UNIVERSITY OF BONN

MASTER THESIS

Improving Let's Hash: Development of a website helping developers with security-related programming tasks

Cüneyt Erem

cuneyterem8@gmail.com

First Examiner
Prof. Dr. Matthew Smith

Second Examiner
Prof. Dr. Michael Meier

Supervisor
Lisa Geierhaas, M.Sc. Dr. Christian Tiefenau

Institute of Computer Science 4

Behavioral Security Group

30.09.2022

ABSTRACT

Password security has always been a significant issue for software developers and users. At first glance, it may seem that software developers care a lot about security. However, many studies have shown that software developers should be at least as careful as users due to a lack of security concerns. For this reason, different infrastructures have been developed to support software developers in many fields. One example is the LetsHashSalt website, a platform that allows software developers to write more secure password authentication. The primary purpose of this thesis is to expand the features of the LetsHashSalt website, make it appealing to more programmers, and help them to write secure password authentication codes in different languages. That is why we added new programming languages to the website, such as Javascript, PHP, Golang, C# languages, and libraries like Pbkdf2 and Regex. In addition, we asked five professional software developers who are experts in their fields to test the security of the code blocks. Finally, we made it a more comprehensive, secure password authentication website that many software developers can use.

DECLARATION OF AUTHORSHIP

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

....

(Place, Date) (Signature)

Table of Contents

Contents I
Abstract . . . I
Declaration of Authorship . . . II
Table of Contents . . . III
List of Figures . . . V
List of Tables . . . V
List of Listings . . . V

1. Introduction 1

2. Related Work 3
1. Password Storage . . . 4
2. Password Policy . . . 5
3. Two Factor Authentication . . . 6
4. LetsHashSalt Website . . . 7

3. Methodology 8
1. Website Design . . . 8
   1.1 Programming Languages Selection . . . 9
   1.2 Password Storage Libraries Selection . . . 10
   1.3 Password Policy Libraries Selections . . . 11
   1.4 Two-Factor Authentication Library Selection . . . 11
   1.5 Adapting Code Snippets for Different Languages . . . 11
2. Study Design . . . 13
   2.1 Survey Questions . . . 13
   2.2 Participants . . . 13
   2.3 Pilot Study and Study . . . 14
   2.4 Evaluation Criteria . . . 14
   2.5 Limitations . . . 15
   2.6 Ethics . . . 16

4. Results 17
1. Not Functionality but Security . . . 17
2. Password Storage Evaluation . . . 18
3. Password Policy Evaluation . . . 22
4. Two Factor Authentication Evaluation . . . 23
5. Evaluations of StackExchange answers . . . 25

5. Discussion 26

6. Conclusion 28

7. Future Work 29

Appendix I

A. Website Images **II**
1. Password Storage . . . II
2. Password Policy . . . III
3. Two Factor Authentication . . . IV
4. Information . . . V

B. Survey **VI**
1. Survey Questions . . . VI

C. Consent Form **IX**
1. Survey Consent Page . . . IX
2. Consent Form . . . X

D. Code Fragments **XIV**
1. Password Storage . . . XIV
1.1 Python . . . XIV
1.1.1 Argon2 . . . XIV
1.1.2 Bcrypt . . . XIV
1.1.3 Pbkdf2 . . . XV
1.2 Java Plain . . . XVI
1.2.1 Argon2 . . . XVI
1.2.2 Bcrypt . . . XVII
1.2.3 Pbkdf2 . . . XVII
1.3 Java Spring . . . XIX
1.3.1 Argon2 . . . XIX
1.3.2 Bcrypt . . . XX
1.3.3 Pbkdf2 . . . XX
1.4 Javascript . . . XXII
1.4.1 Argon2 . . . XXII
1.4.2 Bcrypt . . . XXIII
1.4.3 Pbkdf2 . . . XXIV
1.5 PHP . . . XXV
1.5.1 Argon2 . . . XXV
1.5.2 Bcrypt . . . XXVI
1.5.3 Pbkdf2 . . . XXVII
1.6 Golang . . . XXVII
1.6.1 Argon2 . . . XXVII
1.6.2 Bcrypt . . . XXVIII
1.6.3 Pbkdf2 . . . XXIX
1.7 C# . . . XXX
1.7.1 Argon2 . . . XXX
1.7.2 Bcrypt . . . XXXII
2. Password Policy . . . XXXIII
2.1 Python . . . XXXIII
2.2 Java . . . XXXIV
2.3 Javascript . . . XXXV
2.4 PHP . . . XXXVI
2.5 Golang . . . XXXVII
2.6 C# . . . XXXVIII

3. Two Factor Authentication XL
3.1 Python XL
3.2 Java XLI
3.3 Javascript XLII
3.4 PHP XLII
3.5 Golang XLIII
3.6 C# XLIV

E. Additional Tables **XLVI**

List of Figures

Fig. 3.1 Latest Website Design 9

Fig. A.1 Password Storage Code Snippets on Website II
Fig. A.2 Password Policy Code Snippets on Website III
Fig. A.3 Two Factor Authentication Code Snippets on Website IV
Fig. A.4 Information page on Website V

List of Tables

Tab. 4.1 Participants Background Information 17
Tab. 4.2 Password Storage: Participant 1, 2 and 3 Answers 21
Tab. 4.3 Password Storage: Participant 4 and 5 Answers 21
Tab. 4.4 Password Policy Answers 23
Tab. 4.5 Two Factor Authentication Answers 24

Tab. E.1 Resources Participant Used in Survey XLVI

List of Listings

1. INTRODUCTION

Although technology continues to take place in every part of our lives at a rapid pace, due to some shortcomings in the security field, hackers' attacks continue to do great harm, like the RockYou2021 incident. The problem at the root of this type of attack is that programmers do not securely store the password while writing code [50]. Another study by Acar et al. shows that software developers do not prioritize security while writing code and have deficiencies in writing secure code [3]. In addition, Naiakshina et al. show that while none of the ten software developers who were not told anything could write secure code, only 4 of 10 people who were given information could write secure code. In addition, participants who use the StackOverflow website to write code have much lower security levels than those who use official resources, such that only 17% of them write secure code. [54].

To help developers write secure code for password storage, password policy, and two-factor authentication, Geierhaas et al. established a website called LetsHashSalt. On this website, she has created secure and up-to-date ready-made code blocks consisting of Python and Java code snippets for password storage, Javascript code snippets for password policy, and Python code snippets for two-factor authentication [23].

We decided to develop this website because we thought the LetsHashSalt website would reach more developers and help them write secure password codes. For this reason, we decided to make a more advanced website by adding new programming languages and libraries, examining the accuracy of the selection of programming languages and libraries used on the website, the security, and the up-to-dateness of the code content. We added new programming languages such as Javascript, PHP, GoLang, and C#, and another practical library like Pbkdf2 in the current versions of password storage, password policy, and two-factor authentication sections by considering the OWASP and NIST standards [27] [57]. However, the most critical issue we want to learn during this study is to measure how up-to-date and secure the written code blocks are. Therefore, we created survey questions to conduct this study. They consist of some questions about the background of the software developers, the security status of the existing code blocks, how they can be fixed if they are not secure, and which resources are used while answering the questions.

Since the work we have done needs to be made available to many software developers, the code blocks need to be checked by professional software developers and solve the existing problems and make them ready for use. For this purpose, we decided to make a qualitative study to take a closer look at the thoughts of software developers. With the code review examination by the five software developers, we ensured that we implemented the existing codes securely and selected the proper libraries and up-to-date versions. The study focuses on security rather than functional regulation; thus, we applied only secure recommendations experts gave. Therefore, we pass over their answers in the survey and the resources they used.

What we have obtained as a result of our study is as follows;

Password storage:
- By the given thoughts of the experts, most of the hash and salt code content is used correctly for the security part in Python languages; therefore, only minor changes are applied. Argon2 and Bcrpyt libraries were accurate selections among the has functions. Besides, we decided to add a new library like Pbkdf2. We chose not to include functional recommendations such as rate limit or password manager style in the code content due to the purpose of the study.

Password policy:
- There was no comment from the experts on the selection of regular expressions that were wrong. The only changes we added were using the regex library to test the more inclusive password content. Only for Python code content, we decided to add functional changes to increase readability.

Two-factor authentication:
- None of the experts suggested making changes to the two-factor authentication code content. Therefore, we have decided that current library usage and code content are safe enough, so there is no need to change code content.

2. Related Work

Security and authentication have entered almost every aspect of our lives in the modern world. Although there have been many different security methods over the years, there have always been some vulnerabilities over time. According to the article of Mikalauskas, the RockYou2021 leak having a 100GB TXT file containing 8.4 billion entries of passwords, has been stolen. Data of this size was the most significant leak on the Internet by 2021 [50]. According to the writings of Hill and Swinhoe, there have been many significant data breaches, including Yahoo, Linkedin, Facebook, and millions of data have been stolen [37]. For such reasons, the security of users is a vital area that can affect millions.

There are many studies on password security. For example, Ur et al. show a poor password selection situation due to misconceptions in choosing and remembering passwords [80]. On the other hand, Colnago et al. conducted studies on the secure password power of the Second-factor effect [13]. In addition, many studies have always been done to solve the problems of password security [34] [2] [19]. However, the number of studies that measure the reliability of the code written by software developers is less compared to previous topics [34] [2].

In real-world examples, some studies show that developers have difficulties doing password-based implementations. For example, Acar et al. revealed that 307 GitHub participants, consisting of students and programmers, did not have functional solid, secure coding aspects while writing code in groups of participants [3]. In another study by Acar et al., among student and professional software users, those who use StackOverflow to write code with much lower security than those who use official android and books. For instance, only 17% of users who used Stackoverflow have implemented secure code [54].

According to Naiakshina et al., none of the participants (10 people) who were not told that they should write safe code in the task solutions given to the software students tried to write safe code. Six of the ten who used the recommended libraries could not write a secure code [54]. As seen in studies conducted by many different researchers with similar results, writing secure code is a challenging situation for software developers. These examples indicate that software developers often have less experience writing security code or cannot write sufficiently secure and up-to-date code [15] [21] [53].

Green and Smith identified ten fundamental principles in their study, which are essential steps for creating a usable and secure crypto API. These principles generally aim to make APIs easy to use, maintain, and highly reliable. Suppose software developers are not supported and ignored in these areas. In that case, failures may occur in many security areas in software projects, so as they say, software developers are not the enemy and should be supported [33].

It is vital to carry out broader and more studies on such situations and to support users' secure and up-to-date code writing. The secure password field, password storage, privacy, and two-factor authentication are fundamental issues.

2.1. PASSWORD STORAGE

Password security needs to be protected from different angles. Therefore, the necessary password storage to enable users to access the platform should be carefully put into the database by the software developers. Previously, many passwords were stolen due to leaks as passwords are stored in plain text; the recent RockYou2021 is one of the freshest examples in the memories [50]. In the study of Acar et all., some of the most common mistakes professional software developers make while storing passwords are fundamental problems such as using plain password storage (14.0%), not using salting (23.3%), and using static salt (8.7%) [3].

Naiakshina et al. conducted a qualitative study on how safe password storage is with participants of computer science students. According to this study, programmers always thought of functionality first, then security. No software developer thought of security when asked to store the password. They only thought of security when it was stated that they needed to store passwords securely. The participants used no proper standards, so developers must be given standards and advice. Safe default features should be recommended as there is opt-out security status. Although it is difficult to generalize for all software developers due to the small number of participants, it has emerged due to helping software developers in these items [54].

In another study by Naiakshina et al., students and freelancer developers have the wrong idea about security when storing passwords. For example, some developers have mixed encoding and hashing functions using the Base64 library for password storage. Similar confusion exists when developers use hashing and encryption interchangeably. Another situation is that developers copy and paste code blocks from the Internet even when they need to write secure code. This study shows that developers doing business for actual companies store password storage insecurely [55].

In the security field, companies or organizations have their interests and understandings of security. For example, one of the two crucial organizations supporting secure software content and platforms in the field of cyber security is the nonprofit foundation OWASP and one of the nation's oldest physical science laboratories of NIST in the USA [27] [57]. Therefore, many security engineers follow the recommendations and reports of these two organizations in their coding, so we will also consider the recommendations of these two organizations.

Encryption and hashing are two different methods used to keep data safe. Some programmers may want to use encryption for password security, but passwords should be hashed instead of encrypted, as seen in OWASP's recommended password storage cheat sheet [59]. Since the encryption is a two-way function, the original plaintext can be restored, but the hash cannot be decrypted as it is a one-way function. Therefore, encryption is more suitable for ordinary information in the user profile [59].

An attacker cannot decrypt the hashed password but can crack it. Best practices such as salting, peppering, and work factors should be applied to prevent this issue [59]. Salting is the addition of a unique and randomly generated string to the current password. The key exchange between stored passwords is called pepper. The work factor is the number of iterations of the hash algorithm. The configurations of all these items are taken into account in other elements. Memory, CPU, and a strong hash are created, and many libraries support these functions.

Instead of using less secure hash functions such as MD5 or SHA-1, the recommended hashing algorithms on OWASP's site are Argon2id, Scrypt, Bcrypt, and Pbkdf2 [59]. Argon2 algorithm is the first, current, and number one algorithm in the 2015 Password Hashing Competition [14]. While Bcrypt is the second algorithm recommended by OWASP, the Scrypt algorithm is another effective algorithm written and recommended by Colin Percival [6] [62]. Finally, the Pbkdf2 algorithm is another password hashing algorithm recommended by NIST when using FIPS-140 [32].

2.2. Password Policy

Password usage has entered almost every part of our lives, and every user is asked to select a password for logging into existing private accounts. However, some password policy criteria have been determined for secure password use. In order to determine this password strength, many studies have been done on users before, but studies on software developers are fewer.

Tan et al.'s study give some password policy recommendations for minimum power and length of the password to create a balance between security and usability. For this, they examined composition requirements, blocklists, and neural network-driven minimum-strength requirements to observe that password is effective against attacks [78].

In a study by Ur et al., they wanted to enable users to create passwords that are not harder to remember but more powerful, with a password meter in 1class8 and 3class12 standards, which are effective data-driven password meters for users. The users were then given additional feedback, and the Stringency score was calculated. As a result, 56.5% of users could type a password from their memory. As a result of the study, it was found that 1class8 and detailed text feedback contributed to the efficient password creation of the users [81].

In another study by Ur et al., they measured the relationship between users' behavior and security with the password-strength meter they created. This meter allows users to use longer, more characters, digits, symbols, and upper cases. While creating a password that is not difficult to remember, it was ensured that it had enough power against attacks. This way, users thought longer with these 14 different meters and created longer and stronger passwords [79].

According to Dell'Amico et al. empirical study, id dictionary size is larger, and guessing is harder to predict passwords, leading to avoiding attacks. The passwords in the dataset of MySpace, which has 1.45 million users, are strong enough; PCFGs (Probabilistic Context-Free Grammars) are very useful for passwords. However, the results show that restrictive password policies do not prevent the creation of weak passwords because users spend little effort creating passwords. Therefore, proactive password checker tools are more efficacious [16].

The latest and current recommendations of many studies related to cyber security are given on the pages of OWASP and NIST websites. Most of the password policy recommendations can be followed from these pages [27] [57]. For example, while OWASP recommends the use and length of any character in the password between 8-128, it recommends changing the password when it is possible to pass it to someone else [60]. Among the recommendations made by NIST, the dictionary word recommends not to use usernames, easy-predicted names, or repetitive words. In addition, the password manager should hint at what passwords should and should not include [32].

If we examine some test cases that OWASP has determined to test the reliability of the password policy, here are some important considerations: what characters should be allowed to be used and to what extent (lowercase and uppercase letters, digits, and special symbols), how often should the password be changed and the how many times should the password be used and should the same password be used in different accounts [61]. In addition, Rodrigues and colleagues have created an open-source password complexity and security tool that OWASP recommends. This tool has also been tested on RockYou's famous password list, helping users create more secure passwords [71].

Some of OWASP's most recommended libraries regarding password policy are zxcvbn and Pwned [71]. Zxcvbn library was created by dropbox and is one of the most used password strength checker libraries. Using pattern matching and conservative estimation methods, it tests the strength of a password by detecting repeating or string-shaped words using a password with more than 30 thousand. Thanks to this, it ensures that passwords are more secure, flexible, and usable. Another advantage of this library is that it is coded in many different programming

languages. Therefore, different passwords created in this way can be tested in many different programming languages with this library [85].

2.3. TWO FACTOR AUTHENTICATION

Apart from the two critical steps that increase password storage and password policy security, another method used by many platforms is two-factor authentication. Two-factor authentication is an authentication mechanism that allows access after the user shows two or more factors or proof, from bank ATMs to third-party applications. These questions of evidence or factors are general information, photos, location, etc.

