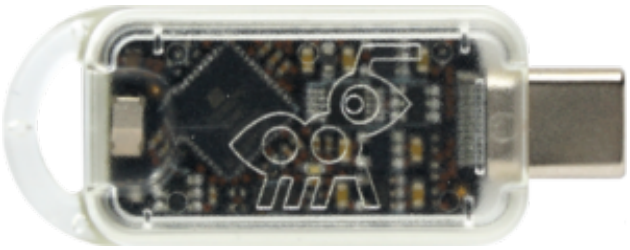
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Passwordless Authentication System Using TKey

Leveraging Asymmetric Cryptography with TKey for Secure Passwordless Authentication

Bachelor's thesis in Computer Science and Engineering

Ismail Al Horan
Amanch Ali
Wilmer Greppe
Max Levin
Albin Skeppstedt
M-Alaa Zabateh

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Bachelor's thesis 2025

Passwordless Authentication System Using TKey

Leveraging Asymmetric Cryptography with TKey for Secure Passwordless Authentication

Ismail Al Horan
Amanch Ali
Wilmer Greppe
Max Levin
Albin Skeppstedt
M-Alaa Zabateh

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Passwordless Authentication System Using TKey
Leveraging Asymmetric Cryptography with TKey for Secure Passwordless Authentication
Ismail Al Horan
Amanch Ali
Wilmer Greppe
Max Levin
Albin Skeppstedt
M-Alaa Zabateh

Supervisor: Yasir Hussain, Department of Computer Science and Engineering.
Examiners: Patrik Jansson and Arne Linde, Department of Computer Science and Engineering.
Graded by teacher: Irum Inayat, Department of Computer Science and Engineering.

Bachelor's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Passwordless Authentication System Using TKey
Leveraging Asymmetric Cryptography with TKey for Secure Passwordless Authentication
Ismail Al Horan, Amanch Ali, Wilmer Greppe, Max Levin, Albin Skeppstedt, M-Alaa Zabateh

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Threats targeting authentication systems are becoming more widespread in the current digital landscape, and passwords are challenged in their role as the dominant authentication method due to poor user habits and cognitive limitations. This thesis describes the design and implementation of a passwordless authentication system developed in Python and Go for a simple web application and proxy server. A two-factor authentication system was implemented utilizing a TKey (Chalmers version) from Tillitis AB with Ed25519 signing, as well as Time-based One-Time Password (TOTP) functionality. Additionally, a recovery mechanism was implemented that utilizes mnemonic phrases to handle the loss of a TKey. The project explores the possibilities of passwordless authentication using a TKey, and in the broader sense hardware-based authentication. It demonstrates that hardware-based authentication schemes relying on cryptographic signatures are effective candidates for tomorrow's authentication systems.

Sammanfattning

Hot mot autentiseringstjänster blir allt vanligare i dagens digitala landskap, och lösenord ifrågasätts som den dominerande autentiseringsmetoden på grund av användares dåliga vanor och kognitiva begränsningar. Denna uppsats beskriver designen och implementationen av ett lösenordslöst autentiseringstystem, utvecklat i Python och Go för en enkel webbapplikation och proxyserver. Ett tvåfaktorsautentisering system implementerades med hjälp av en TKey (Chalmers-version) från Tillitis AB med Ed25519-signering samt Time-based One-Time Password (TOTP)-funktionalitet. Dessutom implementerades en återställningsmekanism som använder minnesfrågor för att hantera förlust av en TKey. Projektet undersöker möjligheterna med lösenordslös autentisering via en TKey, och i ett vidare perspektiv hårdvarubaserad autentisering. Det visar att autentisering baserad på hårdvara och kryptografiska signaturer är en effektiv lösning för framtidens autentiseringstjänster.

Keywords: Cybersecurity, Cryptography, Authentication, Hardware-based authentication.

Acknowledgements

We would like to thank Sasko Simonovski, CEO of Tillitis AB, for his knowledge and kind help with the TKey and existing compatible software. We also want to extend our thanks to our supervisor Yasir Hussain for supporting us through the whole spring term, giving us guidance and valuable insights.

Ismail Al Horan, Amanch Ali, Wilmer Greppe, Max Levin, Albin Skeppstedt and M-Alaa Zabateh, Gothenburg, May 2025

Contents

List of Figures

List of Tables

1

Introduction

As digital technology becomes a more central part of society, authentication systems continue to play a critical role in protecting online services and sensitive information from malicious actors. This trend is expected to continue as our reliance on digital infrastructure increases. Given the high value that attackers can obtain by exploiting authentication procedures, it is no surprise that vulnerable authentication systems still rank among the highest-risk vulnerabilities, as seen in the latest OWASP Top 10 report [1]. It is therefore essential to ensure that these critical systems are kept secure in order to protect confidentiality and maintain user trust.

For decades, the standard method of authentication has been traditional password-based systems, in which the client proves their identity by providing a secret password known only to them. In recent years however, the traditional password-based approach has been challenged in its role as the dominant method for authentication, due to widespread issues with password misuse and the significant number of vulnerabilities and exploits related to passwords.

The human factor is one of the fundamental reasons that passwords are vulnerable as authentication methods due to poor password practices and cognitive limitations. This is evident in user behavior. For example, Riley [2] found that 54.6% of the participants reused the exact same passwords for multiple services, while 33% reused passwords with slight variations. Furthermore, 35% of the participants used passwords of a predetermined length, with an average of 6.84 characters. A study by Florencio and Herley [3] further reinforced these findings, revealing that the average user reused passwords in five to six services and that most passwords lacked sufficient complexity and length. Passwords consisting solely of letters were dominant, while more secure combinations involving various character types were relatively rare.

The security implications of such behavior are significant. Bošnjak et al. [4] demonstrated that 99.5% of approximately 150,000 collected passwords were crackable in 36 hours, using established techniques on commonly available hardware. Password managers could help mitigate this risk by promoting the use of strong, unique passwords without requiring users to remember each one. However, it is important to note that they themselves are a growing target for attacks [5], and even strong passwords are vulnerable to attacks that target the social aspect. Social engineering, phishing attacks, and credential theft are some of the commonly used methods to

gain unauthorized access to other people's accounts. These vulnerabilities highlight the need for more secure authentication methods, including two-factor authentication or passwordless solutions.

One promising avenue in this space is the use of hardware-based security tokens, which provide a higher level of assurance by requiring physical possession of a device during the authentication process. In contrast to passwords, which are considered "something you know", hardware-based security tokens are considered "something you have", which makes it significantly harder for a remote attacker to steal credentials used for authentication.

A practical example of such technology is the TKey, a compact USB device developed by the Swedish company Tillitis AB [6], designed to provide a secure environment for applications that prioritize security. It generates cryptographic keys based on three factors: a unique ID, an optional user secret, and the identity of the application it interacts with [7]. This design ensures that a tampered application fails to generate valid keys, thereby preventing unauthorized access. The device is open source and supports a wide range of use cases, including SSH login, Ed25519 signing, Root of Trust, FIDO2, TOTP, and Passkey [7].

Although the TKey offers a robust solution for secure authentication, its adoption among general consumers may be limited due to its physical nature and technical complexity. However, its potential is particularly well-suited to organizational environments where high security standards are critical as the consequences of unauthorized access can be especially severe. Centralized device management helps organizations enforce consistent authentication policies and reduce security overhead. In such contexts, the TKey can help mitigate risks such as password theft, phishing, and account takeover by offering a more secure and administratively efficient alternative to passwords.

1.1 Problem Statement

Password-based authentication remains widely used despite its well-documented vulnerabilities. Threats such as phishing, brute-force attacks, and credential theft continue to compromise user accounts and organizational data. In enterprise environments even a single breach can result in severe financial loss and reputational damage.

Although hardware-based security tokens and other passwordless approaches have been developed to address these issues, they often face limitations in cost, scalability and integration complexity. The TKey, with its open-source design and flexible architecture, offers a promising alternative for secure passwordless authentication. However, adopting the TKey in a passwordless authentication system involves overcoming several technical and usability challenges. These include key management, web based integration and ensuring a secure and user friendly experience. These limitations underscore the need to investigate whether the TKey can be effectively

adopted for modern web authentication scenarios.

1.2 Aim

This thesis aims to design and implement a web-based authentication system that uses the TKey device to enable passwordless login. The system is designed to offer a secure and user-friendly authentication flow with compatibility for integration into enterprise-grade web platforms.

1.3 Objectives

To achieve this aim, the project will pursue the following objectives:

- System Architecture Development: Design and implement a system architecture comprising a Go-based proxy server that facilitates secure communication between the TKey device and a web server. This proxy is intended to be locally installed on the user's machine.
- Implementation of prototypes: Develop a web application that demonstrates how a TKey-based passwordless login could function in a real-world setting. Although not designed for immediate integration with production environments, the prototype serves as a practical example of what such a system might look like in production. More work would be required to achieve the seamless experience seen in major single sign-on systems.
- Security Considerations: Address potential attack vectors such as man-in-the-middle attacks and device tampering by using secure communication protocols and robust error handling.
- Two-Factor Authentication (2FA): Integrate time-based one-time passwords as a second authentication method to support 2FA.
- Account Recovery Mechanism: Develop a mnemonic phrase-based account recovery mechanism that enables account restoration in case of account loss or inaccessibility to the TKey.

1.4 Scope and Limitations

This project focuses on the development of a working prototype and is limited by the time and resources available on a semester-long academic schedule. The implementation focuses on secure web-based authentication using the TKey device, without addressing broader aspects such as full-featured user account management, cross-platform support, or mobile implementations.

Physical limitations of the TKey itself, such as potential loss, hardware failure, or

user inexperience, are recognized, but fall outside the scope of this project. A basic recovery mechanism will be implemented, but the comprehensive management of the device life cycle, including revocation and replacement of provisioning, is beyond the scope of this project.

Additionally, the project will be developed using only the available hardware (TKey devices) and software environments accessible to the development team. It will not utilize any external infrastructure or third-party services. User experience testing, such as usability studies or feedback, is excluded from the scope of this thesis. The focus remains on demonstrating technical feasibility and implementing core authentication functionality.

2

Theory

This chapter outlines the theoretical foundation needed to understand the cryptographic components of the project. The focus is on explaining core concepts in a general and implementation-independent manner. These include public key cryptography, authentication protocols such as challenge-response, and techniques related to key generation and derivation. Together, these concepts form the basis for understanding how secure, passwordless authentication systems can be designed and what role devices like the TKey play in such systems. This background will support the reader in grasping the technical decisions and implementations discussed in subsequent chapters.

2.1 Hashing functions

One essential component of public key cryptography systems is the hash function. A hash function is a function taking an input and producing an output that satisfies certain conditions [8, Ch 2]. It should take variable length bit-strings and produce fixed length bit-strings [8, Ch 2]. Each output should have an equal probability of occurring [9, Ch 7]. In cryptography, it is necessary that a hash function H is collision resistant, meaning that finding two messages m_1 and m_2 so that $H(m_1) = H(m_2)$ is infeasible [8, Ch 2].

A commonly used hash function in cryptography is SHA-2 [10, Ch 8]. It is a fast algorithm used in routine cryptographic operations, such as producing hashes of messages to sign [10, Ch 8]. It works by taking 512 bit words of the input message and a vector of predefined bit words (based on the fractional part of certain square and cubic roots of primes) and performing different operations such as adding, rotating and choosing on the word [10, Ch 8]. This process is repeated for each 512 bit word in the input bit-string [10, Ch 8]. SHA-256 and SHA-512 are well used variants and their suffixes stand for how long the hash digests are [10, Ch 8].

2.2 Public Key Cryptography

For most of history, since the Roman times until just recently, the only form of encryption used was symmetric cryptography which has a classic example, the Caesar

Cipher [8, Ch 2]. It works by shifting each character in a plain-text n times and is simple but illustrates well how symmetric encryption schemes are constructed [8, Ch 2]. More elaborate techniques have since been developed [11, Ch 1]. However, they share the common trait of requiring both the sender and the receiver(s) sharing a secret key [8, Ch 2]. When n actors in a network wants to have private communication, the number of keys grows by $f(n) = \frac{n(n-1)}{2}$. This becomes an issue regarding efficient distribution of keys [8, Ch 2].

In 1976, Diffie-Hellman proposed a different approach to encryption [12] and public key cryptography was born [9, Ch 9]. The concept of public key cryptography lowers the order of keys in a network to linear growth [8, Ch 2]. It works by each actor having two keys, one public and one private in a pair [8, Ch 2]. In simple terms, encrypting with the public key is a one-way operation unless you have the corresponding private key and in the dawn of the field it worked by utilizing properties of prime numbers [9, Ch 10]. It is inefficient to factorize large primes which makes it infeasible for an adversary to find the inverse of a number in a group [11, Ch 2]. This fact gives the confidentiality property to a ciphertext encrypted with public key cryptography using primes [11, Ch 2].

