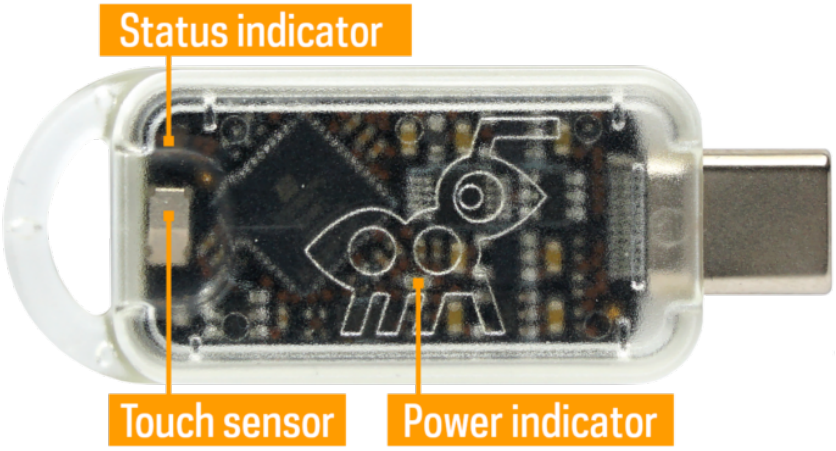
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Passwordless Authentication Using the TKey Device

Bachelor's thesis in Computer science and engineering

Jonas Almström
Carl Forssell
Kalle Johansson
Luqas Lundahl
Isac Snecker
Simon Westlin Green

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Bachelor's thesis 2026

Passwordless Authentication Using the TKey Device

Bachelor's thesis in Computer science and engineering

Jonas Almström
Carl Forssell
Kalle Johansson
Luqas Lundahl
Isac Snecker
Simon Westlin Green

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Passwordless Authentication Using the TKey Device Bachelor's thesis in Computer science and engineering
Jonas Almström Carl Forssell Kalle Johansson Luqas Lundahl Isac Snecker Simon Westlin Green

Supervisor (Handledare): Rhouma Rhouma, Department of Computer Science and Engineering
Examiner: Arne Linde and Patrik Jansson, Department of Computer Science and Engineering Graded by teacher (Rättande lärare): Irum Inayat, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Course DATX11/DIT561
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Abstract

In our increasingly connected digital world, secure access to online services is very important. With the rise of data breaches, phishing attacks, and password reuse, the demand grows for authentication methods that are both highly secure with ease of use. This thesis explores a secure authentication system that removes traditional passwords in favor of using the TKey device, a hardware security token that employs asymmetric cryptography and dynamic key generation. This study covers the necessary theoretical background, the design and implementation of the authentication system, and an analysis of potential security risks. An agile development approach was used to build and test a working prototype, with evaluations made through unit, and usability tests. Additionally, the work discusses the societal and ethical considerations of deploying such technology, including accessibility and data handling, and suggests practical directions for future improvements.

Sammandrag

I dagens allt mer uppkopplade digitala värld, där beroendet av onlinetjänster är överallt, har det blivit av största vikt att skapa säker åtkomst till dem. Med en ökning av dataintrång, phishing-attacker och många personer som återanvänder lösenord så ökar efterfrågan på autentiseringsmetoder som både är mycket säkra och enkla att använda. Denna avhandling utforskar ett säkert autentiseringssystem som tar bort vanliga lösenord, för att istället använda en TKey-enhet. Det är en hårdvarusäkerhetstoken som använder asymmetrisk kryptografi och dynamisk nyckelgenerering. Denna studie täcker den nödvändiga teoretiska bakgrunden, designen och implementeringen av autentiseringssystemet, samt en analys av potentiella säkerhetsrisker. En agil systemutvecklingsmetod användes för att bygga och testa en fungerande prototyp, med utvärderingar gjorda genom enhets- och användbarhetstester. Arbetet diskuterar även de samhälleliga och etiska övervägandena för att implementera sådan teknik, inklusive tillgänglighet och datahantering, och föreslår anvisningar för framtida förbättringar.

Keywords: Passwordless, Authentication System, TKey, Hardware, Cryptography, Security, Web, Challenge-Response, Computer Science

Acknowledgements

We would like to thank Tillitis for providing the Tkey hardware, and support for integrating it.
We would also like to thank Rhouma Rhouma for being our supervisor, and supporting us during the entire process.

Jonas Almström, Carl Forssell, Kalle Johansson, Luqas Lundahl, Isac Snecker, Simon Westlin Green, Gothenburg, January 2026

Contents

List of Figures

List of Tables

Glossary

Abandonware Software that is no longer updated or maintained by its developers..

Agile Development A flexible, iterative approach to software development that emphasizes collaboration, and rapid, incremental delivery of working software.

Application The software that the user want to authenticate to.

Cryptographic Challenge A cryptographic challenge is a test or puzzle designed to verify that a user or system possesses a certain cryptographic key or knowledge, often used to prove identity or secure communication without revealing the key itself.

Cryptographic Signing Cryptographic signing is a process where a person uses a private key to create a unique signature for a message or piece of data, which proves its authenticity and ensures it hasn't been tampered with.

Daemon A software program running as a background process with no direct user interaction.

Malware Software designed to compromise users system, often with harmful intentions.

OTP One-time password.

Passwordless Authentication An authentication method that does not rely on traditional passwords, often using hardware tokens and cryptographic techniques to verify a user's identity.

Signer Application A software component that performs cryptographic signing operations.

Sprint A fixed, time-boxed period in agile development during which a specific set of tasks is completed and reviewed.

Tillitis The company developing the TKey.

TKey A physical USB-C device developed by Tillitis that uses cryptographic signing to authenticate users.

USB security token A portable hardware device that securely stores cryptographic keys and performs cryptographic operations, typically connecting via USB to provide secure authentication.

1

Introduction

Secure authentication is essential in today's connected world. Since our world is increasingly becoming connected and digital, secure access to online services remains especially important. Traditional password-based authentication systems pose a vulnerability in our everyday digital life [3, Ch. 4.2]. Moreover, with the rise of data breaches, phishing attacks, and password reuse, there is an increasing demand for authentication methods that combine robust security with user-friendly simplicity. The IBM X-Force Threat Intelligence Index 2024 [4] reports that attacks exploiting valid credentials increase by 71% from 2023 to 2024. These breaches account for 30% of all incidents, an equal share to that of phishing attacks, highlighting the urgent need for authentication methods that offer both strong security and ease of use. Consequently, this project focuses on developing a robust passwordless authentication using the TKey device, a flexible USB security token that utilizes public-key cryptography and the cryptographic signing protocol [5]. This approach directly addresses the vulnerabilities inherent in traditional password-based systems.

Passwords suffer from inherent usability and security limitations. For example, a Forbes-cited [6] survey of 1,000 US adults finds that 84% practiced unsafe password habits such as using personal information, 24% included their favorite numbers, 40% wrote down their passwords, and 59% shared at least one password with others [7]. These statistics underscore the pressing need for alternative authentication methods. Although tools like password managers exist to help users securely store and generate complex passwords, they still rely on a password that remains vulnerable to exposure.

This work bridges theory and practice in modern authentication. From an academic perspective, the project integrates theoretical insights from asymmetric cryptography with practical, hardware-based implementations. Technically, it demonstrates the feasibility and advantages of secure authentication in web environments, potentially influencing future standards and practices in digital security. In addition, it introduces novel improvements by extending the functionality of the TKey device with a passwordless authentication system. From a user standpoint, the shift away from traditional passwords simplifies the login process and significantly reduces the vulnerabilities associated with conventional methods.

By creating this novel passwordless authentication functionality for the TKey device, this work aims to demonstrate the possibilities of such a solution. Thereby contributing to a safer, more convenient user authentication process. Overall, the project extends the practical utility of the TKey device, making it more adaptable to diverse security challenges in digital environments.

Implementing a seamless TKey-based system presents multiple challenges. This project addresses the issues identified in the background by implementing passwordless authentication with the TKey device. The problem proves to be complex, involving hardware integration, cryptographic theory, application design, and security analysis. The core challenge lies in enabling secure user registration and login via the TKey device.

A thorough review of TKey device documentation is needed to provide insights into the architecture, capabilities, and limitations, which in turn guide the integration strategy to ensure reliable, secure communication with the application. Study of asymmetric cryptography algorithms employed by the TKey device prevents the introduction of vulnerabilities into the authentication flow. A solid foundation in passwordless authentication methods underpins a design that balances security with user experience.

The application architecture manages user registration, public-key storage, and cryptographic verification directly within the software layer. Supporting this, the daemon bridges hardware and software by processing cryptographic operations locally on the user's machine and returning signed challenges. Finally, a security analysis identifies potential vulnerabilities and prescribes mitigation strategies to reinforce system integrity.

The purpose of this project is to develop a secure web application that uses the TKey device for authentication. Below is a summary of the key problems for achieving this:

- Integrating the TKey device into a passwordless authentication system.
- Analyzing asymmetric cryptography algorithms employed by the TKey device.
- Exploring designs for passwordless authentication systems.
- Designing and implementing an application to manage user registration, public-key storage, and authentication verification.
- Developing a daemon to process cryptographic operations locally and bridge communication between the TKey and the application.
- Conducting a security analysis to identify vulnerabilities and defining mitigation strategies to safeguard the system.

This project focuses on software integration rather than hardware development. Therefore, it excludes the creation of applications running directly on the TKey device. Instead, it employs the existing signer application, interfacing via the tkey-client Go package. This approach reduces development time and enhances software longevity since the signer application is actively maintained by Tillitis. Consequently, the research narrows to software integration and authentication processes rather than hardware-specific challenges.

This project limits frontend styling to a basic design because graphical design does not serve as a primary project focus. Although the core security features received thorough investigation, the basic user interface may impair overall usability and limit adoption in scenarios that demand a polished visual experience.

In this chapter, the project's motivation was established by highlighting the vulnerabilities of password-based authentication and defining its objectives and key challenges. Chapter 2 examines the theoretical foundations on which the proposed solution is built.

2

Theoretical Foundations

Following the motivation and objectives set out in Chapter 1, this chapter develops the theoretical foundations that make a TKey-based, passwordless system feasible. Firstly, public-key cryptography and elliptic-curve techniques are examined, after which the principles and practicalities of passwordless authentication and hardware token designs are explored. Finally, architectural patterns and core security considerations are surveyed to guide the system's implementation.

Password-based authentication inherently carries substantial risk because of human-related vulnerabilities, such as weak passwords and reuse [8]. Consequently, passwordless authentication aims to eliminate these risks through methods such as one-time passwords (OTP) or cryptographic hardware security keys.

Various methods have been proposed for implementing passwordless authentication. In particular, this project focuses on an asymmetric cryptography–based approach. Each user holds a unique key pair: the private key resides securely on a hardware device (e.g., the TKey device), while the public key is registered with the authentication system. During login, the system issues a cryptographic challenge that only the device's private key can sign. Since the private key never leaves the hardware, threats like password leaks and phishing attacks become irrelevant.

However, passwordless authentication also faces significant challenges, foremost among them being the need for ubiquitous infrastructure support. In addition, cross-platform compatibility and the risk of lost or damaged authentication devices present further difficulties.

Embedding cryptographic parameters in secure hardware is a robust strategy to ensure both security and practicality in passwordless authentication. This is where hardware security tokens, such as the TKey device, play a critical role. By offering a tamper-resistant environment for cryptographic operations, these tokens ensure the authentication process remains both secure and practical. The following section explores the role of such tokens in greater detail, with particular focus on the TKey device.

2.1 Asymmetric Cryptography

Asymmetric cryptography ensures confidentiality and secure communication by employing a paired key system, with a *public key* for encryption and a *private key* for decryption. Moreover, each user generates a unique key pair: the public key is shared, and the private key is kept secret [3, p. 40]. This arrangement guarantees that interception alone cannot lead to decryption without access to the corresponding private key.

