CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Experimental Security Evaluation of the Tillitis TKey Authentication Ecosystem

Design and Implementation of a Testing Environment for Security Evaluation

Bachelor's thesis in Computer science and engineering

William Bredberg
Maximilian Glantz
Erik Hakeröd
Elias Johansson
Hannah Larsson
Zuzanna Tarkowska

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

Bachelor's thesis 2026

Experimental Security Evaluation of the Tillitis TKey Authentication Ecosystem

Design and Implementation of a Testing Environment for Security Evaluation

William Bredberg
Maximilian Glantz
Erik Hakeröd
Elias Johansson
Hannah Larsson
Zuzanna Tarkowska

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Design and Implementation of a Testing Environment for Security Evaluation
William Bredberg Maximilian Glantz Erik Hakeröd Elias Johansson Hannah Larsson Zuzanna Tarkowska

Supervisor (Handledare): Yasir Hussain, Department of Computer Science and Engineering
Examiners: Patrik Jansson, Department of Computer Science and Engineering
Graded by teacher (medrättande lärare): Irum Inayat, Department of Computer Science and Engineering

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

Abstract

This thesis evaluates the security of the Tillitis TKey authentication ecosystem through simulated attack scenarios and experimental security analysis. Hardware-based authentication tokens such as the TKey are an important part of passwordless systems, which aim to replace traditional passwords and improve security.

To perform the evaluation, a controlled test environment was developed using virtual machines. Several representative attack scenarios were implemented, including person-in-the-middle, replay, injection, fuzzing, and nonce-reuse attacks. These attacks were used to evaluate security properties such as confidentiality, integrity, and authentication strength, based on observed system behavior and collected log data.

The results show that the TKey's authentication mechanism appears resistant to replay and nonce reuse attacks under the tested conditions, and no direct compromise was achieved through injection or fuzzing. However, the person-in-the-middle attack showed that unencrypted communication channels expose authentication metadata and protocol information to interception.

The findings highlight that the security of the TKey ecosystem depends not only on the hardware token itself, but also on the security of the surrounding communication and application layers. This thesis contributes a reproducible testing environment and a practical approach for evaluating authentication system security.

Sammandrag

Detta examensarbete utvärderar säkerheten i Tillitis TKey-autentiseringssystemet genom simulerade attacker och experimentell säkerhetsanalys. Hårdvarubaserade autentiseringsenheter, såsom TKey, är en viktig del av lösenordsfria system som syftar till att ersätta traditionella lösenord och förbättra säkerheten.

För att genomföra utvärderingen utvecklades en kontrollerad testmiljö med hjälp av virtuella maskiner. Flera representativa attacker implementerades, inklusive person-in-the-middle, replay, injection, fuzzing och nonce reuse. Dessa attacker användes för att analysera säkerhetsegenskaper såsom konfidentialitet, integritet och autentiseringsstyrka baserat på observerat systembeteende och insamlad loggdata.

Resultaten visar att TKey-mekanismen framstår som motståndskraftig mot replay- och nonce reuse attacker under de testade förhållandena, och inga direkta intrång uppnåddes genom injektion eller fuzzing. Däremot visade person-in-the-middle-attacken att okrypterad kommunikation exponerar autentiseringsdata och protokollinformation.

Vidare indikerar resultaten att säkerheten i TKey-ekosystemet inte enbart beror på hårdvaran, utan även på säkerheten i de omgivande kommunikations- och applikationslagren. Arbetet bidrar med en reproducerbar testmiljö och en metod för praktisk säkerhetsutvärdering.

Acknowledgements

We would like to express our gratitude to Tillitis for providing the Bellatrix TKeys and supporting this project. We also thank our supervisor, Yasir Hussain, for his support throughout the work. Finally, we thank Chalmers University of Technology for providing the academic environment and resources necessary to conduct this project.

William Bredberg, Maximilian Glantz, Erik Hakeröd, Hannah Larsson, Zuzanna Tarkowska, Gothenburg, May 2026

Contents

List of Figures

List of Tables

Glossary

Attack Surface: All the possible ways an attacker could interact with or attempt to exploit a system.

Authentication: The process of verifying the identity of a user or system

Authentication Flow: The sequence of steps that occur during a login or authentication process.

Authorization: The process of determining what an authenticated user is allowed to access or do within a system

Client / Server / Proxy: Client: the user's device making a request.
Server: the system providing a service.
Proxy: an intermediary that forwards communication between client and server.

Encryption: The process of converting data into a form that cannot be read without the correct key.

Hash: A way of turning data into a fixed-length string of numbers and letters. Even a small change in the input creates a completely different result, so it is useful for checking if data has been changed.

HTTP / HTTPS: HTTP is an unprotected web protocol, while HTTPS is a secure version using encryption.

Packet: A small unit of data sent over a network.

Plaintext: Data that is not encrypted and can be read directly.

Proof-of-Concept (PoC): A simple demonstration showing that a vulnerability or attack works in practice.

Pseudo-Random Number Generator (PRNG): An algorithm that generates numbers that appear random but are actually deterministic.

Relay server: A server that acts as an intermediate node that passes data between two or more parties

Side-channel attack: An attack that gains information from indirect sources, such as timing or power consumption, instead of exploiting software vulnerabilities

SQL: Structured Query Language, a language used to interact with databases.

Threat Model: A description of possible attackers, threats, and ways a system can be attacked.

TLS (Transport Layer Security): A protocol that protects data sent over a network by encrypting it.

1
Introduction

As digital services become more embedded in everyday life and critical sectors, secure login and identity management have become critical security requirements. Authentication is often the first barrier against unauthorized access, and weaknesses can lead to high-impact outcomes such as account takeover, data breaches, and fraud. Password-based authentication remains common, despite it being widely recognized as vulnerable to a multitude of attacks. In addition, human factors such as password reuse and weak password choices further decrease their security [1]. These challenges have contributed to the growing interest in passwordless authentication.

1.1 Background

Passwordless authentication refers to schemes where identity verification does not rely on a traditional password that users must remember and that can be reused or intercepted. Instead, authentication can be based on other factors, such as biometrics or cryptographic keys stored in a token (e.g., "something you are" or "something you have") [1]. In practice, a passwordless solution may still require a local PIN or biometric unlock to enable the device, but the core point is that the account is not solely protected by a reusable server-side password in the traditional sense.

Hardware-backed security keys and tokens are a prominent category within passwordless authentication. They can reduce exposure of long-term secrets to general-purpose software and provide stronger protection against common attack paths that exploit user behavior [1]. However, an important systems security insight remains, cryptographic strength alone does not guarantee end-to-end security. The practical security of an authentication solution depends on the full chain, including the client environment, the communication channel, the protocol logic, and how errors and unexpected inputs are handled.

This bachelor project evaluates the Tillitis TKey, an open-source hardware security token used together with client software and a broader authentication ecosystem [2]. The work is motivated by the need to understand how robust the client-token interaction is under realistic hostile conditions. The focus is explicitly on communication- and protocol-level security in the authentication flow, rather than physical token theft, hardware reverse engineering, or side-channel attacks. Since such attacks require specialized hardware analysis techniques and fall outside the scope of this study. Instead, this work focuses on software- and communication-level threats that can be evaluated through controlled experimental environments. This aligns with Tillitis' own threat modeling, which distinguishes host/interface-driven

software attacks from physical and electrical attack classes [3]. In addition, Tillitis' security reporting illustrates how vulnerabilities can arise at the interface between client software and device applications. This further reinforces that implementation and protocol behavior can become attack surfaces even when the cryptography is working [4].

The findings of this study are relevant to developers, maintainers and organizations, while also contributing to the academic understanding of the field. For developers and maintainers, an experimental evaluation can reveal fragile assumptions, clarify trust boundaries, and motivate concrete countermeasures in protocol design and validation. For organizations considering deployment, the results can provide a basis for deployment decisions by clarifying which security properties hold under realistic channel threats.

Academically, the project contributes a reproducible security evaluation approach that connects threat modeling with controlled experiments. It also strengthens the general argument that authentication security is a system property, the entire chain must withstand realistic attacks, not only the cryptographic part [1].

From a system perspective, the TKey does not operate in isolation, but as part of a larger authentication ecosystem that includes the client application, communication channel, and backend verification logic. Each of these components introduces potential attack surfaces where an adversary may intercept, modify, or inject data. In particular, communication interfaces and protocol implementations represent critical points of exposure, as they handle authentication messages between trusted components.