Instead of using modular prime number arithmetics as in Diffie-Hellman, elliptic curves are mathematical functions commonly used in cryptography and are of the form $y^2 = x^3 + ax + b$ [10, Ch 15]. Under certain values of a and b these curves form groups that can be used instead groups generated by prime numbers [10, Ch 15]. Thus they are useful for cryptographic purposes [10, Ch 15].

For natural number groups in general and prime order groups specifically the group operation is addition of the generator modulo the order of the group [9, Ch 9] whereas elliptic curves has a slightly more complex group operation. The addition of two points is the point where the line between those points intersects with the curve, but reflected through the x-axis [10, Ch 15]. In case of doubling a point, for example P giving P^2, the product is the point intersecting with the tangent of P reflected in the x-axis [10, Ch 15]. This is illustrated in 2.1, $A + B = D$. A group can be constructed by starting from a valid generator [10, Ch 15].

The relevant assumption is that it is easy to compute a point from a base point P by doing the group operation n times, resulting in P^n [10, Ch 15]. However it is infeasible to find the discrete log, n, of P^n (P added to itself n times) by just knowing P and P^n [10, Ch 15]. The reason to use elliptical curves instead of prime order groups is that key sizes are reduced [10, Ch 15]. This is due to the fact that the fastest running algorithm for solving the discrete log of an elliptical curve group is in $\mathcal{O}(\sqrt{n})$ while it is $exp(\tilde{\mathrm{O}}(\log n^{1/3}))$ with prime number groups [10, Ch 15].

An important application of public key cryptography is signing cipher-texts to prove the legitimacy of information [10, Ch 13]. The operation of encrypting a plain-text can be done with both the private and public key and encrypting using the private key is what is known as signing a plain-text [10, Ch 13]. Under the assumption that the expected sender of a cipher-text is the sole owner of a private key and if a

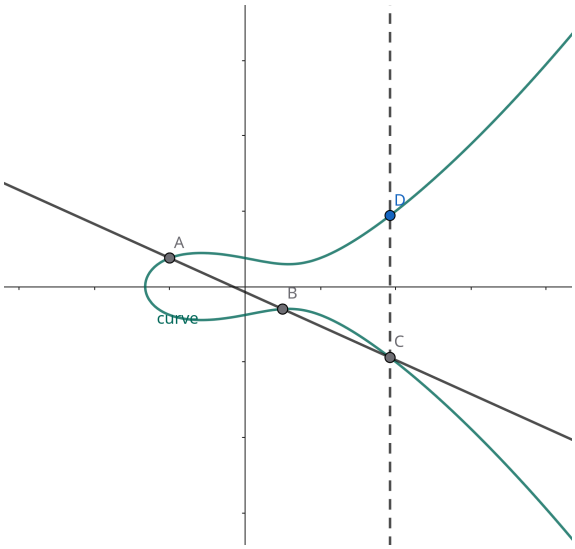


Figure 2.1: Illustration of addition of points on the elliptical curve $y^2 = x^3 - x + 1$.

receiver knows the public key of the expected sender and the expected plain-text, they can verify that the expected corresponding private key was used to sign the text [10, Ch 13].

The process of signing using public key cryptography is called a digital signature scheme [10, Ch 13]. A common digital signature scheme is Ed25519 [10, Ch 15]. It uses the Edwards25519 elliptical curve

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \bmod 2^{255} - 19 \tag{2.1}$$

and SHA-512 for hashing [10, Ch 15]. According to [13] the steps for an Ed25519 signature are as follows where $||$ is the string concatanation operator:

1. Using a random key generation algorithm, generate a 32-byte private key K.
2. Compute the hash $H(K) = h_{0...63}$ which gives 64 bytes, where $h_{0...31} = a$.
3. Compute the public key $A = aB$ where B is a base point on the curve, the generator of the group.
4. To sign a message M, hash K, compute $r = H(h_{32...63}||M) \bmod L$ where L is the order of the group generated by B. Compute the point $R = rB$ on the curve.
5. Compute $h_m = H(R\|A\|M) \bmod L$ and $S = s + h * a \bmod L$. Then the signature is $R\|S$.

Verification given a signature, $R||S$, a message M and a public key A is explained by [13] as

1. Compute $h = H(R||A||M) \bmod L$.

2. The signature is valid if the equation $SB = R + hA$ holds.

To clarify, points on the Edwards curve is on the form (x, y) but they are encoded to be represented as 32 byte numbers [13]. This is done by taking the y-coordinate in little endian, omitting the most significant bit and using that memory space to represent the sign of the x-coordinate [13]. This encoding works because the x-value for each y-value can be solved for using the equation [13].

2.3 Challenge-Response protocol

A Challenge-Response protocol is a form of authentication scheme used to verify the identity of a specific subject, where the subject is tasked with responding to a challenge that was generated by the verifier. The challenge typically takes the form of a randomly generated and unique value, which the subject has to combine or use together with a secret only they have access to. For example, the subject might create the response by applying a cryptographic function with a private key on the challenge, that then acts a personal signature. The verifier can then use the corresponding public key to verify the authenticity of the signature and authenticate the subject if it is valid [14].

Instead of relying solely on directly providing a secret, for example a password, a challenge-response protocol enables the possibility for each challenge to be unique and unpredictable. This helps mitigate various attacks such as Brute-force attacks, an exhaustive attack where the adversary goes through and tries all possibilities until they find the correct one [8, p.927], and Replay-attacks, where authorization is attempted to be gained by submitting an already accepted request [8, p.937].

2.4 Two-Factor Authentication

With the growth of the internet and the increased use of shared digital environments, the need to protect user identity and personal data became more urgent. Passwords were introduced as a basic form of authentication, and for many years, passwords served as the primary method of protecting access to systems and accounts.

However, as the use of passwords increased, attackers developed advanced techniques to compromise them, including several types of attacks and password guessing algorithms. This revealed the weakness of relying on only one layer of protection.

In 1995, AT&T introduced the concept of two-factor authentication(2FA) and patented it in 1998 [15]. Their solution laid the foundation for modern multi-factor authentication systems, where users are required to present more than just a password to gain access. The purpose of this approach was to provide stronger protection for systems and data even in cases where passwords are stolen.

2.4.1 Definition and Purpose

Two-factor authentication (2FA) is a security process that requires users to verify their identity through two different types of authentication factors. This typically includes:

- Something the user knows (such as password or PIN)
- Something the user has (such as hardware token or mobile)
- Something the user is (such as fingerprint or Face-ID)

The key advantage of this layered approach lies in its ability to reduce the risk of unauthorized access and improve security by making it much harder for attackers to gain entry with only the user's password. A second factor, such as a physical device or an authenticator app, is still required to complete the login process [16].

2.4.2 Time-Based One-Time Password (TOTP)

One of the second most commonly used factors in 2FA is the Time-Based One-Time Password (TOTP). TOTP is a type of algorithm that generates short-lived codes that are typically generated through authenticator apps every 30 seconds. The user enters this code during the login process as a second step of authentication [17].

In the context of our project, two-factor authentication is implemented by integrating the Tkey device from Tillits with a time-based one-time password (TOTP) system, provided through authenticator applications (such as Google Authenticator). When used together, the hardware device ensures strong protection against impersonation and key extraction, while the TOTP mechanism adds a practical and widely supported second layer of defense. This approach improves the integrity and confidentiality of authentication systems and provides enhanced resistance against several types of threats and attacks.

In the end, the 2FA process has become a standard recommendation in many security guidelines and is often required in many sensitive applications such as banking and identity management systems.

2.5 Mnemonic Phrase

A mnemonic phrase is a randomly generated sequence of words, which can be created according to the Bitcoin Improvement Proposal 39 standard [18]. The mnemonic phrase is a visual representation of a cryptographic key, commonly used as a password to backup cryptocurrency wallets. [18]

2.5.1 Bitcoin Improvement Proposal (BIP-0039)

Bitcoin Improvement Proposal (BIP-0039) is a standard used to generate cryptographical keys in the form of a mnemonic phrase, which is then converted into a seed [19]. This seed is then used to generate cryptographic keys, making it user-friendly for users to maintain their keys in their deterministic wallets [19]. As defined in BIP-39 [19], the standard can be divided into two parts: generating a mnemonic phrase and generating a seed from a mnemonic phrase.

2.5.2 Generating a mnemonic phrase

To generate a mnemonic phrase, BIP-0039 [19] uses entropy. Entropy is a randomly generated binary sequence consisting of bits 1 or 0. The length of the binary sequence must be within the range of 128 to 256 bits and also a multiple of 32 [19].

Furthermore, the binary entropy sequence is hashed using the SHA-256 algorithm. The result is a 256-bit binary hash, which is then used to generate a checksum. The length of the checksum is calculated using Equation 2.2, where the "length of entropy" refers to the number of bits in the entropy.

To compute the checksum, the n most significant bits of the hash are taken, where n is the length of the checksum. The checksum is appended at the end of the entropy, before the least significant bits. The length of the new binary sequence (that is, entropy || checksum) determines the number of words in the mnemonic phrase 2.3.

Finally, this binary sequence is split into segments of 11 bits. Each 11 bit segment represents an index ranging between 0-2047, corresponding to a word in the word list. Together, every word forms the mnemonic phrase.

$$\text{Length of checksum} = \frac{\text{Length of entropy}}{32} \tag{2.2}$$

$$\text{Length of mnemonic phrase} = \frac{\text{Length of entropy} \,\|\, \text{Length of checksum}}{11} \tag{2.3}$$

2.5.3 Generating a seed from a mnemonic phrase

To generate a seed from a mnemonic phrase, BIP-0039 [19] uses the key derivation function PBKDF2 in combination with the pseudo random function HMAC-SHA512. PBKDF2, when combined with HMAC-SHA512 takes three parameters as inputs: a password, a salt, and iterations, as shown in (Equation 2.4). In the context of BIP-0039, the password corresponds to the mnemonic phrase. By default, the salt is the string `"mnemonic"`, optionally it can be extended by concatenating it with a defined passphrase. The number of iterations is set to 2048. The output of the function is the seed, a 512-bit binary sequence.

$$\text{Seed} = \text{PBKDF2}_{\text{HMAC-SHA512}}(\text{Password} = \text{mnemonic},\ \text{Salt} = \text{"mnemonic"} + \text{passphrase},\ \text{Iterations} = 2048) \tag{2.4}$$

2.6 Password-Based Key Derivation Function 2 (PBKDF2)

Kaliski explains in *PKCS #5: Password-Based Cryptography Specification Version 2.0* [20] that PBKDF2 is a Key Derivation Function (KDF) used to derive cryptographic keys from a password. It enhances security by incorporating additional parameters compared to a regular one-way hash function [21]. These parameters are as follows:

1. Salt: a randomly generated or fixed string that is hashed with the secret data. The salt ensures unique keys even for identical passwords, which in turn protects against dictionary attacks.

2. Iteration count: The iteration count is the number of times the pseudo random function is applied. The higher the iteration count, the higher the computational cost of brute-force attacks.

PBKDF2 applies a pseudo random function (PRF) during the derivation process, such as HMAC. It takes four parameters as input: a password, a salt, an iteration count (`c`), and the desired length of the derived key. The desired key length determines how long the derived key should be.

The algorithm of the PBKDF2 function, as defined by Burt Kaliski [20], is structured as in (Equation 2.5). Kaliski explains the process as in the following steps.

(1) Check if LenDerivedKey is greater than $(2^{32}-1)\cdot(LenHash)$, where LenHash is the length of the output from the hash function used in the pseudo random function. For example, if the PRF is HMAC-SHA512, then LenHash is 512 bits. If LenDeriveKey is greater, the algorithm cannot be performed and the desired key length needs to be reduced.

(2) Next, as in (Equation 2.5b), the derived key is a concatenation of T_i starting from 1 to l.

(3) Furthermore, T_i is defined as in (Equation 2.5c), taking a new parameter i as input for F.

(4) The function F is the c iterations of a chain of XOR operations between the values of U_c as in (Equation 2.5d), where c is the count of iterations.

(5) This step is where the calculation takes place, i.e. where the pseudo random function is applied. The U_c values as described in step (4) are calculated as in

(Equations 2.5e and 2.5f). The first U_1 is calculated by applying the PRF to the password and the salt concatenated with the block index $INT(i)$. Here, $INT(i)$ is a 32-bit binary representation of the block index value.