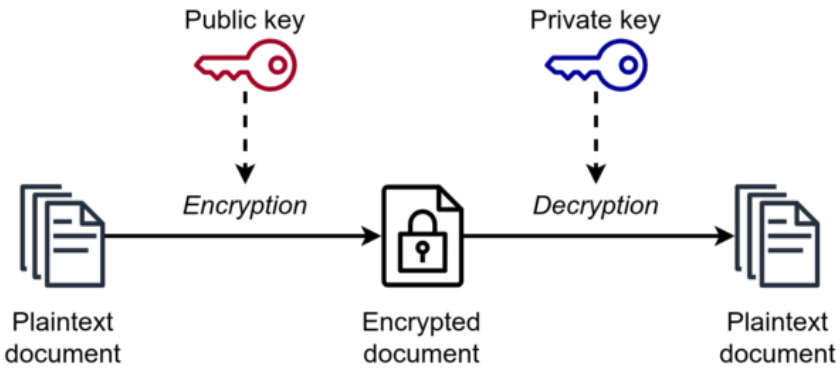


Figure 2.1: *Encrypting and decrypting a text using asymmetric encryption. [1] CC-BY 4.0*

In addition to providing confidentiality through encryption, asymmetric cryptography also facilitates authentication via digital signatures. Specifically, the sender hashes the message and then encrypts the hash with their private key. It is then decrypted by the receiver using the public key and compared with a newly generated hash of the received full message [3, p. 29]. Consequently, by use of digital signatures, systems achieve non-repudiation.

Among asymmetric cryptography algorithms, *elliptic curve cryptography (ECC)* is emphasized for this project. ECC derives its strength from the mathematical challenge known as the *Elliptic Curve Discrete Logarithm Problem* [9]. Moreover, compared to algorithms like *RSA*, ECC achieves equivalent security with significantly smaller keys, thereby reducing processing power requirements for signing and verification [9]. Furthermore, this project employs the *Ed25519* algorithm, whose key-generation curve is defined by

$$y^2 = x^3 + 486662x^2 + x \tag{2.1}$$

and relies on the constant

$$L_{curve25519} = 2^{252} + 27742317777372353535851937790883648493 \tag{2.2}$$

as specified in *High-speed high-security signatures* [10, p. 8] and *RFC 8032* [11]. In summary, key generation is executed via modular arithmetic founded on this constant L value that defines the Ed25519 curve.

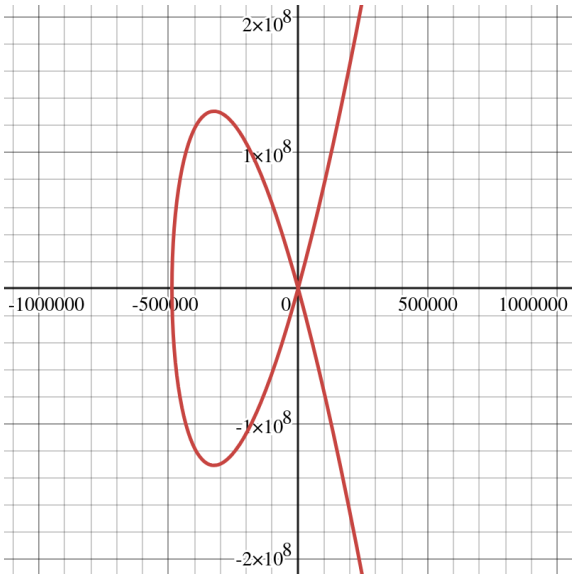


Figure 2.2: *Plot of the Ed25519 curve.*

Furthermore, Ed25519 signature generation differs fundamentally from RSA-based approaches. Specifically, instead of applying complementary mathematical transformations as in RSA, Ed25519 generates a *signature point* on the elliptic curve by combining the private key with the message [12, Digital Signatures/EdDSA and Ed25519].

Finally, signature verification is performed by interpreting the first half of the signature as a point R and the second half as an integer S. The public key is decoded as a point *A'*, and a hash is computed over R, the public key, the message, and the domain separation constant *dom2(F,C)* (described in RFC 8032 [11]). This hash is interpreted as an integer k, and verification is validated by checking a group equation comparing the sum of the public key and the point R with the *generator point* B. The described algorithm can be mathematically described as 2.3 [11, p. 14].

$$SHA512(dom2(F, C) \mid\mid prefix \mid\mid PH(M)) \tag{2.3}$$

The security guarantees provided by public-private key pairs and digital signatures are directly leveraged in passwordless authentication protocols. By issuing cryptographic challenges that can only be signed with a user's private key, securely stored on a trusted device, authentication systems can verify the identity of users without ever transmitting or storing passwords. Thus, this approach directly eliminates many known vulnerabilities associated with traditional password-based authentication methods and demonstrates how the principles of asymmetric cryptography translate into practical solutions for authentication systems.

2.2 Hardware Security Tokens

Hardware security tokens are defined as physical devices engineered to store and protect cryptographic secrets, such as encryption keys and authentication credentials.

Unlike simple portable storage such as a *USB* flash drive containing a password file, non-volatile memory and an embedded *CPU* are typically featured in these tokens. Robust security measures are included, such as secure memory, user-controlled locking mechanisms, and tamper-resistant designs, to ensure that even if an attacker physically acquires the token, extracting its secret data or misusing it is made difficult [13, pp. 306-308].

Figure 2.3: *"Hardware Security Tokens" generated by J. Almström using OpenAI Sora is licensed under the OpenAI Terms of Use. Source:* `https://openai.com/sora`.

Regarding usability, security tokens are used similarly to other portable devices. They can be conveniently carried on a keychain, in a pocket, or stored in a secure location. They are often in small, convenient forms such as USB flash drives, smart cards, or specialized key fobs. Some devices such as smart cards are considered hardware security tokens even when a traditional CPU is not included. In many cases, essential cryptographic functions are performed by a circuit, and the role of the CPU is thereby fulfilled within a more limited scope [13, pp. 306-308].

However, the overall security of these devices is often dependent on user behavior. For instance, if a secure token is left continuously plugged into a computer by the user, an opportunity for unauthorized access is created. This issue could be mitigated through user training and disciplined practices. Nonetheless, a critical dependency on user behavior is retained; secure tokens are made effective only if consistent securing and compliance with security practices are maintained [13, pp.

306-308]. A particular token is the TKey device.

2.3 The TKey Device

The device is presented as the central hardware component of this project. It is designed as a compact USB hardware security token supporting passwordless authentication through asymmetric cryptography. Unlike traditional tokens that store private keys, the TKey device does not store any private keys. Instead, private keys are deterministically generated from a combination of the hash of the running app, a secure unique value assigned to each key, and a user-supplied secret (more details can be seen in Figure 2.5). The risk associated with key exposure is minimized by this approach [5]. Moreover, multi-factor authentication is inherently supported, as hardware token possession is combined with a user secret and dynamic key generation.

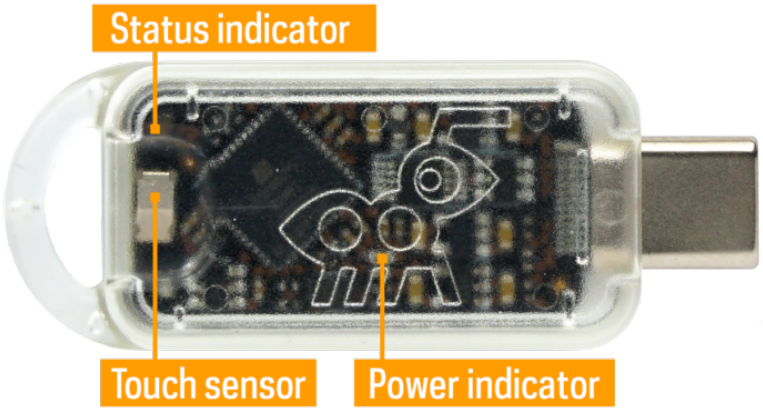


Figure 2.4: *"TKey interaction points" by Tillitis AB is licensed under CC BY-SA 4.0. Source:* `tillitis.se/press`.

Furthermore, the device is designed for both flexibility and adaptability by being open-source. Differentiation from other hardware security tokens is achieved by loading applications at runtime rather than storing them permanently, by which both the efficiency of cryptographic computations and overall system security are enhanced [5].

While the TKey device is capable of performing a challenge-response authentication process, its fundamental security mechanism is based on the dynamic generation of keys. During an authentication attempt, a cryptographic challenge is issued by the host system and subsequently handled by the TKey device using its deterministically generated key. The signed response is then verified by the host system using the corresponding public key. Preservation of authentication integrity is ensured; even if communications are intercepted, the authentication integrity is preserved since no static private key exists within the device.

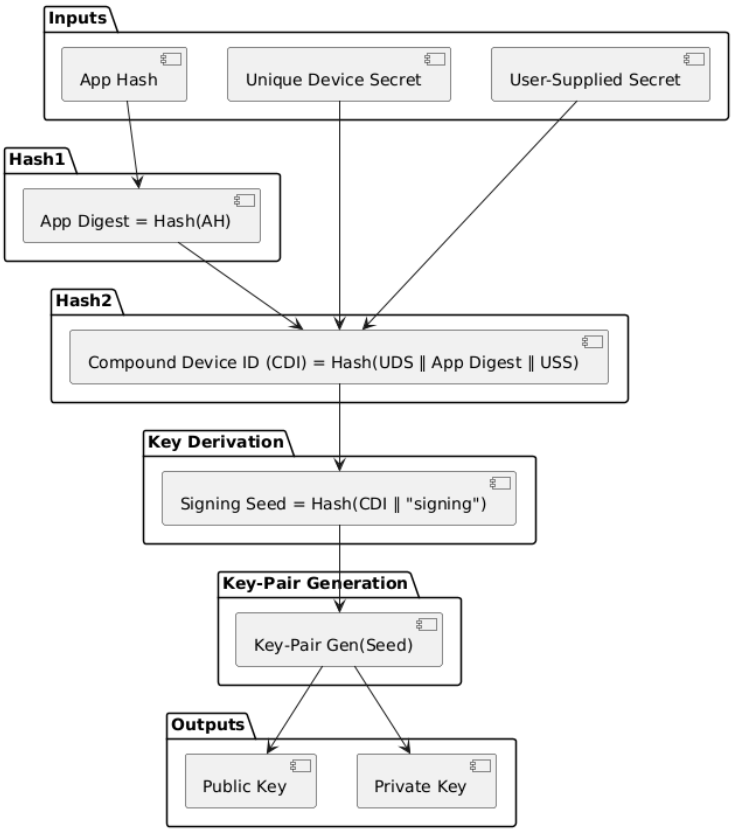


Figure 2.5: *TKey key pair generation overview.*

Multiple applications are supported by the device, including TKey Device Verification, TKey ssh-agent, and the TKey Signature tool, within which Ed25519 cryptography is employed for signing and verifying files [5].

As seen, hardware security tokens such as the TKey device provide a robust foundation for safeguarding cryptographic secrets. However, the power of these devices is realized only when they are incorporated into a well-designed architecture and supported by robust design patterns.

2.4 System Architecture and Design Patterns

Several architectures and design patterns that are common in authentication systems are introduced in this section. These are essential for maintaining a well-structured and scalable system, ensuring secure communication between different components, and improving overall maintainability. By following these established architectures and patterns, flexibility, easier management, and greater adaptability to future changes are ensured in the system. The section begins with coverage of two authentication system architectures and is concluded with coverage of two design patterns.