To investigate these aspects, this project adopts an experimental evaluation approach based on simulated attack scenarios. By constructing a controlled testing environment, it becomes possible to analyze system behavior under realistic threat conditions. Also identify potential weaknesses in the interaction between the TKey and surrounding components.

1.2 Aim

This bachelor's thesis aims to evaluate the security of the Tillitis TKey authentication ecosystem, with a focus on communication and protocol-level threats. The intent is to assess how the system behaves under simulated attack scenarios and to identify potential weaknesses in the interaction between the TKey and its surrounding components.

1.3 Objectives

To achieve the aim of the project, the work is divided into several structured objectives that together enable both experimental and theoretical security analysis of the TKey ecosystem.

- **Threat modeling:** Identify and analyze potential threats, attack surfaces, assets, and trust boundaries within the TKey ecosystem in order to establish a structured understanding of relevant security risks and realistic attack scenarios to be evaluated throughout the project.
- **Controlled environment development:** Develop a simplified and controlled authentication environment that enables secure testing, monitoring, and analysis of interactions within the TKey ecosystem under reproducible conditions.
- **Implementation of automated attack simulations:** Using the developed environment, construct automated and reproducible attacks and implement them in the environment to evaluate and analyze how the TKey behaves under abnormal or malicious conditions.
- **Security evaluation:** Collect and analyze communication data, logs, and system responses generated during the attack simulations in order to identify potential vulnerabilities and assess associated security risks.
- **Countermeasure formulation:** Propose mitigation strategies and security improvements based on the identified vulnerabilities and observed system behavior.

Together, these objectives enable a holistic evaluation of the TKey ecosystem by combining practical experimentation with theoretical security analysis.

1.4 Scope and Delimitations

The project reconstructed a simple authentication application using the Bellatrix TKey model, along with existing open-source TKey-libraries. Vulnerabilities when using hardware-based authentication tokens can be found in different layers of the authentication process such as application, transport, network, data-link, and physical. The evaluation mainly focused on the communication and authentication processes between the host application and the TKey.

Hardware redesign and low-level hardware modification were considered outside the scope of this project. This delimited the project to experimental and simulation-based testing. Furthermore, hardware-level attacks, such as side-channel attacks, are excluded to maintain focus on the logical security of the communication protocol and the interaction between the host and the device.

It should also be taken into account that this project relied on Tillitis documentation, publicly available software tools, and development kits. Additionally, the project focused only on the TKey authentication ecosystem and did not compare the device with other hardware security tokens.

This evaluation prioritized demonstrating vulnerabilities and security behaviors rather than a complete exhaustive security certification. The final report includes documentation, vulnerability analysis, and recommendations for improving system security.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the theoretical foundations relevant to authentication systems, hardware security tokens, and security evaluation concepts. Chapter 3 describes the methodology used to design the experimental environment, simulate attacks and evaluate potential vulnerabilities. This is also complemented by the ethical considerations of said implementations. Chapter 4 presents the results obtained from the experiments conducted. Chapter 5 discusses the findings, including limitations, and implications for the security of the TKey ecosystem, it also concludes the thesis and outlines potential directions for future work.

2
Theoretical Foundations

This chapter provides a theoretical framework necessary to understand the mechanisms of modern security and the security evaluations performed in this study. Firstly, the chapter explains the core principles of authentication. It continues with exploring the transition from knowledge-based methods to passwordless systems relying on public-key cryptography and challenge-response protocols. Then, the hardware components required for secure key storage and the Root of Trust concept are explained. Finally, it outlines common attack vectors and methods used for security evaluation, providing the basis for the security evaluation of this thesis.

2.1 Authentication Systems

Authentication is a fundamental security mechanism used to verify identities before granting access to protected resources. In computer and network security, authentication serves as a fundamental component in identity and access management. This ensures that the user, device, or process is who it claims to be [5]. Reliable authentication is essential for protecting digital systems because it establishes initial trust between a user and a service before any authorization decisions are made.

2.1.1 Authentication Factors

There are different types of authentication mechanisms. They are commonly categorized according to the type of evidence used to verify identity. These categories are known as authentication factors and are traditionally grouped into three categories: knowledge factors ("Something you know"), possession factors ("Something you have"), and inherence factors ("Something you are") [5].

Knowledge-based authentication is based on what users know, such as passwords, personal identification numbers (PINs), or passphrases. This has been the most widely used authentication method due to its simplicity and low cost implementations.

Possession-based authentication is based on something a user has, such as a physical object, a smart card, or a hardware security token. In these systems, authentication of identity is achieved when the user is in possession of the device. An example is generating a one time code or using a stored key.

Inherence-based authentication uses unique biometrics from the user, such as fingerprints, iris, or facial features. These methods, which aim to identify a user based on physical traits, are becoming popular today due to their convenience.

These three factors provide different types of security that are valid for different purposes. Due to the differences, combining these categories has become a common practice to provide better security.

2.1.2 Multi-Factor Authentication and Passwordless Authentication

To strengthen security, many modern authentication systems combine multiple authentication factors. This is known as multi-factor authentication (MFA). An MFA system is required to use at least two of the three authentication factors. This significantly raises the bar for attackers, as breaking one authentication no longer gives you access [5].

Recent authentication architectures are increasingly adopting passwordless approaches to reduce the reliance on reusable shared secrets. These approaches aim to replace the password with cryptographic credentials and hardware-backed authentication, because the knowledge factor is usually considered the weakest link [5],[6].

To counteract the vulnerabilities in password systems, passwordless authentication has become a more secure alternative. Instead of relying on users to memorize passwords, security is about cryptographic proofs with key pairs.

2.1.3 Digital Signatures and Public-Key Authentication

Asymmetric cryptography is used to ensure the authenticity and integrity of a message, through the use of digital signatures [7]. Encrypting with a recipient's public key ensures confidentiality. To prove the origin of the message, a digital signature is used. The digital signature is created by the sender using its private key.

As illustrated in Figure 2.1, the transfer begins with the sender generating a cryptographic hash of the data. The hash is then encrypted with the sender's private key to create the signature. To verify the message, the recipient decrypts the hash with the senders public key. The decrypted hash is then compared with a newly generated hash of the received data, a match confirms that the message is authentic and has not been altered [7].

2.1.4 Challenge-response

An important part of passwordless authentication is the challenge-response protocol. Unlike a password-based system with a shared secret, challenge-response allows the user to prove their identity by demonstrating the possession of a private key without revealing it [7].

This process usually uses a number used once (nonce) [7]. A verifier sends a random challenge to a claimant. The user then performs a cryptographic operation on the challenge, signing it with its private key. This operation is then sent back to the server as a response, which the server verifies using the corresponding public key. This helps mitigate replay attacks, as the challenge is unique for each authentication attempt.

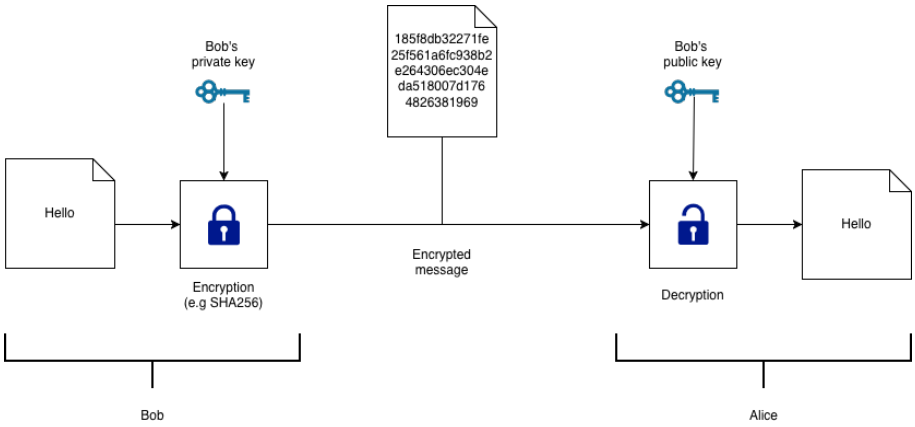


Figure 2.1: Digital signature generation and verification process (Own image, adapted from Stallings [7])

2.2 Hardware Security

Although software-based security provides a necessary layer of protection, it is inherently limited to the underlying security in the operating system and hardware. Hardware security focuses on physical components isolated from the main processes. In which cryptographic functions are calculated and data are stored, such as the Tillitis TKey.