(6) Finally, U_1 is calculated and then each value of U_j is computed as described in (Equation 2.5e). U_j is calculated by applying the pseudo random function to the password and the previous value of U_{j-1}. These U values are then used in (4).

$$\text{DerivedKey} = \text{PBKDF2}_{\text{PRF}}(\text{Password},\ \text{Salt},\ \text{c},\ \text{LenDerivedKey}) \tag{2.5a}$$

$$= T_1 \parallel T_2 \parallel \cdots \parallel T_l, \quad l = \frac{\text{LenDerivedKey}}{\text{LenHash}} \tag{2.5b}$$

$$T_i = F(\text{Password},\ \text{Salt},\ c,\ i), \quad \text{for } i = 1, \ldots, l \tag{2.5c}$$

$$F(Password, Salt, c, i) = U_1 \oplus U_2 \oplus \cdots \oplus U_c\,, \quad \text{for } c = 1, \ldots c \tag{2.5d}$$

$$U_j = \text{PRF}(Password,\ U_{j-1}), \quad \text{for } j = 2, \ldots, c \tag{2.5e}$$

$$U_1 = \text{PRF}(Password,\ Salt \parallel \text{INT}(i)) \tag{2.5f}$$

2.7 Hash Message Authentication Code (HMAC)

Hash message Authentication code (HMAC) as described in *HMAC: Keyed-Hashing for Message Authentication* by Hugo Krawczyk, Mihir Bellare and Ran Canetti [22], is a cryptographic algorithm commonly used to authenticate messages. The purpose of HMAC is to ensure the integrity and authenticity of a message transmitted between two parties.

For example, if Alice wants to send a message to Bob, she can compute an HMAC over the message with the shared secret key that both parties know. When Bob receives the message, he can recompute the HMAC (i.e. he knows the shared secret key). If Bob's recomputed HMAC matches Alice's, Bob can be confident that the message has not been tampered with and that it is sent from Alice.

HMAC is also commonly used as a pseudo random function in password based key derivation algorithms, such as PBKDF2 [20]. HMAC acts as a pseudo random function when combined with a chosen hashing algorithm (e.g., SHA-256) [22, p. 2]. In this context, HMAC ensures that the derived cryptographic key depends on the password and the salt securely.

As described in *Computer Security: Principles and Practice* by Stallings [23], the algorithm for HMAC is defined as shown in (Equation 2.6). The following variables in (Equation 2.6) are fixed: `ipad = 0x36 (hex) = 00110110 (binary)`, `opad = 0x5C (hex) = 01011100 (binary)`. Furthermore, `Hash` refers to the selected hash-

ing algorithm, $\oplus$ represents an XOR operation, as mentioned above `||` is concatenation, and both key and message are represented as byte strings.

William Stallings [23] explains the HMAC algorithm in seven steps:

(1) First, a hashing algorithm is selected. The block size for that selected algorithm will be used throughout the process.

(2) Then, the length of the key is compared to the block size. If the key is shorter than the block size, it is padded with zeros until it matches the block size. For example, if the block size is 64 bytes, and the key is `0x6b3486` (3 bytes), 61 bytes of zeros are padded: `0x6b3486000000...`

But if the key exceeds the block size, it is first hashed with the selected hashing algorithm. If the resulting hashed key is shorter than the block size, it is padded with zeros until it matches the block size.

(3) The following operation is performed: (padKey $\oplus$ ipad), where the padded key from (2) is bitwise XOR'ed with `ipad` n times, where n is the block size from (1). For example, padded key : `0x6b3486000000...` and ipad: `0x36`, the operation is done byte by byte: `0x6b` $\oplus$ `0x36` , `0x34` $\oplus$ `0x36`, `0x86` $\oplus$ `0x36`, `0x00` $\oplus$ `0x36`, etc., resulting in a new byte string `0x5d02b0363636...`.

(4) The result of (3) is concatenated with the message and then hashed with the selected hashing algorithm from (1): `Hash((padKey` $\oplus$ `ipad) || message)`.

(5) The following operation is then performed: `padKey` $\oplus$ `opad`, where the padded key from (2) is bitwise XOR'ed with `opad`, similar to step (3).

(6) The result of (5) is concatenated with the hash calculated in (4).

(7) Finally, the result of (6) is hashed with the selected hashing algorithm from (1).

$$\text{HMAC}(Key, message) = Hash\left((padKey \oplus \text{opad}) \parallel Hash\left((padKey \oplus \text{ipad}) \parallel message\right)\right) \tag{2.6}$$

3

Methods

This chapter describes the technical approach used to develop and evaluate secure, passwordless authentication system based on the TKey hardware device. It begins with an overview of the collaborative development, followed by a high-level summary of the system architecture and underlying technologies. The subsequent sections explain the implementation process and strategies for testing and validation. Finally, the chapter addresses the key limitations and assumptions identified during development.

The objective is to provide a comprehensive technical understanding of the system, with an emphasis on how individual components integrate and interact to support a secure authentication workflow. We include implementation details where relevant, while broader performance and usability considerations are addressed separately in the results chapter.

3.1 Project Workflow and Collaboration

The project was conducted by a team of six students, with tasks divided based on the main components of the system. The work was carried out using a combination of pair programming and independent tasks, depending on the complexity and urgency of the features being developed.

Weekly in-person meetings were held to synchronize progress, discuss technical challenges, and plan upcoming work. These meetings provided a structured environment for raising issues that required team-wide input or guidance from the team's supervisor.

Semi-regular meetings were also held with the assigned supervisor to present project updates, discuss unresolved questions, and receive feedback. These interactions ensured alignment with academic requirements and helped guide the direction of development at key stages.

3.2 System Design and Architecture

The architecture of the passwordless authentication system was developed with a focus on security, modularity, and usability. The system consists of six primary components:

- TKey device: The USB hardware token used for authentication.
- Browser interface: The interface with which the user interacts.
- Web server: Handles the back-end of the web application.
- Proxy server: Handles communication between the TKey and the front end.
- Database: Stores user data.
- Session store: Stores user sessions.

These components are integrated to support secure user registration and login workflows. Authentication is performed using asymmetric cryptography, with additional protection provided by time-based one-time passwords (TOTP) as a second factor. Figure 3.1 illustrates how the system components interact during the authentication process.

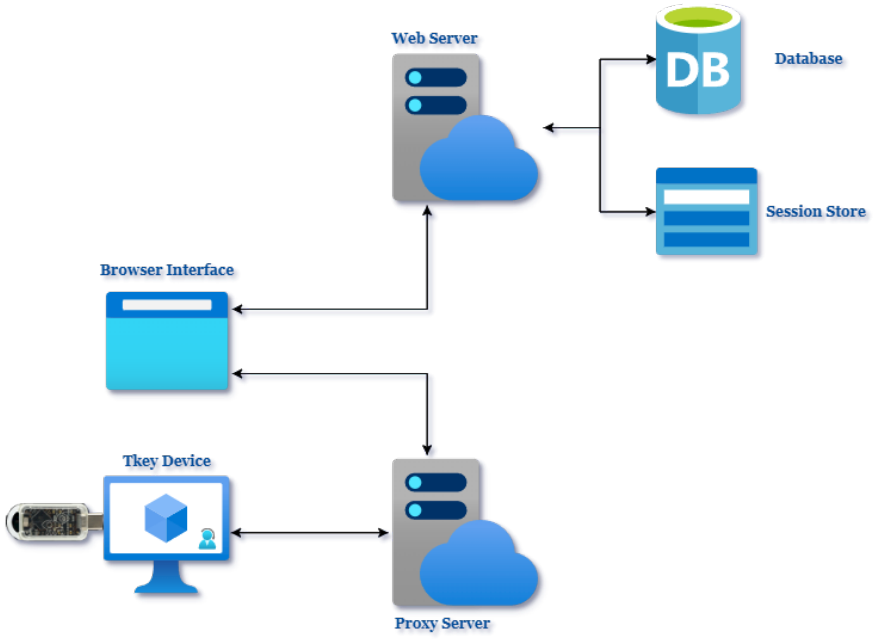


Figure 3.1: System Design

3.2.1 TKey device

The TKey device is a hardware-based cryptographic authenticator that provides secure generation and use of cryptographic keys. The device hosts a secure en-

vironment and keeps all cryptographic keys and operations away from the user's computer. This helps protect sensitive data from software attacks. The TKey generates cryptographic keys by combining three unique inputs:

- A built-in Unique Device Secret (UDS) .
- An optional User Supplied Secret (USS), such as a PIN.
- The binary representation of the loaded application.

This combination ensures that each application running on the TKey has a unique cryptographic identity, significantly protecting against impersonation attacks and unauthorized access. However to keep the application passwordless the User Supplied Secret was not used. As previously mentioned, the TKey supports Ed25519 signing which is utilized to enable authentication with help of the TKey.

3.2.2 Web Server & Browser Interface

The backend web server is responsible for handling all authentication steps included in the registration and login process. This includes generating challenges, verifying signatures and TOTP codes, communication with the database and session management. It processes incoming request and generates responses that the client uses for different tasks, and lastly is responsible for serving the web interface that the user interacts with in the browser.

3.2.3 Proxy Server

The proxy server acts as an intermediary, facilitating communication between the TKey and the client's browser. Its primary responsibilities include:

- Detecting and establishing a secure connection with the TKey device.
- Loading the authentication application onto the TKey.
- Managing the transfer of cryptographic challenges and signatures between the browser and the TKey device.

By isolating cryptographic signing operations to a separate hardware device via the proxy server, the system significantly enhances security and reduces potential attack attempts.

3.2.4 Database and Session store

The database stores user-related information necessary for authentication, including public keys and secret values used in TOTP verification. This data is retrieved and validated during login and registration processes. Additionally, a seed derived from the user's mnemonic phrase, combined with a salt, is stored to support account recovery procedures.

Session storage is handled separately from the client's browser. Instead of storing session data in browser cookies, a reference or token is provided to the client, while the actual session data is maintained on the server in a session store. This allows the server to securely manage session life cycles, including revocation of invalid or expired sessions, thereby reducing the risk of session hijacking or unauthorized reuse.

3.3 Technologies and Tools

This section outlines the technologies and tools selected for the development of the authentication system. Each technology was chosen with consideration of its compatibility with the system's architecture, its ability to support cryptographic operations securely, and the overall efficiency of development and deployment.

3.3.1 Go - Proxy Server

The proxy server was implemented in Go. A significant factor in this decision was that the TKey device's existing software, including its cryptographic functionality, was also written in Go. This alignment allowed for a streamlined integration and facilitated early-stage development by enabling direct reuse of existing libraries from Tillitis AB.

3.3.2 Python and Flask - Web Server

Python was selected as the primary language for the web server, with Flask serving as the web framework. Flask was chosen due to its lightweight structure, ease of use, and support for secure handling of web requests. These features made it a practical choice for implementing the system's core authentication logic, including cryptographic challenge generation and TOTP verification.

3.3.3 HTML, CSS, and JavaScript - Web Interface

The user-facing web interface was developed using plain HTML, CSS, and JavaScript. As the project focused primarily on back-end authentication mechanisms and hardware integration, the front-end design was intentionally kept minimal. These foundational web technologies were sufficient to provide a clear, accessible interface for user interactions such as registration and login, without introducing unnecessary complexity or dependencies.

3.3.4 PostgreSQL - Database

For persistent data storage, PostgreSQL was chosen over lighter alternatives such as SQLite. While initial prototyping began with SQLite, PostgreSQL was ultimately selected due to its robustness, scalability, and advanced security features. It provided reliable storage for sensitive user data such as public keys, TOTP secrets, and recovery information, making it a suitable choice for a system intended to operate in a production environment.

3.3.5 Redis - Session Store

Session data was managed using Redis, an in-memory storage solution. Instead of storing session information in browser cookies, the web server issued session tokens referencing data stored securely in Redis. This enhanced security by keeping sensitive session information server-side and allowed for efficient invalidation of sessions when necessary. Using Redis also reduced the additional writes to the database, if it would have been used to store the session information.

3.3.6 Docker and Docker Compose - Deployment and Environment Management

Docker and Docker Compose were used to containerize all components related to the web application, including the web server, PostgreSQL database, and Redis. The web server was built from source, while the images for PostgreSQL and Redis were pulled from Docker Hub, a repository of Docker images available to the public. This approach ensured consistent development environments and simplified the deployment process.

3.4 Implementation

The implementation of the passwordless authentication system was conducted in several steps, beginning by getting familiar with the TKey technology itself and its relevant applications, as well as the preparation of the development environment. This was followed by building the parallel development of the initial implementations of the proxy server and web application, and once a sufficient level of completeness was achieved, establishing connections between the components. Lastly, additional security features were implemented such as TOTP, and refined the internal processes to increase robustness and security.

3.4.1 Environment setup

In this project, the work initially focused on identifying and configuring the necessary tools, libraries, and applications to ensure that all parts of the system could work properly. During the initial weeks, particular attention was given to exploring and understanding the TKey device and how it works. In addition, all required environments were set up to enable further development throughout the project.

