KEA Copenhagen School of
Design and Technology
Software Development

Final Report for Introduction to Cryptography

Implementing a Secure Image-Based Password System

Teacher name: **Lorena Ronquillo**

Group Assignment:
Mohammad Saiful Islam Bhuiyan
Mohammad Forhad Hossain
Gabor Hartyandi
Kaspars Cinis

Date: 13-12-2017
Copenhagen

1. Introduction **3**
1.1 Problem statement 4

2. Objectives **4**

3. Research on other image-based password systems **5**

4. FURPS **7**
4.1 Functional requirements 7
4.1.1 User stories 7
4.1.1.1 Account Login 7
4.1.1.2 Account Register 7
4.2 Non-functional requirements 8
4.2.1 Usability 8
4.2.2 Reliability 8
4.2.3 Performance 8
4.2.4 Supportability 8

5.0 Security **8**
5.1 Creating and storing passwords 8
5.2 Hashing/ Key Derivation algorithms 10
5.3 Analysis on how to save the image-based password in the database 12
5.4 Password encryption in case the database is compromised 12
5.5 SSL description 13
5.6 Character set influence in security 14

6. Implementation **15**
6.1 Used Technologies 15
6.2 General description of our implementation. 15
6.3 The choices we have taken to secure our implementation 17
6.3.1 Shuffling 17
6.3.2 Endpoints 17
6.3.3 Mapping the frontend image id to a random. 18
6.3.4 Usability 18
6.3.5 Our server setup 18
6.3.6 SSL/TLS 19
6.3.7 Two-factor authentication 19
6.3.8 SQL injection, Xss and csrf protection 19

7. Conclusion **20**

8. References **20**

1. Introduction

In our everyday life, we use many online-services which require a login to protect our personal information, and to save the state of our progress. Login systems do not only need to be secure, but they should also be user-friendly. In this section we will describe the current most used login method; the text-based password login, and introduce the problems, which affect the everyday usage of this type of login system.

Text-based passwords have been widely used as the standard way of authentication across the web for many decades. Generally, for each service, the passwords are stored in a database together with the user information. However, every year billions of passwords are leaked. Just in 2016, over 3 billions of passwords were leaked.[1] This is a big security risk, because, more than half (55%) of users use the same password for most of their accounts.[2]

Of course, once a password is leaked, it is not compromised yet. In the best case scenarios, passwords are hashed (hashing a password is a generally accepted security standard), so optimally the attacker would obtain only the hash and/or a salt. The problem is that these leaked passwords can be brute-forced. Just by a simple dictionary attack it would be possible to retrieve a large part of these passwords, since 26 % of users use simple passwords, such as their birthdays or names.[2]

Especially with the advent of quantum computers, most of the current algorithms used for passwords might become obsolete. As mentioned in Daniel J. Bernstein's “Introduction to post-quantum cryptography”, quantum computers could brute force them very quickly.[3]
While there are better and safer algorithms against quantum computing, this still does not help the billions of passwords that are already leaked, and that are waiting to be cracked.
While it is possible to increase password security by implementing password requirements as minimum character, special character, etc, the downside is, that with such more complex passwords, it is more likely that the user will just write them down, which then makes them vulnerable to social engineering attacks.