A *forward proxy* architecture serves as a critical intermediary in network communications, positioned between clients and external servers to regulate access and enhance security. By intercepting client requests, organizational policies are enforced by the proxy—such as access control, anonymity, and content filtering—before forwarding validated requests to their destinations. As illustrated in Figure 2.6, this architecture conceals client identities from external servers while enabling response processing (e.g., decryption or logging) before relaying results back to clients. Forward proxies are considered indispensable in enterprise environments for threat mitigation and performance optimization through caching mechanisms [14]. However, their effectiveness depends on proper scaling to avoid bottlenecks and meticulous configuration to prevent circumvention by malicious actors.

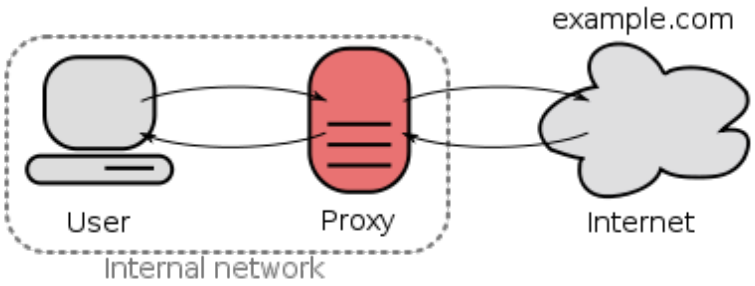


Figure 2.6: *Forward Proxy. [2]*

Complementing network-level architectures, the layered (n-tier) architecture organizes software systems into discrete tiers, each with distinct responsibilities. As depicted in Figure 2.7, the applications are partitioned into discrete tiers. User interactions are managed by the presentation layer, core operations are handled by the business-logic layer, and database interfacing is provided by the persistence layer, while database operations are carried out by the database layer. This separation allows changes to be isolated, code reuse to be promoted, and components to be incrementally refined without destabilizing the entire system [15, pp. 133-139]. By abstracting complexities into layers, the architecture supports scalable development and simplifies testing, making it particularly suited for evolving authentication systems.

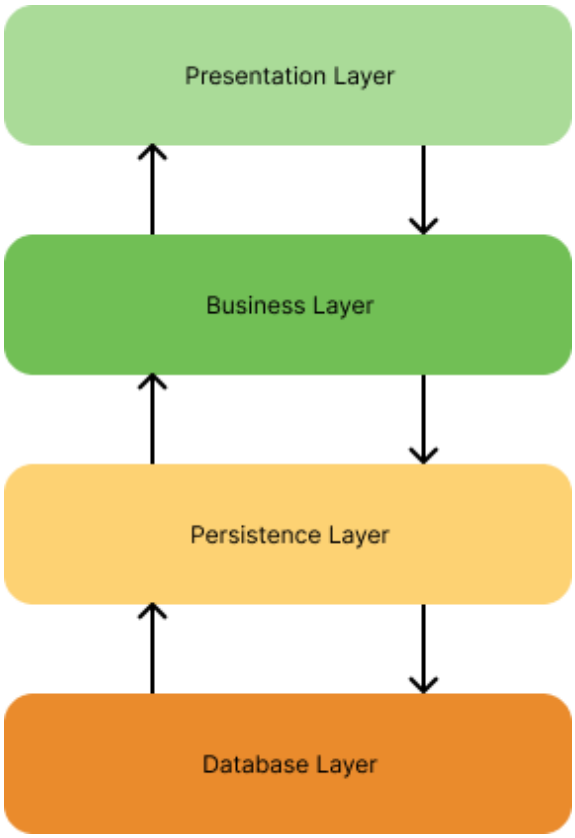


Figure 2.7: *Layered architecture layers.*

Beyond architectural frameworks, recurring security challenges are addressed by design patterns. The validity of sensitive data is limited by *time-based expiration*. Resulting replay-attack risks are mitigated by ensuring transient data usability. Equally critical is input sanitation, a practice vital for NoSQL systems vulnerable to injection attacks due to their schema-less nature. Dynamic queries can be manipulated by malicious inputs, leading to unauthorized data access or corruption. Effective sanitation involves validating input formats, escaping special characters, and employing parameterized queries to treat user input as inert data rather than executable code [16]. For instance, injection of query operators is prevented by ensuring alphanumeric constraints on usernames or labels.

2.5 Security Threats

Some of the most common security threats in the field of computer security are aimed to be presented in this section, which will later serve as the foundation for the threat modeling and vulnerability analysis conducted in this thesis.

A *Denial of Service* (DoS) attack is described as a cyberattack in which the ordinary functionality of a system, network, or other service is intended to be disrupted, ultimately rendering the service unavailable for ordinary users. This attack is typically

achieved when critical system resources such as bandwidth, processing power, or memory are overwhelmed [17, Ch. 4.15].

A *Man-in-the-middle* (MitM) attack is characterized by the interception and possible alteration of communication by an unauthorized third party between two parties, thereby compromising the confidentiality of the communication between said parties. This attack is typically accomplished when the attacker is positioned within the path of communication, enabling them to redirect traffic. As a result, transmitted information can be observed, captured, or manipulated by the attacker with neither party being aware of the situation [17, Ch. 4.28].

A *Social Engineering* attack is defined as a form of cyberattack in which human psychology is exploited to manipulate victims into performing malicious actions that compromise their own security. These types of attacks are distinguished by the fact that all forms of technical defenses are bypassed as they target human behavior rather than system vulnerabilities. One outstanding form of social engineering attack is *Phishing*, in which different communication forms such as emails, SMS, or phone calls are used by attackers so that individuals are lured into performing malicious tasks such as downloading Malware or revealing sensitive information. Another common form of social engineering is referred to as *Domain Name Spoofing* (DNS) in which misleading domain names are used by an attacker (e.g. applesupport.com instead of support.apple.com) to deceive users into trusting fake websites [18].

An *Injection* attack is an attack in which malicious code is injected into a vulnerable application, often with harmful intentions. The cause of these attacks is often improper input validation, allowing hostile input to be processed as a part of the program's code or database query [18, 19]. In relevance to this thesis, there are two types of injection attacks that are most applicable. Firstly, there is *NoSQL Injection*, in which malicious input is used to steal, modify, or delete unauthorized entries from the database [20]. Secondly, there is *Cross-Site Scripting (XSS)*, in which malicious code is attempted to be injected into the web application itself with the intention that the code is run by either the website's server or by the user's machines [21].

A *Session hijacking* attack is when unauthorized control is gained over a communication channel between two parties [17, Ch. 4.18]. In the context of this thesis, it is related to the gaining of access to a session token exchanged between the frontend and the backend. By obtaining this token, the attacker can impersonate a legitimate user and interact with the system under false credentials.

A *Cross Site Request Forgery* (CSRF) attack is when a malicious website is able to hijack your session cookie and perform unwanted actions or requests using it [22]. CSRF attacks can take many forms, but a common attack is carried out by spoofing the origin of an HTTP request and using your cookie to make it look as if you made the request.

This chapter has presented the theoretical foundations for a TKey-based passwordless system, covering public-key and elliptic-curve cryptography, authentication models, hardware token design, architectural patterns, and core security considerations. Chapter 3 describes the methodology used to translate these concepts into a working implementation, detailing development phases, tooling, and the testing strategy.

3

Method

Chapter 2 explored the theoretical foundations of the project, necessary to know before approaching the system development. This chapter outlines the approach and plan for the project. It details the methodology, design strategy, and processes that were followed to develop a secure passwordless authentication system using the TKey.

The project was executed using an agile iterative development strategy, which enabled the breakdown of the overall goal into smaller, more manageable tasks. Continuous feedback and iterative improvements were facilitated throughout the development process by this approach. The work was guided effectively by early prototyping combined with regular evaluations and adaptations to unforeseen challenges. Alignment with the project requirements was maintained, and emerging issues were addressed promptly through frequent meetings and short development sprints. The overall implementation was organized into several distinct phases, as outlined in Section 3.3.

3.1 Core Features and Requirements Elicitation

To lay a foundation for the project, a number of core features were identified as necessary. These features were based on the project description and evaluated as a group. When determining these, most importance was given to creating a robust system, ensuring that a large-scale implementation would require minimal maintenance.

After the core features had been established, a set of requirements was created. These were gathered during brainstorming as a group, and consist of functional, as well as non-functional requirements. The purpose of gathering these requirements was to track the progress, and have some method of measuring the completeness of the prototype created.

3.2 System Architecture and Design

This section covers the systems architecture and design. It covers how the system architecture was planned to look and why it was planned to be designed that way. As shown in figure 3.1, the planned system architecture consisted of three components: the TKey device, the user, and the Application. The private key was managed by the TKey device and all signing operations were handled on behalf of the user, while the corresponding public key was stored by the Application and incoming signatures were verified.

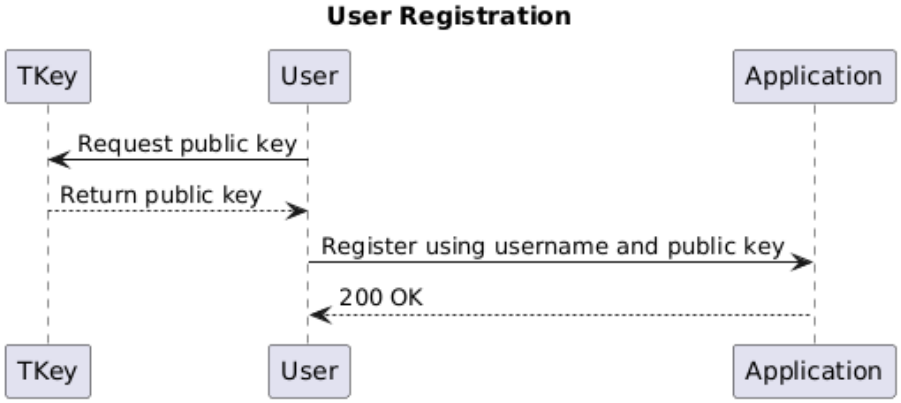


Figure 3.1: *Flowchart showing the initial registration flow concept.*

The registration process was to be initiated by the user requesting a public key from the TKey device. That key, together with the username, would have been forwarded to the application for registration. Upon successful verification, the public key was stored by the application and a confirmation returned, thereby completing the registration.

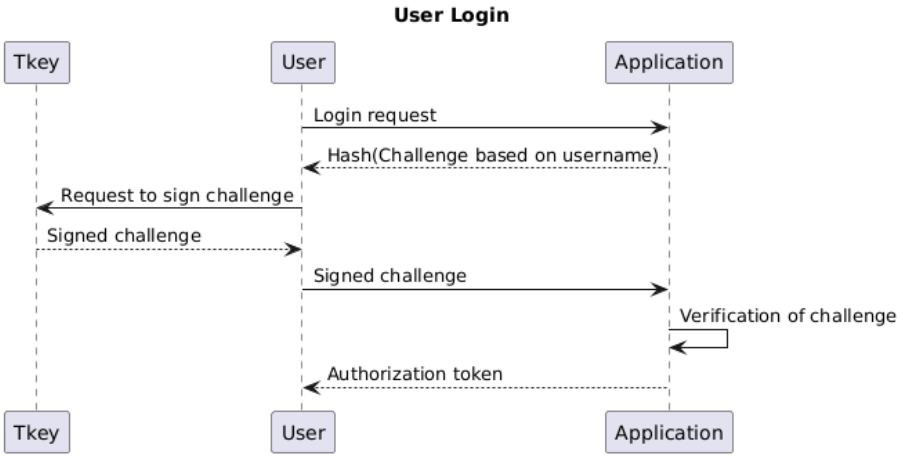


Figure 3.2: *Flowchart showing the initial login flow concept.*

During the planned login process, the username was provided to the Application by the user. A hashed challenge was generated by the Application based on the

received information and was returned to the user. This challenge was forwarded to the TKey device, and it was signed using its private key. The signed challenge was then returned to the user, which forwarded it to the Application for verification. If the signature was valid, an authentication token was issued by the Application; otherwise, an error was returned.