Although two-factor authentication is essential, it is not effective enough for attacks. Twenty years ago, most attacks were passive, and eavesdropping and offline password guessing methods were used a lot. However, as Schneier et al. stated in recent years, with Man-in-the-Middle attacks and Trojan attacks, the user's password can be bypassed, and the password can be stolen. Therefore, it is not suitable for remote authentication over the Internet and is not secure enough for fraudulent transactions [73].

Wang et al. study does not have a specific comprehensive and systematic metric used in smart-card-based password authentication mechanisms [83]. The new schema introduced enhances practicability, simplicity, and security significantly. Thanks to the combination of 'honeywords' and 'fuzzy-verifier,' user card corruption can be understood much more quickly, and a different perspective is provided.

In his study, Reynolds et al. demonstrated the best and worst aspects of YubiKey usability. The first study investigated whether the users' experiences were good or bad when they accessed their Windows, Google, and Facebook accounts using YubiKey [69]. While users found two-factor authentication useless during setup, they found it helpful in their daily lives. Therefore, issues such as standardization of the authentication method, clear indicators, and integration with operating systems are gaining importance.

In a study by Colnago et al., users found two-factor authentication tedious but easy to use and safe [13]. Besides, if more focus is placed on implementation design, adoption mandates, and strategic messaging, two-factor authentication compliance and usage will positively impact users. One of the most important indicators is that the difference between two-factor authentication being 'require' and being 'voluntary' is much lower than expected.

According to a survey conducted by Google, two-factor authentication is one of the three required methods that security professionals feel safe using [42]. According to the advice on NIST's page, two-factor authentication says it should be used almost everywhere it can be used [56]. The email has already become a standard in many companies and fields, including bank accounts, workplace accounts, and health systems.

There are two-factor authentication methods in many different ways. One of the most well-known is OTP (one-time password) and TOTP (time-based one-time password) [88]. In his study, Lee explains that they provide authentication protection against Phishing or Pharming attacks to online banking systems with mobile OTP and QR-code [45]. After the user reads the QR code, the OTP code is sent to the phone, and the user re-enters this code. Thanks to this, the OTP method is one of the most reliable two-factor authentication methods in a significant sector such as banking.

2.4. LETSHASHSALT WEBSITE

Acar et al. have shown in their previous work that developers use websites like Stackoverflow, usually using the copy-paste method [1]. This situation also risks the codes used being outdated or insecure. Therefore, keeping the secure and up-to-date source code in security software is extremely important. As Acar et al. have shown in their other study, many expert software developers have shown that they have problems writing secure code [55]. This situation inspired the creation of LetsHashSalt, a new website that helps software developers to use it quickly and is up-to-date and secure about password security.

Since this work is on developing the LetsHashSalt website, let us talk a little bit about how the first version of this website was. There are two different versions of the website. While the first version enables the creation of code blocks with the wizard's help, the second version is a non-wizard version with three different information pages. As a result of the study, when wizard usage was compared in the categories of usability score, time spent, or the number of clicks, there was no significant difference between them [23]. Therefore, it will not make a difference whether the software developers use the wizard or not while using the site. In our study, however, only the non-wizard version is of interest to us, as there will be a version in which different features are added.

Let us look at the features of the non-wizard version of the LetsHashSalt website in the 'code snippet' section. Hash and pure code blocks are written in Python and Java for password storage, and code blocks are written in javascript for password policy. In addition, python code blocks are written for two-factor authentication. The 'code snippet' is the first page that comes up as soon as the website is opened to prevent the software developers from wasting time. When the page is opened, the code blocks are located in the middle of the page. While the sidebar is located on the right side of the page, the software can follow which section they are in any situation from the sidebar. This design provides users with a high user experience.

The code snippet section, along with the ready-made code blocks written in the password storage, password policy, and two-factor authentication sections, ensures that the software developers write code for password security safely. The password must be kept in the database with hash and salt. The most used and preferred libraries of Agon2id and Bcrypt are used for the selected Java and Python languages. Other known algorithms, such as Scrypt or Pbkdf2, were not included for various reasons.

The use of password policy is the structure that creates essential rules for the passwords of the software developers to meet specific minimum criteria. In this section, a code blog allows manual testing of criteria such as password pattern and length in the widely used Javascript language. Password strength is the handy zxcvbn library. In the two-factor authentication section, there are code blocks for the authenticator mechanism that many companies and applications actively use. For this, a TOTP code example creates a time-based token and allows the user to log in with a one-time password within a certain period.

Apart from the code snippet section, the other two pages on the site are the 'information' and 'about' sections. In the 'information' section, descriptive summary information for password storage, password policy, and two-factor authentication are in code snippets. The software can learn summary information such as why the library is used in which situation and why we need hash and salt. The 'About' section contains information and contact information about which institute and unit the website was created. Thus, a password security website like Stackoverflow, which software can use quickly, has been prepared.

3. METHODOLOGY

As we mentioned in the previous section, while the LetsHashSalt website was being created, it was revealed that experienced programmers had difficulties writing secure code and could not write as securely as they thought, as Acar had shown before [55]. To solve this problem, Lisa created the LetsHashSalt website [23]. We have explained that this website has three different pages: *code snippets*, *information*, and *about* sections.

In this study, we will explain our improvements for the website to reach a wider audience of software developers when they implement software applications to make them more secure. The main goal of this study is to keep the security part of the code blocks at the maximum level instead of the functional part. We first selected Argon2id and Bcrypt in different programming languages in our study from these libraries. Then, we added the Pbkdf2 library and all the parameters in the next step. We have chosen the programming language's official and most used GitHub repository for this. Finally, we have placed the source code references of all these selected libraries under the code blocks of the website for the needs of the software developers.

We designed the survey questions that we asked straightforward, short, and detailed questions as possible. Thus, while the experts evaluated the code blocks, we could get clear feedback on code security. Five software engineers answered the survey questions, and we evaluated their suggestions by evaluation criteria, mainly focused on different resources.

As a result, we extended the existing LetsHashSalt website with six different language options (Python, Java, Javascript, PHP, GOLang, C#), different libraries for password storage (*argon2, bcrypt, and pbkdf2*), password policy libraries (*Regex* and *zxcvbn*) and two-factor authentication library (*OTP*). This section briefly gives information about the improved website version, selecting different libraries with reasons, survey questions, pilot study and study, evaluation criteria, ethics, and study limitations.

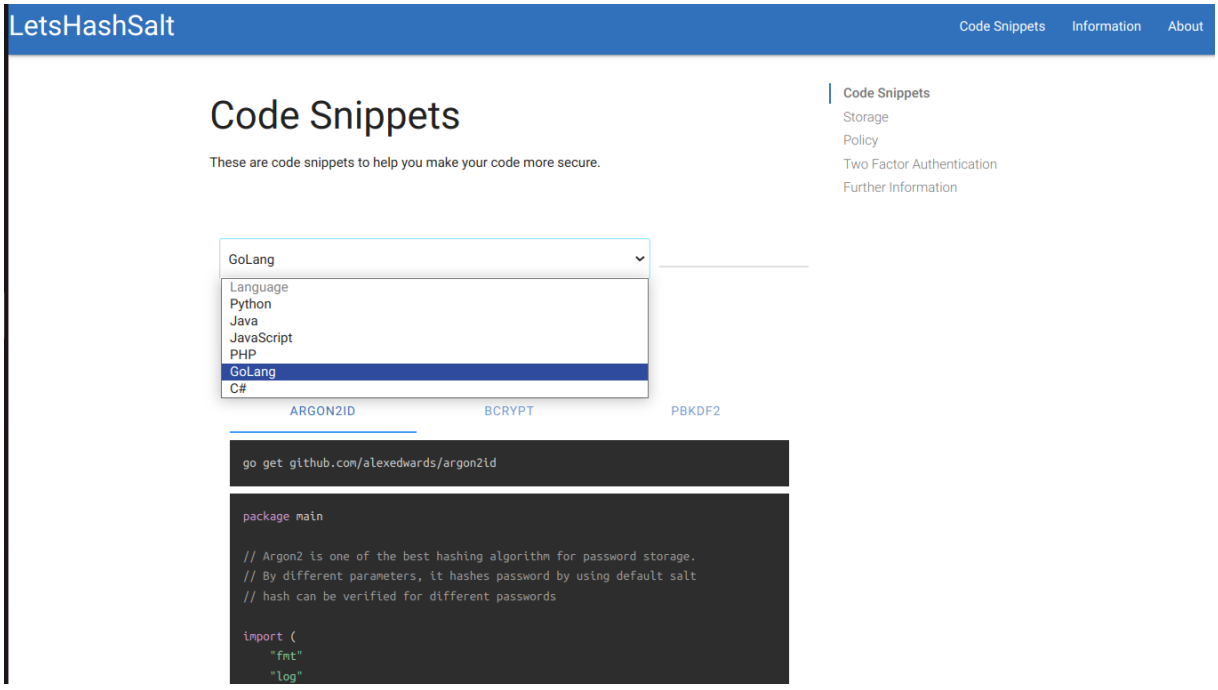
3.1. WEBSITE DESIGN

The LetsHashSalt website has two versions, with and without a wizard. As we explained in the previous section, since the effect of the wizard does not increase user usability, we decided to develop it on the second version without the wizard in this study. The general appearance of the website is very similar to the previous version that can be seen in Appendix A. Three different pages welcome the user. These are the *code snippets*, *information*, and *about* sections. In terms of user convenience, the image that appears when the website is opened for the first time is the *code snippets* section. A sidebar exists on the right side of the page, and in the middle are code blocks and information. Using the sidebar, the user can directly access the specific section on the page. Again, to facilitate use, different language options can be selected by using the button list at the beginning of the page. This selection covers all three different sections, password storage, password policy, and two-factor authentications. Please see Figure 3.1 for Website design.

Unlike the previous version, the programming languages used in all sections are six: *Python, Java, Javascript, PHP, GoLang, and C#*. The libraries used to make hash and salt are *Argon2id, Bcrypt, and Pbkdf2*. Please see Figure A.1 for password storage design. In the password policy section, instead of manually checking the password rulesets, they are checked using *Regex*, and the password strength is also checked with the *zxcvbn* library. Please see Figure A.2 for password policy design. In the two-factor authentication section, authentication is provided by using the *TOTP* library in *OTP*. Please see Figure A.3 for two factor authentication design. In addition, some necessary additions and changes have been made in the code over the previous version.

On top of the explanatory summary information written earlier for password storage, password policy, and two-factor authentication in the code snippets in the *Information* section, some information about which configurations different libraries will work better with has been added. In which case, why the library is used, the reasons for needing it, and reference information such as NIST and OWASP can be learned. The *About* section contains information and contact information about the institute and unit the website was created.

Figure 3.1.: Latest Website Design



3.1.1. Programming Languages Selection

The programming languages used in the previous version of LetsHashSalt were limited in number, as we mentioned in the previous sections. Therefore, we cannot directly say that the software languages are superior to each other because each language has emerged for a specific purpose. However, since this study aims to reach many different software developers, we wanted to highlight the use of the most popular and functional programming languages.

When we search for *'password storage'* on the Github website, which is one of the most used platforms in the software industry, the programming languages that come up with the most repository are Python, Java, javascript, C#, C++, PHP and GoLang [24] [25]. Regex is a syntax easily used on most platforms; since it is not directly related to the programming language, we can easily apply it to many languages [31]. So this syntax does not directly affect which programming language is used much. When we search for *'two-factor authentication'* on the Github website, the programming languages that come up the most are PHP, Javascript, Python, Java, C#, Ruby, HTML, and GoLang [26]. The importance of researching these languages is to ensure that the libraries we have identified give us preliminary information to see if they are supported and implemented in most programming languages.

One of the most important sites where software developers ask questions when researching on the internet is the Stackoverflow website, which is visited by more than 100 million monthly [75]. According to the information in the survey conducted by this website in 2022, some of the most popular programming languages in the eyes of 53,421 professional developers are JavaScript, HTML/CSS, SQL, Python, TypeScript, Java, C#, Bash/Shell, PHP, C++, C, PowerShell and Golang [76]. When we evaluated all this, including our programming knowledge, we decided to write the website in six different languages for all code snippet sections to reach the majority of programmers. These are Python, Java, Javascript, PHP, GoLang, and C#. All password storage, password policy and two-factor authentication for different language code blocks can be seen in the Appendix D.

3.1.2. PASSWORD STORAGE LIBRARIES SELECTION

There are many different password hashing / key derivation functions. The algorithms must make the hashing secure for the password to be stored, but some algorithms are fast and do not hash securely enough, such as *MD5* or *SHA-1*. So these kinds of hashing functions are not suitable for hashing. The four recommended hashing algorithms on OWASP's site are *Argon2id, Scrypt, Bcrypt, and Pbkdf2* [59].

The Argon2 algorithm is the winner of the 2015 Password Hashing Competition and is still the most preferred and recommended first hashing algorithm, it is written in many different languages, and the project is currently supported [14]. It is the top hashing algorithm recommended by OWASP. Argon2id, which has a balanced defense against side-channel and GPU-based attacks, is recommended for the password hash. Minimum memory size, the minimum number of iterations, and the degree of parallelism parameters should be configured for CPU and RAM usage (example: if minimum parameters of memory=15 MiB, iteration=2, and parallelism=1, then the password can be hashed and salted securely enough) [59].

Bcrypt is the password hashing function recommended by OWASP in case Argon2id is unavailable. The work factor should be min 10, and the maximum password length should be 72 bytes [59] [6]. Another algorithm, the Scrypt algorithm, is the password hashing function created by Colin Percival. It is an algorithm that uses high memory developed against large-scale custom hardware attacks [62]. The minimum CPU/memory cost parameter, the blocksize, and the degree of parallelism should be considered (example: if minimum parameters of cost parameter=2$\hat{1}$6 (64 MiB), block size=8 (1024 bytes), and parallelism=1, then the password can be hashed and salted securely enough) [59].

The Pbkdf2 algorithm is the password hashing algorithm recommended by NIST and should be used when FIPS-140 is needed [32]. This algorithm allows the use of internal algorithms such as *HMAC-SHA-256*, and iteration count is vital for the algorithm's security. For example, *PBKDF2-HMAC-SHA1* requires a minimum of 720,000 iterations, while *PBKDF2-HMAC-SHA512* requires 120,000 iterations. When using HMAC, if the password length is greater than the block size, it is automatically pre-hashed. Denial of service vulnerability may occur if the password is too long, so a pre-hash is required [59].

The required minimum criteria are set by default in many modifications, Argon2, Bcrypt, and other algorithms. Some required parameter changes are to change the parameters such as time_cost, memory_cost, parallelism parameters for Argon2, the round parameter for Bcrypt, block size, and iteration for Pbkdf2, according to the computer capacities of the software developers.

3.1.3. Password Policy Libraries Selections

The concept of the password policy is a set of rules created to apply the minimum criteria determined to increase users' account security. Hackers have done a great deal of damage because they guessed easy passwords many times in the past. In order to avoid creating easy passwords against brute force and similar attacks, the guidelines determined by OWASP and NIST are the minimum criteria generally used as a standard today.

Some of the password features recommended by NIST 2017 are as follows; Password must not contain consecutive characters. It should not be changed too often. It must be at least eight characters long, but subscriber-chosen passwords must be at least 64 characters. ASCII, space, and Unicode must be accepted. While creating the password, the system should advise the user with the password-strength meter to create a strong password [32]. The OWASP organization has similar recommendations; passwords must not contain consecutive characters. Password must be at least eight characters long. ASCII, space, and Unicode must be accepted. Frequently used words should not be used. It should not contain complex requirements but should be long enough.

Apart from these similarities, when we consider other items, the standard password policy features should have are as follows; password length should be between 8 and 64 characters long and support ASCII, space, and Unicode properties. In addition, the password must contain at least one uppercase letter, one lowercase letter, one number, and one unique character. Finally, the system should test other features, and the strength of the written password should be checked with the Password-strength meter and shown to the user.

Since all these are easy to write and supported on many platforms, we decided to write them with the regular expression syntax, which is universal [67]. We decided to use the *zxcvbn* library, which belongs to Dropbox and is currently one of the most used libraries, to measure password strength. One advantage of this library is that it is more secure, flexible, and usable and offers many different language support [85]. We selected the official and most used GitHub repositories in each programming language. We have placed the source code references of all these selected libraries under the code blocks of the website for the needs of the software developers.

3.1.4. Two-Factor Authentication Library Selection

Multi-factor authentication or two-factor authentication is the authentication method that allows the system to log in with two or more proofs from the user when entering platforms such as applications or websites. In general, only the user's information is verified over the randomly generated code within a particular time or counter.

According to OWASP and NIST's two-factor authentication guideline, verification can only be something the user knows (password or PIN), something they have (token or phone), or something that exists (biometric data such as a fingerprint) [32] [60]. The mechanism used to test this type of information is called a *One-time password (OTP)*. Many *OTP* libraries have two methods: *counter-based one-time passwords (HOTP)* and *time-based one-time passwords (TOTP)*. Since *TOTP* is frequently used in mobile applications, we decided to use it in the code snippet. Thanks to this, another layer of protection has been added for password security.