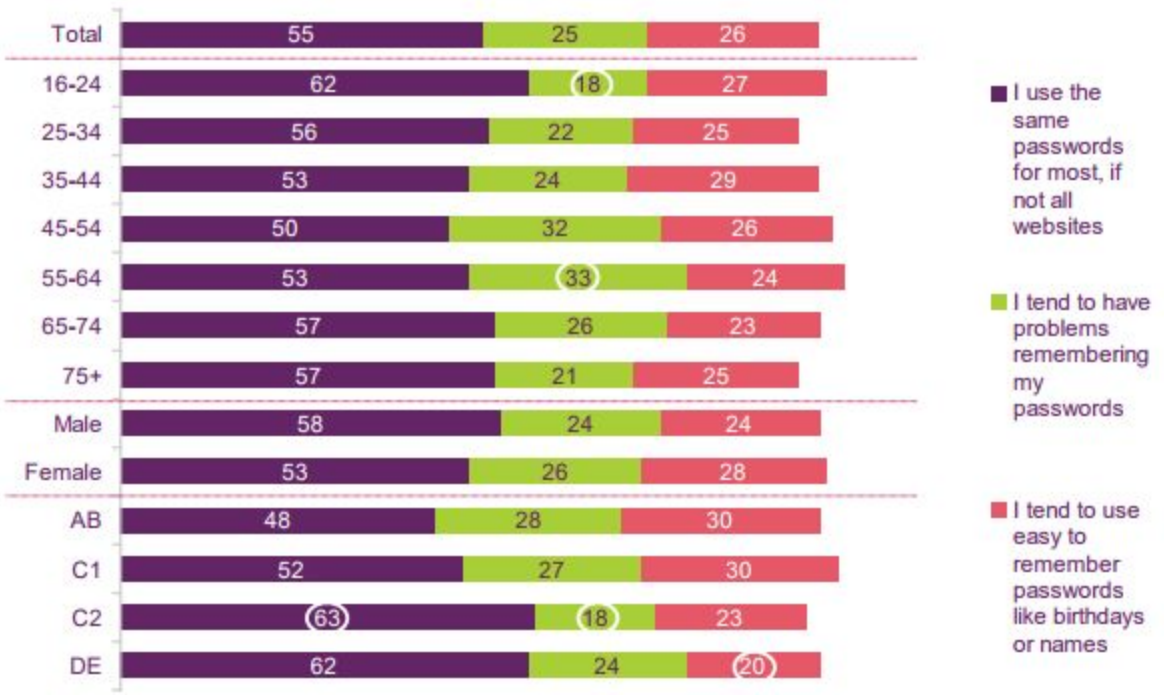

Figure 1. Password usability by different user groups[2]

As discussed, text-based passwords have a lot of flaws, including security issues, as well as the fact that a lot of users are unsatisfied with them. That is why we researched how feasible it is to have an image-based password system to replace the current text-based one.

1.1 Problem statement

What measures have to be applied to implement a secure image-based password system?

2. Objectives

In this section we introduce the aim of our project, and the tasks for the project.

Our aim of researching the topic was to shed a light on new ways of securing passwords that can benefit people with difficulties with using text-driven passwords. This could be users who have a hard time remembering text, and people who are bad at spelling; for example people with dyslexics. Moreover we wanted to investigate if it is possible to implement the login system so that it is secure from different security attacks.

To achieve our aim, we defined the following research objectives:

1. Analyse how to achieve a password complexity which is secure.
2. Analyse how to save the image-based password in the database.
3. Analyse on how to make the password encrypted in case the database is compromised.
4. Define specific factors and prioritized users to develop image-based password systems.
5. Propose ideas for designing a new interface for an image-based password system.
6. Define what the hardware requirements are for using our login system.
7. Use programming language to implement an image-based password system where:
    a. Users can create an account, and choose their password.
    b. Users can login using their chosen password.
    c. Users can change their password.
    d. The hashed password should not be crackable in under an average of 5 years on an averagely fast PC.
8. Design an encryption technique for the password file during the implementation of the image-based password system.

3. Research on other image-based password systems

In our introduction we have identified secondary research, that has been done regarding the security weaknesses of online authentication methods. However, there is a very limited number of research in the area of image-based passwords. In our preliminary investigation of the topic, several research papers support the argument that the main advantages of image-based passwords are, that images are easier to remember than text-based passwords. [7] The main complaint of the users of image-based authentication is that the password registration and login process time is too long. For example, during the user registration form, a user has to choose images from a large set of selections. When on the login page, the user has to choose between many images to identify the limited pass-images. Most of the users are not even familiar with the possibility of using image-based passwords.[8]

Below we provide the references, review and comments of the research papers that we found, that are related to our research proposal.

The article “Security Analysis & Its Implementations Using Image Based Authentication for 3-Level Security System”[7] is about graphical authentication using a pair-based authentication scheme to select click points on an image matrix. It also proposes a 3-Level Security System, involving three levels of security, where the earlier level must be passed in order to proceed to next level. We have analyzed the 3-Level Security Systems proposed solution.[7] We thoroughly investigated their solution and found few difficulties for users to use this authentication system. Some of the users failed to authenticate their password in the 3rd phase while using this system. Few of them did not create their password for the first time because of the complexity of the system. Some users failed to authenticate a valid image password in their first attempt. Some of them were successful in their second attempt. The rest of the users failed to log in. There was no difficulty for users in level 1 and level 2 authentications.
We did not want to implement this idea to our project, but we gained some knowledge from this article. This knowledge gave us a strong foundation to implement in our project.