2.2.1 Hardware Security Tokens vs Software Security Tokens

Security tokens are used as possession based authentication factors ("something you have"). They can be represented as physical or digital objects that the user possesses to authenticate. These tokens can be categorized into hardware and software tokens, depending on how they are stored and processed.

Hardware tokens are physical devices designed to securely generate, store, and use cryptographic keys. For example, smart cards and USB security keys. These tokens provide strong protection due to their resistance to phishing, replay attacks, and malware because the private key never leaves the token [5]. This hardware-level isolation is a critical component in achieving high levels of authenticity of the authenticator [5].

Software tokens instead store cryptographic credentials in the software. Often within mobile applications or operating systems. A common example is time-based one-time password applications (TOTP), such as Microsoft Authenticator and Google Authenticator, where a shared secret is stored on the device and used to generate an authentication code [8]. Software tokens are easy and cheaper to implement than hardware tokens, but generally less secure. That is because they rely on the security of the underlying operating system.

2.2.2 Trusted Execution Environment

A trusted execution environment (TEE) is a secure part of the main processor that ensures that code and data are protected with respect to confidentiality and integrity. TEEs provide an isolated execution separated from the operating system, often referred to as Rich Execution Environment (REE). This is used to prevent many software based attacks.

A widely adopted use of TEE is ARM TrustZone [9]. It partitions system resources into secure and non-secure parts. TEEs ensure strict access control and secure storage that enable sensitive operations, such as secure key storage and authentication, in a protected environment [10]. A modern example of a dedicated and open implementation of these principles is Tillitis' TKey, which achieves similar isolation outside of the main processor.

TEEs significantly improve software security and are a fundamental building block in modern security architecture. However, there are still vulnerabilities, such as side-channel attacks, that can compromise the environment [9].

2.2.3 Hardware Root of Trust

Secure systems typically rely on a hardware root of trust to establish trusted execution and secure boot processes. Establish trust by performing security functions, such as secure boot and secure key storage. The main idea is that hardware rooted security is much harder to bypass than software [5].

A common implementation of this is the trusted platform module (TPM) which acts as a black box for cryptographic functions [11]. One of the most important parts of a TPM is the root of trust for measurement (RTM). This measures each piece of code between the OS and the BIOS and stores it in Platform Configuration Registers (PCRs) during boot. If a rootkit modifies the bootloader, the hash will not match and the system can refuse to release the keys needed to decrypt the drive.

While TEEs provide a secure place to run code, the hardware root of trust ensures the system is authentic from the press of the start button. By combining TEEs and hardware root of trust, modern systems create "a chain of trust" that is hard to break via software alone.

2.3 Tillitis TKey

The Tillitis TKey is a hardware token designed to provide a secure environment for running sensitive applications. Instead of permanently storing applications, the TKey loads an application into temporary memory each time it is connected. The device can perform several cryptographic and authentication related functions, including secure random number generation, encryption, time-based one-time password generation and digital signatures [12]. In addition, the device contains a physical touch sensor that requires direct user interaction before performing operations. This creates an additional verification step tied to MFA, since authentication depends both on possession of the TKey and the user actively confirming the operation.

A central security mechanism in the TKey is the compound device identifier

(CDI). Before a device application is executed, the TKey measures the loaded application using a hash. The CDI is then derived from the unique device secret (UDS), the hash of the device application, and optionally a user supplied secret (USS). Since the UDS differ with each TKey, the same application on different devices will result in a unique CDI. Likewise, if the application is modified in any way, the resulting CDI changes [12]. This measured boot process ensures the integrity of the application and allows cryptographic keys to be securely derived from both the UDS and optionally USS, strengthening resistance to manipulation and unauthorized duplication.

2.4 Common Attacks Against Authentication Systems

No authentication system is immune to exploits. Understanding different threats and how they are executed is crucial for designing and evaluating secure systems. Attackers often target communication channels or stored credentials. A valid cryptographic attack can range from simply intercepting data to changing content and locking the authentic user out.

2.4.1 Person-In-The-Middle

Person-in-the-middle (PITM), also known as Man-in-the-middle attacks, are where an attacker places themselves in the middle of two unknowing parties. It is used to collect confidential information, such as credit card information or login credentials. This information is then used to commit other cyber crimes, such as unauthorized purchases or identity theft [7].

For an attacker to successfully perform a PITM attack, it must first exploit vulnerabilities in network protocols. These can include spoofing attacks, such as ARP spoofing to misdirect packets or false Wi-Fi access points. A critical part of the attack is SSL-stripping, where the attacker downgrades a https domain to an unencrypted http domain. This allows the attacker to read the previously encrypted message in plain text [7].

To prevent PITM attacks, modern systems use multi-layered strategies such as MFA and end-to-end encryption. This ensures that even if the network layer is compromised, the data is still secure [7].

2.4.2 Replay

Replay attacks are where an unauthorized user captures or delays valid data transmissions. By re-transmitting intercepted traffic, the attacker can impersonate the user to access locked systems. Because the data was originally valid, older authentication systems might not detect the intrusion [7].

For a successful replay attack, the attacker usually builds on a PITM attack where the attacker captures data and can replay it. A typical attack can be that the attacker is using PITM to listen to a conversation. When a user tries to log into a

server with a password, the attacker intercepts it. Then they do not need to know the password, just resend the same hashed password to the server to trick the server into believing that they are authorized [7].

To prevent replay attacks, each encrypted component is marked with a session ID and a component number. This works because a unique session ID is created each time a message is sent, preventing the attacker from reusing an already sent and delivered package. There are also other solutions, such as one-time passwords or timestamps [7].

2.4.3 Nonce Reuse

Nonce reuse attacks exploit reuse of a nonce with the same encryption key. If an application has faulty implementation of a nonce handling and generation, duplicate nonce may be sent. Attackers may utilize the faulty nonce handling to try and get an overlap of a nonce to gain authentication and get authentication [13]. In some protocols, nonce reuse may also facilitate replay attacks by allowing previously transmitted messages to be resent and accepted as valid [14].

For a nonce reuse attack to succeed, the same nonce must be reused with the same encryption key. If the system contains occurrences of duplicate nonce and an attacker may utilize this to gain authentication. Reusing a nonce-key pair violates the security assumptions of many authenticated encryption schemes [13].

In order to achieve proper nonce handling and generation, the system can implement counter-based nonce generation, time-based session tokens and/or challenge-response mechanisms that provide freshness guarantees [14].

2.4.4 Injection

Injection attacks are where the attacker is making malicious inputs to a system. That input is then processed as part of the command or query. This can be used to bypass login systems with SQL-injection or manipulate system behavior with malicious input [7].

To implement an injection attack, the attacker needs to take advantage of the trust of the inputs in the systems. The attacker tries to identify the input fields with weak boundary checks. When a weakness is found, the attacker will try to exploit it with inputs such as an SQL statement that bypasses authentication. If the login is bypassed, they have access to the system [7].

To prevent these vulnerabilities, developers must implement strict input validation. They can use parameterized queries and enforce rigorous boundary checks to ensure that the input is treated as data rather than as executable code. In addition, make the system react if the input is different than expected [7].

2.4.5 Fuzzing

Fuzzing is an automated software testing technique that allows the program to be tested on unexpected or malformed inputs. It is commonly used as a complement to the code review, as it can find other vulnerabilities [15].

To perform a fuzzing attack, the attacker usually uses a fuzzing tool. The tool systematically generates data, such as changing bits or extremely large strings, and then tests them to see if the system can handle it. If the fuzzing tool finds something that the system cannot handle, then the attacker can infiltrate the system [15].

To prevent fuzzing attacks, developers should have a high code standard and continue testing during development. This includes strict input validation and secure programming languages [7].

2.5 Vulnerability Assessment and Penetration Testing

Identifying weaknesses in a system requires a systematic approach to testing and measurement. Vulnerability assessment and penetration testing (VAPT) are methodologies used to detect, validate, and categorize security flaws before an attacker. By simulating the techniques used by malicious actors within a controlled and ethical framework, organizations can understand their security posture.