3.1.5. Adapting Code Snippets for Different Languages

As mentioned in the previous sections, the libraries we have determined for password storage hash/salt are *Argon2id, Bcrypt*. The *Pbkdf2* library is later added after survey is analyzed. For password policy, *Regex* and *zxcvbn*. For two-factor authentication, *TOTP* libraries are within

OTP. To adapt all these in different programming languages, we ensured that they are the most up-to-date, most common, and most recommended libraries for their security by OWASP and NIST. A library may have more than one repository; our priority is that the library should be official, secure (up-to-date), and have high Github usage and readability. After all this list of items, we decided to adapt the libraries we researched to six different programming languages.

Within the Python programming language, The libraries used are *argon2-cffi*, *bcrypt* and *pbkdf2* for password storage [72] [6] [48], *re* and *zxcvbn* for password policy [67] [86], and *pyotp* for two-factor authentication [46]. Within the Java programming language; The libraries used are *argon2-jvm*, *jbcrypt* and default *javax.crypto* for password storage [36] [38] [35], default *regex* and *com.nulabinc.zxcvbn* for password policy [58] [41], *com.j256.two-factor-auth* for two factor authentication [84]. Within the Javascript programming language; The libraries used are *argon2*, *bcrypt* and *crypto-js* for password storage [4] [11] [82] [65], *default regex* and *zxcvbn* for password policy [17], *totp-generator* and *otpauth* for two factor authentication [20] [8].

In PHP programming language; The libraries used are default *PASSWORD_ARGON2ID*, default *PASSWORD_BCRYPT* and default *hash_pbkdf2* for password storage [63] [64], default *regex* and *bjeavons/zxcvbn-php* for password policy [12] [43], and two factor authentication with *poles/php* [51]. In the GoLang programming language, libraries are *github.com/alexedwards/argon2id*, *golang.org/x/crypto/bcrypt* and *golang.org/x/crypto/pbkdf2* for password storage [18] [28] [29], default *regexp* and *github.com/nbutton23/zxcvbn-go* for password policy [30] [10], and *github.com/xlzd/gotp* for two-factor authentication [87]. In the C# programming language, The libraries used are *Konscious.Security.Cryptography.Argon2* and *BCrypt.Net-Next* for password storage [39] [9] [5] [7], default *System.Text.RegularExpressions* and *zxcvbn-core* for password policy [40] [70], and lastly *GoogleAuthenticator* for two-factor authentication [66].

While choosing the official ones of Argon2 and Bcrypt libraries, the repositories that we provide more up-to-date support in java language were selected. For GoLang, the *github.com/alexedwards/argon2id* repository has been chosen instead of the Argon2id official library because the readability is much easier, and it is not as complicated to write as the official source. Pbkdf2 was written using the default *crypto* libraries found in programming languages because this is the general spelling. These libraries were used because they have default *regex* boxes in many programming languages. Finally, the *zxcvbn* library is selected from the official documentation or the most up-to-date and popular repository. Due to time constraints, we could not add Pbkdf2 for C#, and we decided to use *GoogleAuthenticator* instead of OTP because it is one of the popular libraries.

3.2. Study Design

3.2.1. Survey Questions

The think-aloud method is a method that takes time to do and is, therefore, a more complicated process than direct surveys, but it is a more effective method for reading software developers' thoughts. However, due to time constraints and insufficient resources, we prepared survey questions that participants could easily reach and send to many people. Furthermore, we took care to keep the survey as short, understandable, and straightforward as possible so that the participants would not get bored. For this reason, we decided to ask questions directly related to the subject instead of asking unnecessary detailed questions that can be seen in Appendix A. After the Consent form, we asked the participant for some brief information about their past that can be seen in Appendix C. Then, the participant of ages, current occupations, years of experience, security experience, implementation knowledge about password storage, password policy, and two-factor authentication are asked. With this, we wanted to measure whether the participant had sufficient competence.

The primary purpose of the questions asked throughout the survey is *'is this code safe?'* and *'if not, how can it be strengthened?'*. After the user chooses one or two of the programming languages he knows best, the participant is welcome on the code snippet page. First, summary information briefly summarizes and explains the code content on the code block screen, followed by the code blocks and the questions.

Each password storage code block has two main methods, such as hash_password and verify methods. The first method hashes and salts the password, and the second method verifies whether the given hash is the same as the password hash. Some of the questions asked are as follows; *'Do you think the implementation of the hash_password method is up-to-date and secure?'* and *'Do you think the implementation of verifying is up-to-date and secure?'* While asking the questions, *'Would you make any other changes to this code snippet?'* With the question, we asked what kind of addition could be made to increase the security. Finally *'Did you use any additional resources while checking the code? If yes, please provide a link or description of the resource.'* With the question, we wanted to learn what resources the participant used and what different backgrounds he used. Similar question contents were asked within the password policy and two-factor authentication methods, and it was questioned how secure the contents of the methods were.

3.2.2. Participants

There are many ways to find professional software engineers from websites like Stackoverflow. Since the qualitative study is the best approach for our work, we primarily aim to measure the security of the codes we write and fix them with a qualitative study. We wanted the written codes to be reviewed by professional software developers who already work in companies with specific experience. That is why we decided that the best place to find participants was the Linkedin website. The LinkedIn website is one of the world's largest professional networking sites, with over 830 million users [47]. Since we could not find anyone to do code reviews for every programming language since the recruitment process is complex, we also asked code review questions on the Stackexchange website, similar to the Stackoverflow website [74].

Within the *'Cyber security'* searches on Linkedin, we sent an invitation to many professional software developers working in Germany, England, Netherlands, America, and many professional software developers. Most are German, British, Dutch, American, Indian, and Turkish. While 96 out of 172 did not respond to the message we sent, 76 people read the message we sent, and 45 started the survey, but only five completed our study's code review survey. On Stackoverflow, two different people reviewed the code blocks we wrote. The University paid 50

euros per person for one programming language review and 100 euros for two programming languages for those who participated via Linkedin. The participants' identities were kept confidential, as they participated by pseudo-name, not the real name. No payment was made to those who participated anonymously on Stackoverflow.

3.2.3. PILOT STUDY AND STUDY

Before asking questions to experienced software developers, three current graduate students were invited to the pilot study to avoid any problems in the survey, fix the mistakes, and create the right survey system. Without any intervention after the start of the survey, the participants finished the survey in about 15-20 minutes. Our aim was not to answer the questions but to measure whether they understood them correctly.

Some sentences in the introductory part of the questionnaire have been changed to be more precise. When we showed the code blocks first and then asked the questions, one participant stated that he could not understand the code content very well because he did not know the subject. For this, a summary paragraph was added to each code blog, which contains short information about the code content and libraries used and what they are used for in a way the participant can understand. This situation supported that it was more understandable to do it this way when asked by another participant. Again, when all the participants were asked whether an extra suggestion block box was necessary to express a different opinion at the end of the survey, they supported that it was better to add it at the end.

Thus, after the pilot study, the basic background questions at the beginning of the questionnaire were written in understandable sentences. The summary information before the code blocks in the code snippet section and the code block questions were short and understandable. The suggestion block question was added at the end of the questionnaire, and the questionnaire took its final form. All this survey work was done with the Qualtrics website, which is also used in university courses, has more than 2 million users, and is very easy to create a survey with its user-friendly interface [68].

After we found the experienced software developers on Linkedin, which we mentioned in the participant section, we sent a private survey link to the people with a personalized pseudonym. The answers of those who did not answer any question or left it halfway were not included in this study. When we look at the short answers given by these participants in the survey, some reasons for not participating include rejecting the consent form, not knowing about password security, or not wanting to give enough time for the survey. On the other hand, some participants did not click on the link or abandoned the survey after clicking.

In the Data and Analysis section of the survey website, all the answers given by the participants appear in detail. The answers given by the participants helped the code blocks develop more with the answers they gave about the general purpose of the survey, what the problems were in the security parts of the code blocks, and how to improve them. The participants understood the questions asked in general terms correctly and answered accordingly.

3.2.4. EVALUATION CRITERIA

Although this study aims to apply professional software developers' suggestions for code review, it is essential to add secure and up-to-date recommendations after evaluating them according to specific criteria. The criteria we used to evaluate the written code blocks is the ISO usability specification, and we decided to interpret it by taking into account the three critical items, effectiveness, efficiency, and satisfaction [49]. Therefore, we have ensured that the code blocks

in the questionnaire are as short and understandable as possible and that the satisfaction and efficiency are high. For this reason, we tried to understand the differences to be made by the participants by looking at the difference in the code comments they made on these two items.

We need to focus on effectiveness, including code functionality and security. However, since the primary purpose of this study is to increase code security rather than functionality, we did not include the functional part in the code changes. However, we still mentioned it. The criteria we use when evaluating the security adequacy of code blocks are the resources we use when choosing password storage, password policy, and two-factor authentication libraries, as we mentioned in the previous sections, especially the OWASP and NIST standards [59] [32].

Some of the crucial items we look at in the code recommendations written by the participants are as follows;

Password storage:

- Suggesting the use of hash/salt instead of encryption
- Recommending the use of *Argon2, Bcrypt, Scrypt, and Pbkdf2* libraries or giving details of the other proposed library
- Recommending OWASP and NIST minimum recommendations for the parameter configurations of the libraries (Random salt, 32-bit salt length, etc.)
- extra changes in code content

Password policy:

- Application of minimum criteria while creating a password (one upper/lower case, one special case, etc.)
- The way *Regex and zxcvbn* libraries are used or giving details of the other proposed library
- extra changes in code content

Two-factor authentication:

- Using *TOTP or HOTP* libraries in the *OTP* library or giving details of the other proposed library
- extra changes in code content

On top of these evaluations, we decided to add the advice of professional software developers on the existing code blocks to extend code changes. However, we did not include the irrelevant and unnecessary recommendations beyond these criteria into the code blocks.

3.2.5. Limitations

Since this study had to be done among professional software developers, we had to find people working online in Germany and the world under Covid-19 restrictions. Therefore, since the study was conducted online, we could not directly observe the analyzes of the participants and ask direct questions. Furthermore, the fact that the study was an online survey and the participation fee was low for software developers, that the number of participants was as few as five, or that many people who started the survey left the survey in half. Therefore, participation fees should be higher in the following studies, and lab studies can yield better results if possible.

For a qualitative study to be practical, the data collected must be sufficient and generalizable. Unfortunately, although the backgrounds of the developers found on Linkedin are sufficient, the number of participants is limited to five is not sufficient to generalize the accuracy of the code blocks already written. Therefore, finding more participants in future studies is even more critical for the study's accuracy.

Although many programming languages are vital for the study, most participants answered for the well-known Python or Java languages. In contrast, for some other programming languages, we could not find participants for code review, so we are not sure of the correctness of these code contents. Furthermore, the fact that there are different interpretations for different programming languages has also led to different levels of security code analysis for each programming language.

The length of the study is 20-40 minutes without getting tired. The questions were asked in a way that would increase code security with very few questions, and no time limit was set. Despite this, only five of the forty-five people who clicked on the survey completed the survey. One of the reasons for this may be that the participants thought the survey was long.

3.2.6. ETHICS

Our work has been conducted under German privacy regulations and EU General Data Protection Regulations. All data, including contact information and name, are stored and taken anonymously. The participants were informed in detail in the consent form before starting the survey. Participants have the right to discontinue the survey at any time.

4. RESULTS

The total number of participants participating in the survey is five people, and the background features are as given in the Table 4.1. While Participant-1 and Participant-2 only made code reviews in Python languages, Participant-3 made code revisions in Python and Golang languages. Participant-4 and Participant-5 made priors in Python and Java languages. Although Javascript, PHP, and C# languages are popular as we mentioned in the previous section, it can be said that these languages are not as popular as Python and Java in the security field. Among the libraries suggested by the participants, apart from Argon2 and Bcrypt, the Pbkdf2 library is also recommended. The use of the regular expression, the use of zxcvbn, and TOTP libraries were found positive by the participants within the framework of safety. In this section, we tried to focus on the details of the code analysis of the participants.

The participants are between the ages of 25-32, and their titles are Cyber Security Engineer, Application Security Engineer, and Penetration Tester. Their professional experience ranges from 2-4 years and all participants are familiar with the implementation of password storage, password policy, and two-factor authentication. The interpretation similarities obtained in these studies and the differences requiring code changes were re-applied to other programming languages by bringing them into comments.

Table 4.1.: Participants Background Information

Participant Background Information					
Information	P1	P2	P3	P4	P5
Age	26	32	26	25	28
Occupation	Msc Student	Cyber Security Engineer	Cyber Security Engineer	Application Security Engineer	Penetration Tester / Java Developer
Years of experience	4	2	3	4	2
Security knowledge	yes	yes	yes	yes	yes
Pwd storage knowledge	Moderate	Very high	Moderate	Very high	Extremely high
Pwd policy knowledge	Very high	Extremely high	Moderate	Extremely high	Extremely high
2FA knowledge	Very high	Extremely high	Very high	Extremely high	Extremely high
languages	Python	Python	Python and Golang	Python and Java	Python and Java

P1: Participant-1, P2: Participant-2, P3: Participant-3, P4: Participant-4, P5: Participant-5

4.1. NOT FUNCTIONALITY BUT SECURITY

The primary purpose of this study is to examine security differences instead of functionality in existing code snippets, thus maximizing code security. Functionality may vary depending on the platform, code content, parameters, and variables. However, the security part should be

updated and presented to the developers in its most potent form. That is why we asked users which libraries and parameters or changes should be made for password storage, password policy, and two-factor authentication.
The participants are informed that they could use any source they wanted before proceeding to the survey questions, and there was no time limit. In addition, we stated that they should mark the questions they are unsure about, so we aimed to create the most accurate and safe code blocks so they can take it to the maximum level from a professional point of view.

4.2. PASSWORD STORAGE EVALUATION

The answers given in the code review by participant-1, participant-2, and participant-3 to the password storage survey questions can be seen in the Table 4.2, the answers of participant-4 and participant-5 can be seen in the Table 4.3.
Two of participants stated that Argon2 is decent choice for hashing function. Participant-1 states as following;

> "Argon2 is ok as well known cryptographic password hashing algorithm".

Participant-3 mentioned the advantages and disadvantages between Argon2d, Argon2i and Argon2id variants over each other and Argon2id is the right choice for password security.

> "Argon2id is a hybrid of Argon2i and Argon2d, using a combination of data depending and data-independent memory accesses, which gives some of Argon2i's resistance to side-channel cache timing attacks and much of Argon2d's resistance to GPU cracking attacks. That makes it the preferred type for password hashing and password-based key derivation."

> "The RFC also says that Argon2id resists timing attacks, so if you don't know which one to use, you should choose it."

Participant-3 additionally proposes the pyrgon library instead of the argon2 library;

> "You cant specify a salt argument with argon2 library. hash_password function use randomly salt value while hash generation. If you want more controle on salt you can use pyargon2 library."

Participant-4 and Participant-5 only ticked the options where the use of Argon2 was correct without making any comments in the survey.

When we look at the sources used for Argon2, Participant-1 did not specify any source, while participant-2 wrote "my repository" and participant 5 stated "StackOverflow". However, Participant-3 and participant-4 used many resources while writing the code and these can be seen in the Table E.1. Participant-3 uses the source of the official argon2-cffi library for content and comparison of the Python Argon2 library, while for GoLang, he uses the official argon2 library for Argon2. He recommends looking at the source code before used as extra information.

> "You used argon2id library. You can choose to use official library and their function: https://pkg.go.dev/golang.org/x/crypto/argon2#IDKey"

The sources used by Participant-4 for Argon2 are to measure whether the used libraries have vulnerability and prefer to use the library according to the reliability level.

> "Make sure all your libraries are not vulnerability affected. In this case, you can find below the current version. If you use a High/Critical affected library, it can be dangerous to your software. You can also use open-source tools like Owasp dependency check and Snyk to get flaws of your libraries."

Participant-2 did not recommend any of the Argon2 or Bcrypt libraries, as they recommended using encryption instead of hash and salt method in password storage.

> "its also convertible to original value of password, salting is a basic algorithm, it should be encrypted"

Many participants stated that the Bcrypt library is used correctly. Participant-1, Participant-3, and Participant-4 did not comment on the security part of the Bcrypt library and they stated that it was implemented correctly. Participant-2 indicated different libraries instead of Bcrypt.

> "instead of Bcrypt, i'd prefer to use below ones: PyCryptodome, Criptography, PyNaCl"

Participant-5 did not comment on Bcrypt, but suggested using the Pbkdf2 library instead.

> "NIST has issued Special Publication SP 800-132 on the subject of storing hashed passwords. They recommend PBKDF2. This doesn't mean that they deem bcrypt insecure, just nothing says about bcrypt."