In this architecture, the Proxy design pattern was used to handle the private-key signing operations without modifying the existing Application. Where the user functions as the proxy. As explained in *The Design Patterns Companion* [23, pp. 53-54], a proxy adds behavior externally while preserving the original object's cohesion, enabling security features such as access control and logging. Testability was also enhanced by allowing the proxy to be substituted with a mock during testing. Overall, a flexible, secure, and maintainable design was achieved through the employment of a proxy.

3.3 Implementation Phases

This section outlines the distinct phases of the project, with detailed descriptions of each stage.

Phase 1: Project Kickoff and Initial Research In this initial phase, multiple startup meetings were held to align the team on objectives and scope. Subsequently, requirements were gathered and initial research was conducted to understand the challenges associated with the main problem, while relevant literature and technical documentation were reviewed. The insights gained during this phase were used to lay the foundation for the subsequent development work.

Phase 2: Design and Architecture Planning Following the initial research, the system architecture was designed and planned, and appropriate design patterns were selected. During this phase, subproblems and design choices were defined, which provided a clear blueprint for the development phase.

Phase 3: Development Phase During the development phase, backend development was the primary focus and constituted the bulk of the work. The core functionalities outlined in the requirements were implemented and validated against the planned requirements. Towards the end of this phase, frontend development efforts were directed to ensure that the user interface and related features met the defined criteria.

Phase 4: Enhancement and Documentation Phase During this phase, additional product features that, although considered out of scope for the initial release, or classified as non-functional, were implemented to enhance the overall robustness and user experience of the system. Simultaneously, initial project documentation was drafted to capture the development process and decisions made.

Phase 5: Final Documentation and Code Refinement In this phase, comprehensive documentation was produced and final adjustments were made to the code base. This included polishing the implementation to ensure stability and maintainability, and updating the documentation to reflect the final state of the project.

Phase 6: Project Wrap-up In the final phase, the entire process was reviewed, the documentation was finalized, and preparations were made for the submission and presentation of the thesis. This wrap-up phase ensured that all project deliverables were completed and that lessons learned were documented for future reference.

3.4 Testing

Throughout the project phases, several testing methods had been considered and were partially implemented to ensure the functionality and reliability of the system. Initially, during the planning phase, three testing approaches had been identified as relevant for this thesis: *unit testing*, *integration testing*, and *manual UI/UX testing*

Unit Testing For this project, the GO testing framework had been utilized to implement unit tests. They were primarily applied to verify the correctness of critical functions within the Daemon components. However, owing to time constraints and the relative simplicity of some modules, unit testing had not been applied thoroughly across the system.

Integration Testing Integration testing was prioritized in the project due to the system's reliance on communication between multiple components: e.g., the daemon, the TKey device, and application. These tests were carried out by validating the data flow and ensuring the authentication process functioned as expected across the whole system. These tests were carried out manually to confirm the correct interactions between the different modules.

Manual UI and UX Testing To assess both the user interface and user experience, manual tests were conducted with a group of users who had no prior relationship with the project. In order to make sure these tests were of high quality and usable for validation, the user group testing the project was chosen to include a wide range of potential users such as elders, young people, users with technical and very limited technical backgrounds. Before performing these user tests, a structured approach which can be found in detail in Appendix B was designed and used to minimize the deviation in testing between users and testers.

Git Strategy and CI Pipeline Throughout the development phase, the git feature-branching strategy was used during implementation, with manual pull request reviews being used as quality assurance and sanity checks before features were merged into the main branch. In addition, a CI (Continuous Integration) pipeline was applied to confirm the successful compilation of the project before features

were merged, in order to avoid merge conflicts and to uphold the availability and functionality of the main branch.

3.5 Tools and Documentation

During the project, a range of tools and technologies was utilized to support both development and deployment.

Development tools The following tools were employed:

- Visual Studio Code (VS Code): A powerful, extensible code editor that provides robust support for debugging, version control, and collaborative development.
- Windows Subsystem for Linux (WSL): Enables a native Linux environment on Windows machines.
- Go: A statically typed, compiled programming language, ideal for building backend services.
- React: A JavaScript library for building use interfaces, ideal for building frontend applications.

Hardware and Software Platforms The project was developed on the following platforms:

- TKey: A hardware/software solution that provides secure key management and authentication.
- Linux: A versatile operating system.

Supported Operating Systems In its current state, it was confirmed that the daemon ran on the following operating systems:

- Linux (Running Gnome 48).
- Windows 11.
- Mac.

Support for other operating systems is not guaranteed.

Third-Party Integrations The following third-party tools were also integrated into the project:

- Tillitis software: Open-source tools and software compatible with the TKey [24].
- Gorilla sessions: Part of gorilla web toolkit, used to manage user sessions.
- Gorilla CSRF: A library used to implement Anti-CSRF tokens.
- go-pinentry-minimal: A library to allow secure entry of a user supplied secret.
- Go Crypto library: Library to manage cryptographic keys in go.
- Testify: Library for unit tests in go.
- godotenv: Library to allow for .env files in order to simplify development.

Documentation For code documentation, inline comments were utilized, and a README file was maintained in the repository. The use of GoDocs to automatically generate documentation for functions was also explored, ensuring that the codebase was both accessible and maintainable. Throughout the project, progress was tracked using a logbook, a time log was maintained to document individual contributions, and the project report was used as a central repository of gathered information and reflections.

With the methodological framework firmly in place, which covers the planning of requirements, system architecture, phased implementation, and testing strategy, the blueprint for a passwordless authentication system is established. Chapter 4 shifts focus from planning to actual outcomes by detailing the system's behavior and the user experience of the system.

4

Results

Based on the methodology outlined in Chapter 3, this chapter presents the actual results. An overview of the end-to-end system is provided, followed by detailed sequence diagrams, component interactions, and compiled results from user experience tests. These results show how theoretical designs and implementation phases translate into a functional, secure passwordless authentication prototype.

4.1 System Overview

The final system produced consists of a daemon that users install on their machines and an application that implements the login system. In its current form, the application serves as a proof of concept; you can use any application as long as it provides the proper endpoints. For deployment, the proof-of-concept application and database are served by Docker containers behind a forward proxy that enforces HTTPS-only communication and restricts access to approved systems.

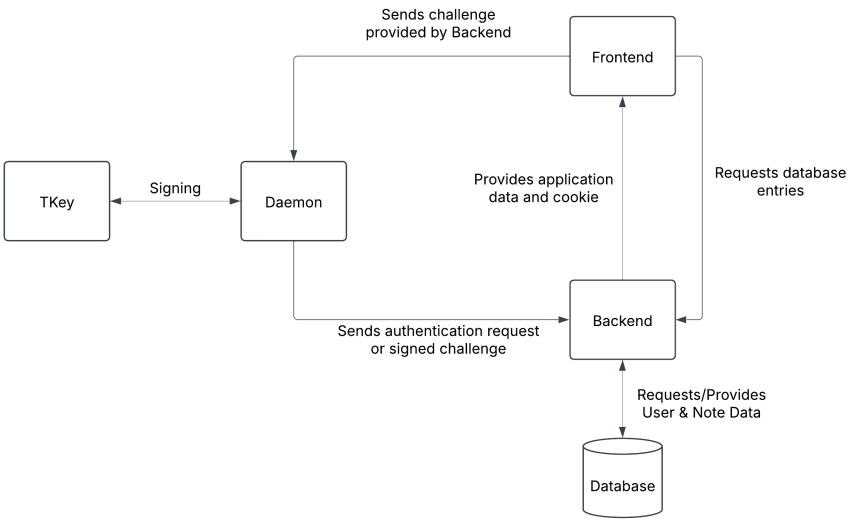


Figure 4.1: *Diagram showing flow of information.*

4.2 Core Features and Requirements

The core functionality, which was determined to be critical for the system, as well as the requirements designed to be able to measure the progress of development are as follows.

- Registration and login with the support of a TKey device is possible.
- Communication with the TKey device is only handled by the daemon, and a challenge can be generated, and verified for authentication.
- A basic authentication interface is provided for users.

Functional Requirements

- The Application must be capable of storing user information.
- The Application must be able to store a public key associated with each user.
- The Application must effectively communicate with the Daemon.
- The Application must grant access only when authentication is successful.
- The Daemon must be able to operate on a Linux operating system.
- The Daemon must be able to send a cryptographic challenge to the TKey device.
- The Daemon must successfully retrieve a signed cryptographic challenge from the TKey device.

Non Functional Requirements

- The system should be easy and intuitive for users to register and log in.
- Authentication using the TKey should be performed quickly, ensuring minimal delay.
- The Application should display clear and informative error messages when authentication fails.

The system's components will now be explained in detail to give a proper understanding of how they work, starting with the daemon.

4.3 Daemon

The daemon serves as the system's central point of contact. It runs on the user's device and intermediates between the application's frontend and backend, enabling services to communicate with the TKey device. Figure 4.2 and Figure 4.3 illustrate this information flow in detail.

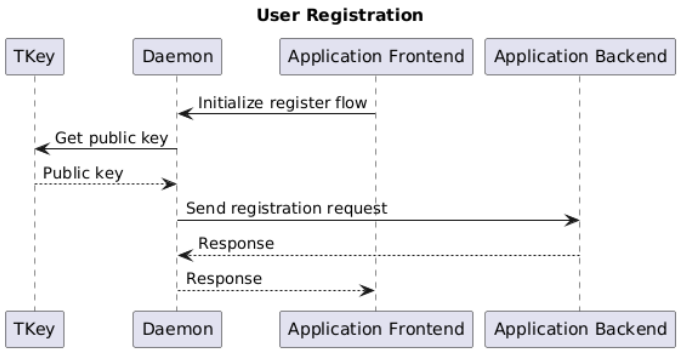


Figure 4.2: *Sequence diagram showing the registration flow.*

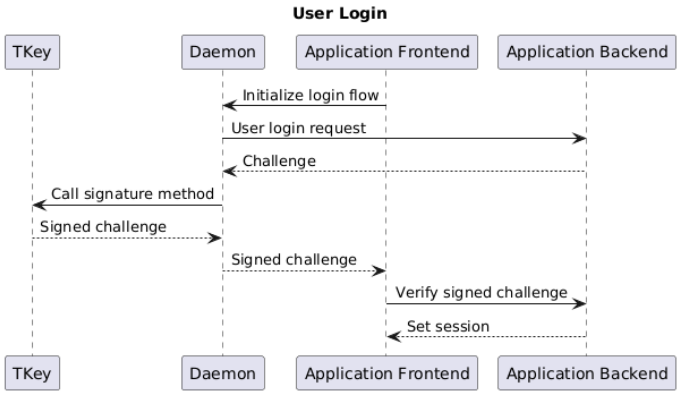


Figure 4.3: *Sequence diagram showing the login flow.*

The daemon uses three software libraries developed and published by Tillitis: tkeysigner, tkeyclient, and tkeyutil. These libraries allow it to retrieve the TKey device's public key and sign cryptographic challenges.

Users can interact with the TKey authentication system through the application that integrates the daemon as its authentication method. As Figure 4.1 shows, the daemon enables end-users to authenticate with a TKey device. If the daemon isn't running, the application cannot authenticate, register, or execute any user tasks.

The application sends requests to the daemon, and the daemon processes them accordingly. Currently, the daemon supports three request types: *login*, *register*, and *add public key*, which are presented in pseudocode in Appendix C.

During registration, users can add a User Supplied Secret (USS) to their key. This allows users to have a password alongside their TKey device for authentication. The

daemon handles the USS functionality instead of the application. Users enter the USS via operating-system prompts rather than through the application's frontend. As a result, prompt appearance varies across operating systems. These prompts appear as an input box in front of the web browser (see Figure 4.4).