The TKey device from Tillitis AB is the main component in the project, making it essential to load and configure the necessary environments on it to ensure it functions correctly. One of the key components loaded onto the device during each use was the signer application, which is responsible for signing the challenges received from the web server. Since the TKey operates using the Go programming language, it was necessary to install Go and its related libraries to build the proxy server. This server acts as a management and routing point between the TKey device and the web server.

For the web server, Python along with the Flask framework was utilized to implement both the server and communicate with the database. All required libraries were installed and configured accordingly. Initially, SQLite was selected as the database framework because of its simplicity. However, as the project progressed it was replaced with PostgreSQL due to its reliability in production and easy integration with Docker-Compose. For the front end, standard web technologies such as HTML, CSS, and JavaScript create a simple and user-friendly web interface.

3.4.2 System interaction

The challenge-response protocol provides the underlying foundation for the authentication system and consists of two separate parts:

Part 1: Cryptographic challenge-response

- When the user tries to register or login, the back-end web server generates a unique cryptographic challenge consisting of a random nonce and the current time.
- Then the challenge is sent to the client which forwards it to the proxy server.
- The proxy server loads the signer application onto the TKey and afterwards submits the challenge.
- The TKey securely signs the challenge using its internally derived private key and returns the signature to the proxy server.
- In turn, the proxy server sends the signed challenge back to the browser, which then forwards it to the web server who in turn verifies the signature using the user's stored public key in the database.

Part 2: TOTP Verification

- Following successful cryptographic verification, the web server verifies the provided TOTP code from the submission form.

3.4.3 Registration Flow

The user registration is important to securely introduce the new user to the system and link their identity to a unique public key in the database. This consists of several steps to help ensure that the utilized credentials are authentic and have not been tampered with.

The process starts when a new user opens the registration page in their web browser and is asked to enter a username. When the form is submitted, the web server and client go through part 1 of the challenge-response protocol described above. In addition to sending the signed challenge back to the web server, the proxy also submits the public key of the TKey which is required for verification of the signature.

If the web server is able to verify the signature with the provided public key, the public key is saved in the active session as the registration proceeds.

Following a valid signature and available username, the web server generates a TOTP secret and encodes it into a QR code by a Python library called qrcode. The QR code is presented to the user who is required to scan it with an authenticator app and provide the correct 6-digit TOTP code to verify that the configuration is successful. If the submitted code is validated successfully, then the user is presented with a mnemonic phrase generated with the Mnemonic python library, which serves as a recovery method should the user loose their TKey.

Only when the user finally confirms the registration process, the associated information with that user is saved to the database. This flow ensures that each registered user is securely associated with both a physical device (the TKey) and a TOTP secret, as well as that any error within the process is handled correctly.

3.4.4 Login Flow

The login process is similar to the registration, although with a few fundamental differences. The process begins by the user being prompted with entering their username and the current valid TOTP code. If the user is found, then the client and web server again go through the same challenge-response protocol as described previously. The difference now being that no public key is sent from the proxy as the web server should have the required key stored in the database. If both the signature and TOTP code is verified to be correct, then the web server marks the session as valid and authenticates the user. This two-step process renders the system far more resistant to compromise since an attacker would have to have both the TKey device and the code from the user's authenticator app. It helps protect against phishing, Man-in-the-Middle (MitM) attacks, and stolen credentials.

3.4.5 Proxy & TKey integration

In order to allow the browser to communicate securely with the TKey device, we used a proxy server that acts as an intermediary between the two components. This proxy server is an important part of the system since the backend server cannot communicate directly with the TKey via USB interface.

The main job of the proxy server is to relay messages between the TKey device and browser. For example, in both registration and login steps, when the backend server generates a cryptographic challenge, it relays this challenge through the browser to the proxy which in its turn relays it to the TKey via USB. When the TKey has finished signing the challenge using its own internal private key, the proxy collects the resulting signed value and sends it back to the browser and in turn to the backend in order to be verified.

To make this communication possible, Go and the required Go packages had to be installed. The proxy server is then compiled and listens on a specific port (e.g. 8000)

for HTTP requests from the frontend.

The proxy server automatically loads the signer app onto the TKey during every startup, which signs the cryptographic challenges from the backend web server. The hash of this app is also added to the key derivation process inside the TKey, which adds an extra level of security.

In the implementation of the authentication system, we handled two main points in the proxy:

- Handling registration-related signing.
- Handling login challenge signing.

In summary, the proxy server plays a critical role in securely linking the backend web server to the TKey device, without exposing the cryptographic operations to the rest of the system and maintaining the strict security properties provided by TKey.

3.4.6 Recovery implementation

In the project, a secure account recovery process was implemented, which allows users to recover access in case of loss or unavailability of their TKey device. This process is meant to provide high-security levels while ensuring continuity of accounts.

The recovery process starts at the end of the registration phase, where the backend generates a 12-word mnemonic recovery phrase from the mnemonic Python library, according to Bitcoin Improvement Proposal 39 (BIP39).

For safe storage, the mnemonic is converted to a seed during registration. This seed is then processed using the Password-Based Key Derivation Function 2 (PBKDF2) with HMAC-SHA256 as the pseudo random function. The PBKDF2 parameters are as follows: a randomly generated 16 byte salt, and the number of iterations is set to 100,000. The output of PBKDF2 is a derived key, and together with the salt stored within the PostgreSQL database. In addition, the user is shown the 12-word recovery phrase and instructed to store it privately.

When account recovery is required, the user accesses the recovery page and enters their username. If the username is valid, the user is redirected to a page to enter the mnemonic phrase. The system then converts this phrase into a seed and applies PBKDF2 using the stored salt associated with the username. If the newly derived key matches the stored key, the user is authenticated and can proceed.

The user is then required to insert a new TKey device. The backend server initiates a new cryptographic challenge-response procedure via the proxy server similar to the original registration process. Internally, the new TKey signs the challenge, and the backend server verifies the response using the public key derived from the new device.

By combining hardware and mnemonic-based recovery, this implementation maintains passwordless architecture integrity with the additional security of providing users with a safe way to recover their accounts without undermining cryptographic defenses.

3.5 Testing and Validation

To validate the functionality and reliability of the passwordless authentication system, a series of manual tests covering both expected user behavior and potential misuse scenarios was conducted. These tests were carried out using the implemented components and the Tillitis TKey device (CTH version) and a TOTP authenticator application such as Google Authenticator. The system was accessed through its deployed web interface using a range of different browsers as highlighted in Table A, to ensure consistent behavior across configurations.

The core functionality test focused on the full authentication workflow: registering a new user with a TKey and setting up two-factor authentication via a QR code, followed by successfully logging in using both the TKey and the generated TOTP code. This confirmed that all major components, including key registration and recovery, session handling, and cryptographic signing, worked together as intended.

The system's robustness was tested by simulating common edge cases. These included attempting to register a user with an already taken username, re-registering a TKey that was already in use and submitting empty or malformed input. These scenarios were designed to ensure that the application could gracefully handle invalid requests and provide appropriate feedback to the user.

In addition, it was evaluated how well the system communicated errors. For example, submitting a duplicate username correctly triggered a "user already exists" message, and attempting to register an already-used TKey returned a more generic message indicating a failure in TKey communication. While the latter response was technically correct, it highlighted a possible improvement in the clarity of error messaging.

All test cases followed a consistent format: we defined the scenario, performed the necessary steps, and recorded the resulting behavior. The outcomes of these tests, including how the system responded to each case, are documented in Chapter 4.

3.6 Limitations and Assumptions

Despite achieving the core objectives of the project, several limitations and assumptions were identified during the development and testing phases. These factors affect the scope, security, and usability of the system and should be considered when evaluating its potential for production deployment.

One of the key technical limitations lies in the communication between the user's

web browser and the locally hosted proxy server. This channel is currently not encrypted, as the proxy server communicates over HTTP rather than HTTPS. While this interaction occurs on the same machine, and thus does not traverse external networks, it remains vulnerable to local man-in-the-middle attacks if the host device is compromised. As this project is a proof of concept rather than a product designed for real-world use, this was an acceptable risk. Future versions that are designed for production could solve it by creating a solution that works independently of a proxy server on the clients machine with for example WebAssembly.

The account recovery process also presents a more serious theoretical risk. If an attacker gains knowledge of both the username of the user and the mnemonic recovery phrase, they could potentially register a new TKey and thus gain full access to the user's account while locking out the legitimate user. Although this scenario requires significant prior knowledge and is not a direct vulnerability of the system, it underscores the importance of secure handling of recovery information and careful user education.

Furthermore, the system assumes a secure and uncompromised client environment. This includes trust in the browser, the operating system, and the local proxy server. If the user's machine is infected with malware or accessed by an adversary with elevated privileges, all aspects of the security of the system could be undermined, including access to TOTP secrets or manipulation of proxy traffic. This limitation is inherent to most client-side security models, but is important to recognize explicitly.

The system relies on asymmetric cryptographic algorithms provided by the TKey device, such as those used in its signing application. These algorithms are currently not designed to be resistant to quantum computing attacks. While quantum threats are not yet practical, they may pose a risk to current public-key cryptography standards in the future. As such, post-quantum cryptographic techniques have not been implemented in this project but would need to be considered in a long-term or production-grade deployment.

Finally, it is assumed that the HTTPS deployment is correctly configured and that the certificates used for encryption and identity verification are valid and trusted by the client. If these configurations are incorrect or expired, the entire communication channel could be exposed to interception or impersonation.

Taken together, these limitations do not significantly impact the functionality of the system as a proof of concept. However, they highlight several areas where improvements would be required in order to transition to a production-ready, enterprise-grade authentication solution.

4

Results

We evaluate the results of the developed authentication system, the system's functionality, performance, and security based on empirical testing. Functional tests were conducted across multiple platforms and browsers to verify authentication flow and user experience. Performance metrics were recorded to assess responsiveness and stability, while a security evaluation was carried out to identify any remaining vulnerabilities. Finally, known issues and limitations are discussed, along with a summary of key findings.

4.1 Overview of Results

The main objective of the project was to develop a secure, passwordless authentication system using the TKey hardware device from Tillitis. The final implementation meets this goal- users can register and log in using a challenge-response protocol backed by asymmetric encryption, without relying on traditional passwords.

The system integrates three main components, web frontend, backend server, and a local proxy, to support hardware-backed authentication and two-factor verification. Authentication combines TKey-based cryptographic verification with a second factor such as Time-based One-Time Passwords (TOTP), e.g., via Google Authenticator.

Testing confirmed that the full authentication flow works reliably in standard use cases. The system enforces secure communication over HTTPS using TLS, ensuring confidentiality and integrity between client and server. User credentials, public keys, and TOTP secrets are stored securely in a PostgreSQL database.

4.2 System Performance

The system has been evaluated in terms of responsiveness, stability, and scalability. During internal testing, we observed that login and registration processes are completed in less than one second under normal conditions. This includes challenge generation, signature validation, and TOTP verification, demonstrating the system's ability to handle authentication quickly and efficiently.

No critical delays or timeouts were observed during these operations. The system

demonstrated sufficient responsiveness to support real-time user interaction and is considered efficient for small- to medium-scale enterprise environments.

4.3 Functional Testing

The application was manually tested in a controlled environment to verify the correctness of the authentication flow and related security mechanisms. Testing was conducted across a wide range of browsers on Windows and Linux, including Chrome, Brave, Firefox, and others. On macOS, testing was limited to Brave and Chrome. A detailed overview of browser and OS compatibility can be found in Table A.1.

4.3.1 Core Interaction Flow

The passwordless login process functioned as intended. Users were able to authenticate using the TKey device without requiring a traditional password. The system successfully detected the device's presence, prompted for physical confirmation (i.e., a touch on the device), and verified the user's identity.

The authentication process is generally smooth, with sessions being established correctly after successful device verification. Upon arriving on the website, the user is greeted with two options: to log in or register, as shown in Figure 4.1.