2.5.1 Ethical Hacking

Ethical hacking is a proactive security measure in which authorized people attempt to find vulnerabilities in systems before they can be exploited by malicious actors. They are governed by a strict legal and ethical framework to prevent abuse of power or possible vulnerabilities from being leaked. To simulate a realistic attack cycle, the process typically follows three phases: reconnaissance, staging the attack, and reporting [16].

2.5.2 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) is a standardized framework to assess security numerically from 0-10, based on different metrics such as exploitability metrics, vulnerability metrics, environmental metrics, or supplemental metrics. The purpose of this scoring is to characterize the vulnerabilities and to produce a score that reflects their severity. The scoring can help organize different weaknesses and prioritize the response to then manage each and one in order [17].

3
Method

This chapter describes the methodology for the project. It details the process and the rationale for the methods used in each section. The chapter first presents the societal and ethical considerations before structuring the method, then followed by an overview of the project. The sections that follows presents the authentication setup. It then describes the structure for the test environment, how the security analysis was performed and the structure for the attacks. Finally, the method for data collection, evaluation of the attack results and the analysis of identified vulnerabilities is presented.

3.1 Methodology

To evaluate the security properties of the Tillitis Tkey ecosystem and assess the security of the Tillitis TKey ecosystem, the project followed an experimental research approach based on attacks specified further down in this chapter. First an authentication setup was created that communicates over the network. With the program setup it needed to be implemented in an environment that will give consistent data. The next step was an iterative process where the attacks were implemented. First iteration of attacks were based on research done on the TKey and its associated libraries. Following iterations used the data from each attack, which was analyzed for possible vulnerabilities and based on that analysis the new attacks were created. The collected results were analyzed using a risk assessment framework and scored using CVSS. Based on these findings, mitigation strategies were proposed.

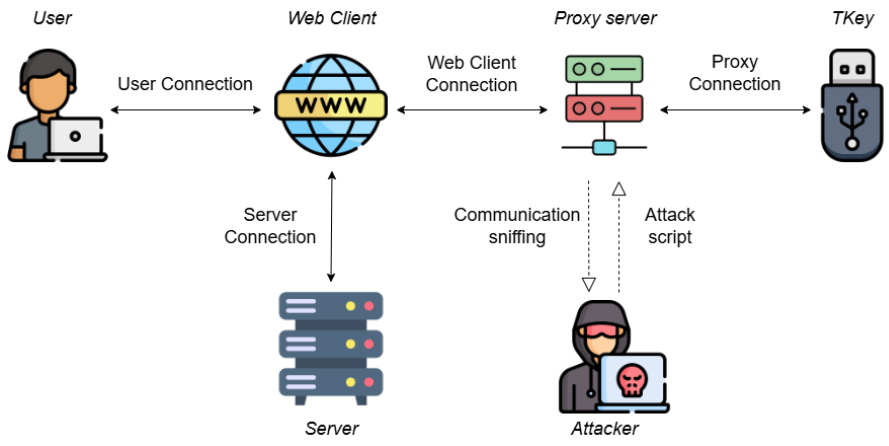


Figure 3.1: Image of constructed ecosystem

The project was conducted using an iterative development methodology with Kanban-based task management. Version control and CI/CD practices were used to ensure code consistency, quality, and traceability throughout development.

3.2 Societal and Ethical Considerations

The purpose of this ethical analysis is to evaluate the projects societal and ethical implications, formulate an ethically based approach to minimize any potential misuse or unintended security consequences.

An evaluation of any security based hard- or software is generally a necessity for its reliability, the moral constraints lie heavily in the implementation and result following of said implementation. The ethical risk while exposing flaws in safety is the possibility of exposing potential vulnerabilities that can later be exploited by malicious third parties, which makes our project an example of security-relevant (dual-use) research.

Research-ethics discussions on the topic highlight the need for analysis on risk awareness, stakeholder impact, and mitigation strategies such as controlled experimentation and careful communication of results. In other words, security-relevant research requires awareness of dual use risks [18]. Thus this project has a responsibility to mitigate the dual-use risk as much as possible. Furthermore if the final report comes to the conclusion that the Tkey is in any way vulnerable or untrustworthy, it might damage the reputation of Tillitis which in turn could also affect the company economically.

On the contrary, finding possible ways for the Tkey to be exploited raises necessary awareness for the existence of weaknesses, making it possible to patch them and increase the security of the token as well as the credibility of Tillitis.

Doing the assessment through penetration testing may be perceived as ethically problematic, as it involves deliberately attempting to breach security of the system which is being evaluated. However, research demonstrates that ethical hacking is very effective in identifying vulnerabilities and contributes to proactively strengthening a systems security. Penetration testing done in an informed and authorized way does not increase risk, it reduces the probability of exploitation by malicious third parties[19]. From an ethical standpoint, this contributes to harm minimization and preservation of system integrity. Neglecting security evaluation may lave vulnerabilities undiscovered, potentially increasing security risk for end users This would mean lack of accountability for the safety of the end users.

Another important aspect in regards of using attack simulation for the project, is the fact that these means are authorized by Tillitis themselves. Moreover, the fact that the Tkey is an open-source project suggests that penetration testing is encouraged for the sake of evaluating and improving the safety of the token. Open-source projects can also be used as reference for other developers on how security of this kind can be implemented, improving overall knowledge and research beyond Tillitis, which this project would be a contributor to.

This means that penetration testing is necessary for projects of this nature, but it is essential to carry them out in a proper and ethical manner. The method of this project must ensure that research is done responsibly to prevent misuse.

Allowing misuse could lead to a breach in safety and integrity of the end users or organizations using the Tkey, as well as reputation/economical harm to Tillitis. The methodological design therefore plays a critical role in ensuring responsible experimentation. The attack simulations must be done in a controlled, isolated environment that is detached from the official code so that no attacks are deployed in a way that interacts with the end users. Implementation and documentation must be kept private to ensure no findings are at risk of exploitation. Documentation will also be kept to a general description, there must be no actionable explanations on how to execute the attacks before publication. This also helps the project follow a responsible and coordinated vulnerability disclosure (CVD), which is a model on how disclosure should be handled.

With this methodology any negative outcomes can be avoided and make sure the project results in outcomes that are beneficial and ethical. Such include improved safety for end users/organizations, insights in potential vulnerabilities for Tillitis to address and research on security practices through the final report. In the event that we are unable to find any vulnerable attack surfaces, we can still make the conclusion that Tillitis provides a robust authentication key. Meaning that either outcome is a valuable one and the resources put into the project are not wasted, rather the promise of safety for the end users is enforced. In addition such conclusion can still be of value within research or educational purposes.

3.3 Threat Model

This threat model was constructed to create a structured approach to detect any vulnerabilities or threats to the Tkey ecosystem. This provided a systematic analysis of the security in the system by identifying critical assets, potential threats, and mitigation strategies for protecting the system.

3.3.1 Assets

The primary assets considered for this evaluation are cryptographic keys, authentication credentials, integrity of the authentication process, firmware memory and preloaded apps. Protecting these assets from being compromised is critical to guarantee no loss in integrity, avoid unauthorized access or prevent execution of malicious code within the TKey system.

- Private Key – The secret key stored inside the TKey and should never be exposed.
- uds – Unique device secret. Provided during Tkey provisioning and should never be replaced. Used for application secrets.
- cdi – Compound device identity. Computed when loaded with app using. uds, uss and hash of device binary. used for application secrets.
- uss – User supplied secret. It is Provided by the user and should be kept secret.
- udi – Unique device id. Its provided during TKey provisioning and replaced.
- signature – Returned from the TKey when signing a nonce.
- preloaded apps – Used to provide the tkey with functionality.

- firmware memory – Memory holding the firmware inside the TKey.

3.3.2 Threat Actors

The threat model assumes an external remote attacker capable of intercepting and manipulating network communication. Their intention is to abuse the authentication process between the Tkey and the host application, alternatively find existing exploitable weaknesses in the Tkey ecosystem. The attacker is assumed to have no physical access to the Tkey, hence no ability to access the hardware of the token. It is assumed that the attacker has no privileges within the system, no resources beyond a personal computer, and operates within the network data flow.

3.3.3 Threat Vector

Two main threat vectors are defined. The first threat vector is Software-based attacks over the network communication. These attacks are performed over http. They target either the communication between the web client and the proxy for the TKey or between web client and the server. Second type of vector targets the host-device, where the attacker has gained control over the computer. The attacker can exploit the TKey directly through the USB serial interface.