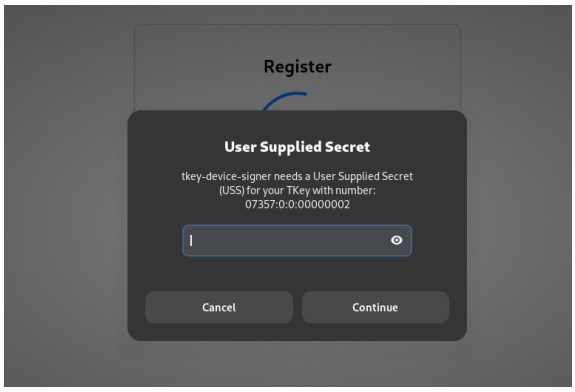


Figure 4.4: *USS System prompt running in Gnome Desktop Environment.*

With the daemon in place, the system application is now presented, which demonstrates that the daemon works with a web application to enable passwordless authentication.

4.4 Application

The application uses a standard web architecture with a frontend and a backend. To demonstrate functionality, a placeholder note-keeping app allows users to create and store notes. The interface includes user-friendly features and provides clear error messages for common issues. For example, the system notifies users when they register with an already taken username or when they try to delete the only public key linked to their account.

When users enter the site, a welcome page displays instructions on how to use the web interface with the daemon. The page also includes a button that redirects users to a registration form, where they enter a username and a label to identify the TKey device (see Figure 4.5). During registration, users run the daemon and connect a TKey device. After users enter their information and click "Register", the system prompts them to enter an optional USS (password), which they can leave blank (see Figure 4.4). If registration succeeds, the system displays a success message. After creating an account, users log in using only their username and USS. If login succeeds, the system refreshes the home page to render the placeholder app.

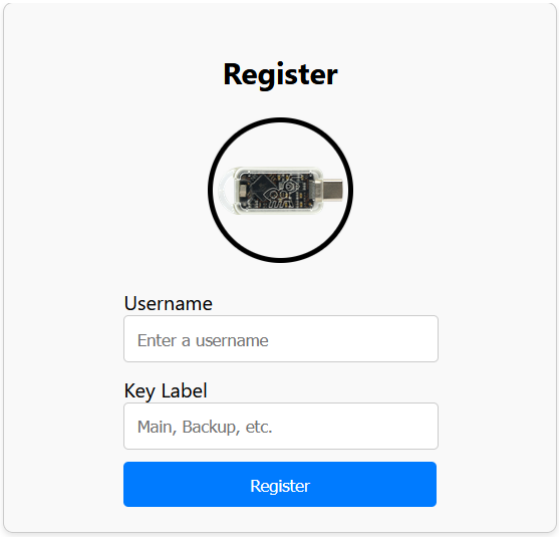


Figure 4.5: *Registration form where users can register an account.*

A settings page was also implemented where users can view the TKey devices associated with their account (see Figure 4.6). They can choose to delete or add new TKey devices; each new device receives the same authentication level as the one used to create their account. Each user may hold up to three TKey devices simultaneously. On the settings page, users delete their accounts by pressing the delete button and confirming the action.

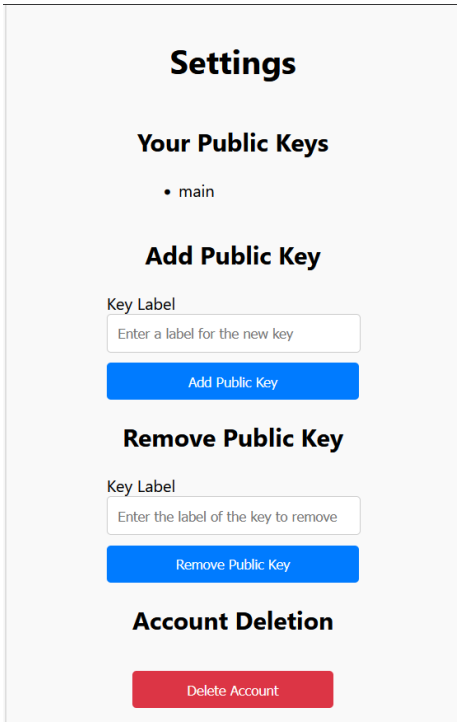


Figure 4.6: *Screenshot of the settings page.*

The backend handles authentication logic and database operations. The implementation is a REST API, which the frontend and daemon communicate. The API provides the following endpoints; the first three are mandatory for the authentication system.

- **/api/register**: Handles logic for registering user.
- **/api/login**: Returns a randomly generated challenge used for logging in.
- **/api/verify**: Verifies if a signed challenge is valid.
- **/api/getuser**: Returns the username of the current session.
- **/api/add-public-key**: Adds an additional TKey to a user.
- **/api/remove-public-key**: Removes a TKey from a user.
- **/api/get-public-key-labels**: Returns the TKey labels for a user.
- **/api/unregister**: Deletes users account.
- **/api/create-note**: Creates a note and saves it.
- **/api/get-user-note**: Gets all notes for an user.
- **/api/update-node**: Updates a note and saves it.
- **/api/delete-note**: Deletes a note.
- **/api/logout**: Terminates the users session in the server and deletes browser cookie.

To authenticate and authorize users, the backend generates a challenge and sends it to the daemon for signing. When the TKey device finishes signing the challenge, it returns the signed challenge to the frontend, which forwards it to the backend for verification. Once the backend verifies, it creates a cookie, attaches it to the response, and sends the response back to the user. The system later uses this cookie to authorize access to protected API endpoints. The session middleware validates the cookie before it invokes each handler. When users want to terminate a session, they press the Logout button in the navigation bar after logging in. This action removes the backend session data and deletes the cookie from the browser. If users do not log out manually, the cookie expires one hour after issuance.

The application backend utilizes a non-relational database created in *MongoDB* to store user information. The database contains two collections, *Users* and *Notes*. The former collection is being used to store user information, while the latter is

used to store the notes for users.

Table 4.1: *Schema for the Users collection.*

Field	Description
`ID`	Unique identification number only used by MongoDB and not used directly for the functionality of the applications.
`Username`	Unique username to identify accounts.
`PublicKeys`	A list of public keys linked to a username which can be used to generate a challenge to be signed by its respective private key for authentication.

Table 4.2: *Schema for the User Notes collection.*

Field	Description
`ID`	Unique identification number only used by MongoDB and not used directly for the functionality of the applications.
`Username`	The owner of the note (references a user's username).
`Name`	Name of the note.
`Note`	The text content of the note.

The application backend uses two repositories, that interface between the database and the rest of the application. Each repository provides the functions the application needs to operate correctly. The database-interaction methods for the *User Repository* and the *Note Repository* interacting with the database can be found in tables 4.3 and 4.4. Each operation encapsulates specific functionality for creating, retrieving, updating, and managing user and note data.

Table 4.3: *Interface methods for UserRepository.*

Method	Description
`CreateUser`	Creates a new user and stores the corresponding information in the database.
`GetUser`	Retrieves an existing user's information from the database.
`UpdateUser`	Updates the information of an existing user.
`DeleteUser`	Removes a user and all associated data from the database.
`AddPublicKey`	Adds a public key to an existing user.
`RemovePublicKey`	Removes a specific public key from a user.
`GetPublicKeyLabels`	Returns a list of key labels associated with a user.

Table 4.4: *Interface methods for the* `NotesRepository`.

Method	Description
`CreateNote`	Creates a new note associated with a user. Stores the note, its title, and the username in the database.
`GetNotes`	Retrieves all notes belonging to a specified user.
`GetNote`	Fetches a single note based on its unique identifier.
`UpdateNote`	Updates an existing note with new data, such as a revised note body or different title.
`DeleteNote`	Deletes a note from the database using its unique identifier.

4.5 User Experience

The prototype's usability and performance are assessed through structured user tests with 23 participants who report a broad range of technical abilities. Each participant receives the same brief oral introduction to the TKey device, then completes three timed tasks in sequence:

1. *Registration*, measured from the moment the user navigated away from the landing page until account creation completed.

2. *Login*, measured from successful registration to successful authentication.

3. *Account removal*, measured from a logged in state to confirmation of removal.

During the tasks, the test leaders avoid answering questions so participants discover the workflow on their own. After the tasks, participants submit free-text motivations and rate:

- Their own *technical ability* (1 = very low ... 5 = very high).

- The system's *usability* (1 = very poor ... 5 = excellent).

- *Willingness to use* the prototype instead of a traditional password (1 = definitely not ... 5 = definitely yes).

The following tables and pie charts present the collected data.

Table 4.5: *Self-reported technical ability (1 = very low … 5 = very high).*

Technical Ability	Count
1	2
2	1
3	10
4	8
5	2

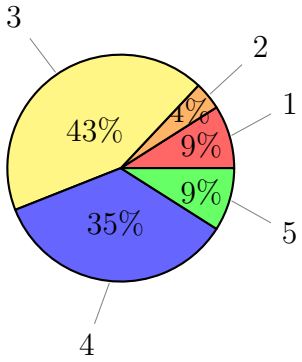


Figure 4.7: *Technical-ability distribution (%).*

Table 4.6: *Usability ratings (1 = very poor … 5 = excellent).*

Usability Rating	Count
1	2
2	2
3	7
4	11
5	1

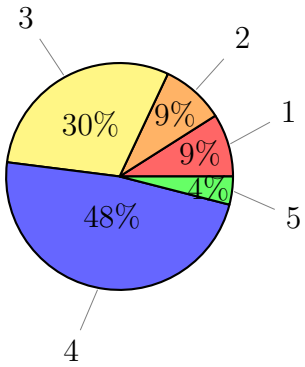


Figure 4.8: *Usability distribution (%).*

Table 4.7: *Willingness to use prototype instead of a traditional password (1 = definitely not … 5 = definitely yes).*

Willingness to use	Count
1	5
2	7
3	4
4	4
5	3

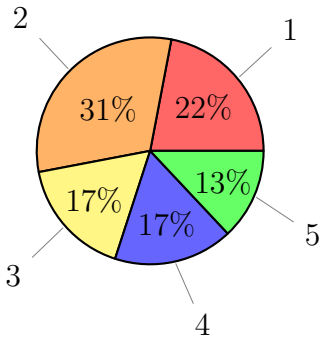


Figure 4.9: *Willingness distribution (%).*

Table 4.8: *Task completion times across all participants.*

Task	Average (s)	Fastest (s)	Slowest (s)
Registration	36.1	10	71
Login	16.5	5	47
Account Removal	29.8	10	120

The written feedback given by the participants clusters around six recurring themes:

1. *Terminology*: Testers repeatedly described "USS" (User-Supplied Secret) as "another password," "USS-ruta," or simply "password," and many noted uncertainty about whether it was optional or required.

2. *Key Interaction*: Participants did not understand what "key label" referred to, and most hesitated over where or when to touch the TKey when its LED blinked green.

3. *Guidence Needs*: Users asked for in-context prompts (e.g. "Touch your TKey now"), a brief quick-start overlay, and more obvious navigation cues, particularly a clear way to exit the settings screen.

4. *UI Discoverability*: Several comments flagged button placement and color choices as unintuitive, leading participants to click around in search of primary actions.

5. *Physical Form Factor*: Descriptions such as "clunky" and "risky to lose" appeared in multiple responses, reflecting concern about carrying a separate USB token.

6. *Mobile Compatability*: The absence of a mobile-friendly option was noted as a barrier, with testers indicating they would use TKey only for high-value services without mobile support.

In this chapter, the prototype implementation of the passwordless authentication system was presented, detailing the end-to-end architecture, daemon integration, and application flow. Chapter 5 will discuss these results and determine if the goals of the project were achieved.

5

Discussion

Drawing on the findings from Chapter 4, this chapter explains differences from the initial design, analyzes the daemon and application implementation, evaluates overall system security, reviews the user experience, examines relevant societal and ethical considerations, and concludes with proposed directions for future improvement.