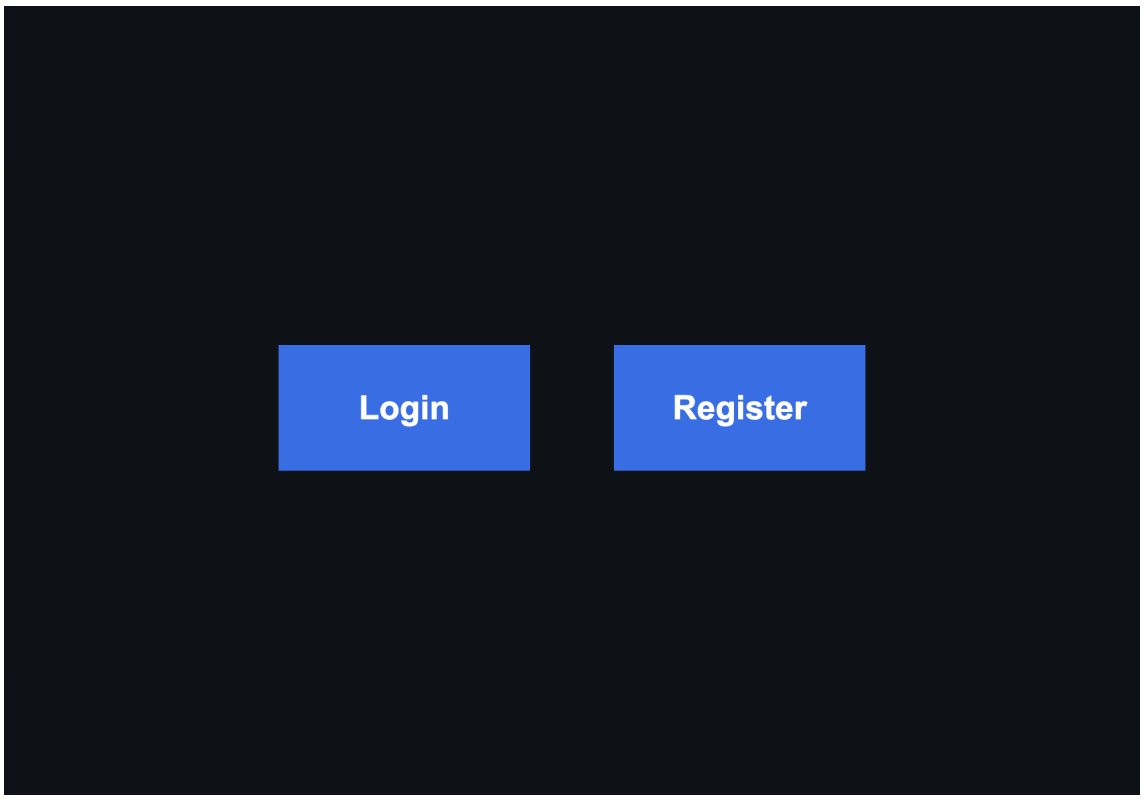


Figure 4.1: Home page with options to log in or register.

When a new user registers an account, they are first prompted to enter a username, as shown in Figure 4.2. Afterward, a QR code appears (Figure 4.3), which the user must scan using an authenticator app on their phone or a browser extension. Upon scanning, the user receives a TOTP in their authenticator, which are then used for authentication. The following step displays a 12-word recovery mnemonic (Figure 4.4), which the user must save securely in case they need to recover their account if their TKey device is lost. Once the user clicks on "Finalize Account," their account is created and stored in the database.

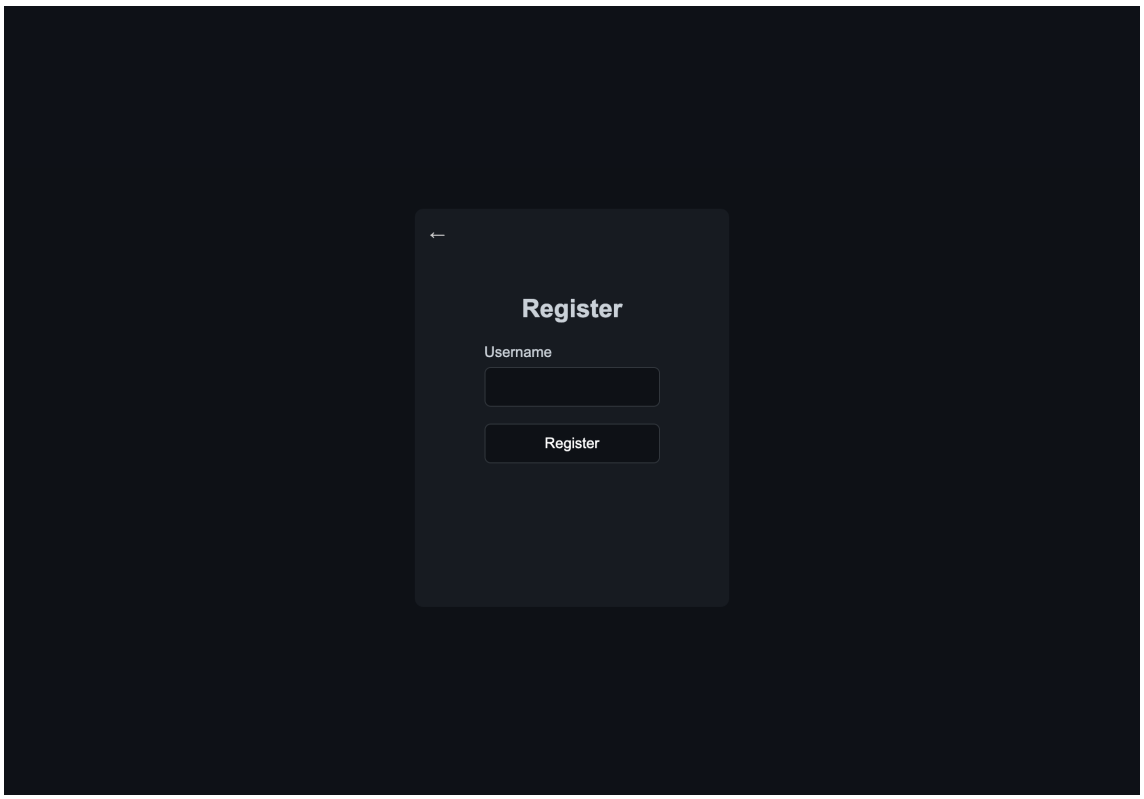


Figure 4.2: Step 1 of registration: Prompt for username entry.

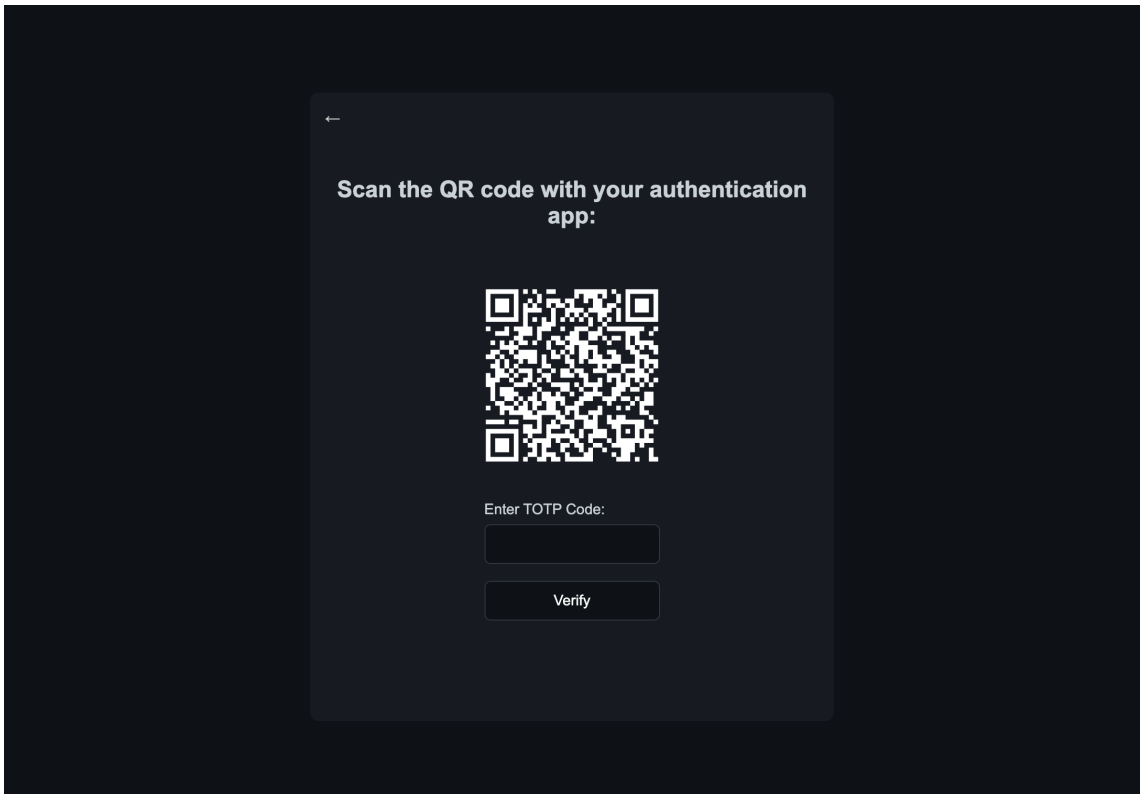


Figure 4.3: Step 2 of registration: QR code scan and TOTP entry.

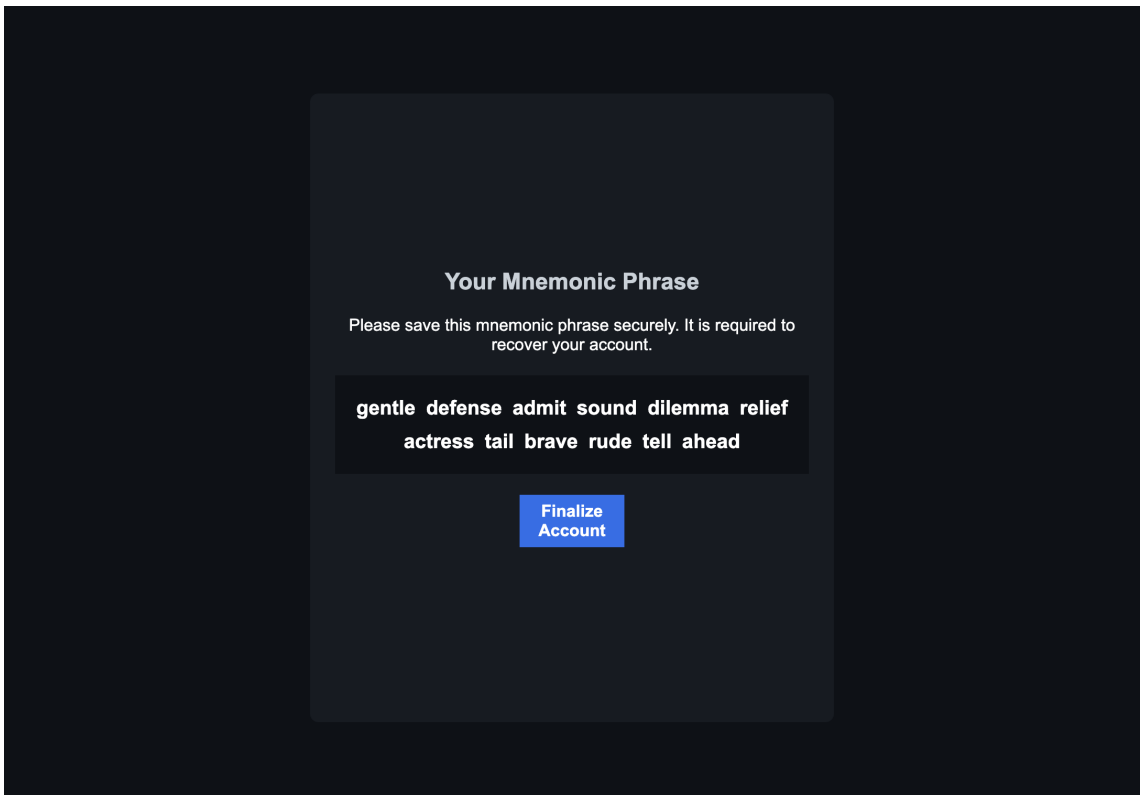


Figure 4.4: Step 3 of registration: Display 12-word recovery mnemonic.

After registration, the user is redirected to the login page, as shown in Figure 4.5, where they can enter their username and TOTP. Figure 4.6 demonstrates a case where the user enters an incorrect TOTP. However, if the credentials are correct, the user is successfully logged in, as shown in Figure 4.7, where they can choose to either log out or delete their account.