The article “Graphical Password System Using Image Segmentation”[8] is about a graphical authentication scheme based on the Hash Visualization technique. In this graphical password system, a set of random pictures is generated by a program from which a user is asked to select a certain number of images. These selected images will then be used later for authentication. One of the advantages we found in this article is that it is very difficult to break image-based passwords using the traditional attack methods such as brute force search, dictionary attack, or spyware. The main disadvantage of this authentication system is that the password registration and login process time takes too long. After analyzing the proposed solution of the article, we could see that they have tried to implement a strong and user-friendly security system. We found that this graphical authentication system had lot of scope to improve. In this system, users need to upload an image to create a password. After uploading the image, the system divides the image into a 2*2 grid. Users can select any image block as a password by dragging the mouse. In this case, hackers can sometimes predict the image. Another problem in this article was, that when images are saved as a password in the database, they are not encrypted. We found that, the authentication system is not secured between client and the server side during registration. We gathered a few ideas and knowledge from the article that has helped us to implement our project.

Another article that we found relevant was “Image Based Registration and Authentication System”.[9] In this article the authors propose integrating text-based passwords with images to provide more security of authentication systems. In their system they use hash function for authentication. At the login page, the user submits

the user ID and an image as credentials to the system. If the image matches with the one stored in the system, the user is authenticated. After analyzing the the article we found that they have implemented a very user friendly graphical user interface. The system has used images as password but does not store the images in the database. In this case images are read byte wise and SHA-1 hash function has been used for hashing. Algorithm SHA-1 produces a 20 byte output which requires less memory. This output is saved in the database as a password. The article helped us to find our main idea of our project.

4. FURPS

4.1 Functional requirements

4.1.1 User stories

4.1.1.1 Account Login

User arrives on the login page. A random ordered set of images are displayed on the screen along with a field for typing in the email. User defines his/her email address, and chooses her/his previously registered password out of set of images. If email matches with the image password saved in the database, user is redirected to the index page. In any other cases an error message is shown.

4.1.1.2 Account Register

User arrives on the login page, and clicks on registration button. The system redirects to the register page. On the register page a number of randomly ordered images will be presented along with fields where user defines his/her personal info such as name and email address. User defines his/her password by clicking on the displayed images.

4.2 Non-functional requirements

4.2.1 Usability:

- The web application should have an intuitive design with small room for wrong user inputs.

4.2.2 Reliability:

- User information will be encrypted and stored safely.
- System uptime will be at least 80 %.
- Database backups will be done regularly.

4.2.3 Performance:

- Login in the web application will provide a strong security service.
- Login in service startup time will be < 10 seconds.
- Login in response time will be within 5 seconds.

4.2.4 Supportability:

- Login in the web application will run across all modern browsers.
- Login in the web application will support the English language.
- The system will be running on Linux based servers.

5.0 Security

To achieve a secure login system there are different security measures which have to be considered. In this section we will detail the measures that ensured that our image-based login system is secure.

5.1 Creating and storing passwords

Creating and storing a password, either image based or text based, needs to follow security and usability measures to prevent loss sensitive user data. These measures are summarized by NIST Digital Identity Guidelines (NIST). The guidelines state that password complexity should be avoided. In other words no special letter should be required for creating a password from a user. NIST comes with an example that adding numbers, capital letters and special characters to a non-appropriate password such as “password” would result in a “1Password!” password, that dictionary attacks would be ready to crack. At the same time, the guideline also states that password length is, “(...) a primary factor in characterizing password strength. Passwords that are too short yield to brute force attacks as well as to dictionary attacks using words and commonly chosen passwords.”[6] NIST advise to limit the amount of login attempts to prevent online attacks. They state that every 10th failed password try should require some extra user action. The minimum requirements of the length of the password should be 8 characters, but developers are encouraged to give users the option to specify 64 character passwords. The guidelines also advise that the password field is as