The resources provided by participant-3 show both official and sample tutorial content that uses Bcrypt, while participant-4 measures the current situations of the libraries used against vulnerabilities. participant-4 defines as follows;

> "I use the link to ensure the library version is not affected by any significant flaw."

The important point here is that while participant-3 controls the resources for the Python and Golang libraries, Participant-4 controls the resources of the java plain and spring libraries in addition to Python libraries.

If we look at it from another angle, some participants also made some suggestions for functionality changes in password storage code blocks, apart from security. Participant-1 proposed a rate-limit mechanism for the password.

> "Adding a rate limiting mechanism to preserve brute forcing, dictionary attacks"

Participant-3 suggested not to use hard-coded arguments.

> "Not necessary use of arguments. We are using code linter tools in CI/CD process. There are bound to be overlooked bugs, unnecessary operations and code improvements."

Participant-4 repeated the same advice. He suggested to use the tool instead of hard-coded password example.

> "You can use systems like HashiCorp Vault (called a secret management process) or at least get your input and stored as variable."

> "This recommendation can apply to all code snippets to make sure that you don't have any hard-code credential/password."

Since the functional part is not our immediate priority in all evaluations, the points we will focus on will be the security part. As a result of the evaluation, we took decisions as follows;

Participant-1 evaluation;

- We decided not to make any changes because the selection and code content of the Argon2 and Bcrypt libraries is correct
- We ignored the addition of the rate.limiting mechanism as it is a functional change

Participant-2 evaluation;

- OWASP stated that hash and salt should be used instead of encryption when storing passwords [59]. That is why participant interpretation is wrong
- Instead of using the Python bcrypt library, PyCryptodome, Criptography, PyNaCl libraries are recommended. However, the official bcrypt library resource says it is sufficiently secure and up-to-date, so we keep the official library [6]

Participant-3 evaluation;

- In Python, 'pyargon2' usage is suggested instead of 'argon2' library to provide more parameter control for Argon2 content, but as it can be seen in the official source, parameter changes can be made within the official library, so we did not consider this recommendation [14]
- It is an excellent recommendation to use the official 'crypto/argon2' library for GoLang, which we have evaluated. However, we decided to continue using the alexedwards' argon2id library, which is popular on Github and has high readability for software developers who do not have high-security knowledge [18]
- No change was needed for the Bcrypt library since it is strong enough
- We decided not to add the Code linter tools recommendation because it is functional

Participant-4 evaluation;

- The use of Argon2 and Bcrypt libraries is correct and safe
- We checked whether the libraries used for Java are safe with the mvnrepository and Snyk websites, and we saw that the latest versions are safe. At the same time, there is a vulnerability in outdated libraries [52] [77]
- We did not consider using the secret management tool because it is functional

Participant-5 evaluation;

- Argon2 is safe to use, but also Bcrypt is a safe library recommended by OWASP [59]
- Also, since Pbkdf2 is the library recommended by NIST and this participant recommended it, we decided to add the Pbkdf2 library to the website [32]

Table 4.2.: Password Storage: Participant 1, 2 and 3 Answers

Participant 1, 2 and 3 Survey Question and Answers				
Password Storage Argon2 Question	P1:pyt	P2:pyt	P3:pyt	P3:go
Do you think the implementation of hash_password is up-to-date and secure?	not sure	no	not sure	not sure
Do you think the implementation of verify is up-to-date and secure?	yes	no	yes	not sure
Would you make any other changes to this code snippet?	yes	yes	no	yes
Did you use any additional resources while checking the code?	no	yes	yes	yes
Password Storage Bcrypt Question				
Do you think the implementation of hash_password is up-to-date and secure	yes	no	yes	yes
Do you think the implementation of verify is up-to-date and secure?	yes	no	yes	yes
Would you make any other changes to this code snippet?	yes	yes	no	yes
Did you use any additional resources while checking the code?	no	no	yes	yes

P1: Participant-1, P2: Participant-2, P3: Participant-3, pyt: Python, go: Golang

Table 4.3.: Password Storage: Participant 4 and 5 Answers

Participant 4 and 5 Survey Question and Answers				
Password Storage Argon2 Question	P4:pyt	P4:java pln/spr	P5:pyt	P5:java pln/spr
Do you think the implementation of hash_password is up-to-date and secure?	yes	yes/yes	yes	yes/not sure
Do you think the implementation of verify is up-to-date and secure?	yes	yes/yes	yes	yes/yes
Would you make any other changes to this code snippet?	no	no/no	no	no/no
Did you use any additional resources while checking the code?	yes	yes/yes	yes	no/no
Password Storage Bcrypt Question				
Do you think the implementation of hash_password is up-to-date and secure	yes	yes/yes	not sure	yes/not sure
Do you think the implementation of verify is up-to-date and secure?	yes	yes/yes	not sure	yes/not sure
Would you make any other changes to this code snippet?	yes	no/no	no	no/no
Did you use any additional resources while checking the code?	no	yes/yes	no	no/no

P4: Participant-4, P5: Participant-5, pyt: Python, java pln/spr: Java plain/spring

4.3. PASSWORD POLICY EVALUATION

The answers given in the code review by all participants to the password policy survey questions can be seen in the Table 4.4.

Participant-1 made a comment on the use of Regex and zxcvbn libraries for password policy states "not sure", while Participant-4 states that the use of regex is correct, however, the use of zxcvbn-python library used in the PyPI library is not correct, whereas he did not specify the reason for this suggestion.

While Participant-2 did not state that the use of Regex and zxcvbn was wrong, Participant-3 and Participant-5 stated that there were errors in the use of Regex and it should be changed. Participant 3 states the following for Python review;

> "number = "(?=.* \d)" complex usage and returns a lot of null matches. Maybe this regex is more simple, useful, and effective: number = "[0-9]". "

Participant 3 mentions the following for Golang review;

> "This regex always matched: .*[^\\d \\w].
> Try this: .*[^A-Za-z0-9]"

Participant-5 states that there is an error in the Regex for Python however he did not specify what the error is. He only typed the following;

> "number = "(?=.* \d)"
> upper_lower = "(?=.*[a-z])(?=.*[A-Z])"
> special = "[^A-Za-z0-9]"
> length = ".{8,64}$". "

In contrast, Participant-5 states that Regex for Java is true. All participants defined that implementation of zxcvbn library is true.

Participant-1, Participant-2, and Participant-5 did not provide any source for password policy. Participant-3 used official regular expression resources, as well as websites with policy and regex, consisting of tutorials and blog posts. Participant-5, on the other hand, used websites that checked the zxcvbn library for vulnerability. The resources used by both participants can be seen in the Table 4.1.

Furthermore, some participants also made some suggestions for functionality changes in password policy code blocks, apart from the security side. Participant-2 proposed a restriction process for password attempts.

> "I will totally add a procedure for restricting incorrect password attempts as a finite loop"

> "It can be exploitable via buffer-overflow"

Participant-4 suggested secret management tools or user input password examples.

> "You can use user input to make it more realistic and not be hard-coded."

> "String password = "Abcdefghi1.";
> It's good to leave it as hard-coded as an example. Use secret management software in a production environment to safely store your passwords."

Participants evaluation;

- As we mentioned earlier, we did not add the functional changes to the code snippets.
- Participants did not comment on whether the Regex and zxcvbn libraries were the right choice, so we cannot comment on the selection of these libraries.
- Following the advice of Participant-3 and Participant-5, we applied the given changes in Regex usage to code blocks.
- We found Participant-4's recommendation not to use the "zxcvbn-python" library is not applicable because this library is recommended by the official PyPI [86].
- It is not recommended to add or remove any of at least one lowercase character, one uppercase character, one number, one special character, and the 8-64 password length attributes. Therefore, we did not need to apply any other existing changes to the code content.

Table 4.4.: Password Policy Answers

Participant Survey Question and Answers					
Password Policy Question	P1	P2	P3:p/g	P4:p/j	P5:p/j
Do you think the implementation of this code snippet is up-to-date and secure?	not sure	no	yes/yes	not sure/no	no/yes
Would you make any other changes to this code snippet?	no	yes	yes/yes	no/yes	no/no
Did you use any additional resources while checking the code?	no	no	yes/yes	yes/yes	no/no

P1: Participant-1, P2: Participant-2, P3: Participant-3, P4: Participant-4, P5: Participant-5, p: Python, g: go, java: Java

4.4. Two Factor Authentication Evaluation

The answers given in the code review by all participants to the two-factor authentication survey questions can be seen in the Table 4.5.

Participant-1, Participant-2, and Participant-5 stated that the implementation of the TOTP library belonging to the OTP library inside the two-factor authentication code blocks is completely correct for the Python and Java programming languages. However, they did not provide any resources or detailed information about their choices. We can say that they found the code blocks given in the two-factor authentication section to be sufficiently secure and does not need any changes.

Participant 3 states that different libraries could be used for alternative situations for two-factor authentication. However, the code blocks written with 'pyotp' are safe enough. The comment made by the participant is as follows;

> "It is a basic usage of pyotp. If you want more security you must search alternative libraries but pyotp is common and accepted by most developers. This is also valid for the other code examples I have mentioned."

He also found the GoLang code blocks sufficiently secure, similar to his previous comment. He stated that TOTP and HOTP usages are sufficient, but TOTP is generally used.

> 'The HOTP and TOTP standards are produced by OATH, the Initiative for Open Authentication. Generally used TOTP.'

Participant 4 did not make any different markings for the two-factor authentication section and stated that the given code blocks were sufficiently secure. However he commented on the java programming language that the part given for the example password for the function should be changed.

> 'We have to store securely secret-key to the database, we can also use Vault to improve the security storage mechanism'

Participant-3 has suggested general security checker websites instead of resources for OTP usage. Participant-4 uses the library we use for the otp library in java language and he only used the tutorial source for Python instead of an official source that can be seen in Table E.1.

Participants evaluation;

- For the functional advises given by the participants, we did not to make any changes in the code content like use of the Vault tool.
- Many participants did not comment on whether the OTP and TOTP libraries were the right choice. Only Participant-3 acknowledged our choice for the TOTP library.
- It is not recommended to add or remove any of the content we applied for the two-factor authentication. Therefore, we decided not to change anything about code content.

Table 4.5.: Two Factor Authentication Answers

Participant Survey Question and Answers					
Two Factor Authentication Question	P1	P2	P3:p/g	P4:p/j	P5:p/j
Do you think the implementation of this code snippet is up-to-date and secure?	yes	yes	not sure/yes	yes/not sure	yes/yes
Would you make any other changes to this code snippet?	no	no	no/yes	no/no	no/no
Did you use any additional resources while checking the code?	no	no	yes/yes	yes/yes	no/no

P1: Participant-1, P2: Participant-2, P3: Participant-3, P4: Participant-4, P5: Participant-5, p: Python, g: go, java: Java

The resources used by the participants are given the the additional Table E.1.

4.5. Evaluations of StackExchange answers

Apart from these questions, we also asked questions in the code review section of the Stackexchange website to get extra code review suggestions [74]. For example, some recommendations are the password storage with Argon2 in Python and the password policy with a regex and zxcvbn libraries.

Participant 1: Based on the advice of the first Stackexchange contributor, Argon2 is the right choice, but he recommended some additions to the code blog [44].
Some of the functional recommendations are as follows:

- change in use of return

  > 'I'd return the hash immediately instead of assigning it to a variable. I'd avoid an explicit return None from a function which just drops off the end'

- use of parallelism

  > '... running it in an already multi-threaded environment like many web servers, I'd be tempted to dial back the parallelism parameter to the PasswordHasher object.'

The recommendation for security is to use the 'check_needs_rehash' function.

> '... call check_needs_rehash alongside verify, so that may want exploring. It's a key part of ensuring that you stay up-to-date and secure.'

Participant 2: The second Stackexchange participant did not state a sentence about the use of zxcvbn and the security part of the Regex in any way [22].
The functional recommendations given in the password policy are as follows;

- 'Elevate the data that drives the code to the status of named constants.'
- 'The regular expressions are overly complex and maybe naive'
- 'Return more declarative data'
- 'Consider unifying the checks.'
- 'Get in the habit of parameterizing the main function'

Among these recommendations, the second item that should be paid attention to is the use of Regex. The participant mentioned that Regex is limited to ASCII characters and has a complex structure [22]. However, we decided to continue using Regex as we thought that using Regex would not cause any problems in the framework recommended by OWASP. In addition, we decided to add the functional parts that increase the readability of the code for the password policy part.

5. DISCUSSION

In order to create a website that many software developers can benefit from and to maximize the security and up-to-date code blocks, we tried to delve deeper into the recommendations and thoughts of experienced software developers with a qualitative approach. So we continued our work by participating in our code review survey of five professional software developers we found on Linkedin and two participants commenting on the Stackexchange website. Although we mentioned to participants that they should focus on security instead of functionality while doing code reviews at the beginning of the survey, many of them made functional recommendations. Although most of these recommendations are helpful when writing code in practice, we decided not to put them on the website for this study except password policy part because including functional changes will increase readability effectively.

While all five participants made code review for Python language, two participants did code reviews in Java language, and one participant did a code review for the GoLang programming language. Unfortunately, since we could not find participants for Javascript, PHP, and C# languages, we could not directly apply the changes in the code contents. To overcome this issue, we have decided to select the same library selections and apply similar changes in the code blocks for these languages as we applied in Python, Java, and GoLang by trying to infer. There can be several reasons that we could not find professionals in the recruiting process for these languages. For example, many people know Python and Java programming languages in the password security field, but fewer people dominate the other languages. Another reason can experts in other fields are not ready to attend code review surveys as Python and Java experts. The library selections are similarly based on the experts' resources they used for these languages. Inference of the code content is based on code structure, and library versions experts used.

Three participants supported the usage of Argon2 and Bcrypt libraries in password storage; one participant suggested the usage of Argon2 and stated that it would be more accurate to use Pbkdf2 instead of Bcrypt. No participant suggested MD5 or SHA256, which are weak hash functions. One participant suggested using encryption instead of hash and salt functions. However, as mentioned in the previous section, OWASP explains the reasons for using hash and salt instead of encryption for password storage. None of the participants commented on the parameters used in the libraries, and they confirmed that the default parameters given for each programming language are secure enough. Participants commented on whether the libraries used were up-to-date [52] [77]. There can be several reasons for not mentioning the parameter changes inside the code content. Firstly, experts think default parameters are already secure enough to hash and salt passwords. Secondly, experts ignored details and only checked the code structure. Either way, we assumed that using the default parameters is strong enough and should be changed according to developers' needs.

For password policy, most participants did not mention that using regex and zxcvbn libraries is a genuine choice. The main reason can be that regular expressions are well-known libraries and widely used syntax; zxcvbn is the official dropbox library that shows it is sufficient. However, they did not mention other libraries, so it is still open question to be the proper choice of libraries. Using at least one uppercase character, one lowercase character, one unique character, and one number checked in the given password is considered valid to check the minimum password features. We have made some functional changes for readability, such that one participant suggested changing the primary function content, using a dictionary for the composition method, and regulating the check_strength method. The participants have stated that the code

blocks are safe enough, except for some minor changes in using the regex library. However, the participants added check mechanisms that needed to be corrected due to functional deficiencies. Some were options, such as adding a restriction with a password loop or using a password manager instead of a hardcoded password. For two-factor authentication, all participants found the use and code content of the TOTP library safe and do not change anything. Even if we did not apply any change in this section, we think that experts have only limited knowledge that does not know the details of the two-factor authentication mechanism. Another reason can be that they did not pay enough attention to the code content quality because this section was the last in the survey, which may lead experts to pass fast.
The resources that experts use vary that should be scrutinized. For example, participants 1, 2, and 5 did not use any resources, so they used their background knowledge directly, which may present their professional views or misremembering thoughts. Because they did not specify any resources, we cannot interpret the quality. Participant 2 used seven resources for the Python language. He used three official websites for each library except regex and four tutorial websites. Participant 2 furthermore used eleven resources for the Golang language. He used one official Argon2 library website and one official regular expression library website. Five resources are tutorial websites, whereas four are code review websites to check code quality, giving a different perspective. Almost all official resources are the same resources we used. However, tutorial resources are different, so it helps us to examine different examples for code content when we apply his suggestions. Participant 4 used one official website for Argon2 and two tutorial websites for the rest of the libraries for Python language. He used ten resources for the Java language in total, where two out of three official websites are different resources he used from our resources. Two resources are tutorial websites, and two resources are functional tutorial websites. Three resources are about library check websites. These are important because we can also check anytime whether current versions of the libraries have any vulnerabilities. Overall, these insights added different check perspectives to increase code content in terms of security highly. The resources can be seen in the Table E.1.

To conclude, we added the Pbkdf2 library next to the use of Argon2 and Bcrypt. Finally, we continued to use the regex, zxcvbn, and TOTP libraries as they are, but we made small changes in the code content. Thus, we have prepared three code block final versions after regulating the resources expert used and their recommendations on the survey.