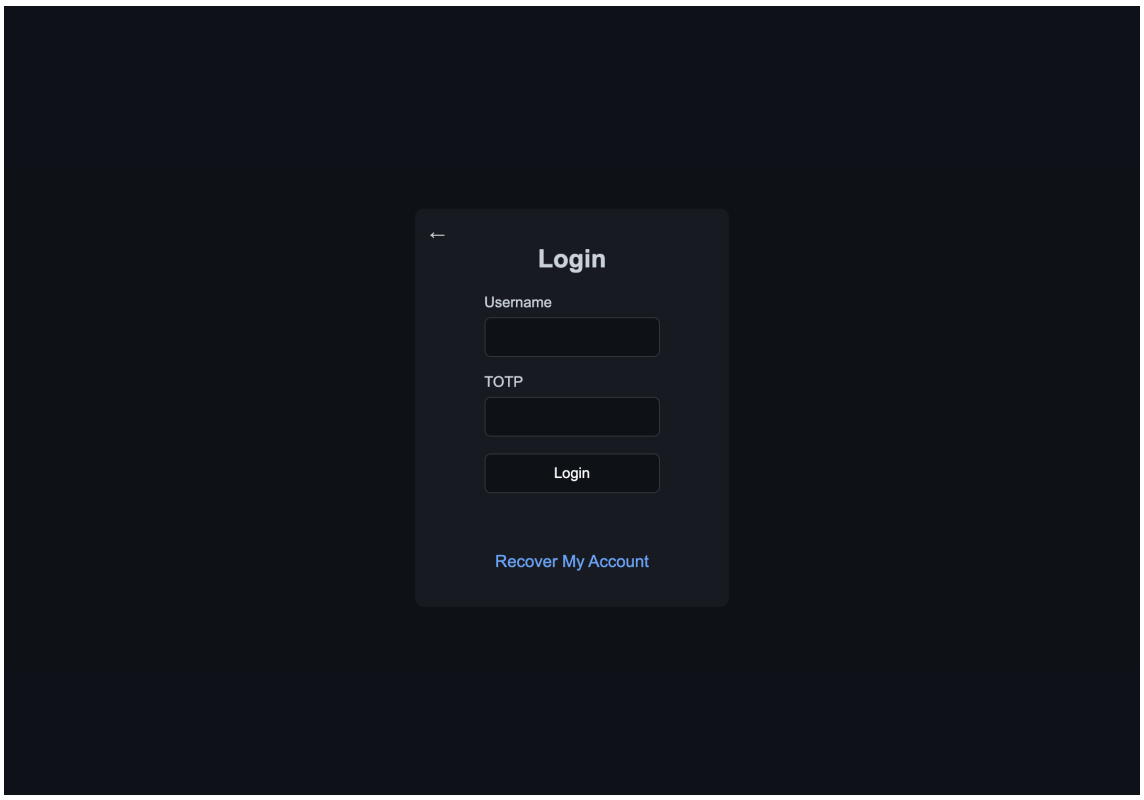


Figure 4.5: Login page: Prompt for username and TOTP entry.

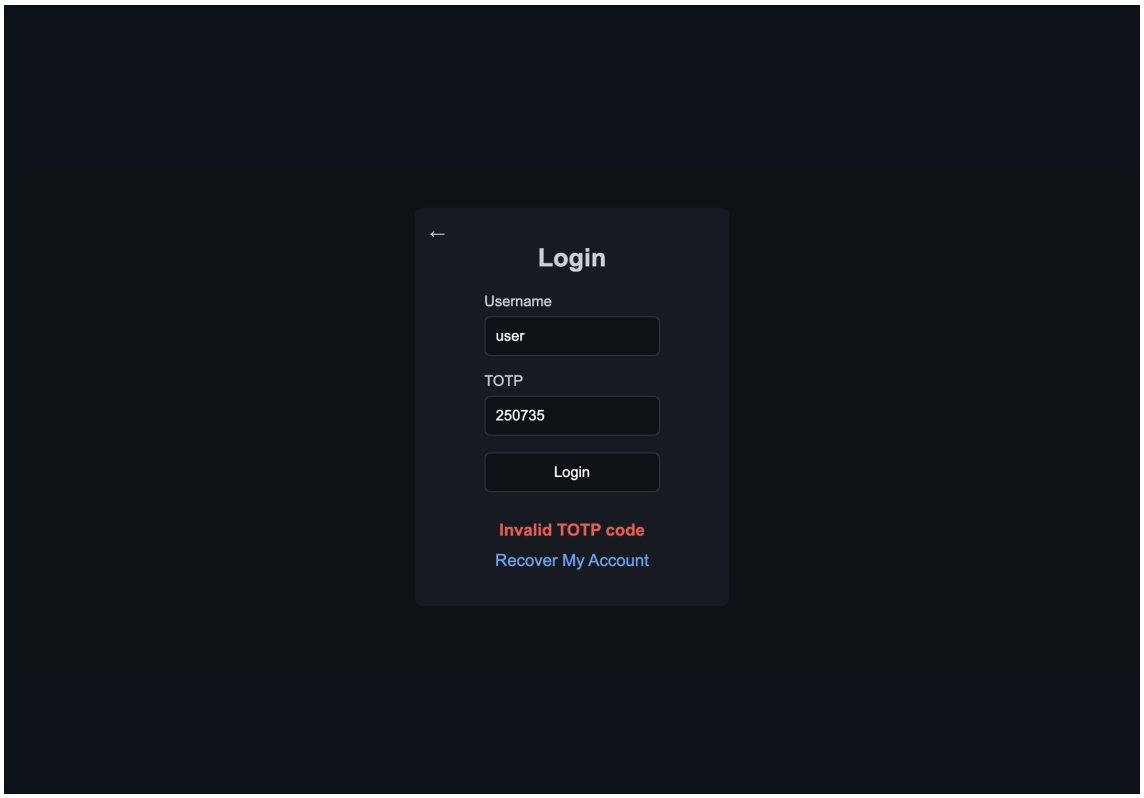


Figure 4.6: Login page: Incorrect TOTP entered.

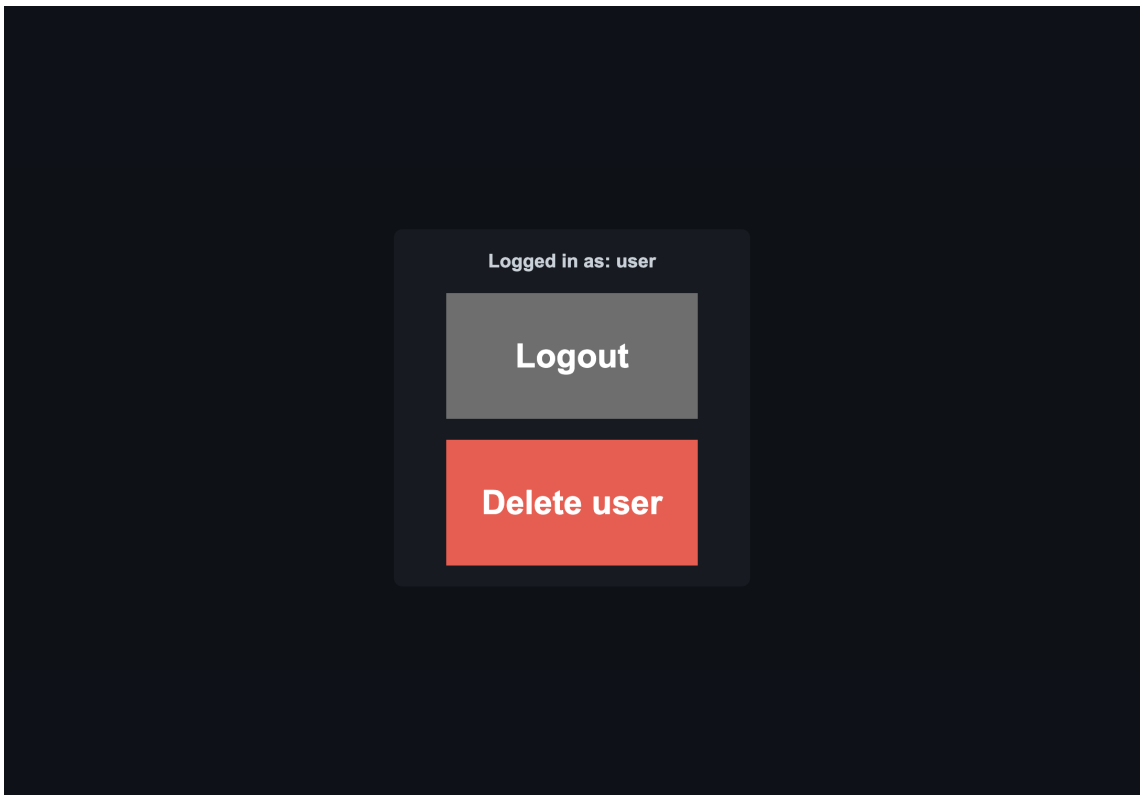


Figure 4.7: Home page after successful login.

For account recovery, the user can click "Recover My Account" on the login page (Figure 4.5) to initiate the process. First, they are prompted to enter their username (Figure 4.8). Next, they need to input their 12-word mnemonic phrase (Figure 4.9). In Figure 4.10, they are asked to insert their new device and press verify. If everything goes smoothly, Figure 4.11 shows the success message, indicating that the account has been successfully recovered.

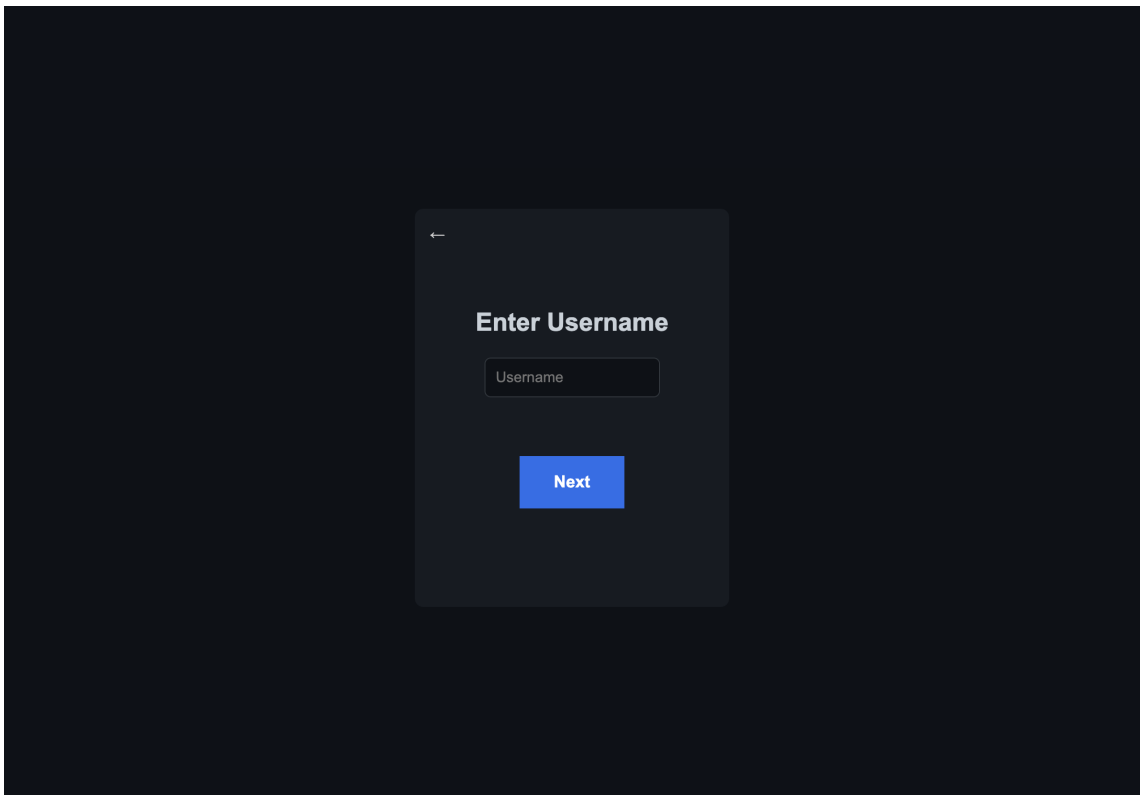


Figure 4.8: Step 1 of recovery: Prompt for username entry.

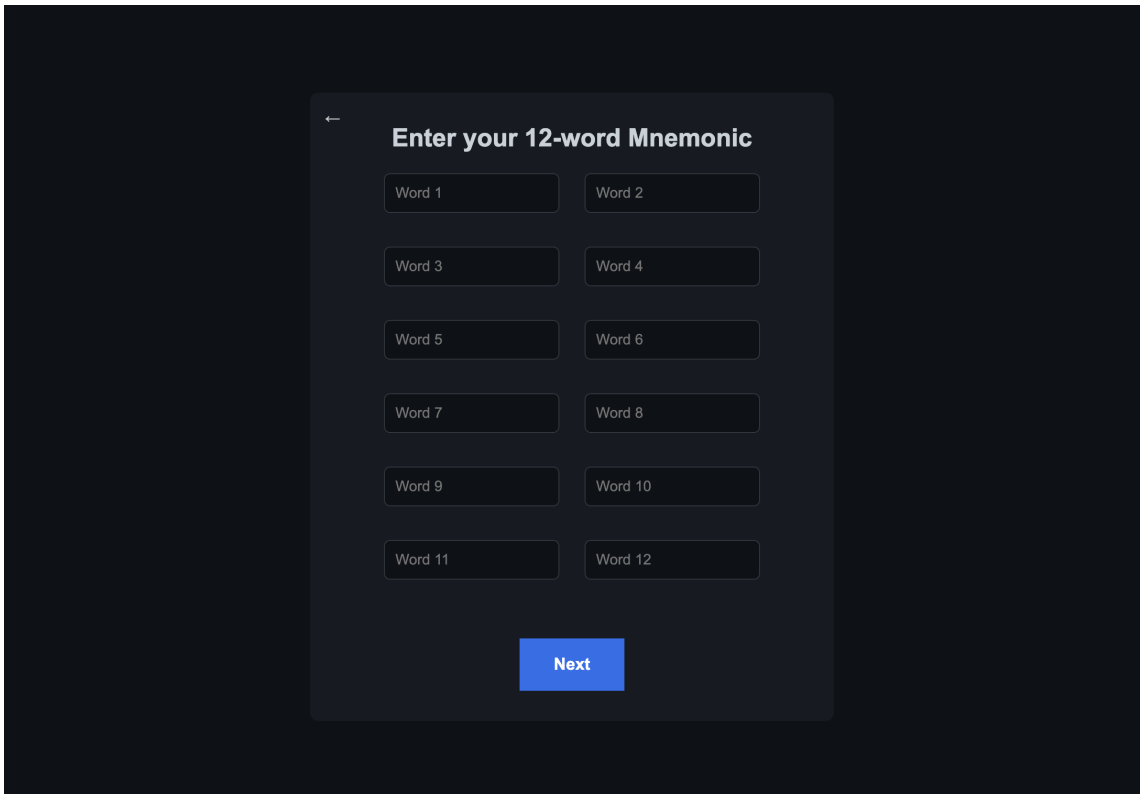


Figure 4.9: Step 2 of recovery: Enter 12-word mnemonic.