user-friendly as possible. For example, “To make allowances for likely mistyping, verifiers MAY replace multiple consecutive space characters with a single space character prior to verification, provided that the result is at least 8 characters in length.”[6] Moreover usage of only ASCII characters is encouraged. Also the user should be banned from using passwords leaked from previous database breaches, dictionary words, repetitive characters and the words which can be specific to the context. Developers should offer guidance of how to make a secure password, as well as a password-strength meter. In addition the user should be allowed to paste passwords into the password field, and developers “(...) SHOULD offer an option to display the secret — rather than a series of dots or asterisks — until it is entered.”[6] The communication channel should be encrypted with SSL/TLS to prevent eavesdropping and MitM attacks. The passwords should be salted and hashed with a keýderivation function, which makes the the cost of the attack high. The recommended derivation functions include Password-based Key Derivation Function 2 (PBKDF2) and Balloon. “A memory-hard function SHOULD be used because it increases the cost of an attack. The key derivation function SHALL use an approved one-way function such as Keyed Hash Message Authentication Code (HMAC). (...) Secure Hash Algorithm 3 (SHA-3), CMAC or Keccak Message Authentication Code (KMAC), Customizable SHAKE (cSHAKE), or ParallelHash. The chosen output length of the key derivation function SHOULD be the same as the length of the underlying one-way function output. The salt SHALL be at least 32 bits in length and be chosen arbitrarily so as to minimize salt value collisions among stored hashes. Both the salt value and the resulting hash SHALL be stored for each subscriber using a memorized secret authenticator. For PBKDF2, the cost factor is an iteration count: the more times the PBKDF2 function is iterated, the longer it takes to compute the password hash. Therefore, the iteration count SHOULD be as large as verification server performance will allow, typically at least 10,000 iterations.”[6] Hash should be stored separated from the password, in case of database breach.
NIST recommends two-factor authentication among password authentication. One-time passwords sent by email or SMS.[6] According to auth0.com [10], sending one-time passwords by SMS was not recommended by NIST previously because SMSs can be redirected by a malware to an attacker phone. However in the newest guidelines it is recommended again, prompting that the “Verifiers SHOULD consider risk indicators such as device swap, SIM change, number porting, or other abnormal behavior before using the PSTN to deliver an out-of-band authentication secret.”[6]

We used the above report to implement our user interface user-friendly and choosing a secure key derivation function for hashing. The choices we made are further detailed in the Implementation section.

5.2 Hashing/ Key Derivation algorithms

A specific password hashing function is mainly used to protect systems from attackers, or more accurately, to prevent attackers from gaining access to systems. In particular, an attacker will try more hashes of passwords per second by using a graphics processing unit (GPU).

The hashing process is a one-way process to convert a given data into an unique string of a fixed length. The process will always return the same string from that data. This allows the system to validate the password without knowledge of the original data.

SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA).[11] SHA is generally not introduced in our project. Not because of security flaws but because the speed of the hashing function and the easy implementation on a GPU. Moreover, it was announced that a collision was found in SHA1,[12] which means that different data might result in the same output and that the cryptographic algorithm is broken.

MD5 algorithm , which produced 128-bit hash value is a hash function extensively used in internet world.[20] Though it was developed for providing cryptographic hash functions, a lot of vulnerabilities found in this algorithm. Security weakness of MD5 has been found in 2012 by the Flame malware. Finally Carnegie Mellon Software Engineering Institute considers MD5 as a wrecked and incongruous cryptographic algorithm for advance use.[13]
Password-Based Key Derivation Function 2 (PBKDF2) is a key derivation functions , which is designed to decrease vulnerability and weakness of encrypted keys for brute force attacks.[21] It performs the Hash-based message authentication code(HMAC) as many times as specified by the ‘iterations’ parameter. One of the main weakness of PBKDF2 is that, it can be easily implemented in a small circuit as well as very little RAM. Hacker make brute force attack by using graphics processing units (GPU) or application specific integrated circuits with low cost.[14] On the other hand, implementing brute force attack against bcrypt key derivation function requires a huge amount of RAM that is fixed for a given amount of CPU time. So bcrypt is much stronger in terms of brute force attack.[16]

Faster cryptographic hashes for example, SHA and MD5 have several design requirements. It should be efficient and fast to calculate. It has security vulnerabilities and many GPU work on fast and easy hashes. GPU can crack hundreds of millions of

these hashes each second. The faster the algorithm runs, the quicker the password can be hacked. On the other hand, slow hashes for example Bcrypt, PBKDF2 and Scrypt have different design goals. They are designed to be inefficient and difficult to calculate. So it is therefore almost impossible for attackers to do dictionary attacks.

In our project, we are using the slow hashing function BCrypt, which is a implementation of OpenBSD's Blowfish password hashing code.[17] BCrypt is a password hashing function which incorporates a salt to protect against a rainbow table attacks. The algorithm has an adaptive functionality, so increasing iterations counts in the process of make hashing functions slower and respectively the system gets much stronger against brute force attack even though the attacker has increased computation power.