3.3.4 Risk Assessment

To assess the risk of an attack a risk assessment strategy has been implemented. This risk is assigned as a product of $likelihood \times impact$, visualized in the matrix seen in Figure 3.2. Likelihood describes how likely it is that the attack occurs, while impact refers to the threat posed on the ecosystem if the breach succeeds. Depending on the risk severity different mitigation responses can be implemented as described further down in this chapter. The severity ranges from low to high with five levels total and contributes to a proper prioritization of action when securing the ecosystem. This helps allocating resources and present the changes needed to non technical stakeholders with a easy to understand visual. The comprehensive overview also contributes to proactive work by bringing awareness to even lower-severity risk cases.

3.3.5 Vulnerability Assessment

We started by identifying vulnerabilities, each vulnerability identified in the previous stage was analyzed based on our environment. On a successful attack, it was scored via CVSS. Based on the technical characteristics of each vulnerability, values were assigned for the different parameters in the CVSS model. This was done to create an objective assessment. All identified weaknesses were then categorized, where the resulting scores were used to rank the vulnerabilities from 0 to 10.

This method was chosen to provide a transparent and repeatable analysis, ensuring that the study's conclusions are based on an established industry standard. The concrete results and specific scores for each threat are reported.

Likelihood \ Impact	Negligible	Minor	Moderate	Significant	Severe
Very likely	Medium low	Medium	Medium high	High	High
Likely	Low	Medium low	Medium	Medium high	High
Possible	Low	Medium low	Medium	Medium high	Medium high
Unlikely	Low	Medium low	Medium low	Medium	Medium high
Very unlikely	Low	Low	Medium low	Medium	Medium

Figure 3.2: Risk assessment matrix used to calculate the risk as a product of likelihood and impact of an attack.

3.3.6 Mitigation Strategies

The mitigation strategy is derived from the risk matrix, meaning the approach is based on the severity of the risk as seen in Figure 3.3. The color of the risk matrix corresponds to the color in the mitigation scale. This is the starting point for approaching the vulnerability. In the case of a strategy not being applicable or appropriate to a specific risk it moves over one step to the right on the scale.

For the lowest risk levels no action is strictly required, being aware of the risk is enough. In the case of a somewhat larger risk a plan for action should be prepared in the case of occurrence. Changes should be made to make sure the negative outcomes in a potential breach are not as severe, once the risk goes beyond the green part of the scale. When approaching the right side of the scale the impact or likelihood should be reduced to bring the attack back down within the risk matrix. In the event of reaching the critical-severity, affected functionalities should be disabled until the issue is resolved to avoid major negative consequences on the ecosystem.

Mitigation strategies

Risk acceptance	Contingency planning	Risk mitigation	Risk reduction	Risk avoidance
Acknowledge the risk but no action is necessary	Prepare for potential risk events	Reduce negative effects of risk	Reduce likelihood or impact	Removal of vulnerable functionality

Figure 3.3: Mitigation strategy scale showcasing how attacks should be approached depending on the risk they pose to the ecosystem.

3.4 Test Environment

For designing the test environment the qualifications were reproducibility, isolation, controlled environment, and highly automatable. The environment that fulfilled these criteria best was a setup that used three virtual machines. By using VMs we create an isolated environment that ensures all users have identical setups. To automate this environment as much as possible docker-compose is used to download the tools and setup necessary elements for the VMs. Having an isolated environment makes it so that experimentation and potentially malicious attacks can be executed safely without exposing any vulnerabilities.

3.4.1 Authentication Setup

The authentication setup is an application programmed in Go. Go was selected due to the availability of official Tillitis libraries and its suitability for networked applications. The application uses a web client and a TKey client that communicate through a relay server with some proxy functionality. This made it so that messages could be intercepted in two places: between TKey client and server, and between web client and server. The TKey takes the USS that is stored with the user, generates a public key and then stores the public key with the user. The program was built according to the flowchart seen in Figure 3.4 and simulates logging in to the web client by sending a request to the server to verify the TKey. The server sends a challenge to the TKey client which request TKey to sign it and returns a signed message to the server. The server verifies the challenge with the public key that the user has stored and returns if it passed verification or not to the web client. When

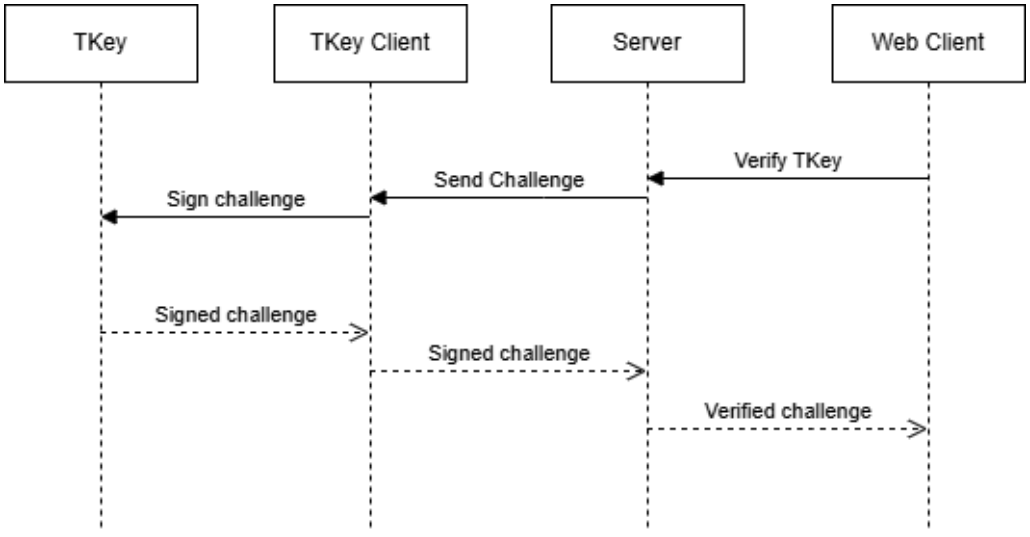


Figure 3.4: Flow chart for logging in using first iteration authentication setup

creating the attacks we discovered that polling was going to be an issue and that it was needlessly complicated. A second, final iteration was created to make it simpler and more modular for implementing new attacks. In the final iteration of the setup it was implemented with two servers as in Figure 3.5, one as a proxy for the TKey

and the other to simulate a regular server for the website. Web client is what the user interacted with that decided what the system should do. We decided on this setup because of the simplified communication using two servers and with the new web client controlling both servers, it made the setup more modular.

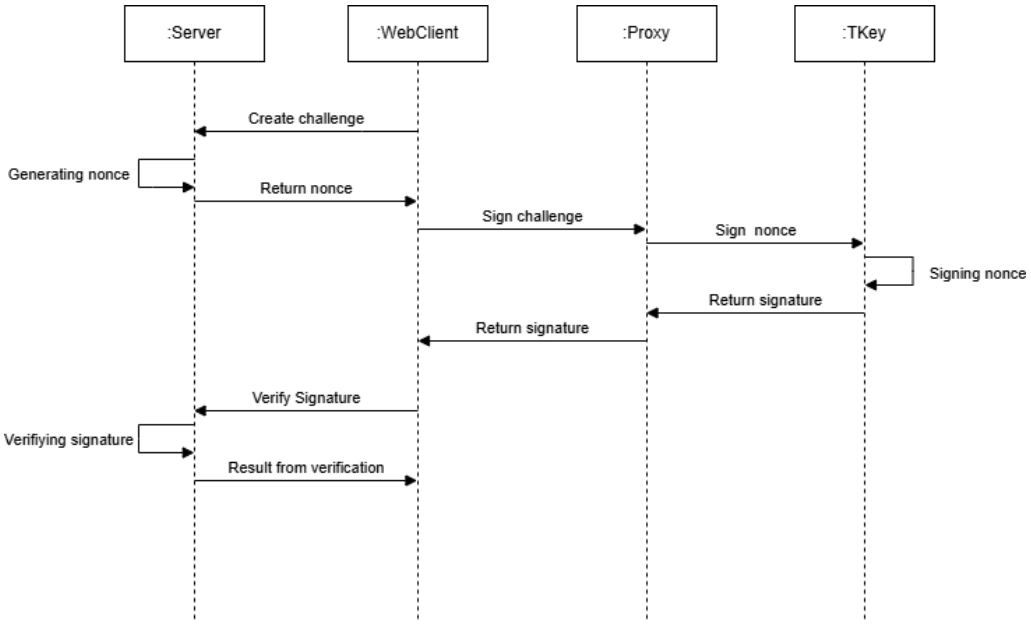


Figure 3.5: Flow chart for logging in using the final iteration of the authentication setup.