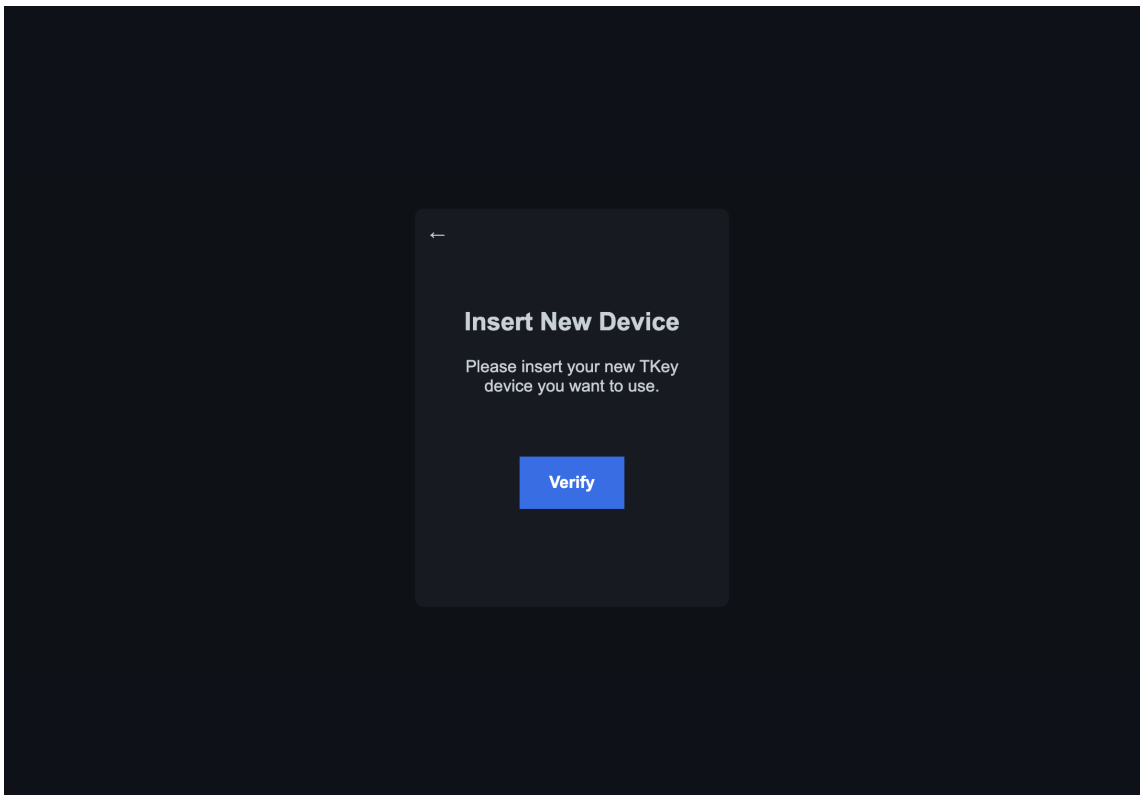


Figure 4.10: Step 3 of recovery: Insert and verify new TKey device.

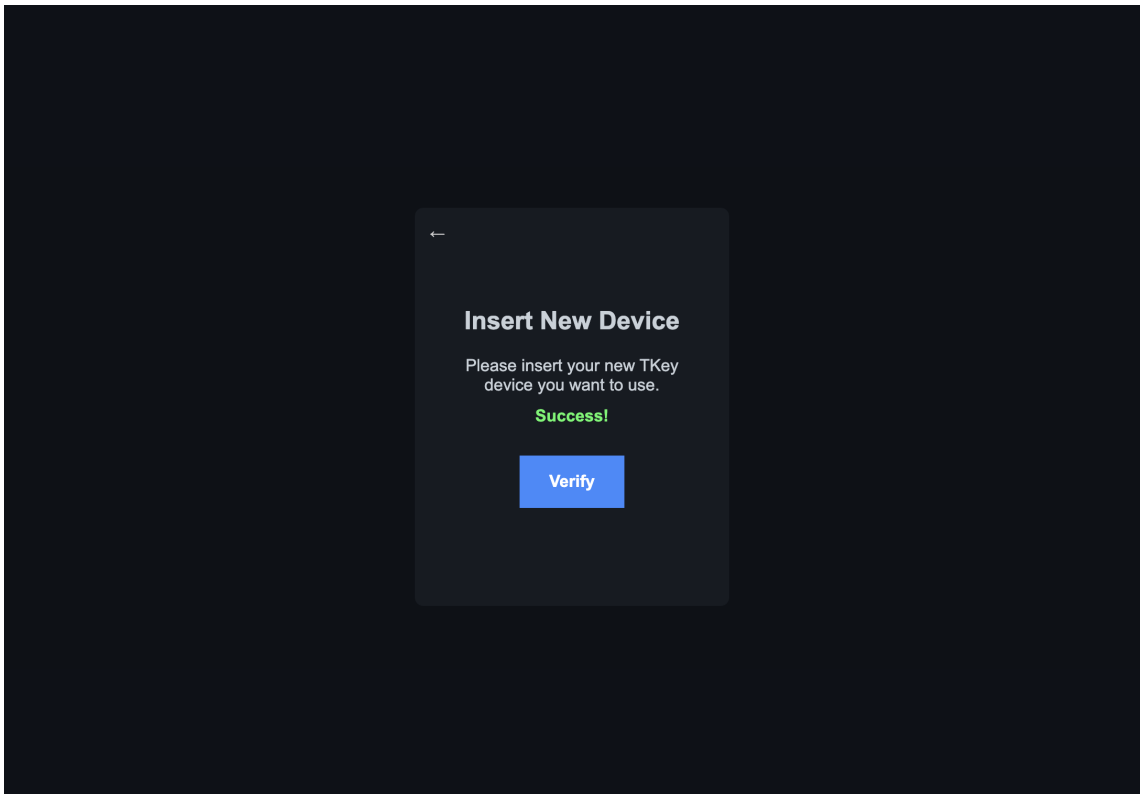


Figure 4.11: Step 3 of recovery: Successful recovery notification.

4.3.2 User Experience Issues

During testing, some minor user experience issues were identified:

- Device Responsiveness: The TKey device occasionally responds slowly to touch inputs. This is not part of the systems implementation but affects the overall user experience.

- TOTP Timeout Risk: The 2FA flow relies on a TOTP. Because the device prompts require user interaction and the TOTP refreshes frequently, there is a risk that the code may expire before the device verification is completed. This could lead to failed login attempts. A potential solution is to accept the current TOTP for verification first and defer device verification until afterward. This could improve reliability without compromising security.

4.3.3 Account Recovery Limitation

A significant limitation in the current implementation is the lack of a fallback mechanism in the event of lost 2FA. At present, the recovery flow is only applicable when the TKey device is lost. If the user loses their 2FA then recovery of that component is impossible. Due to time constraints this was never done in our case, but in an essential part of a production-ready system.

4.3.4 Browser Compatibility

The application was tested across multiple mainstream browsers on Windows and Linux. It performed reliably in Chrome, Firefox, and Edge to name a few. However, compatibility issues were encountered in several privacy-focused browsers, including Brave, Mullvad Browser, and Tor Browser.

In particular, Brave's privacy feature, Shields, blocked communication between the web application and the locally running proxy service that interfaces with the TKey device. This behavior likely results from Brave interpreting local HTTP requests as potential tracking activity—an intentional feature of its privacy model [24]. As a result, the authentication flow could not be completed while Shields were enabled.

Disabling Shields for the specific site allowed the application to function correctly, but this workaround poses usability and trust concerns. Users unfamiliar with the cause of failure may perceive the application as broken or insecure. Similar issues were observed in Mullvad Browser and Tor Browser, where security policies prevent or interfere with local communication.

These findings indicate that web authentication systems relying on local services must account for privacy-enhancing browser features and adopt communication strategies that are compatible with such environments. A detailed summary of browser and OS compatibility is provided in Table A.1.

4.4 Security Evaluation

This section evaluates the security aspects of the implemented system, with particular attention to common web application threats and vulnerabilities. The evaluation considers both server-side and client-side mitigations and highlights any remaining risks.

4.4.1 Brute Force Attacks

To mitigate brute force attacks on login and account recovery routes, a rate-limiting mechanism is implemented. Each IP address and account is restricted to a maximum of five login or recovery attempts per minute. This significantly reduces the effectiveness of automated password guessing attacks.

Although distributed brute-force attacks (across multiple IP addresses) could theoretically bypass IP-based rate limits, the account-level restriction ensures that such attacks are not feasible at scale.

4.4.2 Cross-Site Scripting (XSS)

The system is resistant to cross-site scripting (XSS) attacks due to Flask's Jinja2 templating engine, which automatically escapes user input by default [25]. This

built-in behavior mitigates common vectors such as stored and reflected XSS by preventing the injection of malicious scripts into rendered HTML. Our implementation avoids manual overrides of this mechanism to maintain its effectiveness. These protections align with broader best practices in preventing script injection, as outlined by Jim et al. [26], who advocate for embedded policy enforcement and default-deny approaches to script execution.

4.4.3 Cross-Site Request Forgery (CSRF)

CSRF protection is implemented through the use of CSRF tokens on all forms and state-changing API requests. These tokens ensure that requests originate from legitimate user sessions and are not forged by external sites. Proper implementation of CSRF tokens across all sensitive endpoints prevents attackers from exploiting authenticated sessions to perform unauthorized actions. This approach aligns with established best practices for CSRF mitigation [27], and prevents unauthorized commands from being executed on behalf of authenticated users.

4.4.4 Man-in-the-Middle (MitM) Attacks

All communication between the client (browser) and the backend server is secured using HTTPS, which includes TLS encryption and server authentication. This protects against man-in-the-middle (MitM) attacks that could compromise the confidentiality and integrity of transmitted data. As emphasized in IEEE's study on secure communication with hardware tokens [28], establishing a trusted channel between the client and the token is critical to prevent credential leakage and unauthorized access.

However, in the current implementation, communication between the browser and the locally running proxy server occurs over unencrypted HTTP. This channel lacks encryption and message signing, making it susceptible to interception or forgery if the local machine or network is compromised. An attacker with local access could potentially manipulate or observe this communication, posing a security risk.

4.4.5 Credential Compromise and Account Takeover

If an attacker obtains both a valid username and the corresponding mnemonic recovery phrase, they can register a new TKey device for that account. This effectively grants them access and potentially locks out the legitimate user. While this is not a flaw in the design of the system, it underscores the importance of secure credentials handling.

4.4.6 SQL Injection

The application uses the psycopg2 library to interact with a PostgreSQL database. All database queries are implemented using parameterized statements, which prevent SQL injection attacks by ensuring that user input is never directly interpolated into SQL strings.

This approach is consistent with best practices and considered effective in eliminating this class of vulnerabilities.

4.4.7 Summary of Security Posture

As shown in Table 4.1, the system has strong protections against common web vulnerabilities, particularly brute force, CSRF, and SQL injection. However, three moderate risks remain. The first relates to communication between the browser and the local proxy, which lacks encryption and could be targeted by attackers with local access. The second is user-dependent: if a mnemonic phrase is compromised, account takeover becomes possible. Although not a flaw in the design itself, it highlights the need for user education. The third concern loss of access to the second authentication factor. Currently, there is no recovery mechanism if a user loses access to their TOTP-based 2FA, making the account permanently inaccessible.