6. CONCLUSION

As demonstrated in many previous studies, preventing software developers from making mistakes while ensuring password security and helping them write secure code is an important idea. One of the most critical problems that ensure that software developers cannot write secure code is the difficulty in finding reliable and up-to-date resources. Geierhaas et al. website work is an essential and helpful tool to solve the resource problem. The website contains basic source codes, including password storage, password policy, and two-factor authentication code snippets.

In this study, we decided to continue the study only with the standard version since there was no significant difference between the usability of the wizard version of the existing LetsHashSalt website and the usability of the standard version. Therefore, we have updated the content of programming languages Python and Java on the website. In addition to adding diverse programming languages such as Javascript, PHP, Golang, and C#, we have added new libraries such as Pbkdf2, and Regex. Furthermore, we reviewed the most up-to-date and secure methods available for the website and edited the code snippets for the use of software developers.

After reaching out to professional software developers, we conducted a code review survey with five people and two anonymous people's recommendations on the StackExchange website. The results we obtained as a result of the study are as follows; Using the hash and salt method instead of encryption is the right choice when performing password storage. Although the use of Argon2 and Bcrypt libraries is the right choice, the Pbkdf2 library is the library that should be preferred in any case. No recommendations are given for the use of Scrypt or other similar libraries. It is the right choice to use general Regex syntax for password policy and to measure password strength with the zxcvbn library. Having at least one uppercase, one lowercase letter, one special character, and one number in the password content having a length of 8-64 are sufficient criteria. Therefore, when using Regex, it is crucial to ensure it covers all these features. Two-factor authentication is a standard process used in many platforms, and it is vital to use it for final auditing. For this process, it is generally the right choice to use TOTP or HOTP methods within the OTP library.

The primary purpose of this study is to focus on the security part, so although the recommendations for functional changes are not applied to the code content, some recommended examples are as follows; Some recommended password storage changes are checking passwords with code linter tools and checking passwords with secret management processes like HashiCorp Vault. Some of the recommended changes in the password policy are restricting incorrect password attempts with loops and adding the most common password list. No changes are recommended in two-factor authentication. In conclusion, the existing programming languages and the libraries' contents have been changed due to the evaluations.

7. FUTURE WORK

While the personal accounts of millions of people on many different platforms are protected with passwords, password security will continue to be very important in our lives in the coming years. While it is imperative to help the software developers in different ways while providing this, many new features and differences can be put into this work in the future.
For this website, we currently use three different libraries for password storage in six different languages, two different libraries for password policy, and one for two-factor authentication. In this study, the number of software developers controlling the code blocks was limited to five. What needs to be done in the following study is that at least two people who look at each programming language from different points of view should control it. If possible, these people should have at least five years of experience and have worked specifically for password security. Programmers who check code in this way will make more accurate suggestions.
In addition, if the participants do a code review with the think-aloud method, when there are no Covid-19 and time restrictions, it will help for a more detailed evaluation. These details will provide a broad perspective, such as utilizing different libraries for different platforms or comparing libraries' specific parameters. Furthermore, performance and memory comparisons of library usage on different platforms can be valuable for software developers. In addition, the study will become a more effective and valid website if more extensive studies are carried out to increase functionality and usability besides security. For this reason, additions can be made in different ways from different perspectives, and the website can be more practical.

BIBLIOGRAPHY

[1] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016. doi: 10.1109/SP.2016.25.

[2] Y. Acar, S. Fahl, and M. L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8, 2016. doi: 10.1109/SecDev.2016.013.

[3] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl. Security developer studies with GitHub users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, Santa Clara, CA, July 2017. USENIX Association. ISBN 978-1-931971-39-3. URL https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar.

[4] R. Althoff. node-argon2. https://github.com/ranisalt/node-argon2, 2022.

[5] K. Aragon. Konscious.security.cryptography. https://github.com/kmaragon/Konscious.Security.Cryptography, 2022.

[6] P. C. Authority. bcrypt. https://github.com/pyca/bcrypt, 2022.

[7] BCrypt.net. bcrypt.net. https://github.com/BcryptNet/bcrypt.net, 2022.

[8] M. Bellstrand. totp-generator. https://github.com/bellstrand/totp-generator, 2022.

[9] B. Burman. How to use argon2 for password hashing in c#. URL https://www.twelve21.io/how-to-use-argon2-for-password-hashing-in-csharp.

[10] N. Button. zxcvbn-go. https://github.com/nbutton23/zxcvbn-go, 2021.

[11] N. Campbell. node.bcrypt.js. https://github.com/kelektiv/node.bcrypt.js, 2021.

[12] CodexWorld. How to validate password strength in php. URL https://www.codexworld.com/how-to/validate-password-strength-in-php/#:~:text=Password%20must%20be%20at%20least,at%20least%20one%20special%20character.

[13] J. Colnago, S. Devlin, M. Oates, C. Swoopes, L. Bauer, L. Cranor, and N. Christin. "it's not actually that horrible": Exploring adoption of two-factor authentication at a university. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3174030. URL https://doi.org/10.1145/3173574.3174030.

[14] P. H. Competition. phc-winner-argon2. https://github.com/P-H-C/phc-winner-argon2, 2019.

[15] N. Cubrilovic. Rockyou hack: From bad to worse. URL https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/.

[16] M. Dell'Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010. doi: 10.1109/INFCOM.2010.5461951.

[17] Dropbox. zxcvbn. https://github.com/dropbox/zxcvbn, 2017.

[18] A. Edwards. argon2id. https://github.com/alexedwards/argon2id, 2021.

[19] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, CCS '13, page 73–84, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450324779. doi: 10.1145/2508859.2516693. URL https://doi.org/10.1145/2508859.2516693.

[20] H. M. Fernández. otpauth. https://github.com/hectorm/otpauth, 2022.

[21] D. Florêncio, C. Herley, and P. C. Van Oorschot. An administrator's guide to internet password research. In *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, LISA'14, page 35–52, USA, 2014. USENIX Association. ISBN 9781931971171.

[22] FMc. Password policy with regex and zxcvbn. URL https://codereview.stackexchange.com/questions/277198/password-policy-with-regex-and-zxcvbn.

[23] L. Geierhaas, A.-M. Ortloff, M. Smith, and A. Naiakshina. Let's hash: Helping developers with password security. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, pages 503–522, Boston, MA, Aug. 2022. USENIX Association. ISBN 978-1-939133-30-4. URL https://www.usenix.org/conference/soups2022/presentation/geierhaas.

[24] GitHub. Let's build from here, . URL https://github.com/about.

[25] GitHub. Repository results, . URL https://github.com/search?q=password+storage.

[26] GitHub. Repository results, . URL https://github.com/search?q=two+factor+authentication.

[27] Foundation. About OWASP. online, 2013. URL https://www.owasp.org/index.php/About_OWASP. [Accesssed online 04-11-2013].

[28] Go. bcrypt, . URL https://pkg.go.dev/golang.org/x/crypto/bcrypt.

[29] Go. pbkdf2, . URL https://pkg.go.dev/golang.org/x/crypto@v0.0.0-20220926161630-eccd6366d1be/pbkdf2.

[30] Go. regexp, . URL https://pkg.go.dev/regexp.

[31] J. Goyvaerts. Regular expressions reference. URL https://www.regular-expressions.info/reference.html.

[32] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, and J. P. Richer. Digital identity guidelines authentication and lifecycle management section 5.1.1.2. URL https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-63b.pdf.

[33] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security Privacy*, 14(5):40–46, 2016. doi: 10.1109/MSP.2016.111.

[34] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security Privacy*, 14(5):40–46, September 2016. ISSN 1558-4046. doi: 10.1109/MSP.2016.111. Conference Name: IEEE Security Privacy.

[35] L. Gupta. Java – create a secure password hash. URL https://howtodoinjava.com/java/java-security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples.

[36] M. Halbritter. argon2-jvm. https://github.com/phxql/argon2-jvm, 2021.

[37] M. Hill and D. Swinhoe. The 15 biggest data breaches of the 21st century. URL https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html.

[38] J. Hooke. jbcrypt. https://github.com/jeremyh/jBCrypt, 2015.

[39] M. Ignite. Install the .net sdk or the .net runtime on ubuntu, . URL https://learn.microsoft.com/en-us/dotnet/core/install/linux-ubuntu.

[40] M. Ignite. Regex class, . URL https://learn.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regex?view=net-6.0.

[41] N. Inc. zxcvbn4j. https://github.com/nulab/zxcvbn4j, 2013.

[42] I. Ion, R. Reeder, and S. Consolvo. New research: Comparing how security experts and non-experts stay safe online. URL https://security.googleblog.com/2015/07/new-research-comparing-how-security.html.

[43] B. Jeavons. zxcvbn-php. https://github.com/bjeavons/zxcvbn-php, 2021.

[44] Josiah. Password hashing for safe storage with argon. URL https://codereview.stackexchange.com/questions/277162/password-hashing-for-safe-storage-with-argon.

[45] Y. S. Lee, N. H. Kim, H. Lim, H. Jo, and H. J. Lee. Online banking authentication system using mobile-otp with qr-code. In *5th International Conference on Computer Sciences and Convergence Information Technology*, pages 644–648, 2010. doi: 10.1109/ICCIT.2010.5711134.

[46] P. W. A. Libraries. pyotp. https://github.com/pyauth/pyotp, 2022.

[47] Linkedin. About linkedin. URL https://about.linkedin.com.

[48] D. Litzenberger. python-pbkdf2. https://github.com/dlitz/python-pbkdf2, 2011.

[49] C. M. R. Lopez, J. E. C. Lopez, A. B. Buchely, and D. F. O. Lopez. Ergonomic requirements for office work with visual display terminals (vdts) —. 1998.

[50] E. Mikalauskas. Rockyou2021: largest password compilation of all time leaked online with 8.4 billion entries. URL https://cybernews.com/security/rockyou2021-alltime-largest-password-compilation-leaked/.

[51] F. Morselli. otphp. https://github.com/Spomky-Labs/otphp, 2022.

[52] Mvnrepository. What's new in maven. URL https://mvnrepository.com.

[53] S. Nadi, S. Krüger, M. Mezini, and E. Bodden. "jumping through hoops": Why do java developers struggle with cryptography apis? In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 935–946, 2016. doi: 10.1145/2884781.2884790.

[54] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith. Why do developers get password storage wrong? a qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 311–328, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134082. URL https://doi.org/10.1145/3133956.3134082.

[55] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith. "if you want, i can store the encrypted password": A password-storage field study with freelance developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300370. URL https://doi.org/10.1145/3290605.3300370.

[56] NIST. Back to basics: Multi-factor authentication (mfa). URL https://www.nist.gov/back-basics-multi-factor-authentication.

[57] N. I. of Standards and Technology. Security requirements for cryptographic modules. Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 140-2, Change Notice 2 December 03, 2002, U.S. Department of Commerce, Washington, D.C., 2001.

[58] Oracle. java.util.regex. URL https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html.

[59] OWASP. Password storage cheat sheet, . URL https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

[60] OWASP. Authentication cheat sheet, . URL https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html.

[61] OWASP. Testing for weak password policy, . URL https://wiki.owasp.org/index.php/Testing_for_Weak_password_policy_(OTG-AUTHN-007).

[62] C. Percival. scrypt. https://github.com/Tarsnap/scrypt, 2022.

[63] PHP. password_hash, . URL https://www.php.net/manual/en/function.password-hash.php.

[64] PHP. hash_pbkdf2, . URL https://www.php.net/hash-pbkdf2.

[65] P. Pisarek. Javascript crypto-js pbkdf2 examples. URL https://javascript.hotexamples.com/examples/crypto-js/-/PBKDF2/javascript-pbkdf2-function-examples.html.

[66] B. Potter. Googleauthenticator. https://github.com/BrandonPotter/GoogleAuthenticator, 2022.

[67] Python. Regular expression operations. URL https://docs.python.org/3/library/re.html.

[68] Qualtrics. Make every interaction an experience that matters. URL https://www.qualtrics.com.

[69] J. Reynolds, T. Smith, K. Reese, L. Dickinson, S. Ruoti, and K. Seamons. A tale of two studies: The best and worst of yubikey usability. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 872–888, 2018. doi: 10.1109/SP.2018.00067.

[70] T. Richards. zxcvbn-cs. https://github.com/trichards57/zxcvbn-cs, 2021.

[71] B. Rodrigues, J. Paiva, V. Gomes, C. Morris, and W. Calixto. Passfault: an open source tool for measuring password complexity and strength. 03 2017.

[72] H. Schlawack. argon2-cffi. https://github.com/hynek/argon2-cffi, 2021.

[73] B. Schneier. Two-factor authentication: Too little, too late. *Commun. ACM*, 48(4):136, apr 2005. ISSN 0001-0782. doi: 10.1145/1053291.1053327. URL https://doi.org/10.1145/1053291.1053327.

[74] Stackexhange. Explore our questions. URL https://codereview.stackexchange.com.

[75] Stackoverflow. How many developers visit stack overflow?, . URL https://stackoverflow.co/advertising/audience/.

[76] Stackoverflow. 2022 developer survey, . URL https://survey.stackoverflow.co/2022/#most-popular-technologies-language-prof.

[77] SYNK. Find and fix vulnerabilities in 5 minutes. URL https://snyk.io.

[78] J. Tan, L. Bauer, N. Christin, and L. F. Cranor. Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1407–1426, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370899. doi: 10.1145/3372297.3417882. URL https://doi.org/10.1145/3372297.3417882.

[79] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor. How does your password measure up? the effect of strength meters on password creation. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 65–80, Bellevue, WA, Aug. 2012. USENIX Association. ISBN 978-931971-95-9. URL https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/ur.

[80] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor. "i added '!' at the end to make it secure": Observing password creation in the lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 123–140, Ottawa, July 2015. USENIX Association. ISBN 978-1-931971-249. URL https://www.usenix.org/conference/soups2015/proceedings/presentation/ur.

[81] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. Emami Naeini, H. Habib, N. Johnson, and W. Melicher. Design and evaluation of a data-driven password meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 3775–3786, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346559. doi: 10.1145/3025453.3026050. URL https://doi.org/10.1145/3025453.3026050.

[82] E. Vosberg. crypto-js. https://github.com/brix/crypto-js, 2021.

[83] D. Wang and P. Wang. Two birds with one stone: Two-factor authentication with security beyond conventional bound. *IEEE Transactions on Dependable and Secure Computing*, 15(4): 708–722, 2018. doi: 10.1109/TDSC.2016.2605087.

[84] G. Watson. two-factor-auth. https://github.com/j256/two-factor-auth, 2020.

[85] D. L. Wheeler. zxcvbn: Low-Budget password strength estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX, Aug. 2016. USENIX Association. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler.

[86] D. Wolf. zxcvbn-python. https://github.com/dwolfhub/zxcvbn-python, 2022.

[87] xlzd. gotp. https://github.com/xlzd/gotp, 2022.

[88] G. Yeo. otplib. https://github.com/yeojz/otplib, 2019.