We are developed an image-based password system in java programming language, so jBCrypt is a Java™ implementation which is used in our system for hashing password.

5.3 Analysis on how to save the image-based password in the database

To save the image-based password user has to select several images from the password field to create the password. Each image generates a unique string of four characters, a combination of numeric digits,uppercase and lowercase letters, in server side.

Java BCrypt generates a random salt. A "cost" factor is predefined by default value 10 in the algorithm, which is changeable. jBcrypt generates an encryption key by using a salt and a cost factor from a given password. The system knows the salt and ciphertext length, so it concatenates them and stores them as a string in database.

When an user wants to login to the system, authentication process in jBcrypt derives a salt from stored string. Then it applies cost and salt to the input password and generates a string. Finally it compares the stored string with the newly generated string. If the comparison matches, the user is authenticated.

A bcrypt "hash" that is stored in the database looks like this:
$2a$10$JozQv9pEcmSHNOB1CSkZbOq1AW3H6Cxvm1mEu1vZjH/UgX4ly69.C
It has three fields ,enclosed by "$":
"2a" defines version of bcrypt algorithm that was used.
"10" defines the cost factor. So 2^{10} iterations of the key derivation function are used in the system.

"The rest of the hash string includes the cost parameter, a 128-bit salt (base-64 encoded as 22 characters), and 184 bits of the resulting hash value (base-64 encoded as 31 characters)."[18]
JozQv9pEcmSHNOB1CSkZbOq1AW3H6Cxvm1mEu1vZjH/UgX4ly69.C is the salt and the ciphertext, concatenated and encoded in a modified Base-64. In total of 60 bytes (480 bits). So storing millions of hashes in a database should be taken into account.

5.4 Password encryption in case the database is compromised

The BCrypt utility generates salts randomly and appends them to the output of the function so that they are remembered later on. If an attacker gains access to the root of the system, it would be easy to get the salt and hashed password. As salt is generated randomly, the attacker needs to generate different rainbow tables for every possible salt. After all, salts make it harder to produce a rainbow tables.

The reason why we used salts is to stop precomputation attacks, such as rainbow tables. The attacker could create a database of hashes and their equivalent plaintexts, searching hashes and determining plaintext for a specific hash. As an unique salt is used to hash each password, an attacker would have to attack every single password individually, because same password would have completely different hashes stored in database. The attacker may have to use an expensive, power-hungry GPU or a FPGA (Field-programmable gate array) farm to run attacks against targets. So an attacker would have to run a full brute-force search a million times to get whole million passwords.

5.5 SSL description

SSL (Secure Sockets Layer) is a security protocol that provides encrypted communication between a web server and a browser in an online communication. SSL uses a combination of a public key and private key (symmetric cryptography) encryption, to secure a connection between a web server and a browser (client system). SSL runs above the transport layer and the network layer, which are responsible for the transport of data between processes and the routing of network traffic over a network between client and server, respectively, and below application layer protocols such as HTTP and the Simple Mail Transport Protocol. Besides supporting the web pages, SSL has been implemented for applications including email, file transfer, instant messaging and voice over IP. “The "sockets" part of the term refers to the sockets method of passing data between a client and a server program in a network or between processes in the same computer”.[19] Due to numerous protocol and implementation flaws and

vulnerabilities, SSL was deprecated for use on the internet by the Internet Engineering Task Force (IETF) in 2015 and has been replaced by the Transport Layer Security (TLS) protocol.
OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping. OpenSSL contains an open-source implementation of the SSL and TLS protocols. OpenSSL supports a number of different cryptographic algorithms. In our project we used OpenSSL library, which provides solution for SSL / TLS connection between the client side and the server side. Here it uses Java's built-in security for calling OpenSSL library as a process from Java.

5.6 Character set influence in security

Because a part of our research aimed to look for aspects of how the image-based system could be more user-friendly than a text-based password system, we couldn't focus entirely on security and we had to make some compromises. For example, after multiple prototype iterations, we settled on a set of 30 images. The higher amount of images used for the login, the harder it would be for an attacker to brute-force the password, thus making it more secure. But since the images are shuffled for security reasons, a higher number of images would mean that the users would have to spend more time looking for their chosen image, which would take time and might be mildly annoying. We found that 30 images fitted nicely on an average monitor size,[4] and it would be easy for users to look across all images without the need for scrolling.