5.1 Daemon

This section explores several key aspects of the daemon developed in this project thesis, focusing primarily on what design choices were made during development, how these choices affect the end product, and how the daemon could be developed further.

The Daemon is as described in 4.3 developed to run on the user's system. The choice to design the system this way has several benefits, but also some drawbacks which are discussed further.

To start, the way the daemon is developed in this thesis decouples its functionality from the specific application also developed in this thesis. This means that any software or web-based application can choose to integrate the daemon as an authentication method, as long as the mentioned application correctly implements the necessary endpoints required by the daemon, described in 4.3

This decoupling and flexibility of the daemon allows it to serve as a unified authentication system across multiple services with only one daemon instance running on the user's device. Furthermore, the TKey authentication system developed in this thesis can also be used in conjunction with other authentication methods, including traditional passwords, biometric authentication, and other two-factor authentication systems, which may be implemented and used by these external applications.

This method of implementing the daemon to serve as a passwordless authentication system can easily be compared to other types of already existing two-factor authentication systems. Such as the Swedish-developed BankID, where a single app

running on the user's device can be used to authenticate across multiple services such as banks, authorities, and other websites choosing to integrate them. [25]

However, while being labeled as "passwordless", the project incorporates the built-in USS functionality of the TKey device. In some respects, this can be interpreted as a form of password in connection to the TKey device, which in turn challenges the notion of the system being entirely passwordless. The integration of USS is a deliberate design choice to enhance the security of the application by preventing unauthorized use of the TKey device in the event of theft (which is further discussed in Section 5.3). Nevertheless, given the focus of the thesis on developing a passwordless authentication system, this feature is made optional to enable a fully passwordless experience if desired.

As previously mentioned in section 4.3, the daemon depends on three different Tillitis libraries. The *tkeysigner*, *tkeyclient* and *tkeyutil*. Several reasons motivate the decision to use these libraries, as opposed to developing custom external software specifically for the daemon.

Firstly, the choice of relying on Tillitis's existing libraries and code helps save a lot of time for the development team. This allows the team to put more effort into securing both the daemon and application, resulting in a better, safer, and more reliable end product.

Furthermore, the use of Tillitis's libraries also adds more security and longevity to the daemon, as these libraries are expected to be maintained and further developed by Tillitis. This ensures that potential exploits and security vulnerabilities are more likely to be patched in the future, and can easily be fixed in the daemon by updating the libraries. In contrast, if external software is developed specifically for the daemon, it most likely becomes abandonware.

Throughout the planning and development phase of the daemon, a few different approaches to the final agreed-upon solution are researched and evaluated. There are two primary contenders which are not chosen for development.

A *Full React/Typescript* based solution, which proposes developing the entire authentication functionality as part of a React Frontend, in which the web browser communicates directly with the TKey device to authenticate. A *WebAssembly (WebASM)* solution, which consists of developing a GO-based application and then compiling it into a Web Assembly program which can be run inside the user's browser. While both of these solutions are feasible within this project thesis and its scope, we decide to go the path of the daemon-based solution instead for several key reasons.

First, the way web browsers allow communication with USB devices can vary a lot, and some browsers may not allow USB communication at all. Choosing either the React/Typescript or WebAssembly solution reasonably results in our project not being compatible with all web browsers. Second, leading Linux distributions such as

Fedora, Ubuntu, and Linux Mint increasingly rely on sandboxed packaging formats like Flatpak and Snap [26]. These formats restrict direct USB access and render those implementations unusable [27]. Third, the daemon-based solution creates an interface for developers to integrate the TKey authentication system into their own applications and websites with no concerns of compatibility, as long as the daemon can run on the user's system.

The discussion now continues with the implementation of the application.

5.2 Application

The application is designed with a clear separation between frontend and backend components, utilizing React and Go respectively to create a modular and maintainable system. The architecture follows a layered approach that enforces separation of concerns while enabling efficient parallel development.

On the frontend, React's component-based architecture provides an effective way to structure the user interface into reusable modules. The presentation layer focuses exclusively on UI rendering and user interactions, while custom hooks handle business logic and state management. Communication with the backend is standardized through utility functions that implement security measures like CSRF protection, ensuring consistent and secure API interactions.

The backend implementation in Go follows a four-layer architecture that cleanly separates responsibilities. HTTP handlers and middleware form the presentation layer, handling request processing and security validation. Business logic for authentication and cryptographic operations resides in the application layer, working with domain entities defined in the domain layer. Database interactions are abstracted through repository patterns in the data access layer, maintaining independence from other components. This clear separation allows team members to work on different parts of the system simultaneously with minimal coordination overhead.

For data persistence, MongoDB is selected as the optimal database solution. Its document-oriented approach and schema flexibility proved particularly valuable during the iterative development process, allowing for easy adaptation of data structures as requirements evolved. The BSON document format naturally accommodates nested data like public key arrays without requiring complex relational mappings. While relational databases offer advantages for certain use cases, MongoDB's simplicity, seamless integration with Go, and ability to handle semi-structured data make it the better choice for this proof-of-concept implementation. The official MongoDB Go driver provides reliable database connectivity while minimizing development overhead.

This combination of technologies and architectural decisions results in a system where frontend and backend components can evolve independently while maintaining clean integration points. The layered structure not only facilitated parallel

development but also improved testability, as each component can be validated in isolation. By carefully selecting tools that aligned with the project's goals of simplicity and flexibility, the implementation successfully demonstrates the core authentication concepts while maintaining a codebase that can be easily extended or modified as needed.

5.3 Security Considerations

This section introduces several security considerations that our project may be subject to. It covers potential threats and attack vectors, an analysis of potential vulnerabilities, and security measures and mitigation strategies against potential threats.

To start, a threat model of the proposed authentication system is defined. This is done by laying out the foundational assumptions, scope boundaries, critical assets, likely adversaries, and the attack surfaces.

It is assumed that the TKey device itself contains no exploitable hardware or software flaws, and that the chosen cryptographic algorithms resist brute-force attacks. The scope of the analysis limits itself to the authentication system as implemented in this thesis. Firmware or software vulnerabilities within the TKey device or its Tillitis libraries are not considered, nor are attacks targeting third-party applications beyond the application's API.

There are some system assets which are especially important to protect. The following table lists the system's most sensitive elements. Loss or compromise of any of these critically undermines security.

Table 5.1: *Sensitive assets to be protected.*

Asset	Description
USS (User Supplied Secret)	The user's chosen authentication secret, used alongside the TKey
Session cookies & Tokens	Credentials issued after successful login, required for accessing protected endpoints
Physical TKey Devices	Hardware tokens whose possession grants authentication capability
Database Entries	Stored user records and user-generated content

Three different attack surfaces exist. First, the USB-C interface on the user's machine is vulnerable to physical tampering. Second, API endpoints of both the local daemon and the application backend are exposed to attack. Third, session cookies and authentication tokens stored in the application frontend are at risk of compromise. The following table lists the three primary threat vectors that could abuse

these attack surfaces.

Table 5.2: *List of potential adversaries.*

Attacker	Description	Goal
External attackers	Hacker from outside the system	Gain unauthorized account access
Physical Attackers	Attempts to physically get hold of other users TKey devices	Steal TKey devices to impersonate users
Malware Attackers	Infects users' devices	Gain unauthorized account access and or steal data

Combining the sensitive assets, attack surfaces, and threat vectors produces a table that lists various known threats to the system, together with associated attack vectors and potential impacts.

Table 5.3: *Known threats to the system.*

Threat	**Attack Vector**	**Description**	**Impact**
Theft of TKey device	Lost or stolen TKey device	Physical attacker steals users TKey device	Unauthorized access if no USS is set
Phishing Attack	Fake authentication system	Attacker tricks user into revealing sensitive information	Users USS or other sensitive data is leaked
Malware	Machine compromised by Malware	An attacker infects users machine with malware	Keylogging which leads to a compromised USS
Denial of Service	Application endpoints	Attacker overloads server with requests	Authentication service is unavailable for legitimate users
Session Hijacking	Leaked session token or cookie	Attacker gets hold of users session token or cookie	Unauthorized access to users account for a limited time
Man in the middle attack	Compromised network	Attacker steals challenge/response by intercepting request	Unauthorized access to users account

A vulnerability analysis can be performed using the threat model. Possible vulnerabilities are identified and considered during the design of the system. For each vulnerability, its location in the architecture, a brief explanation of the weakness, the potential impact, and a severity score are noted.

Severity is rated using the *Common Vulnerability Scoring System (CVSS) 4.0*, main-

tained by the *Forum of Incident Response and Security Teams (FIRST)* [28]. Scores are produced using FIRST's online CVSS calculator [29], applying the *Base* and *Threat* metric groups (as seen in figures 5.1 and 5.2) and parameter values deemed most realistic for our system. A detailed calculation example of a CVSS score can be found in Appendix A. The following table presents the findings:

Table 5.4: *Analysis of potential vulnerabilities and their severity.*

Vulnerability	**Whereabout**	**Description**	**Impact**	**Severity (CVSS Score)**
No USS or other authentication method	Registration / Login	User chooses to not use a USS or other authentication method	Attacker can get access to users account if TKey is stolen	6.8
NoSQL Injection	MongoDB Database	Database is subject to NoSQL Injections	Attacker can steal, delete or modify database	10
Cross Site Scripting	Daemon	Daemon does not filter input from requests allowing remote code execution	Attacker can run code on application server	10
Non encrypted data transfer	Daemon-application communication	Packets sent between daemon and application are not encrypted Allowing Man in the middle attack	Data interception / Man in the middle attack	8.6
Lack of request limitations	Application endpoints	Application does not have a rate limit to the number of requests	Denial of Service	8.2

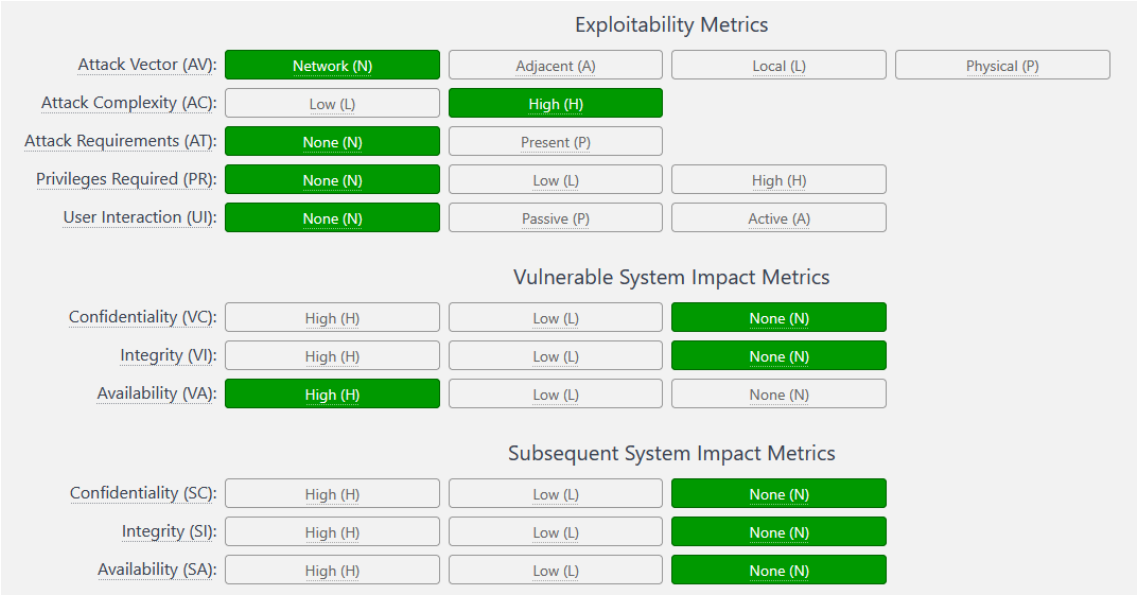


Figure 5.1: *Base metrics.*

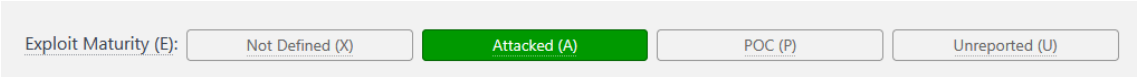


Figure 5.2: *Threat metrics.*

This vulnerability analysis shows some vulnerabilities nearing and some reaching the most severe CVSS score (10). To ensure that vulnerabilities are mitigated, threats associated with each vulnerability are identified and appropriate mitigation strategies defined. The following table lists security measures and mitigation strategies for various threats associated with previously identified vulnerabilities.