. APPENDIX

A. Website Images

A.1. Password Storage

Figure A.1.: Password Storage Code Snippets on Website

A.2. Password Policy

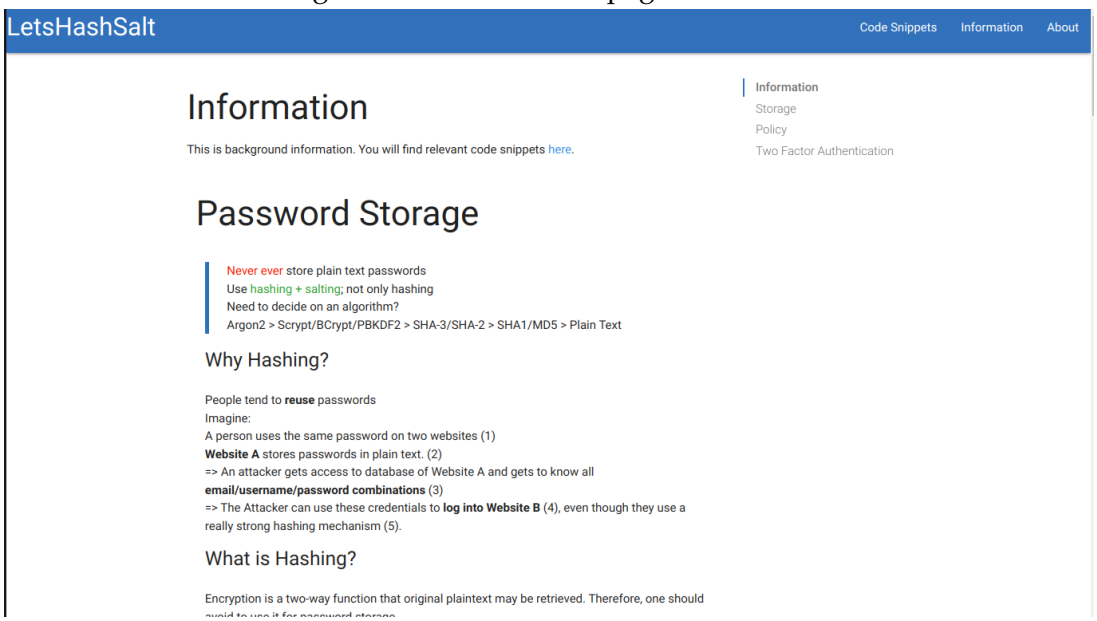
Figure A.2.: Password Policy Code Snippets on Website

A.3. TWO FACTOR AUTHENTICATION

Figure A.3.: Two Factor Authentication Code Snippets on Website

A.4. INFORMATION

Figure A.4.: Information page on Website



B. Survey

B.1. Survey Questions

1. Please state your age (Free text)
2. Please state your current occupation (MSc Student, Software Engineer etc.) (Free text)
3. How many years of professional experience do you have in software development? (Free text)
4. Do you have experience in security concepts? (Yes/ No)
5. Are your tasks in your current occupation related to security concepts? (Yes/ No)
6. How familiar are you with security concepts for implementation of password storage in general?
   - Not familiar at all
   - Slightly familiar
   - Moderately familiar
   - Very familiar
   - Extremely familiar
7. How familiar are you with implementation of password policy in general?
   - Not familiar at all
   - Slightly familiar
   - Moderately familiar
   - Very familiar
   - Extremely familiar
8. How familiar are you with implementation of two factor authentication in general?
   - Not familiar at all
   - Slightly familiar
   - Moderately familiar
   - Very familiar
   - Extremely familiar
9. In what programming language do you have the most experience? (Please select at most two languages)
   - Python
   - Java
   - Javascript
   - PHP
   - GOLang
   - C#

10. The main purpose of this study is to question whether the provided password code blocks (code snippets) are safe or how to improve them securely.
    - The code is presented in three categories: password storage, password policy and two factor authentication.
    - You are not obligated to answer questions that you are unsure of.
    - While checking the code blocks, you can use ANY resource you want to examine the code's security.
    - Estimated study duration is 10-30 minutes
    - You will only be asked questions about the programming languages you choose.
    - We wish you a good survey.
11. (Selected Language Name) Password Storage: This code snippet implements password hashing for safe storage.
    - It uses Argon2 as hashing algorithm
    - The method hash_password is used to hash and salt a password
    - The method verify is used to check whether a password matches a hash
    - (Example Code snippet)
12. Do you think the implementation of hash_password is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
13. Do you think the implementation of verify is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
14. Would you make any other changes to this code snippet? (No / Yes (Free Text))
15. Did you use any additional resources while checking the code? If yes, please provide a link or description of your resource. (No / Yes (Free Text))

16. This code snippet implements password hashing for safe storage.
    - It uses Bcrypt as hashing algorithm
    - The method hash_password is used to hash and salt a password
    - The method verify is used to check whether a password matches a hash
    - (Example Code snippet)
17. Do you think the implementation of hash_password is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
18. Do you think the implementation of verify is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
19. Would you make any other changes to this code snippet? (No / Yes (Free Text))
20. Did you use any additional resources while checking the code? If yes, please provide a link or description of your resource. (No / Yes (Free Text))

21. Password Policy: This code snippet shows an implementation of a password policy check.
    - The method composition uses regexes to verify that the password meets composition criteria (at least one number, one special character, one capital letter and length between 8 and 64)
    - The method check_strength uses the library zxcvbn to check a password's strength
    - (Example Code snippet)
22. Do you think the implementation of this code snippet is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
23. Would you make any other changes to this code snippet? (No / Yes (Free Text))
24. Did you use any additional resources while checking the code? If yes, please provide a link or description of your resource. (No / Yes (Free Text))

25. Two-Factor Authentication: This code snippet shows an implementation of two factor authentication.
    - It uses the library otp
    - It generates and verifies a totp
    - It creates a provisioning uri for the user
    - (Example Code snippet)
26. Do you think the implementation of this code snippet is up-to-date and secure? (Yes/ No (Free Text)/ I am unsure (Free Text))
27. Would you make any other changes to this code snippet? (No / Yes (Free Text))
28. Did you use any additional resources while checking the code? If yes, please provide a link or description of your resource. (No / Yes (Free Text))

29. Do you have any other suggestions, questions or comments? (Free Text)

C. Consent Form

C.1. Survey Consent Page

Welcome!

Thank you for participating in our survey.

The purpose of this survey is to examine the security of several code fragments. These code fragments are part of a larger project by the Behavioural Security Research Group at the Institute for Computer Science of the University of Bonn. Our aim is to ensure that the code you will see here is as up-to-date and secure as it can be. You can choose to examine code fragments in up to two programming languanges, and you will receive 50 Euros per language for full and meaningful contributions. Please notify us after you have finished the survey to receive your payment.

If you have any questions or concerns, please do not hesitate to contact us:

Cüneyt Erem (s6cuerem@uni-bonn.de)
Lisa Geierhaas (geierhaa@cs.uni-bonn.de)

Participant Consent

Please read the consent form which is linked below carefully and keep the document.

I have read and understood the consent form and I agree to take part in this survey.

(i consent/ i do not consent)

C.2. CONSENT FORM

Participant Information for

Empirical Study of Code Security

The purpose of this document is to specify the terms of my participation in the research project.

Purpose
An empirical study of security concepts for password storage, password policy and two factor authentication

Procedure
-The participant is asked to revise the code blocks regarding their security within an online survey.
-The expected time to revise the code blocks is estimated to be approximately 10-30 minutes. The actual time is dependent on the individual time the participant requires to complete the survey.

Study
Conducted by researchers of the Rheinische-Friedrichs-Wilhelms Universität Bonn led by Prof. Dr. Matthew Smith.

Involved Scientists
Cüneyt Erem (s6cuerem@uni-bonn.de)
MSc. Lisa Geierhaas (geierhaa@iai.uni-bonn.de)
Phd. Christian Tiefenau (tiefenau@cs.uni-bonn.de)
Prof. Dr. Matthew Smith (smith@cs.uni-bonn.de)

Risks
No foreseeable mental or physical risks to you associated with participation in this study.

Benefits
-Learning of new methods to gain more programming experience.
-Contribution to scientific research.

Confidentiality
The information collected in this study will be kept confidential and stored pseudonymously during the study and anonymously thereafter. The pseudonyms are stored to enable you to access and delete your data on request.
After the study, the pseudonyms will be destroyed and all data will then be anonymous. All analysis of the data will only occur on the anonymous data and at no point will results be linked to identifying information.
Optionally, if you wish to be contacted by the researchers in the future, you can additionally consent to your contact details being stored. This data will be stored separately from the study data. All data will be stored using strong encryption.
In all cases uses of data will be subject to standard data use policies at the Rheinische-FriedrichWilhelms Universität Bonn and will conform to the EU Data Protection Directive.

Data Analysis and Publication
Researchers will have access to this data only if they agree to preserve the confidentiality of the data and if they agree to the terms I have specified in this document. No identifying information will be referenced in the publication or presentation of the study.

Name and address of the controller
The controller within the meaning of the General Data Protection Regulation and other national data protection laws of the Member States as well as other provisions of data protection law

Universität Bonn
Institut für Informatik
Endenicher Allee 19A
53115 Bonn
Germany

Name and address of the data protection officer
Official data protection officer:
Dr. Jörg Hartmann
Genscherallee 3
53113 Bonn
Email: joerg.hartmann@uni-bonn.de

Deputy:
Eckhard Wesemann
Dezernat 1, Abt. 1.0
Regina-Pacis-Weg 3
53113 Bonn
Email: wesemann@verwaltung.uni-bonn.de

Duration of storage
The data will be erased as soon as they are no longer necessary to achieve the purpose for which they were collected.

Deletion obligations
You can request from the person responsible to delete the concerning personal data immediately.

Right to lodge a complaint with a supervisory authority
Without prejudice to any other administrative or judicial remedy, you have the right to lodge a complaint with a supervisory authority, in particular in the Member State where you are staying, working or suspected of infringing, if you believe that the processing of personal data concerning you is in breach of the EU General Data Protection Regulation.
The supervisory authority with which the complaint has been lodged shall inform the complainant of the status and results of the complaint, including the possibility of a judicial remedy under Art. 78 GDPR. The competent supervisory authority is: Landesbeauftragte für Datenschutz und Informationsfreiheit Nordrhein-Westfalen, Postfach 20 04 44, 40102 Düsseldorf.

Informed Consent Form

1.	I have read and understood the information about the project, as provided above.
2.	I have been given the opportunity to ask questions about the project and my participation.
3.	My participation in this study is voluntary. There is no explicit or implicit coercion to participate.
4.	I have the right not to answer any of the questions if I feel uncomfortable in any way during the study. I also have the right to withdraw from the study entirely. I understand I can withdraw at any time without giving reasons and that I will not be penalized for withdrawing nor will I be questioned on why I have withdrawn.
5.	The procedures regarding confidentiality have been clearly explained (e.g., use of names, pseudonyms, anonymization of data, etc.) to me. I optionally consent to my contact information (e.g., email address) being stored. This information will be stored separately from the research data and will only be used to contact me in matters pertaining to this study.
6.	I have been given the explicit guarantees that the researcher will not identify me by name or function in any reports using information obtained from this study and that my confidentiality as a participant in this study will remain secure. I was informed that all audio recordings and handwritten documents, if available, will be destroyed after transcription. In all cases, subsequent uses of records and data will be subject to standard data use policies at the Rheinische-FriedrichWilhelms Universität Bonn and will conform to the EU Data Protection Directive.
7.	The use of the data in research, publications, sharing and archiving has been explained to me.
8.	I agree that other researchers will have access to this data only if they agree to preserve the confidentiality of the data and if they agree to the terms I have specified in this form.
9.	I have read and understood the points and statements of this form. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this study.
10.	I have been given a copy of this consent and the participant information (please download this document and save it).

D. Code Fragments

D.1. Password Storage

D.1.1. Python

D.1.1.1. Argon2

```
# Argon2 is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
# hash can be verified for different passwords

import argon2
from argon2.exceptions import VerifyMismatchError

# hash_length, salt_length, type: argon2id are default, if needed increase the values
# if needed regulate parameters time_cost, memory_cost, parallelism for your system
# example: PasswordHasher(1, 8, 1, 16, 16)
ph = argon2.PasswordHasher()

def hash_password(password):
    return ph.hash(password)

def verify(pw_hash, password):
    try:
        return ph.verify(pw_hash, password)
    except VerifyMismatchError:
        return False

def main():
    # example password, change with your database password or input password
    pw_hash = hash_password('s3cr3t')
    print(pw_hash)                          #print hash as string
    print(verify(pw_hash, 's3cr3t'))        #print true
    print(ph.check_needs_rehash(pw_hash))   #print false
    print(verify(pw_hash, 's3cr4t'))        #print false

if __name__ == '__main__':
    main()
```

D.1.1.2. Bcrypt

```
# Bcrypt is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
```

```
# hash can be verified for different passwords

import bcrypt

# gensalt specifies round 12, if needed increase the value
def hash_password(password):
    pw_hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
    return pw_hash

def verify(pw_hash, password):
    return bcrypt.checkpw(password.encode(), pw_hash)

def main():
    # example password, change with your database password or input password
    pw_hash = hash_password('s3cr3t')
    print(pw_hash)                          #print hash as string
    print(verify(pw_hash, 's3cr3t'))        #print true
    print(verify(pw_hash, 's3cr4t'))        #print false

if __name__ == '__main__':
    main()
```

D.1.1.3. PBKDF2

```
# Pbkdf2 is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
# hash can be verified for different passwords

from pbkdf2 import crypt

# iteration and salt are default, if needed increase the values
def hash_password(password):
    return crypt(password)

def verify(pw_hash, password):
    if pw_hash == crypt(password, pw_hash):
        return True
    else:
        return False

def main():
    # example password, change with your database password or input password
    pw_hash = hash_password('s3cr3t')
    print(pw_hash)                          #print hash as string
    print(verify(pw_hash, 's3cr3t'))        #print true
```

```
    print(verify(pw_hash, 's3cr4t'))        #print false

if __name__ == '__main__':
    main()
```

D.1.2. JAVA PLAIN

D.1.2.1. ARGON2

```
// Argon2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import de.mkammerer.argon2.Argon2;
import de.mkammerer.argon2.Argon2Factory;
import de.mkammerer.argon2.Argon2Factory.Argon2Types;
import de.mkammerer.argon2.Argon2Helper;

public class Argon2_class {
    public static String hash_password(String password) {
        // hash_length, salt_length, type: argon2id are default,
        // if needed, increase the values
        // if needed, regulate parameters maxMillisecs, memory, parallelism for your system
        Argon2 argon2 = Argon2Factory.create(Argon2Types.ARGON2id);
        int iterations = Argon2Helper.findIterations(argon2, 1000, 65536, 4);
        char[] passwordArray = password.toCharArray();
        String hash = argon2.hash(iterations, 65536, 4, passwordArray);
        argon2.wipeArray(passwordArray);
        return hash;
    }

    public static Boolean verify(String password, String hashed) {
        Argon2 argon2 = Argon2Factory.create(Argon2Types.ARGON2id);
        char[] passwordArray = password.toCharArray();
        return argon2.verify(hashed, passwordArray);
    }

    public static void main(String[] args) {
        // example password, change with your database password or input password
        String pw_hash = hash_password("s3cr3t");
        System.out.println(pw_hash);    //print hash as string

        System.out.println(verify("s3cr3t", pw_hash));      //print true
        System.out.println(verify("s3cr4t", pw_hash));      //print false
    }
```

```
}
```

D.1.2.2. BCRYPT

```
// Bcrypt is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import org.mindrot.jbcrypt.*;

public class Bcrypt_class {
    public static String hashPassword(String password) {
        // gensalt specifies round 10, if needed increase the value
        return BCrypt.hashpw(password, BCrypt.gensalt());
    }

    public static Boolean verifyPassword(String password, String hashed) {
        return BCrypt.checkpw(password, hashed);
    }

    public static void main(String[] args) {
        // example password, change with your database password or input password
        String hash = hashPassword("s3cr3t");
        System.out.println(hash);    //print hash as string

        System.out.println(verifyPassword("s3cr3t", hash));     //print true
        System.out.println(verifyPassword("s3cr4t", hash));     //print false
    }
}
```

D.1.2.3. PBKDF2

```
// Pbkdf2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;

public class Pbkdf2_class {
```

```
private static String hash_password(String password)
        throws NoSuchAlgorithmException, InvalidKeySpecException
{
    // iteration and salt are default, if needed increase the values
    int iterations = 1000;
    char[] chars = password.toCharArray();
    byte[] salt = getSalt();

    PBEKeySpec spec = new PBEKeySpec(chars, salt, iterations, 64 * 8);
    SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");

    byte[] hash = skf.generateSecret(spec).getEncoded();
    return iterations + ":" + toHex(salt) + ":" + toHex(hash);
}

private static byte[] getSalt() throws NoSuchAlgorithmException
{
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    byte[] salt = new byte[16];
    sr.nextBytes(salt);
    return salt;
}

private static String toHex(byte[] array) throws NoSuchAlgorithmException
{
    BigInteger bi = new BigInteger(1, array);
    String hex = bi.toString(16);

    int paddingLength = (array.length * 2) - hex.length();
    if(paddingLength > 0)
    {
        return String.format("%0"  +paddingLength + "d", 0) + hex;
    }else{
        return hex;
    }
}

private static boolean verify(String originalPassword, String storedPassword)
        throws NoSuchAlgorithmException, InvalidKeySpecException
{
    String[] parts = storedPassword.split(":");
    int iterations = Integer.parseInt(parts[0]);

    byte[] salt = fromHex(parts[1]);
```

```
        byte[] hash = fromHex(parts[2]);

        PBEKeySpec spec = new PBEKeySpec(originalPassword.toCharArray(),
                salt, iterations, hash.length * 8);
        SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] testHash = skf.generateSecret(spec).getEncoded();

        int diff = hash.length ^ testHash.length;
        for(int i = 0; i < hash.length && i < testHash.length; i++)
        {
            diff |= hash[i] ^ testHash[i];
        }
        return diff == 0;
    }
    private static byte[] fromHex(String hex) throws NoSuchAlgorithmException
    {
        byte[] bytes = new byte[hex.length() / 2];
        for(int i = 0; i < bytes.length ;i++)
        {
            bytes[i] = (byte)Integer.parseInt(hex.substring(2 * i, 2 * i + 2), 16);
        }
        return bytes;
    }

    public static void main(String[] args)
            throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        // example password, change with your database password or input password
        String pw_hash = hash_password("s3cr3t");
        System.out.println(pw_hash);    //print hash as string

        System.out.println(verify("s3cr3t", pw_hash));        //print true
        System.out.println(verify("s3cr4t", pw_hash));        //print false
    }
}
```