For the final iteration of the setup the new way for simulating logging in became: web client request challenge from server, server generates nonce and returns it to the web client, web client sends request to proxy to sign the nonce, proxy tells TKey to sign the nonce and returns the signature to the web client, web client requests the server to verify the signature, the server returns if the signature passed verification, the web client tells the user if it passed verification and managed to log in or if it failed.

3.4.2 Simulating Attacks

The attacks are simulated privately to ensure non-exposure of the vulnerabilities and make sure that potential vulnerabilities are not publicly disclosed during development. This was ensured by using private Github repositories during the simulations and closed documentation during development for potentially exploitative weaknesses.

The attacks are implemented as shell scripts on the VMs. By using shell scripts multiple attacks can be executed on the kali VM with a single terminal command. Kali comes preinstalled with multiple tools for attacking and logging, for example bash.

For the security evaluation and enable continuous analysis from the simulated attacks we needed to log the attacks. The logs were integrated into the attacks using python since the script for the attacks were also in python. To log responses

from the TKey the application generated its own logs using a Go implementation of logs. The attacks log generated an unique id that it sent with http message to the Tkey. The TKey logs registered that id with the TKey log. We did this so that there was a reliable way to connect the TKey logs with the attack logs. The logs were generated into JSON so that the information from the logs could easily be extracted, manipulated and shown in different ways.

We implemented a shell script to be able to view the logs in different ways. This was done so that we could easily analyze the information from the logs and be able to view them in a readable way. An example of the structure and content of the generated log output is provided in Appendix A, illustrating how events are recorded.

3.5 Security Evaluation

Evaluating the security of the Tkey during this project relied heavily on the analysis derived from the attack simulations. This evaluation can be divided into two phases.

The first one being accomplished in parallel with the implementation of simulations, which can be described as a feasibility-based security evaluation. During this phase the security against a certain attack was evaluated by looking at the degree in which said attack could be implemented within the simulation environment.

In the case of an attack not being feasible for implementation under the conditions of this study, the results may suggest that there are effective mitigation mechanisms in place within the Tillitis firmware. Note that this should not be interpreted as proof of security outside the scope of this project.

If no such safeguards were identified during implementation the evaluation moved forward to the second stage, which is the achievable simulation of the attack. This part of the analysis was conducted through observations once the simulations were fully implemented, this is where the actual behavior and response of the Tkey were evaluated when presented with threats. This part of the process is supported by the logs explained in the previous section, and which are showcased in the result section. Depending on the results, the analysis could by this point determine what mitigation mechanisms are present in the Tkey ecosystem or identify lack of security at certain points of the authentication process.

Security depends heavily on the implementation of the system and what constraints and architectural protections are set in place to strengthen security and lessen the attack surface of the authentication ecosystem. This approach allows for a practical evaluation of the system, where risks can also be assessed by reviewing the likelihood and possibility of exploitation. A feasible attack can have a larger chance of breaching security, any constraints making the attack infeasible contribute to minimizing the security breach risk.

The two phases of the analysis could then be translated into a complete evaluation, where a conclusion was made consisting of potential threats posed on the Tkey or alternatively how the Tkey is well protected against certain attacks with the countermeasures we identified.

3.5.1 Attack Selection

When implementing the attacks we started off with a passive Person-in-the-middle-attack as a form of communication interception analysis. This was a way of getting a more comprehensive understanding of what we could consider implementing further depending on the confidentiality of communication, seeing if an unauthorized party would be able to depict or extract something from the messages between components. The vulnerabilities assessed here were lack of encryption or exposure of sensitive data in transit such as keys or commands that could further guide other, more advanced, attacks.

Fuzzing was implemented as a baseline to check if the Tkey had any obvious vulnerabilities or defects, or if there would be any unexpected behavior from unusual inputs. Since the Tkey system is heavily reliant on communication between itself and the host machine, any malformed data let through would have a big impact on its reliability. This attack would give a clearer indication of any input handling weaknesses, lack of validation or defects in parsing.

Many injection attacks could later be built upon the framework of the fuzzing attack, in some cases paired with the potential information gained with the PITM weaknesses. An example of this is a command injection with commands found within the communication layer and Tillitis developers page. Instead of altering the payload, these commands can be injected by using the same basic framework used for nonce manipulation in the fuzzing attack.

The aim behind this attack would be to further expand on assessing the vulnerabilities within input validation, command integrity and separation between data. With this implementation we go further by checking whether the control logic, execution flow or overall hardware behavior can be modified. In short, check if an improper command could affect how the system executes.

A replay attack was implemented to test whether the challenge-response authentication procedure could be exploited and that once verified signatures could be reused. By saving a valid signature and sending it again during a later login attempt, we could check if the system accepts old authentication data. The aim was to see whether each signature is properly tied to a fresh challenge, and if old messages are rejected instead of granting access.

Lastly, we aimed to evaluate the resistance against the authentication mechanism by implementing a nonce reuse attack. This would help us assess the effectiveness of the access control mechanisms and any potential system exposure under repeated attempts. This attack is an expanded version of a replay attack and focuses primarily on the handling and generation of nonce. The nonce reuse attack saves and uses a previously accepted nonce derived from a successful authentication obtained from logs of previously conducted replay attacks. This saved nonce was then used multiple times on a large number of different challenge requests. By performing the attack on a large number of challenges, eventual weaknesses within the authentication algorithm and nonce handling may be exposed.

3.6 Limitations

A project of this nature requires a strong fundamental implementation of attack simulations that are slowly built into more advanced attacks that investigate more complex parts of the system. What was possible to achieve during this project was heavily influenced by the limited time granted for this evaluation. Furthermore a decision had to be made regarding how this time would be distributed between the development of the environment, simulation of a number of attacks as well as the initial research required surrounding the functionality and implementation of the Tkey. The time had to also be evenly distributed to balance the depth and complexity of each attack while also being able to implement several attacks, thereby covering a reasonable surface for a comprehensive evaluation of the Tkey authentication ecosystem. The project is also limited by the previous experience and knowledge of group members surrounding any practical implementation of security evaluations of this nature. Hence, a big portion of this project is part of the learning curve and the project scope as well as extent should be interpreted as such.

3.7 Use of AI

AI has been used in this project to help with code syntax, review code, additional guidance on attack patterns and common attacks, text structure, identifying relevant keywords and search terms. The use of AI has been performed ethically, which means that no work is directly copied without proper consideration of the content at hand and how the dependencies interact.

4
Results

This chapter present the experimental results obtained from the implemented attack simulations and the resulting security evaluation of the TKey ecosystem. The first section presents the simulation environment showing its final structure and functionality. This is followed by the second section which consists of the security evaluation tied to each of the simulated attacks. For each simulation the properties that were tested are described and the observed behavior and security implications of each attack scenario, each evaluation is complemented by a CVSS score to further describe their threat on the Tkey.

Table 4.1: Overview of evaluated security properties

Attack method	Person-in-the-middle	Fuzzing	Injection	Replay	Nonce reuse
Security Property Tested	Confidentiality & Integrity	Input robustness	Input trust boundaries	Freshness	Authentication strength
What is Evaluated	Whether messages between the Tkey and host can be intercepted or altered	Handling of unexpected or malformed data sent to the system	Whether crafted input can manipulate execution logic or commands	Whether previously captured authentication data can be reused to gain access to the application	Resistance to repeated guessing or forged authentication attempts
Success criteria	Communication is captured	Unintended behavior	Executes unintended command	Get a valid verification	Get a valid verification
Outcome	Successful	Unsuccessful	Unsuccessful	Unsuccessful	Unsuccessful
Observed Effect	Retrieved communication logs to/from TKey	TKey crashes or signs the payload	Signs the payload	Failure response from verify	Script was still running
Impacted component	Network communication flow	TKey	TKey	Application, verify	Application, verify

4.1 Test Environment

The developed test environment enabled reproducible attack execution and dynamic switching between attack scenarios during runtime. It enabled implementation of multiple attacks while also making it simple to test the different attacks. The setup with two servers increased modularity of the environment and simplified the implementation of new attacks and only minimal changes to the system were needed. The implemented logging function made it so that the TKeys reaction to an attack could be recorded and reliable correlated to the attack that caused the reaction. It is also easily modifiable and modular for logging the attacks. Using shell script allowed both displaying logs and running attacks to be automated.