To save the password we needed to transfer the user selected images into text. We could have just used the string that contains a list of user selected images, but this posed a security issue, because users can find out the numbers assigned to images, since they're also available on front-end. So if the database would somehow get leaked, the attackers could just try to brute force the passwords based on image numbers. To prevent this we implemented the image to a text mapping mechanism, which in our case meant, to just assign a text string to each image. This was done purely on the server, so if the attacker was to hack the database, they still wouldn't know the character set we were be using, and it would prevent them from even trying to brute force the passwords.

In the current implementation, if an attacker gets root access to the server, the attacker would be able to see how the mapping works, and would be able to try brute forcing the passwords. We considered more advanced mapping algorithms, such as pseudo-random maps generating individually for each user, but it would just be a small

problem for an attacker with root access to the server. So because of the low increase in security we didn't feel the need to implement it.

To make sure our password system was secure against brute forcing attacks, we had to make sure that on average it would take a long time to crack the passwords. In our case, we set 5 years as the minimum goal, that would take to crack a single leaked password. From the beginning we targeted the minimum password length, as 8 images. This means that the minimum number of possible image combinations is = 30^8 = 656 100 000 000 combinations.
On average, an attacker would need to try half of the possible combinations. A modern CPU can calculate around 200 BCrypt hashes per second,[5] so in average, the minimum time required to crack one password would be around 50 years. The time could be drastically decreased by using multiple parallel GPUs.

6. Implementation

6.1 Used Technologies

We used the Java Spring-Boot framework to develop our system with Javascript, and the template language Tymeleaf. Spring-Boot offers fast development without lots of setting. Tymeleaf template language is integrated with the Spring-Boot framework, and offers functionality for example against csrf attacks. We set up a cloud server at DigitalOcean for serving our backend and mysql database.The implementation can be checked on Github.[15]

6.2 General description of our implementation.

Different interface design proposals

The interface of the image-based password system is an important factor, because it defines how usable the whole system is. Because of that we considered multiple alternatives until we chose the one, that we thought, was the most easiest to use.

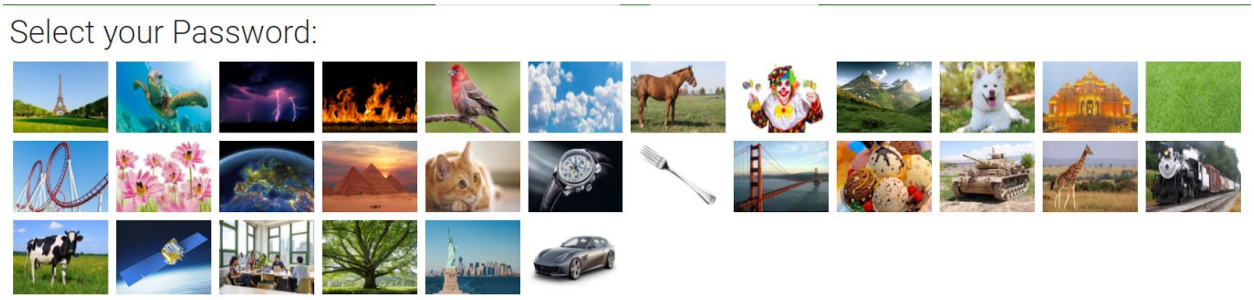

Figure 2. Final interface design

Our chosen interface, consisted of a simple panel of all the images in our image set, and the user could freely click on the images, to make their password (Figure 2). Before we agreed on the current solution we also had different proposals. One of them was a design, similar to a Google Captcha (Figure 3).

Figure 3. Google Captcha image selection

The proposal was, that to choose a password, the user would be presented with a container of 9 images. Upon selecting a single image, the images would change to a different set of 9 images, thus allowing the user to select the next, different image. The

process would be repeated until the password was entered. In our prototypes, this design was more difficult to use, and wasn't as user intuitive.
The final interface proposal that we made was a combination of image- and number-based system. First the user would have to choose an image, from a set of 20 images. Then after choosing an image, the user would be presented with a keypad, and a 4 or more digit number would be required to enter. Because this solution wasn't strictly image based, we decided not to use it.