D.1.3. JAVA SPRING

D.1.3.1. ARGON2

```
import org.springframework.security.crypto.argon2.Argon2PasswordEncoder;

public static String hashPassword(String password) {
    // Argon2PasswordEncoder(int saltLength, int hashLength, int parallelism,
            int memory, int iterations)
```

```
    // Spring Security uses default values that should be adjusted to your system
    Argon2PasswordEncoder encoder = new Argon2PasswordEncoder();
return encoder.encode(password);
}

// Verify that password matches hash:
public static Boolean verifyPassword(String password, String hashed) {
    Argon2PasswordEncoder encoder = new Argon2PasswordEncoder();
    return encoder.matches(password, hashed);
}
```

D.1.3.2. BCRYPT

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public static String hashPassword(String password) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
return (encoder.encode(password);
}

// Verify that password matches hash:
public static Boolean verifyPassword(String password, String hashed) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
    return(encoder.matches(password, hashed));
}
```

D.1.3.3. PBKDF2

```
// Pbkdf2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;

public class Pbkdf2_class {

    private static String hash_password(String password)
            throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        // iteration and salt are default, if needed increase the values
```

```
        int iterations = 1000;
        char[] chars = password.toCharArray();
        byte[] salt = getSalt();

        PBEKeySpec spec = new PBEKeySpec(chars, salt, iterations, 64 * 8);
        SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");

        byte[] hash = skf.generateSecret(spec).getEncoded();
        return iterations + ":" + toHex(salt) + ":" + toHex(hash);
    }

    private static byte[] getSalt() throws NoSuchAlgorithmException
    {
        SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
        byte[] salt = new byte[16];
        sr.nextBytes(salt);
        return salt;
    }

    private static String toHex(byte[] array) throws NoSuchAlgorithmException
    {
        BigInteger bi = new BigInteger(1, array);
        String hex = bi.toString(16);

        int paddingLength = (array.length * 2) - hex.length();
        if(paddingLength > 0)
        {
            return String.format("%0"  +paddingLength + "d", 0) + hex;
        }else{
            return hex;
        }
    }

    private static boolean verify(String originalPassword, String storedPassword)
            throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        String[] parts = storedPassword.split(":");
        int iterations = Integer.parseInt(parts[0]);

        byte[] salt = fromHex(parts[1]);
        byte[] hash = fromHex(parts[2]);

        PBEKeySpec spec = new PBEKeySpec(originalPassword.toCharArray(),
                salt, iterations, hash.length * 8);
```

```
        SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] testHash = skf.generateSecret(spec).getEncoded();

        int diff = hash.length ^ testHash.length;
        for(int i = 0; i < hash.length && i < testHash.length; i++)
        {
            diff |= hash[i] ^ testHash[i];
        }
        return diff == 0;
    }
    private static byte[] fromHex(String hex) throws NoSuchAlgorithmException
    {
        byte[] bytes = new byte[hex.length() / 2];
        for(int i = 0; i < bytes.length ;i++)
        {
            bytes[i] = (byte)Integer.parseInt(hex.substring(2 * i, 2 * i + 2), 16);
        }
        return bytes;
    }

    public static void main(String[] args)
            throws NoSuchAlgorithmException, InvalidKeySpecException
    {
        // example password, change with your database password or input password
        String pw_hash = hash_password("s3cr3t");
        System.out.println(pw_hash);    //print hash as string

        System.out.println(verify("s3cr3t", pw_hash));        //print true
        System.out.println(verify("s3cr4t", pw_hash));        //print false
    }
}
```

D.1.4. JAVASCRIPT

D.1.4.1. ARGON2

```
// Argon2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

const argon2 = require('argon2');

// hash_length, salt_length, type: argon2id are default, if needed increase the values
// if needed regulate parameters time_cost, memory_cost, parallelism for your system
async function hash_password(password) {
```

```
  try {
    return await argon2.hash(password, { type: argon2.argon2id });
  } catch (err) {
    console.log("error1: " + err)
  }
}

async function verify(pw_hash, password) {
  try {
    if (await argon2.verify(pw_hash, password)) {
      return true;
    } else {
      return false;
    }
  } catch (err) {
    console.log("error2: " + err)
  }
}

async function main() {
  // example password, change with your database password or input password
  const hash = await hash_password("s3cr3t");
  console.log(hash);                          //print hash as string

  console.log(await verify(hash, "s3cr3t"));    //print true
  console.log(await verify(hash, "s3cr4t"));    //print false
}

main();
```

D.1.4.2. BCRYPT

```
// Bcrypt is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

const bcrypt = require('bcrypt');

// saltRound specifies round 12, if needed increase the value
async function hash_password(password) {
    const saltRounds = 12;
    try {
        return await bcrypt.hash(password, saltRounds);
    } catch (err) {
        console.log("error1: " + err);
```

```
    }
}

async function verify_password(pw_hash, password) {
    try {
        if (await bcrypt.compare(password, pw_hash)) {
            return true;
        } else {
            return false;
        }
    } catch (err) {
        console.log("error2: " + err);
    }
}

// example password, change with your database password or input password
async function main() {
    const hash = await hash_password("s3cr3t");
    console.log(hash);                                          //print hash as string

    console.log(await verify_password(hash, "s3cr3t"));     //print true
    console.log(await verify_password(hash, "s3cr4t"));     //print false
}

main();
```

D.1.4.3. PBKDF2

```
// Pbkdf2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

const crypto = require('crypto-js');

const salt = crypto.lib.WordArray.random(128 / 8);

// iteration and salt are default, if needed increase the values
async function hash_password(password) {
    const key = crypto.PBKDF2(password, salt, {
        keySize: 128 / 32,
        iterations: 1000
    });

    return key.toString(crypto.enc.Base64);
}
```

```
async function verify_password(pw_hash, password) {
    const key = crypto.PBKDF2(password, salt, {
        keySize: 128 / 32,
        iterations: 1000
    });
    const hash = key.toString(crypto.enc.Base64);

    if (hash === pw_hash) {
        return true;
    }

    return false;
}


async function main() {
    // example password, change with your database password or input password
    const hash = await hash_password("s3cr3t");
    console.log(hash);                          //print hash as string

    console.log(await verify_password(hash, "s3cr3t"));     //print true
    console.log(await verify_password(hash, "s3cr4t"));     //print false
}

main();
```

D.1.5. PHP

D.1.5.1. ARGON2

```
# Argon2 is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
# hash can be verified for different passwords

function hash_password(&$password) {
    # hash_length, salt_length, type: argon2id are default
    # if needed regulate parameters time_cost, memory_cost, threads for your system
    $options = [
        'memory_cost' => PASSWORD_ARGON2_DEFAULT_MEMORY_COST,
        'time_cost' => PASSWORD_ARGON2_DEFAULT_TIME_COST,
        'threads' => PASSWORD_ARGON2_DEFAULT_THREADS
    ];
```

```
    return password_hash($password, PASSWORD_ARGON2ID, $options);
}

function verify(&$password, &$pw_hash) {
    if (password_verify($password, $pw_hash)) {
        return 'true';
    } else {
        return 'false';
    }
}

# example password, change with your database password or input password
$password = 's3cr3t';
echo $hash = hash_password($password), "\r\n";    #print hash as string

$password1 = 's3cr3t';
$password2 = 's3cr4t';
echo $verify = verify($password1, $hash), "\r\n";           #print true
echo $verify = verify($password2, $hash), "\r\n";           #print false
```

D.1.5.2. BCRYPT

```
# Bcrypt is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
# hash can be verified for different passwords

function hash_password(&$password) {
    # if needed regulate cost parameter
    $options = [
        'cost' => 12,
    ];

    return password_hash($password, PASSWORD_BCRYPT, $options);
}

function verify(&$password, &$pw_hash) {
    if (password_verify($password, $pw_hash)) {
        return 'true';
    } else {
        return 'false';
    }
}

# example password, change with your database password or input password
$password = 's3cr3t';
```

```
echo $hash = hash_password($password), "\r\n";    #print hash as string

$password1 = 's3cr3t';
$password2 = 's3cr4t';
echo $verify = verify($password1, $hash), "\r\n";          #print true
echo $verify = verify($password2, $hash), "\r\n";          #print false
```

D.1.5.3. PBKDF2

```
# Pbkdf2 is one of the best hashing algorithm for password storage.
# By different parameters, it hashes password by using default salt
# hash can be verified for different passwords

function hash_password(&$password) {
    # iteration and salt are default, if needed change the values
    # $salt = openssl_random_pseudo_bytes(16);
    $iterations = 1000;
    return hash_pbkdf2("sha256", $password, $iterations, 20);
}

function verify(&$password, &$pw_hash) {
    if (hash_password($password) == $pw_hash) {
        return 'true';
    } else {
        return 'false';
    }
}

# example password, change with your database password or input password
$password = 's3cr3t';
echo $hash = hash_password($password), "\r\n";      #print hash as string

$password1 = 's3cr3t';
$password2 = 's3cr4t';
echo verify($password1, $hash), "\r\n";             #print true
echo verify($password2, $hash), "\r\n";             #print false
```

D.1.6. GOLANG

D.1.6.1. ARGON2

```
package main

// Argon2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords
```

```
import (
"fmt"
"log"

"github.com/alexedwards/argon2id"
)

// hash_length, salt_length, type: argon2id are default, if needed increase the values
// if needed regulate parameters time_cost, memory_cost, parallelism for your system
func hash_password(password string) string {
hash, err := argon2id.CreateHash(password, argon2id.DefaultParams)
if err != nil {
log.Fatal(err)
}
return hash
}

func verify(pw_hash string, password string) bool {
match, err := argon2id.ComparePasswordAndHash(password, pw_hash)
if err != nil {
log.Fatal(err)
}
return match
}

func main() {
// example password, change with your database password or input password
hash := "" + hash_password("s3cr3t")
fmt.Println(hash) //print hash as string

fmt.Println(verify(hash, "s3cr3t")) //print true
fmt.Println(verify(hash, "s3cr4t")) //print false
}
```

D.1.6.2. BCRYPT

```
package main

// Bcrypt is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import (
"fmt"
```

```
"log"

"golang.org/x/crypto/bcrypt"
)

// gensalt specifies round 14, if needed increase the value
func hash_password(password string) string {
hash, err := bcrypt.GenerateFromPassword([]byte(password), 14)
if err != nil {
log.Fatal(err)
}
return string(hash)
}

func verify(pw_hash string, password string) bool {
err := bcrypt.CompareHashAndPassword([]byte(pw_hash), []byte(password))
return err == nil
}

func main() {
// example password, change with your database password or input password
hash := "" + hash_password("s3cr3t")
fmt.Println(hash) //print hash as string

fmt.Println(verify(hash, "s3cr3t")) //print true
fmt.Println(verify(hash, "s3cr4t")) //print false
}
```

D.1.6.3. PBKDF2

```
package main

// Pbkdf2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

import (
"bytes"
"crypto/rand"
"crypto/sha256"
"fmt"
"log"

"golang.org/x/crypto/pbkdf2"
)
```

```
//iteration and salt are default, if needed increase the values
var salt = generateRandomBytes(16)

func generateRandomBytes(n uint32) []byte {
b := make([]byte, n)
_, err := rand.Read(b)
if err != nil {
log.Fatal(err)
return nil
}

return b
}

func hashPassword(password string) []byte {
return pbkdf2.Key([]byte(password), salt, 4096, 32, sha256.New)
}

func verifyPassword(password string, hash []byte) bool {
var pw_hash []byte = pbkdf2.Key([]byte(password), salt, 4096, 32, sha256.New)
var compare int = bytes.Compare(hash, pw_hash)

if compare == 0 {
return true
} else {
return false
}
}

func main() {
// example password, change with your database password or input password
hash := hashPassword("s3cr3t")
fmt.Printf("%x\n", hash) //print hash as string instead of bytes

fmt.Println(verifyPassword("s3cr3t", hash)) //print true
fmt.Println(verifyPassword("s3cr4t", hash)) //print false
}
```

D.1.7. C#

D.1.7.1. ARGON2

```
// Argon2 is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
```

```
// hash can be verified for different passwords

using System;
using System.Security.Cryptography;
using Konscious.Security.Cryptography;
using System.Text;

public class Argon_class
{
    private static byte[] CreateSalt()
    {
        var buffer = new byte[128];
        var generator = RandomNumberGenerator.Create();
        generator.GetBytes(buffer);
        return buffer;
    }

    // hash_length, salt_length, type: argon2id are default,
    //if needed increase the values
    // if needed regulate parameters time_cost, memory_cost,
    //parallelism for your system
    private static byte[] hash_password(string password, byte[] salt)
    {
        var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password));
        argon2.DegreeOfParallelism = 16;
        argon2.MemorySize = 8192;
        argon2.Iterations = 40;
        argon2.Salt = salt;

        return argon2.GetBytes(128);
    }

    private static bool verify(byte[] pw_hash, string password, byte[] salt)
    {
        var new_hash = hash_password(password, salt);
        return pw_hash.SequenceEqual(new_hash);
    }

    public static void Run()
    {
        var salt = CreateSalt();
        // Console.WriteLine($"salt '{ Convert.ToBase64String(salt) }'.");

        // example password, change with your database password or input password
```

```
        var pw_hash = hash_password("s3cr3t", salt);
        Console.WriteLine($"'{ Convert.ToBase64String(pw_hash) }'.");  //print hash as string

        var result1 = verify(pw_hash, "s3cr3t", salt);
        Console.WriteLine(result1 ? "True!" : "False!");                    //print true

        var result2 = verify(pw_hash, "s3cr4t", salt);
        Console.WriteLine(result2 ? "True!" : "False!");                    //print false
    }
}
```

D.1.7.2. BCRYPT

```
// Bcrypt is one of the best hashing algorithm for password storage.
// By different parameters, it hashes password by using default salt
// hash can be verified for different passwords

using System;
using BCryptNet = BCrypt.Net.BCrypt;

public class Bcrypt_class
{
    //gensalt specifies round 11, if needed increase the value
    private static string hash_password(string password)
    {
        string pw_hash = BCryptNet.HashPassword(password);
        return pw_hash;
    }

    private static bool verify(string pw_hash, string password)
    {
        return BCryptNet.Verify(password, pw_hash);
    }
    public static void Run()
    {
        // example password, change with your database password or input password
        string pw_hash = hash_password("s3cr3t");
        Console.WriteLine(pw_hash);                     //print hash as string

        var result1 = verify(pw_hash, "s3cr3t");
        Console.WriteLine(result1);                     //print true

        var result2 = verify(pw_hash, "s3cr4t");
        Console.WriteLine(result2);                     //print false
```

```
    }
}
```

D.2. Password Policy

D.2.1. Python

```
# Password policy is important for defining minimum requirements
# for creating strong passwords
# it creates patterns with having at least one
# upper letter/lower letter/number/special character/8-64 length
# check password strength with zxcvbn library

import re
from zxcvbn import zxcvbn
from dataclasses import dataclass
from typing import Tuple

PATTERNS = dict(
    number = re.compile('[0-9]'),
    lower = re.compile('[a-z]'),
    upper = re.compile('[A-Z]'),
    special = re.compile('[^A-Za-z0-9]'),
    length = re.compile('.{8,64}$'),
)

def composition(password):
    return {
        label : bool(rgx.search(password))
        for label, rgx in PATTERNS.items()
    }

STRENGTHS = {
    0: 'Worst',
    1: 'Bad',
    2: 'Weak',
    3: 'Good',
    4: 'Strong'
}

@dataclass(frozen = True)
class PasswordStrength:
    score: int
    label: str
    warning: str
    suggestions: Tuple[str]
```

```
def check_strength(password):
    z = zxcvbn(password, user_inputs=['John', 'Smith'])
    return PasswordStrength(
        z['score'],
        STRENGTHS[z['score']],
        z['feedback']['warning'],
        tuple(z['feedback']['suggestions']),
    )

def main():
    # example password, change with your database password or input password
    password = 'Expassword0.'

    x = composition(password)
    print("composition: ", x)              #print each requirement as true

    y = check_strength(password)
    print("check_strength: ", y)           #print strength features

if __name__ == '__main__':
    main()
```

D.2.2. JAVA

```
// Password policy is important for defining minimum requirements
// for creating strong passwords
// it creates patterns with having at least one
// upper letter/lower letter/number/special character/8-64 length
// check password strength with zxcvbn library

import com.nulabinc.zxcvbn.Strength;
import com.nulabinc.zxcvbn.Zxcvbn;
import java.util.HashMap;
import java.util.Map;

public class Policy_class {

    public static String composition(String password) {
        String number = "(?=.*[0-9]).{2,}";
        String upper = "(?=.*[A-Z]).{2,}";
        String lower = "(?=.*[a-z]).{2,}";
        String special = "(?=.*[^A-Za-z0-9]).{2,}";
        String length = ".{8,64}";
        return "" + password.matches(number) + ", " + password.matches(upper) + ", " +
```