The environment setup used three VMs as shown in the Figure 4.1. The program is divided on two VMs so they are forced to communicate over the internet. The network uses a flat isolated topology through the docker bride network. The three VMs have their own network node and are on the same subnet, having static IP addresses. The VMs CPU, GPU, and storage have no explicit set limits, so they are limited to the inherited resources from the host system or if a explicit limit has been set in the docker settings. The client VM and TKey signer VM uses ubuntu 22.04 LTS, while attacker VM is running Kali Linux rolling release.

Each VM has its own container and are configured in privileged mode. They are all loaded with the authentication setup and have a shared volume that allows shared access to logs and files. Client and TKey signer both have /dev/ttyACM0, and /dev/bus/usb devices mounted to enable USB communication. Only TKey signer has extended access to /dev:/dev which allows dynamic finding of the TKey without restarting the containers.

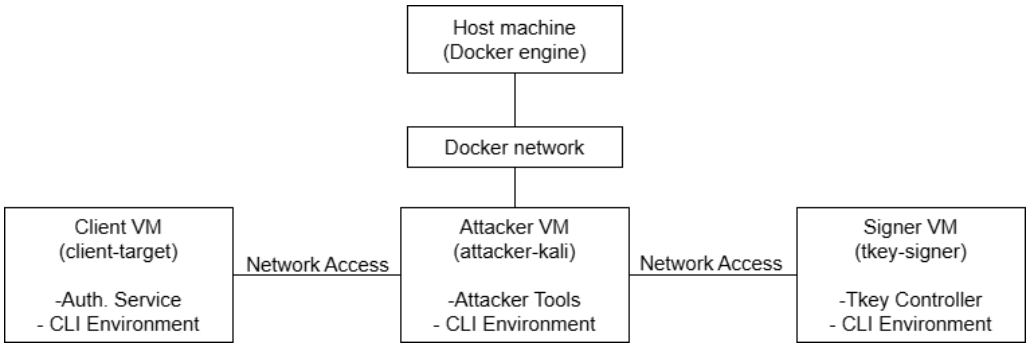


Figure 4.1: Environment chart

4.2 Security Evaluation

This part presents the results of each attack implemented. Most of the attacks did not result in a vulnerability and were not feasible as shown in Table 4.2. The attacks were further evaluated with the risk assessment from the threat model.

Table 4.2: Compact overview of CVSS and risk assessment

Attack method	CVSS	Risk	Feasibility	Feasibility reason
Person-in-the-middle	8.8	Medium	Feasible	Unencrypted communication
Fuzzing	0.0	Low	Infeasible	Cannot be automated
Command injection	0.0	Low	Infeasible	No command interpreter
Replay	0.0	Low	Infeasible	New nonce generated each time
Nonce reuse	0.0	Low	Infeasible	Unrealistic time to compromise

4.2.1 Person-In-The-Middle Attack

The results of the PITM attack show that it successfully intercepted communication between the TKey and the host computer. The device continued to function normally during the attack, and no security warnings were triggered. Because the connection relied on unencrypted HTTP instead of TLS, all captured packets could be read in plaintext.

To improve readability, the captured log was summarized into a chronological table, shown in Table 4.3. The table highlights the most relevant events in the communication, while the complete unfiltered log showing packet examples, extracted fields, interception evidence, and protocol structure are shown in Appendix A.1. This level of access allowed for extraction of application data, including protocol metadata and the hash generated from the nonce and shared secret.

Table 4.3: Chronological summary of the PITM attack log

Step	Time	Event	Description
1	08:10:49.484	HTTP request observed	The proxy observed an HTTP POST request to the `/challenge` endpoint.
2	08:10:49.487	Data logged	The transmitted challenge data was logged by the attacker.
3	08:11:05.808	HTTP request observed	A second HTTP POST request to the `/challenge` endpoint was observed.
4	08:11:05.809	Data logged	The transmitted data from the second request was also logged.

4.2.2 Fuzzing Attack

Three different fuzzing attacks were evaluated targeting the nonce: Null injection, maximum integer injection, and an oversized payload injection consisting of 1 MB zero-valued input. The first attack, null injection resulted in the TKey returning the error nonce length null. This suggests that input-length validation mechanisms are present. The second attack did not affect the TKey adversely. The sign method maintained expected system stability. It signed the maximum integer valued nonce and returned it. Similar to the null injection injecting 1 mb of zeros resulted in a length error.

4.2.3 Command Injection Attack

The Injection attack tried to inject commands through the nonce as shown in Figure 4.2, in an attempt to exploit the TKey framing protocol. The command it tried to send was the byte 0x08 and its goal was for it to be interpreted as a *show cdi* command. A successful command injection would be if the TKey only signed part of the nonce and interpreted the rest of it as a separate framing message

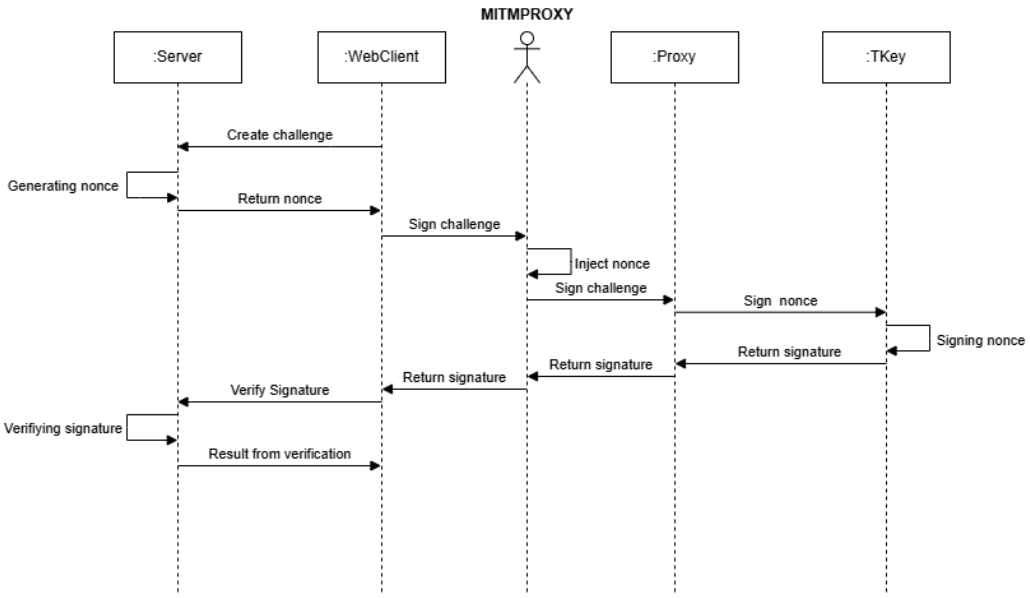


Figure 4.2: Environment flowchart for injection attack

The attack was infeasible, TKeys response to the command injection was to sign the whole nonce. This is because executing the sign function creates a framing message where the function creates the header before it handles the nonce. The framing format consists of a single header byte followed by a command/response byte and lastly data bytes. The header byte specifies the length of data and domain for command/response. Sign handles the payload by dividing the data into chunks and sends it to the TKey using framing message. The protocol only interprets the framing metadata while the message payload is handled as uninterpreted data, so it does not parse the payload in the way that it could interpret commands from it. Packet examples from this attack is shown in Appendix A.6

4.2.4 Replay Attack

The implemented replay attack tested whether previously valid authentication data could be reused to gain access. The attack targets the challenge–response mechanism by capturing a valid signature during a successful login and attempting to reuse it in a later login attempt.

Figure 4.3 shows the sequence of the attack. During the first login, the system behaves normally: the server generates a challenge (nonce), the TKey signs it, and the signature is returned and verified successfully. At this stage, the proxy captures and stores the signature. During the second login attempt, a new challenge is

generated by the server. Instead of forwarding this new challenge for signing, the proxy replaces the outgoing data with the previously captured signature. This simulated an attacker attempting to reuse old authentication data.

