorithm," December 2005. [Online]. Available: https://tools.ietf.org/html/rfc4226. [Accessed 7 May 2021].

[7] S. M. M. P. J. R. D. M'Raihi, "TOTP: Time-Based One-Time Password Algorithm," May 2011. [Online]. Available: https://tools.ietf.org/html/rfc6238. [Accessed 7 May 2021].

[8] D. Liu, Next Generation SSH2 Implementation, ScienceDirect, 2009.

[9] M. B. R. C. H. Krawczyk, "HMAC: Keyed-Hashing for Message Authentication," February 1997. [Online]. Available: https://tools.ietf.org/html/rfc2104. [Accessed 7 May 2021].

[10] L. C. S. Turner, "Updated Security Considerations for," March 2011. [Online]. Available: https://tools.ietf.org/html/rfc6151. [Accessed 7 May 2021].

[11] Tutorialspoint, "Cryptography Hash functions," [Online]. Available: https://www.tutorialspoint.com/cryptography/cryptography_hash_functions. [Accessed 7 May 2021].

[12] S. M. D. N. A. A. N. M. N. N. A. S. H. A. A. Maetouq, "Comparison of Hash Function Algorithms Against Attacks," 2018. [Online]. Available: https://thesai.org/Downloads/Volume9No8/Paper_13-Comparison_of_Hash_Function_Algorithms.pdf. [Accessed 7 May 2021].

[13] National Institute of Standards and Technology, "Secure Hash Standard," August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. [Accessed 7 May 2021].

[14] B. Stroustrup, "What can C++ do for embedded systems developers?," NDC Conferences, 24 September 2018. [Online]. Available: https://www.youtube.com/watch?v=VoHOLDdfDhk&ab_channel=NDCConferences. [Accessed 8 May 2021].

[15] cppreference, "C++ compiler support," 4 March 2021. [Online]. Available: https://en.cppreference.com/w/cpp/compiler_support. [Accessed 7 May 2021].

[16] "Wikipedia: QR code," 7 May 2021. [Online]. Available: https://en.wikipedia.org/wiki/QR_code. [Accessed 7 May 2021].

[17] ISO, September 2006. [Online]. Available: https://www.iso.org/standard/43655.html. [Accessed 7 May 2021].

[18] DensoWave, "QR code," DensoWave, [Online]. Available: https://www.qrcode.com/. [Accessed 7 May 2021].

[19] Thonky, "QR Code Tutorial," 9 March 2020. [Online]. Available: https://www.thonky.com/qr-code-tutorial/. [Accessed 7 May 2021].

[20] STMicroelectronics, "AN4621 Application note," December 2019. [Online]. Available: https://www.st.com/resource/en/application_note/dm00148033-stm32l4-and-stm32l4-ultralowpower-features-overview-stmicroelectronics.pdf. [Accessed 8 May 2021].

[21] S. Thornton, "Display options for MCUs: LCD, LED, and OLED," June 2017. [Online]. Available: https://www.microcontrollertips.com/display-options-mcus-lcd-led-oled/. [Accessed 8 May 2021].

[22] Solomon Systech, "SSD1608," February 2015. [Online]. Available: https://cdn-learn.adafruit.com/assets/assets/000/099/574/original/SSD1608.pdf. [Accessed 8 May 2021].

[23] STMicroelectronics, "VL53L1X," November 2021. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf. [Accessed 8 May 2021].

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

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 – GitHub repository with the work

https://github.com/nth-eye/Thesis