```
                password.matches(lower) + ", " + password.matches(special) + ", " +
                password.matches(length);
    }

    public static String check_strength(String password) {
        Zxcvbn zxcvbn = new Zxcvbn();
        Strength strength_levels = zxcvbn.measure(password);
        Map map = new HashMap();
        map.put(0, "Worst");
        map.put(1, "Bad");
        map.put(2, "Weak");
        map.put(3, "Good");
        map.put(4, "Strong");

        return "" + strength_levels.getScore() + ", " +
                map.get(strength_levels.getScore()) + ", " +
                strength_levels.getFeedback();
    }

    public static void main(String[] args) {
        // example password, change with your database password or input password
        String password = "Expassword0.";

        //print each requirement as true
        System.out.println("composition: " + composition(password));

        //print strength features
        System.out.println("check_strength: " + check_strength(password));

    }
}
```

D.2.3. JAVASCRIPT

```
// Password policy is important for defining minimum requirements
// for creating strong passwords
// it creates patterns with having at least one
// upper letter/lower letter/number/special character/8-64 length
// check password strength with zxcvbn library

function composition(password) {

    var number = /(?=.*[0-9])/;
    var upper_lower = /(?=.*[a-z])(?=.*[A-Z])/;
    var special = /[^A-Za-z0-9]/;
```

```
    var length = /.{8,64}$/;

    return [number.test(password), upper_lower.test(password), special.test(password),
            length.test(password)];
}

function check_strength(password) {
    var zxcvbn = require('zxcvbn');
    var result = zxcvbn(password);
    var strength = {
        0: "Worst",
        1: "Bad",
        2: "Weak",
        3: "Good",
        4: "Strong"
    }

    return [result.score, strength[result.score], result.feedback.warning,
            result.feedback.suggestions];
}


// example password, change with your database password or input password
var password = 'Expassword0.';

console.log('composition: ' + composition(password));  //print each requirement as true
console.log('check_strength: ' + check_strength(password)); //print strength features
```

D.2.4. PHP

```
# Password policy is important for defining minimum requirements
# for creating strong passwords
# it creates patterns with having at least one
# upper letter/lower letter/number/special character/8-64 length
# check password strength with zxcvbn library

require_once 'vendor/autoload.php';
use ZxcvbnPhp\Zxcvbn;

function composition(&$password) {
    $number = preg_match_all('/(?=\S*[0-9])/', $password);
    $upper = preg_match_all('/(?=\S*[A-Z])/', $password);
    $lower = preg_match_all('/(?=\S*[a-z])/', $password);
    $special = preg_match_all('/(?=.*[^A-Za-z0-9])/', $password);
    $length = strlen($password) > 7 && strlen($password) < 65;
```

```
    $array = array();
    array_push($array, ($number ? 'true' : 'false'), ($upper ? 'true' : 'false'),
            ($lower ? 'true' : 'false'), ($special ? 'true' : 'false'),
            ($length ? 'true' : 'false'));

    return $array;
}

function check_strength(&$password) {
    $strength = array(
        0 => "Worst",
        1 => "Bad",
        2 => "Weak",
        3 => "Good",
        4 => "Strong",
    );

    $zxcvbn = new Zxcvbn();
    $result = $zxcvbn->passwordStrength($password);

    $array = array(
        'score' => $result['score'],
        'stregth' => $strength[$result['score']],
        'warning' => $result['feedback']['warning'],
    );

    return $array;
}

# example password, change with your database password or input password
$password = 'Expassword0.';

$x = composition($password);                #print each requirement as true
print_r($x);

$y = check_strength($password);             #print strength features
print_r($y);
```

D.2.5. GOLANG

```
package main

// Password policy is important for defining minimum requirements
// for creating strong passwords
```

```
// it creates patterns with having at least one
// upper letter/lower letter/number/special character/8-64 length
// check password strength with zxcvbn library

import (
"fmt"
"regexp"

"github.com/nbutton23/zxcvbn-go"
)

func composition(password string) string {
number, _ := regexp.MatchString(".*[0-9]", password)
upper_lower, _ := regexp.MatchString(".*[a-zA-Z]", password)
special, _ := regexp.MatchString(".*[^A-Za-z0-9]", password)
length, _ := regexp.MatchString(".{8,64}", password)

return fmt.Sprintf("%t, %t, %t, %t", number, upper_lower, special, length)
}

func check_strength(password string) string {
result := zxcvbn.PasswordStrength(password, nil)
strength := map[interface{}]interface{}{
0: "Worst",
1: "Bad",
2: "Weak",
3: "Good",
4: "Strong",
}

return fmt.Sprintf("%v, %v", result.Score, strength[result.Score])
}

func main() {
//example password, change with your database password or input password
var password string = "Expassword0."

fmt.Println("composition: ", composition(password))  //print each requirement as true
fmt.Println("check_strength: ", check_strength(password)) //print strength features
}
```

D.2.6. C#

```
// Password policy is important for defining minimum requirements
// for creating strong passwords
```

```
// it creates patterns with having at least one
// upper letter/lower letter/number/special character/8-64 length
// check password strength with zxcvbn library

using System;
using System.Text.RegularExpressions;
using Zxcvbn;

public class Policy_class
{
    private static string composition(string password)
    {
        Regex number = new Regex(@"(?=.*[0-9])");
        Regex upper_lower = new Regex(@"(?=.*[a-z])(?=.*[A-Z])");
        Regex special = new Regex(@"[^A-Za-z0-9]");
        Regex length = new Regex(@".{8,64}$");

        Match match1 = number.Match(password);
        Match match2 = upper_lower.Match(password);
        Match match3 = special.Match(password);
        Match match4 = length.Match(password);

        bool[] test = new bool[4];
        test[0] = match1.Success;
        test[1] = match2.Success;
        test[2] = match3.Success;
        test[3] = match4.Success;

        return "" + test[0] + ", " + test[1] + ", " + test[2] + ", " + test[3];
    }

    private static string check_strength(string password)
    {
        var result = Zxcvbn.Core.EvaluatePassword(password);
        Dictionary strength = new Dictionary();
        strength.Add(0, "Worst");
        strength.Add(1, "Bad");
        strength.Add(2, "Weak");
        strength.Add(3, "Good");
        strength.Add(4, "Strong");

        string last = "" + result.Score + ", " + strength.FirstOrDefault(x => x.Key ==
                result.Score).Value + ", " + result.Feedback.Warning + ", " +
                result.Feedback.Suggestions;
```

```
        return last;
    }

    public static void Run()
    {
        // example password, change with your database password or input password
        string password = "Expassword0.";

        string a = composition(password);       //print each requirement as true
        Console.WriteLine(a);

        string b = check_strength(password);    //print strength features
        Console.WriteLine(b);
    }
}
```

D.3. TWO FACTOR AUTHENTICATION

D.3.1. PYTHON

```
# OTP with TOTP and HOTP are used for password authentication to increase security
# TOTP is commonly used version, create totp secret and verify secret within valid
#time period
# create provisioning_uri and use it by scanning barcode with Google Authenticator

import pyotp
import time

# if needed, change interval parameters
def generate_totp_secret():
    return pyotp.TOTP(pyotp.random_base32(), interval=30)

def generate_uri(second_factor, user_mail, issuer_name):
    return second_factor.provisioning_uri(user_mail, issuer_name)

def main():
    # create totp secret and verify within valid time period
    totp_secret = generate_totp_secret()
    totp_secret_first = totp_secret.now()

    print(totp_secret.verify(totp_secret_first))        #print true
    time.sleep(30)
    print(totp_secret.verify(totp_secret_first))        #print false

    # change mail and issuer parameters
    # provisioning_uri can be used for QR-code scan by Google Authenticator
```

```
    provisioning_uri = generate_uri(totp_secret, "example@mail.com", "issuer name")
    print(provisioning_uri)                              #print provisioning uri

if __name__ == '__main__':
    main()
```

D.3.2. JAVA

```
// OTP with TOTP and HOTP are used for password authentication to increase security
// TOTP is commonly used version, create totp secret and verify secret within
//valid time period
// create provisioning_uri and use it by scanning barcode with Google Authenticator

import java.util.concurrent.TimeUnit;
import com.j256.twofactorauth.TimeBasedOneTimePasswordUtil;

public class Totp_class {
    public static int generate_totp_secret(String shared_secret) throws Exception{
        return TimeBasedOneTimePasswordUtil.generateCurrentNumber(shared_secret);
    }

    public static String generate_uri(String keyId, String shared_secret) {
        return TimeBasedOneTimePasswordUtil.generateOtpAuthUrl(keyId, shared_secret);
    }

    public static void main(String[] args) throws Exception {
        //create totp secret and verify within valid time period
        String shared_secret = TimeBasedOneTimePasswordUtil.generateBase32Secret();
        int totp_secret_first = generate_totp_secret(shared_secret);

        // if needed, change interval parameters
        System.out.println(TimeBasedOneTimePasswordUtil.validateCurrentNumber(
                shared_secret, totp_secret_first, 30));       //print true
        TimeUnit.SECONDS.sleep(30);
        System.out.println(TimeBasedOneTimePasswordUtil.validateCurrentNumber(
                shared_secret, totp_secret_first, 30));       //print false

        // change mail and issuer parameters
        // provisioning_uri can be used for QR-code scan by Google Authenticator
        String keyId = "example@mail.com";
        System.out.println(generate_uri(keyId, shared_secret));
        // String imageURL = TimeBasedOneTimePasswordUtil.qrImageUrl(keyId, shared_secret);
    }
}
```

D.3.3. JAVASCRIPT

```
// OTP with TOTP and HOTP are used for password authentication to increase security
// TOTP is commonly used version, create totp secret and verify secret within valid
//time period
// create provisioning_uri and use it by scanning barcode with Google Authenticator

const OTPAuth = require('otpauth');

// if needed, change interval parameters
function generate_totp_secret(shared_features) {
return new OTPAuth.TOTP(shared_features);
}

function generate_uri(totp_secret) {
let uri = totp_secret.toString(); // or "OTPAuth.URI.stringify(totp)"
return OTPAuth.URI.parse(uri);
}

function main() {
// create totp secret and verify within valid time period
let totp_secret = generate_totp_secret({
issuer: 'issuer name',
label: 'name@mail.com',
});
let totp_secret_first = totp_secret.generate();

// print true
console.log(totp_secret.validate({ token: totp_secret_first, window: 1 }) == 0
        ? true : false);

// change mail and issuer parameters
    // provisioning_uri can be used for QR-code scan by Google Authenticator
console.log("" + generate_uri(totp_secret));
}

main();
```

D.3.4. PHP

```
# OTP with TOTP and HOTP are used for password authentication to increase security
# TOTP is commonly used version, create totp secret and verify secret within valid
# time period
# create provisioning_uri and use it by scanning barcode with Google Authenticator
```

```
require_once(__DIR__ . '/vendor/autoload.php');
use OTPHP\TOTP;

function generate_totp_secret() {
    #use random secret
    $secret = 'JDDK4U6G3BJLEZ7Y';

    # if needed, change parameters
    $otp = TOTP::create($secret, 20, 'sha512', 8, 100);

    return $otp;
}

function generate_uri(&$otp, &$user_mail, &$issuer_name) {
    $otp->setLabel($user_mail);
    $otp->setIssuer($issuer_name);

    return $otp->getProvisioningUri();
}


$otp = generate_totp_secret();

# create totp secret and verify within valid time period
$totp_secret_first = $otp->now();
echo $otp->verify($totp_secret_first) ? "true" : "false", "\r\n";       #print true
sleep(30);
echo $otp->verify($totp_secret_first) ? "true" : "false", "\r\n";       #print false

# change mail and issuer parameters
# provisioning_uri can be used for QR-code scan by Google Authenticator
$user_mail = 'example@mail.com';
$issuer_name = 'issuer name';
$otp_uri = generate_uri($otp, $user_mail, $issuer_name);
echo $otp_uri, "\r\n";
```

D.3.5. GOLANG

```
package main

// OTP with TOTP and HOTP are used for password authentication to increase security
// TOTP is commonly used version, create totp secret and verify secret within valid
//time period
// create provisioning_uri and use it by scanning barcode with Google Authenticator
```

```
import (
"fmt"
"time"

"github.com/xlzd/gotp"
)

// if needed, change interval parameters
func generate_totp_secret() *gotp.TOTP {
return gotp.NewDefaultTOTP("4S62BZNFXXSZLCRO")
}

func generate_uri(totp *gotp.TOTP, user_mail string, issuer_name string) string {
variable := totp.ProvisioningUri(user_mail, issuer_name)
return variable
}

func main() {
// create totp secret and verify within valid time period
totp := generate_totp_secret()
totp_secret_first := totp.Now()

fmt.Println(totp.Verify(totp_secret_first, int(time.Now().Unix())))
time.Sleep(time.Second * time.Duration(30))
fmt.Println(totp.Verify(totp_secret_first, int(time.Now().Unix())))

// change mail and issuer parameters
// provisioning_uri can be used for QR-code scan by Google Authenticator
fmt.Println(generate_uri(totp, "name@mail.com", "issuer name")) //print provisioning uri
}
```

D.3.6. C#

```
// OTP with TOTP and HOTP are used for password authentication to increase security
// TOTP is commonly used version, create totp secret and verify secret within valid
//time period
// create provisioning_uri and use it by scanning barcode with Google Authenticator

using System;
using Google.Authenticator;

class Totp_class
{
    static void Main(string[] args)
    {
```

```
        //if needed, change interval parameters
        string key = Guid.NewGuid().ToString().Replace("-", "").Substring(0, 10);
        string issuer = "issuer name";
        string accountTitle = "name@mail.com";

        // create totp secret and verify within valid time period
        TwoFactorAuthenticator tfa = new TwoFactorAuthenticator();
        SetupCode setupInfo = tfa.GenerateSetupCode(issuer, accountTitle, key, false, 3);

        string second_factor_now = tfa.GetCurrentPIN(key, false);

        //print true
        Console.Write(tfa.ValidateTwoFactorPIN(key, second_factor_now) + "\r\n");

        // change mail and issuer parameters
        // provisioning_uri can be used for QR-code scan by Google Authenticator
        var provisionUrl = string.IsNullOrWhiteSpace(issuer)
            ? $"otpauth://totp/{accountTitle}?secret={key.Trim('=')}"
            : $"otpauth://totp/{Uri.EscapeDataString(issuer)}:{accountTitle}
            ? secret={key.Trim('=')}&issuer={Uri.EscapeDataString(issuer)}";

        Console.Write(provisionUrl + "\r\n");

    }
}
```

E. Additional Tables

The tables in this section contain more detailed information given by the participants.

Table E.1.: Resources Participant Used in Survey

P1 Python	no resources used
P2 Python	"my repo"
P3 Python	https://argon2- cffi.readthedocs.io/en/stable/parameters.html https://argon2.online https://pypi.org/project/bcrypt https://zetcode.com/python/bcrypt https://regex101.com https://regex-generator.olafneumann.org https://pyotp.readthedocs.io/en/stable
P3 Golang	https://pkg.go.dev/golang.org/x/crypto/argon2 https://www.codementor.io/@supertokens/how-to-hash-salt-and-verify-passwords-in-nodejs-python-golang-and-java-1sqko521bp https://www.oreilly.com/library/view/regular-expressionscookbook/9781449327453/ch04s19.html https://www.ibm.com/docs/en/spim/2.0.0?topic=administration-passwordpolicies https://www.digicert.com/blog/creating-password-policy-best-practices https://www.regextester.com https://regexr.com https://golangci-lint.run https://deepsource.io https://sonarcloud.io https://viezly.com
P4 Python	https://pythonlang.dev/repo/hynek-argon2-cffi https://medium.com/@taimoor.mirza9595/integrating-2fa-mfa-using-pyotp-193ad858f80d
P4 Java	https://mvnrepository.com/artifact/de.mkammerer/argon2-jvm/2.6 https://docs.spring.io/springsecurity/site/docs/current/api/org/springframework/security/crypto/argon2/Argon2PasswordEncoder.html https://www.programcreek.com/java-api-examples/?api=org.springframework.security.crypto.argon2.Argon2PasswordEncoder https://security.snyk.io/package/maven/org.mindrot:jbcrypt/0.4 https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORKSECURITY-570203 https://www.vaultproject.io/ https://www.vaultproject.io/docs https://www.tabnine.com/code/java/classes/com.nulabinc.zxcvbn.Zxcvbn https://github.com/j256/two-factor-auth https://docs.microsoft.com/en-us/azure/active-directory/authentication/concept-password-ban-bad#global-banned-password-list
P5 Python	"Stackoverflow"
P5 Java	no resources used

P1: Participant-1, P2: Participant-2, P3: Participant-3, P4: Participant-4, P5: Participant-5