Table 5.5: *Security measures and mitigation strategies against potential threats.*

Threat	**Mitigation Strategy**	**Mitigation Type**	**Description**
Theft of TKey device	Allow User Supplied Secret	Preventative	Allow user to use a User Supplied Secret in order to use device
Theft of TKey device	Allow backup TKeys to recover account	Responsive	User can use backup keys to recover account and lock out attacker
Session Hijacking	Short lived session tokens	Preventative	The session tokens are short lived
Denial of Service	Rate limit for application	Preventative	Application should limit amount of requests an IP address can make
Session Hijacking	HTTP-Only Cookies	Preventative	Prevent e.g. malicious plugins from accessing the cookies
Man in the middle attack	Encrypted communication	Preventative	Communication between daemon and application is encrypted with HTTPS
Cross Site Request Forgery (CSRF)	Anti-CSRF Tokens	Preventative	Tokens are sent from the server after login that are stored in the browsers JavaScript. These tokens are then used to verify HTTP Requests

To summarize, because the TKey device derives its private key internally and never releases it, the key itself posed no measurable risk. Instead, the most prevalent issues are general account management issues. The worst-case scenario security flaws are:

- *Insecure API endpoints in the daemon* allow an attacker to sidestep the challenge-response flow.
- *Replay attacks that grant unauthorized sessions* occur when the surrounding web application accepts the same signed message more than once.
- *User-data leakage from the application*, for example through wrongly configured database queries or missing access checks.

Only the first issue is inherent to the daemon code delivered in this project. The latter two originate in the consuming web application's ordinary account-management and authorization logic. For the daemon code to work, the application exposes just three identity-related endpoints: *register*, *login*, and *verify*. Everything else belongs to the application's own business layer and lies outside the scope of the work presented here.

With the security covered, the discussion moves on to whether users can or want to use the prototype in everyday scenarios. The following section discusses the results of the user experience tests.

5.4 User Experience

The user study reveals mixed but instructive feedback on the daemon's usability, clarity, and perceived value. Despite participants representing a wide range of self-assessed technical abilities, the majority complete core tasks without external help, indicating a baseline level of functional usability. The average task times, 36.1 seconds for registration, 16.5 seconds for login, and 29.8 seconds for account removal, suggest a well-balanced model once users become familiar with the flow.

Usability ratings skew positively, with over 50% rating the system a 4 or 5, though the presence of low scores (two participants rated it 1, two rated it 2) reflects inconsistency in user experience. Willingness to adopt the system is notably lower, with 52% expressing reluctance (ratings of 1 or 2), indicating skepticism toward replacing traditional passwords with the TKey system in its current form.

Qualitative feedback clarified the cause of this reluctance. Terminology confusion, especially around the USS input, leads many to assume it is a password field, undermining the system's core premise of passwordless authentication. Poor labeling and insufficient guidance during TKey device interaction (e.g., when to touch the key, what blinking signals mean) further degrade user confidence. UI weaknesses such as unclear button pliancy and lack of navigation cues worsen these issues.

Hardware concerns are also prominent. Participants find the physical TKey device "clunky" and inconvenient, particularly when compared to existing authentication solutions. The absence of mobile compatibility reinforces this drawback, positioning the device as viable only for niche or high-security use cases.

These findings point to three areas requiring substantial improvement:

- Precise and consistent language, particularly around security concepts.
- Real-time, contextual guidance to reduce hesitation and missteps.
- Improved form factor and mobile integration to broaden use-case appeal.

Some of the collected feedback is implemented into the daemon, such as a more comprehensible text prompt when entering the USS.

Despite current limitations, the user tests confirm technical feasibility and offer insight into improving the product, as well as providing a better understanding of broader societal and ethical aspects.

5.5 Societal and Ethical Aspects

This section provides an analysis of the societal and ethical dimensions of the project. The analysis outlines both advantages and potential drawbacks of the project from ethical and societal perspectives and justifies the decisions made during development.

By eliminating traditional passwords, the system could reduce vulnerabilities, including weak passwords, phishing, and credential reuse. This enhancement in digital security could foster increased trust in digital services. Given the growing frequency of password-related breaches, these enhancements are crucial for both individual users and organizations.

Although the benefits of a passwordless system are substantial, ensuring equitable access remains essential. If the system requires technical expertise that exceeds the average user's abilities, it may exclude certain demographic groups. To mitigate this, the design prioritizes user-friendly interfaces featuring clearly visible buttons, informative error messages, and an intuitive user flow. Such an approach could bridge the digital divide and extend enhanced security benefits to a wider audience.

Streamlining the authentication process is expected to enhance security while simultaneously improving user convenience and productivity. Eliminating the need to remember and manage multiple passwords can reduce daily friction in digital interactions, potentially boosting overall user satisfaction and system efficiency.

The ethical handling of user data is important in the system. Although the TKey securely stores cryptographic credentials, the application is designed to collect only the minimal necessary personal information: a username, a public key, and a label for the public key. While theoretically only a public key is needed for identification, this design choice further enhances user-friendliness. If a data breach occurs, the limited information collected would restricts the direct harm to individuals by primarily exposing identifiers rather than sensitive data. The company utilizing the system is responsible for ensuring compliance with data protection regulations, including GDPR, and managing personal data accordingly. This approach reinforces the commitment to security while acknowledging the company's role in GDPR compliance.

Relying on hardware like the TKey demands a high level of transparency. How the system operates is clearly communicated, the security benefits it may provide, and

potential associated risks, such as the implications of a lost or stolen TKey. In the case of a lost or stolen TKey, a user can have multiple keys associated with their account, which allows them to regain access to their account if a key is lost. Users can also remove individual keys or delete their account entirely, ensuring no residual traces remain.

In developing the passwordless authentication system, the project carefully weighs both the benefits and the potential ethical challenges. The decisions, ranging from the adoption of hardware-based security to the design of accessible user interfaces, reflect a balanced consideration of ethical values such as privacy, transparency, and inclusivity. These choices aim to maximize the system's benefits while mitigating risks and upholding ethical integrity.

End users represent a critical dimension in the impact of any technological system. While the design of such systems can incorporate robust security measures and ethical guidelines, these efforts do not control the behavior or intent of the end user. The design decisions and mechanisms described in this thesis are intended to protect users from criminal behavior. However, these same elements might also be exploited to provide a secure platform for illegal activities.

This possibility highlights a broader challenge of how to prevent misuse by end users. The difficulty of this issue lies in the dual-use nature of technology, where innovations intended for beneficial applications can also be adapted for harmful purposes.

With project results discussed and shortcomings identified, focus now shifts to improving the system prototype in future work.

5.6 Future Work

This section will cover possible improvements that can be made in the future for the system.

Societal and Ethical Improvements To broaden accessibility, the system could be adapted to comply with the international Web Content Accessibility Guidelines (WCAG) [30]. This adaptation would involve redesigning the user interface, navigation, and interactive elements to ensure compatibility with assistive technologies such as screen readers, thus ensuring a more inclusive experience. Ultimately, while the process of integrating the WCAG guidelines presents trade-offs, the long-term benefits make it a definitive improvement to include a broader audience.

Multifactor Authentication Improvements The integration of multifactor authentication represents a significant improvement and will substantially enhance the overall security of the system. Although the current implementation relies on a robust challenge-response mechanism of the TKey device together with a user-supplied secret, adding additional factors such as biometrics or one-time passwords (OTP) would introduce additional layers of security. For example, including a biometric

sensor could validate a user's identity before the TKey device processes the cryptographic challenge. Although these enhancements were beyond the scope of the development phase, they offer a relevant direction for future work. Ultimately, by integrating multifactor authentication, a more resilient and secure system can be created at the cost of ease of use, which may play a large role in an authentication system.

Cross-platform Authentication While performing user tests of the passwordless authentication system using the TKey device, a strong demand for mobile support was identified. Many users highlighted that they frequently access the same accounts on both desktop and mobile devices and therefore expect a consistent cross-platform authentication experience. To address this need, future development could focus on extending the current desktop-centric application to support mobile platforms. This will involve adapting the existing user interface for smaller screens, ensuring compatibility with mobile operating systems (primarily Android and iOS) and integrating with mobile-specific APIs for secure communication with the TKey device over USB-C or through potential wireless interfaces. In addition, adjustments to the authentication flow and session management may be required to accommodate mobile-specific security considerations and platform constraints.

Single Sign-On (SSO) It is possible to implement an authentication mechanism that allows users to access multiple applications or systems using a single set of credentials. This eliminates repeated logins by delegating authentication to a central authority, which issues tokens or assertions after verifying the identity of the user once. Subsequent systems trust these tokens and grant access without requiring further authentication [31]. SSO reduces credential sprawl, improves security through centralized policy enforcement, and simplifies the user experience. If the TKey device is integrated into an identity provider, it could serve as the authentication factor in a login system. In this model, third-party websites would rely on the identity provider for user verification, eliminating the need to handle the login process directly. This approach is more complex to implement, but could significantly improve interoperability and usability compared to the current standalone implementation.

Persistent TKey Connection Currently, the TKey is only required during registration or when signing in. Once authenticated, users can disconnect the TKey without affecting the session. While acceptable during account creation, this poses a security risk post-login, as session integrity is no longer tied to TKey presence. Enforcing a continuous TKey device connection during active sessions would enhance security by immediately terminating access upon disconnection. However, the existing implementation relies on unidirectional HTTP communication between the web application and the daemon, making real-time status monitoring resource-intensive. To address this, a persistent bidirectional connection, such as WebSocket or a similar protocol, could be implemented to enable low-latency state synchronization. This would allow for immediate session invalidation when the TKey device is removed.

Plug and Play The system as designed is reliant on a user-installed application in order to make the authentication system work. In an ideal scenario, the authentication key should be able to work as plug and play, without requiring additional software. This would simplify the use of systems such as WebAuthn, and make it easier to follow other standards for hardware security tokens. One method to achieve this could be to develop a driver for the TKey device, enabling it to emulate any Universal 2nd Factor (U2F) device by binding its hardware-specific functions to a generic interface.

6

Conclusion

This thesis explored the design, development, and evaluation of a passwordless authentication system using the TKey device. The goal was to provide a secure and user-friendly alternative to traditional password-based login methods. By eliminating the need for shared secrets and by utilizing the principles of cryptography, the system helps prevent common attacks such as phishing, credential theft, and data breaches.

The developed prototype demonstrated that a TKey-based approach to authentication was both technically feasible and practical for real-world use. It also demonstrated security improvements, supporting the idea that passwordless authentication solutions offered notable benefits over traditional methods.

Some challenges still remain, for example handling device loss, ensuring service and device compatibility, and improving overall user experience. In addition, exploring other two-factor methods in conjunction with the TKey, such as biometric recognition or one-time passwords, presented great opportunities for future work.

An important aspect of this project was evaluating how users interacted with the system in practice. Although the project proved usable in test scenarios, user feedback indicated that clearer instructions, broader device compatibility, and more effective and safer recovery methods were necessary for large-scale deployment. These findings highlighted the critical need to balance security with an accessible user experience, especially when introducing new authentication technologies.