Our frontend Login and Signup pages show 30 images and an email field. Each image has a id number which represents the image on the frontend. When the signup form is submitted, the image id number is mapped one by one to a different random text. After we had built our password with this random text, we hashed the password and saved the hashed password with the email to the database. When the user wants to login, we check if the provided text password matches the password in the database. Whenever they match we redirect the user to the authenticated home page.

6.3 The choices we have taken to secure our implementation

6.3.1 Shuffling

Showing images in the same order on our login pages on each request could introduce vulnerability, in case of password peeping; for example if a person looks the pattern of someone's images from behind. Another vulnerability issue could be that people might always choose a password based on where the images are places in the grid. They might choose the first five images, or a pattern of images in the grid. Because of these concerns we chose to shuffle our images securely in every request. We used Knuth Fisher Yates shuffling algorithm, which makes sure that the frequency of the possible shuffled result is distributed evenly.

The algorithm in our code looks the following:

```
private static void shuffleList(List<String> a) {
    int n = a.size();
    SecureRandom random = new SecureRandom();
    random.nextInt();
    for (int i = 0; i < n; i++) {
        int change = i + random.nextInt(n - i);
        swap(a, i, change);
    }
}

private static void swap(List<String> a, int i, int change) {
    String helper = a.get(i);
    a.set(i, a.get(change));
    a.set(change, helper);
}
```

Figure 4. Knuth Fisher Yates algorithm

In the implementation we use SecureRandom which creates a crypto secure random.

6.3.2 Endpoints

To secure endpoints we used Spring Security, and we blocked all endpoints except /signin /signup and the endpoints ending with .html, .js, .css, which are necessary to show our frontend. Whenever the users are not authenticated they are redirected to the signin page or they get a 404 page not found response code.

6.3.3 Mapping the frontend image id to a random.

When the users have to choose their password images, we map the image to their corresponding frontend id. In our case it is numbers: every image has a number from 1 to 30. The created password can be inspected in the browser after the submit. In this way an attacker would be able to know our character set, which could lead to dictionary attack. That is why we map the front end id to a eight character long securerandom text. We generated our random text by the help of the www.random.org service, which provides crypto random texts. We allowed numeric digits, uppercase letters, lowercase letters, and each string should be unique. Mapping images to a string helped build a long secure password for the user. If the user chooses 8 images as a password, that would create a 64 characters long secure text-based password.

6.3.4 Usability

Implementing the frontend, we needed to make sure that the provided password system would be user-friendly. Our app provided 30 images for password creation. The more images chosen, the more secure, but paginating our images to smaller devices could mean that our users only would choose from the images on the first page all the time, which would make our character set vulnerable for brute force attacks. We chose 30 images because that is a minimal character set which makes the brute force attack time-costly.

6.3.5 Our server setup

As mentioned before we used a Droplet on DigitalOcean to deploy our web application. Our server comes with Ubuntu 16.4 OS. To secure our server we needed to make several changes on the default setup.

To connect our server we used ssh instead of logging in with username password. Ssh is an asymmetric encryption, which can provide a secure communication channel for secure login. We used Ssh-2 rsa key as the private key. To ssh into our server our local computer had to have the private key and the server needed the public key. Only computers with the private key can connect to the server. We also setup a password for login which means that even if an attacker got hold of our private key, the attacker would need to know the password.

On our OS we setup a user that has sudo rights. To make a sudo command the user needs to specify another password. We also disabled the root ssh, so the user is not allowed to ssh into our remote server with root access. In that way even if the attacker has our private key and our password to the ssh, he still needs to know our username and our user password to execute a command on our server. On our server we also banned all port except which we use. Port 22 is open for ssh connection, which we use for sshing into our server, and port 443 is open for our web application.

We use mysql for our database, and we also created a user on mysql for our web application. The created user only has access to the image-based password systems database. Our mysql server only serves on a localhost.
We also use fail2toban for scanning logfiles and banning ips, which tries to ddos attack our server or tries to guess passwords.

6.3.6 SSL/TLS

On our domain we setup a free ssl/tls with let's encrypt.

6.3.7 Two-factor authentication

Two factor authentication could add an extra security layer for logging in. The login requires an extra one-time-password which the users can get from a standalone application on their device, or the password can be sent as an email or sms. However when implementing sms password sending, the developer has to make sure that the sms is not redirected to an attacker.

6.3.8 SQL injection, Xss and csrf protection

We used Spring built-in security features and templating language on frontend Tymeleaf to avoid xss and csrf attack.
We used prepared statement, when querying from database which protects from sql injections.