Threat Category	**Mitigation Present**
Brute Force	✓ Rate limiting
Cross-Site Scripting (XSS)	✓ Auto-escaping
Cross-Site Request Forgery (CSRF)	✓ CSRF tokens
Man-in-the-Middle (Client-Server)	✓ HTTPS/TLS
Man-in-the-Middle (Local API)	× None
TKey Recovery	✓ Backend validation
2-Factor Authentication Recovery (2FA)	× None
Mnemonic Theft Scenario	× User-dependent
SQL Injection	✓ Parameterized queries

Table 4.1: Summary of Security Posture

5

Discussion

This project set out to explore the viability of a password-less authentication system using the TKey hardware device. The resulting prototype demonstrates a functional and secure authentication framework that enables users to register, log in, and recover accounts using a physical token and a mnemonic phrase, without relying on passwords. Through the development of a web application, a locally hosted Go-based proxy, and integration with TOTP-based 2FA, the system fulfills its core objectives: to implement a secure architecture, deliver a working system, and evaluate key security threats and usability challenges.

The system offers a compelling alternative to traditional password-based schemes. Its architecture significantly mitigates risks such as brute-force attacks, phishing, and password reuse, while also improving the user experience by removing the burden of password memorization. Notably, account recovery is supported by relying instead on a combination of the TKey device and a locally stored or memorized mnemonic phrase.

5.1 Use Case Suitability

While the system functions well as a proof-of-concept, it is most appropriate for controlled environments, such as within enterprises or institutions. In these settings, centralized IT support can more effectively facilitate the issuance, maintenance, and recovery of physical authentication tokens. For instance, rather than placing the full burden of secure mnemonic phrase storage on individual users, organizations could maintain shared recovery devices or secured credential vaults, thereby reducing the risk of access loss due to personal mismanagement or hardware failure.

Although such centralized mechanisms enhance account recoverability, they also entail a shift in authority from individual users to organizational administrators. To preserve trust and security, these systems should be complemented by safeguards such as role-based access controls, oversight procedures, and detailed logging. The broader ethical and privacy implications of this model are examined in Section 5.4.

5.2 Design Limitations

Despite its strengths, the system exhibits several limitations. Chief among these is the reliance on a locally running proxy service to mediate between the browser and the TKey device. This architectural choice, while functional, undermines ease of deployment and user accessibility, as it requires additional configuration steps and may conflict with privacy-focused browser settings. Ideally, the system would function out of the box, leveraging built-in browser APIs or extensions.

Another limitation is the reliance on TOTP-based 2FA. Although TOTP remains a well-established standard, it introduces temporal constraints and risks associated with time drift or user delays. Moreover, if users lose access to their 2FA mechanism, the current implementation provides no method of recovery. This gap presents a serious availability risk and suggests that a fallback or recovery pathway for lost 2FA, such as multi-factor verification or administrative reset, is essential for a production-ready system.

Additionally, the mnemonic recovery phrase, while secure and user-controlled, functions as a form of shared secret. If compromised, it can lead to account takeover. Therefore, users must be explicitly warned about its sensitivity and encouraged to store it using secure methods, such as encrypted vaults or offline physical media. These design tensions—between decentralization, usability, and resilience—highlight the complex landscape of secure authentication.

5.3 Future Work

While this project successfully demonstrated that a password-less authentication system utilizing public-key cryptography is a feasible solution to combat password weaknesses, there is till areas that need continued study to ensure the systems stay secure. One significant area is the rise of Quantum Computing, which makes it possible to solve problems like the discrete logarithm problem, an essential foundation of modern public-key cryptography algorithms such as RSA and elliptic curves like ed25519 [29]. Government agencies such as the NCSC have presented timelines for organizations to migrate over to solutions utilizing post-quantum cryptography by 2035 [30], highlighting the importance of this research area. Future work could investigate the possibility of using quantum-resistant algorithms in authentication systems, not only ensuring security in the long-term, but also adding benefits such as eavesdropping detection [31].

Future development could focus on eliminating the reliance on a proxy by integrating directly with browser-native security APIs, such as WebAuth. This would allow for secure device communication without external services, significantly improving user experience and compatibility. Additionally, implementing account recovery mechanisms for lost 2FA, such as biometric fallback, secure reset tokens, or multi-factor re-verification, would enhance the robustness of the recovery process.

In enterprise scenarios, supporting centralized recovery methods, e.g., administrative overrides or shared recovery tokens, would reduce lockout risks while necessitating privacy-preserving oversight structures. Expanding the system to support multi-device login or Single Sign-On (SSO) functionality could make it viable for broader organizational use. Furthermore, enhanced auditing and logging would be required for traceability, especially in regulated or compliance-focused industries.

These enhancements would collectively move the system closer to real-world deployment, balancing usability, security, and operational requirements.

5.4 Ethical and Privacy Considerations

The design of the system prioritizes user security by eliminating passwords and relying instead on hardware-based authentication using the TKey device, combined with a mnemonic recovery phrase. While this significantly reduces the risk of phishing and credential theft, it introduces a different type of responsibility: users must securely store and manage both the hardware device and the recovery phrase. If either is lost without backup, account access may become permanently unrecoverable. This raises ethical concerns around user burden, particularly for individuals who may lack the technical skill or resources to manage hardware tokens securely.

To mitigate these risks in enterprise environments, organizations may centralize recovery by storing credentials internally or permitting IT administrators to reset access. While this enhances usability and prevents permanent lockouts, it also redistributes power away from the individual. Administrative personnel could, whether intentionally or inadvertently, access user accounts, raising questions about trust, surveillance, and impersonation.

From an ethical standpoint, any such override mechanism must be designed with robust procedural and technical safeguards. These may include tamper-evident audit logs, multi-party recovery approvals, and real-time user notifications in the event of administrative intervention. Striking the right balance between user autonomy and organizational control remains a fundamental challenge. Future research should explore governance models that ensure operational continuity while upholding individual rights and transparency.

5.5 Conclusion

This project has demonstrated that password-less authentication using hardware tokens is not only feasible but also promising as a next-generation identity solution. The prototype highlights how a thoughtfully designed combination of hardware, cryptographic recovery, and minimal user burden can outperform conventional login flows in both security and user experience. Nonetheless, transitioning from prototype to a deployable system will require addressing several practical, security, and ethical challenges. As password-less technologies mature, systems like this could play an integral role in shaping safer and more resilient digital identities.

Bibliography

[1] OWASP Foundation, *OWASP Top 10 - 2021*, https://owasp.org/Top10/, Accessed: 2025-04-20, 2021.

[2] S. Riley, "Password security: What users know and what they actually do," *Usability News*, vol. 8, no. 1, pp. 2833–2836, 2006.

[3] D. Florencio and C. Herley, "A large-scale study of web password habits," in *16th International World Wide Web Conference*, ser. WWW '07, Banff, Alberta, Canada, 2007, pp. 657–666. DOI: 10.1145/1242572.1242661. [Online]. Available: https://doi.org/10.1145/1242572.1242661.

[4] L. Bošnjak, J. Sreš, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1161–1166. DOI: 10.23919/MIPRO.2018.8400211.

[5] Picus Labs, *Picus Security Finds 3X Increase in Malware Targeting Password Stores*, https://www.picussecurity.com/resource/press-release/the-rise-of-perfect-heist-attacks, Accessed: 2025-04-26, 2025.

[6] "Tillitis - the creator of tkey." Accessed: 2025-04-01, Tillitis AB. (), [Online]. Available: https://tillitis.se/.

[7] "Tkey." Accessed: 2025-04-01, Tillitis AB. (), [Online]. Available: https://tillitis.se/products/tkey/.

[8] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 5th ed. Harlow, United Kingdom: Pearson, 2024.

[9] B. Barak, *An Intense Introduction to Cryptography*. Boaz Barak, 2021. [Online]. Available: https://intensecrypto.org/public/.

[10] D. Boneh and V. Shoup, *A Graduate Course in Applied Crytography*. Harvard University, 2023, Manuscript in progress, version 0.6. [Online]. Available: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_6.pdf.

[11] R. Pass and A. Shelat, *A Course in Cryptography*. Pass/shelat, 2010. [Online]. Available: https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf.

[12] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Tang, "High-speed high-security signatures," *J. Cryptogr. Eng.*, 2012.

[14] M. Nieles, K. Dempsey, and V. Y. Pillitteri, "An introduction to information security," National Institute of Standards and Technology, Gaithersburg, MD,

Tech. Rep. NIST Special Publication (SP) 800-12, Rev. 1, 2017. DOI: 10.6028/NIST.SP.800-12r1.

[15] C. Arthur. "Kim dotcom loses two-factor authentication patent fight." Accessed: 2025-05-02, The Guardian. (2013), [Online]. Available: https://www.theguardian.com/technology/2013/may/23/kim-dotcom-authentication-patents.

[16] Microsoft. "The importance of two-factor authentication." Accessed: 2025-05-02, Microsoft. (2022), [Online]. Available: https://www.microsoft.com/en-us/microsoft-365-life-hacks/privacy-and-safety/importance-of-two-factor-authentication.

[17] Microsoft Support. "What is multifactor authentication?" Accessed: 2025-05-02, Microsoft. (2023), [Online]. Available: https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661.

[18] Zimperium. "Mnemonic phrase." Accessed: 2025-04-27. (2025), [Online]. Available: https://zimperium.com/glossary/mnemonic-phrase.

[19] M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe, *BIP 39: Mnemonic code for generating deterministic keys*, https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki, Final, Bitcoin Improvement Proposal, 2013. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.

[20] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, RFC 2898, Sep. 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2898.

[21] PullRequest. "Understanding the benefits of key derivation functions: A deep dive into pbkdf2." Accessed: 2025-04-28. (Jan. 2024), [Online]. Available: https://www.pullrequest.com/blog/understanding-the-benefits-of-key-derivation-functions-a-deep-dive-into-pbkdf2/.

[22] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, Feb. 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2104.

[23] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2th. Pearson, 1999, ISBN: 0132775069.

[24] Brave Software, *What is "Shields"?* https://support.brave.com/hc/en-us/articles/360022973471-What-is-Shields, Accessed: 2025-05-15, n.d.

[25] Pallets Projects, *Security considerations – flask documentation (3.1.x)*, https://flask.palletsprojects.com/en/stable/web-security/, Accessed: 2025-04-17, n.d.

[26] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, ACM, 2007. DOI: 10.1145/1242572.1242654.

[27] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM conference on Computer and communications security (CCS '08)*, ACM, 2008. DOI: 10.1145/1455770.1455782.

[28] H. K. Lu and A. Ali, "Communication security between a computer and a hardware token," in *Proceedings of the IEEE Conference on Third International Conference on Systems (icons 2008)*, IEEE, 2008. DOI: 10.1109/ICONS.2008.36. [Online]. Available: https://ieeexplore.ieee.org/document/4497126.

[29] A. Sahoo, I. K. A. K, and S. M. Rajagopal, "Comparative study of cryptographic algorithms in post quantum computing landscape," in *2024 5th International Conference on Data Intelligence and Cognitive Informatics (ICDICI)*, 2024, pp. 36–40. DOI: 10.1109/ICDICI62993.2024.10810828.

[30] National Cyber Security Centre, *Timelines for migration to post-quantum cryptography*, Accessed: 2025-05-15, 2025. [Online]. Available: https://www.ncsc.gov.uk/guidance/pqc-migration-timelines.

[31] M. J. Hossain Faruk, S. Tahora, M. Tasnim, H. Shahriar, and N. Sakib, "A review of quantum cybersecurity: Threats, risks and opportunities," in *2022 1st International Conference on AI in Cybersecurity (ICAIC)*, 2022, pp. 1–8. DOI: 10.1109/ICAIC53980.2022.9896970.

A

Appendix 1

Table A.1: Browser and OS Compatibility Test Results

Browser	**Windows 10/11**	**Ubuntu**
Chrome	Functioned as expected	Functioned as expected
Firefox	Functioned as expected	Functioned as expected
Edge	Functioned as expected	Functioned as expected
Opera	Functioned as expected	Functioned as expected
Opera GX	Functioned as expected	Not tested
Vivaldi	Functioned as expected	Functioned as expected
Chromium	Functioned as expected	Functioned as expected
Brave	Required disabling Brave Shields[1]	Required disabling Brave Shields[1]
Mullvad Browser	Required disabling script-blockers[2]	Required disabling script-blockers[2]
LibreWolf	Functioned as expected	Functioned as expected
Tor Browser	Not functional[3]	Not functional[3]

[1]Brave Shields prevented full communication between the browser and proxy-server. Disabling it entirely for the site or disabling its tracker protection restored full functionality.

[2]Included extensions prevented JavaScript execution. Full functionality required manual disabling of privacy tools.

[3]Due to the way Tor routes connections through Tor nodes, communication with the proxy server hosted on the local machine was not possible.