Additionally, the current need for a *Daemon* to be installed and running on the authenticating machine, currently acts as a barrier against more seamless deployment, and smoother user experience. This situation presented a significant opportunity to explore the elimination of the Daemon entirely while maintaining security, authentication flow, and modularity.

Ultimately, the project demonstrated that moving towards a passwordless future was feasible and that hardware security tokens such as the TKey device could play an important role in this transition by offering users a secure and convenient way to authenticate online.

Bibliography

[1] Wikimedia Commons, "File:Asymmetric encryption scheme.png — Wikimedia Commons, the free media repository," 2024. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Asymmetric_encryption_scheme.png&oldid=854206923 (Accessed: Mar. 19, 2025).

[2] Wikipedia contributors, "Proxy Server," 2014. [Online]. Available: https://web.archive.org/web/20140331192003/https://en.wikipedia.org/wiki/Proxy_server (Accessed: Apr. 9, 2025).

[3] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*. Springer Berlin Heidelberg, 2015. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=ece49cef-6bf8-3712-afb0-1fb93720ff1a (Accessed: Jan. 30, 2025).

[4] IBM, "X-Force Threat Intelligence Index 2024," 2024. [Online]. Available: https://www.ibm.com/downloads/documents/us-en/10c31775c0d40a37 (Accessed: Mar. 24, 2025).

[5] Tillitis, "Tkey," n.d. [Online]. Available: https://tillitis.se/products/tkey/ (Accessed: Jan. 30, 2025).

[6] D. Winder, "Password warning: 50% of internet users open to reuse attack," 2025. [Online]. Available: https://www.forbes.com/sites/daveywinder/2025/03/14/password-warning-50-of-internet-users-open-to-reuse-attack/ (Accessed: Mar. 20, 2025).

[7] All About Cookies, "84% of Internet Users Practice Dangerous Password Behaviors [Survey]," 2025. [Online]. Available: https://allaboutcookies.org/password-users-behavior-survey (Accessed: Mar. 20, 2025).

[8] T. Oduguwa and A. Arabo, "Passwordless Authentication Using a Combination of Cryptography, Steganography, and Biometrics," 2024. [Online]. Available: https://doi.org/10.3390/jcp4020014 (Accessed: Apr. 29, 2025).

[9] SSL Support Team, "What is Elliptic Curve Cryptography

(ECC)?" 2024. [Online]. Available: https://www.ssl.com/article/what-is-elliptic-curve-cryptography-ecc/ (Accessed: Feb. 25, 2025).

[10] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," in *Proc. 13th Int. Conf. Cryptographic Hardware and Embedded Systems*, ser. CHES'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 124—-142.

[11] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8032

[12] S. Nakov, *Practical Cryptography for Developers*. n.p., 2018. [Online]. Available: https://cryptobook.nakov.com/ (Accessed: Mar. 4, 2025).

[13] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=7d838396-57cb-3542-b970-9a0091631d63 (Accessed: Feb. 28, 2025).

[14] M. A. Jabbar, B. B. Gupta, and S. R. Sharma, "Review and Analysis of Proxy Servers and Their Security Issues in Computer Networks," *J. Scientific and Technical Advancements*, pp. 234–239, 2016. [Online]. Available: https://www.ijsta.com/papers/IJSTAV2N4Y16/IJSTA-V2N4R54Y16.pdf (Accessed: Apr. 6, 2025).

[15] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2021. [Online]. Available: https://mrce.in/ebooks/Software-Fundamentals%20of%20Software%20Architecture.pdf (Accessed: Apr. 6, 2025).

[16] W. Han, Z. He, and Z. Wang, "Securing NoSQL Databases: A Study on Input Handling and Injection Prevention," 2021. [Online]. Available: https://www.cs.ru.nl/~erikpoll/papers/secure_input_handling.pdf (Accessed: Apr. 7, 2025).

[17] C. Thomas, P. Fraga-Lamas, and T. M. Fernández-Caramés, *Computer Security Threats*. IntechOpen, 2020. [Online]. Available: https://mts.intechopen.com/storage/books/9234/authors_book/authors_book.pdf. (Accessed: Apr. 17, 2025).

[18] IBM Cloud Team, "Types of Cyberthreats," 2024. [Online]. Available: https://www.ibm.com/think/topics/cyberthreats-types (Accessed: Apr. 18, 2025).

[19] OWASP, "OWASP Top Ten," 2024. [Online]. Available: https://owasp.org/www-project-top-ten/ (Accessed: Apr. 18, 2025).

[20] PortSwigger, "NoSQL injection," n.d. [Online]. Available: https://portswigger.net/web-security/cross-site-scripting#what-is-cross-site-scripting-xss (Accessed: Apr. 22, 2025).

[21] ——, "Cross-site scripting," n.d. [Online] Available: https://portswigger.net/web-security/cross-site-scripting#what-is-cross-site-scripting-xss (Accessed: Apr. 22, 2025).

[22] W. Zeller and E. W. Felten, "Cross-Site Request Forgeries: Exploitation and Prevention," 2008. [Online]. Available: https://people.eecs.berkeley.edu/~daw/teaching/cs261-f11/reading/csrf.pdf (Accessed: Apr. 19, 2025).

[23] S. L. Bain, *The Design Patterns Companion*. Project Management Institute, 2020. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=156fc28a-2e0e-3edc-8267-a2f7f985db92. (Accessed: Jan. 31, 2025).

[24] Tillitis, "Applications," n.d. [Online]. Available https://tillitis.se/download/ (Accessed: Jan. 30, 2025).

[25] BankID, "About BankID," n.d. [Online]. Available https://www.bankid.com/en/privat/om-bankid (Accessed: Apr. 3, 2025).

[26] G. Whittaker, "The Future of Linux Software: Will Flatpak and Snap Replace Native Desktop Apps?" 2025. [Online]. Available: https://www.linuxjournal.com/content/future-linux-software-will-flatpak-and-snap-replace-native-desktop-apps (Accessed: Apr. 6, 2025).

[27] Flatpak Team, "Flatpak documentation," n.d. [Online]. Available: https://docs.flatpak.org/en/latest/index.html (Accessed: Apr. 6, 2025).

[28] Forum of Incident Response and Security Teams (FIRST), "FIRST Strategy Framework," n.d. [Online]. https://www.first.org/about/strategy/ (Accessed: Mar. 9, 2025).

[29] ——, "Common Vulnerability Scoring System Version 4.0 Calculator," n.d. [Online]. Available: https://www.first.org/cvss/calculator/4.0 (Accessed: Mar. 9, 2025).

[30] W3C Web Accessability Initiative (WAI), "WCAG 2 Overview," 2024. [Online]. Available: https://www.w3.org/WAI/standards-guidelines/wcag/ (Accessed: Mar. 19, 2025).

[31] T. Bazaz and A. Kahlique, "A Review on Single Sign on Enabling Technologies and Protocols," 2016. [Online]. Available: https://www.researchgate.net/publication/309225903_A_Review_on_Single_

Sign_on_Enabling_Technologies_and_Protocols (Accessed: May 2, 2025).

A

Sample CVSS Calculation

Calculation example of Denial of Service Threat

- **Attack Vector:** *Network* as attack can be done outside of local network of application
- **Attack Complexity:** *High* as generally, a lot of computing power is required to perform
- **Attack Requirements:** *None* as attack can be performed at any time
- **Privileges Required:** *None* as it can be performed by any attacker without any privileges
- **User Interaction:** *None* as attack can be carried out without any interaction by users
- **Confidentiality:** *None* as no data is being leaked by attack
- **Integrity:** *None* as no data is altered by attack
- **Availability:** *High* as service will either be put down or slowed down making it harder or impossible for normal users to access data
- **Subsequent systems:** *None* for all parameters as neither confidentiality, integrity or availability of subsequent systems will be affected by a Denial of Service attack.
- **Exploit Maturity:** *Attacked* as Denial of Service attacks are common and already been used in practice against other services.

These parameters result in a CVSS score of **8.2**

B

Manual UI/UX Testing procedure

Procedure
The test began with a brief verbal explanation of what the TKey is and its core functionality, including visible feedback like the blinking LED. The user was then handed a computer, a TKey and the application homepage open. The user would then be verbally told to do a specific task, such as creating an account. The time taken for the user to independently complete the following tasks was recorded:

- Account Registration
- Login Process
- Account Deletion

No assistance was provided during the test unless strictly necessary, in order to observe the user's natural interaction with the system. The observations, comments and other remarks were documented, and the user was asked to rate various aspects of the experience. These aspects and comments were the following:

Ease of Use: How intuitive the user thought the system was. 1 = Impossible to use on their own, 5 = Fully intuitive, no guidance needed.

Likelihood of replacing Passwords with TKey: How likely the user was to replace their passwords with a TKey under the assumption it was available on all their platforms. 1 = Never, 5 = Definitely Would.

Questions asked by the user: Observations documented by the tester of questions the user asked during the interaction.

Actions user got stuck on: Observations documented by the tester of actions the user got stuck at during the interaction.

Other comments: Comments provided by the user after the testing and documented by the tester.

C

Pseudocode

Registration request

Input: appUrl, username, label
Output: response, error
begin
    `// Get the public key from tkey`
    pubKey ← GetTkeyPubKey()
    if *pubKey retrieval failed* **then**
        return (nil, error)
    `// Construct the registration URL`
    regUrl ← appUrl + "/api/register"
    `// Send the request with pubKey, username, and label to application`
    (response, error) ← sendRequest(regUrl, pubKey, username, label)
    if *error exists* **then**
        return (response, error)
    return (response, nil)

Login request

Input: appUrl, username
Output: username, signedChallenge, error
begin

```
// Fetch the generated challenge from the server
challengeResponse ← GetChallenge()

if challenge fetch failed then
    return (nil, error)

// Sign the challenge
signedChallenge ← signChallenge(user, challengeResponse)

if if signing failed then
    return (nil, error)

return (username, signedChallenge, nil)
```

Add public key request

Input: ResponseWriter w, Request r
Output: JSON response with pubkey or error status
begin

```
// Get the public key from tkey
pubKey ← GetTkeyPubKey()

if pubKey retrieval failed then
    return error

return (pubKey)
```

D

Use of AI tools

AI has been used in several different ways during this project. It can be divided into two different groups.

System Implementation To accelerate development, we used AI code completion and recommendation tools. By providing high-level descriptions of desired functionality, we obtained implementation suggestions and code snippets. All AI-generated code was then reviewed, tested, and finalized by the project team to ensure correctness and to prevent plagiarism. The primary tool used for this purpose was GitHub Copilot.

Report Writing AI language tools supported the refinement of our text by improving grammar, structure, coherence, and overall readability. These tools were not used to collect data, generate factual content, or author entire sections. Instead, they provided editorial suggestions. Every AI-recommended change was carefully reviewed and fact-checked by the authors. It has also been used to generate images to include in the report.

Models Several different models and versions have been used between January-May 2025. This includes:

- ChatGPT
- Claude
- DeepSeek
- Mistral

Prompt examples

> **Prompt:** "I want to use present tense. Read the text and provide a table with excerpts of original and proposed change. *text snippet*"

AI Suggestion:

Original	Proposed
text in past tense	text in present tense
text in past tense	text in present tense
text in past tense	text in present tense

Authors' action: Reviewed each proposed change, ensured consistency with present-tense narrative, and applied to the final draft.

Prompt: "How can this be done? Provide bullets that suggest different approaches. *code snippet*"

AI Suggestion:

- Suggested
- Approach
- Here

Authors' action: Reviewed each suggested approach, chose the most suitable one and implemented it.

Verification We take full responsibility for report content correctness. We have followed good research practice: no plagiarism, falsification, or fabrication. AI suggestions were used only as editorial or developmental aids and were always verified by the authors.