7. Conclusion

There are several measures to implement a secure image-based password system. Security of the application is not only a matter of the used encryptions, but also a matter of how user-friendly it is to use the password system.
Using a key derivation algorithm is essential for keeping the passwords encrypted in the database. The key derivation algorithm ensures that our encrypted data can be brute forced, if the server is compromised.
Salting each password individually with different salt protects the keyderivated password from dictionary attack.
Mapping the typed password to another character set on the backend, makes sure that the attacker doesn't know the character set, only if the server is compromised.
The size of the character set is also a factor which is important for brute forcing all possible characters; the bigger the character set the harder it is to try all possibilities.
The security of the server is also a key to achieving a secure system. Setting up ssh for a secure connection and creating an individual user account, and making the root account inaccessible for our server are important factors. Another important measure is to make our database secure. This is achieved by creating a user account for each web application.

It is also important to query our database with prepared statements to avoid SQL injection. Sql injection can compromise the database.
Making the user interface user-friendly is the last measure. Thinking about how the users use the interface key to avoid non-secure passwords. For example if we provide pagination for our images, this can cause that the users only pick passwords from the first page, which makes our character set lower, which then causes vulnerability for a bruteforce attack.

8. References

This report is the shortened modified version of our Research and Dissemination project: Image-based Password Systems(A Research of an Image-based Password System as an Alternative to a Text-based Password System).

1. Shape, (2017). “2017 Credential Spill Report”.

2. Ofcom (2013), “Adults’ media use and attitudes report”.

3. Daniel J. Bernstein (2009), “Introduction to post-quantum cryptography”, Springer-Verlag Berlin Heidelberg.

4. W3schools, “Browser Display Statistics”, https://www.w3schools.com/browsers/browsers_display.asp, (accessed 23 November 2017)

5. Katja Malvoni, Josip Knezovic (2014), “Are Your Passwords Safe: Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware”, Solar Designer

6. NIST, (2017) National Institute of Standards and Technology, 2017 SP-80063 Authentication and Lifecycle Management, https://doi.org/10.6028/NIST.SP.800-63b

7. Rahul S. Mate, Pramod P. Gadekar, Suhas B. Sathe, Prof. Mangesh K. Manke (2014) “Security Analysis & Its Implementations Using Image Based Authentication for 3-Level Security System” http://www.ijemr.net/DOC/SecurityAnalysisAndItsImplementationsUsingImageBasedAuthenticationFor3LevelSecuritySystem(92-94)3d627e46-91de-407b-8bb8-fa34c656e082.pdf

8. Krishan Chand, Ashish Anand (May 2016)“Graphical Password System Using Image Segmentation”
http://www.ijcter.com/papers/volume-2/issue-5/graphical-password-system.pdf

9. Srinath Akula, Veerabhadram Devisetty(April 2013) “Image Based Registration and Authentication System”
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.465.297&rep=rep1&type=pdf

10. auth.com Retrieved 2017-11-01

11. https://en.wikipedia.org/wiki/SHA-2. Retrieved 2017-11-14

12. Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), Yarik Markov (Google), Alex Petit Bianco (Google), Clement Baisse (Google). "Announcing the first SHA1 collision" February 23,2017. Retrieved 2017-02-23.

13. Chad R, Dougherty (31 Dec 2008). "Vulnerability Note VU#836068 MD5 vulnerable to collision attacks". Vulnerability notes database. CERT Carnegie Mellon University Software Engineering Institute. Retrieved 2017-11-14.

14. Colin Percival. scrypt. As presented in "Stronger Key Derivation via Sequential Memory-Hard Functions". presented at BSDCan'09, May 2009.

15. https://github.com/ghartyandi/ImagebasedPasswordSystem

16. "New 25 GPU Monster Devours Passwords In Seconds". The Security Ledger. December 4,2012. Retrieved 2013-09-07.

17. Provos, Niels; Mazières, David; Talan Jason Sutton 2012 (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81–92.

18. https://en.wikipedia.org/wiki/Bcrypt.Retrieved 2017-11-14

19. https://en.wikipedia.org/wiki/Transport_Layer_Security.Retrieved 2017-11-16
20. https://en.wikipedia.org/wiki/MD5.Retrieved 2017-11-14
21. https://en.wikipedia.org/wiki/PBKDF2. Retrieved 2017-11-14