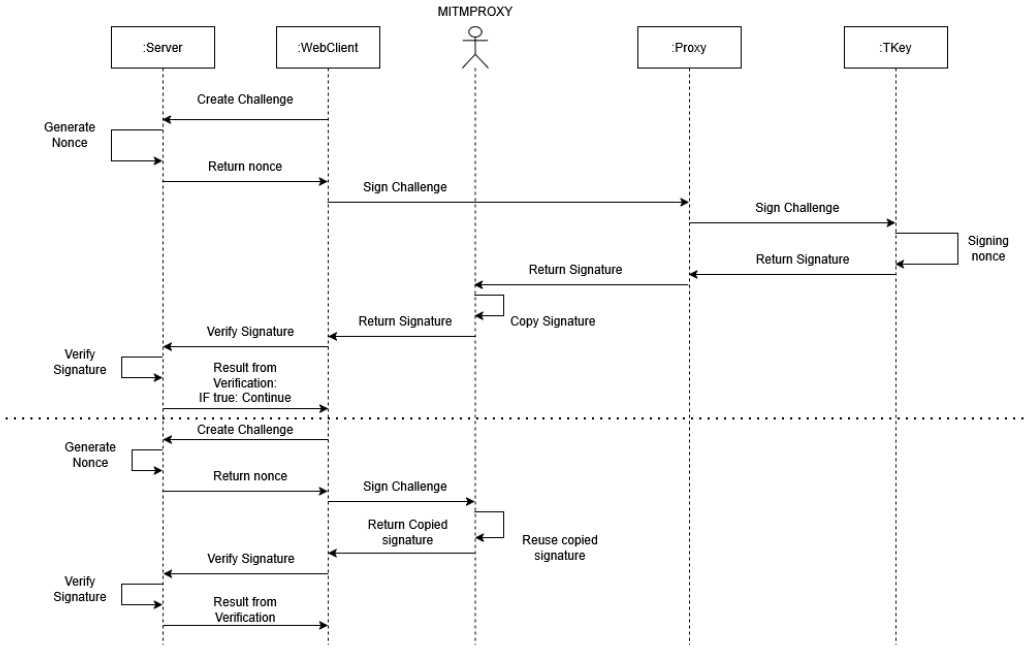


Figure 4.3: Replay attack flowchart

To make the log output easier to interpret, the raw log was reformatted into a chronological table together with the client-side output, where the authentication status is displayed after each request. The table presents the same event sequence as the original log, but only includes the fields that are most relevant for understanding the replay attack. The complete raw log is provided in Appendix A.2, and the corresponding client-side output is shown in Appendix A.3.

The behavior of the attack can be observed in the logs in Table 4.4. The log entries show:

- "Signature copied": the signature from the first successful login is stored
- "Signature replayed": the stored signature is reused in a later request
- repeated replay attempts to confirm consistent behavior

Despite the replay attempts, the authentication fails after the first login. This indicates that the system correctly binds each signature to its corresponding challenge and rejects reused signatures, as seen in Appendix A.3.

4.2.5 Nonce Reuse Attack

To expand on the replay attack and test the robustness of nonce generation and handling, a nonce reuse attack was performed. The Figure 4.4 depicts the flowchart of the nonce reuse attack. The attack started as the second part of the replay attack, but repeated each time it got rejected. This attack is assumed to have at least one verified signature saved from the logs of previously performed replay attacks. The nonce reuse attack then repeats the challenge-response mechanism to try and get the forged signature to verify in computationally feasible time.

Table 4.4: Chronological summary of the replay attack log with verification outcome

Step	Time	Event	Description	Result
1	13:25:11.849	Attack initiated	The replay attack script was started	–
2	13:25:19.237	Request intercepted	A request was captured and assigned an ID	–
3	13:25:21.030	Signature copied	Signature from a valid login was stored	–
4	13:25:24.871	Signature replayed	Stored signature reused in a new request	Rejected
5	13:25:33.370	Signature replayed	Reused signature in a second attempt	Rejected

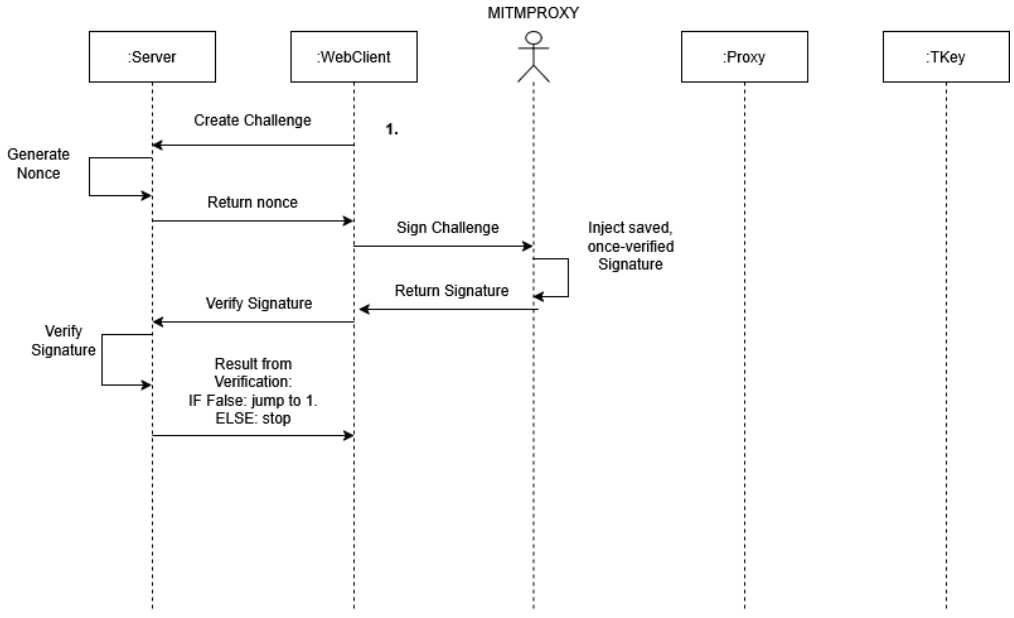


Figure 4.4: Nonce reuse attack flowchart

The table 4.5 shows that during the entire runtime, an accepted request did not occur. These results are based on the logs from the nonce reuse attack. Because the results were over 1.6 million rows, an example of the captured data is given in Appendix A.5, were the forged signature that was used is also presented.

Table 4.5: Chronological summary of the nonce reuse log with verification outcome

Step	**Time**	**Result**
1	11:27:43.909	Rejected
2	11:27:43.916	Rejected
3	11:27:43.925	Rejected
...	...	Rejected
1,664,439	14:03:43.621	Rejected

As seen in table 4.6 the performed nonce reuse attack ran for 2 hours, 32 minutes and made 1664439 challenges, resulting in about $1664439 \div ((2 \times 60) + 32) \simeq 11000$ requests per minute or about 182 requests per second. During this time the nonce reuse attack failed to get the forged signature to validate.

Table 4.6: Summary of the nonce reuse log

Metric	**Value**
Authentication requests	1,664,439
Average requests/sec	182
Runtime	2h 35m 59.712s
successful forgeries	0
Failed Verifications	1,664,439

5
Discussion

This final chapter analyses the contents of the results and discusses security within the TKey ecosystem. Countermeasures are stated on how to improve security within the TKey and future work on security evaluation is proposed. Lastly, the discussion leads to a conclusion that summarizes the thesis and analyze the main purpose of the report and how well the project achieved said purpose.

5.1 Security of Systems

As briefly mentioned in Section 3.5, the security of software that implements authentication is dependent on the weakest link of the entire authentication ecosystem. With each library or component that is used, there is a risk that a vulnerable component is added. If a component of the system becomes compromised it may significantly weaken the overall security posture of the entire system. Therefore, reducing the amount of components and complexity is usually beneficial for keeping software secure.

By design, a TKey cannot function without a structure built around it. The software against which an authentication attempt is made, has to store information for the TKey to verify itself against. This creates three different attack surfaces: the server that stores information, the TKey itself that stores the secret, and the communication channel between the server and the TKey. Understanding these attack vectors is vital for developing secure software.

5.2 Test Environment

To evaluate the security of the TKey ecosystem, the environment was designed to simulate a realistic network-based authentication flow. With that, it introduces certain constraints and security properties that must be taken into account.

The use of virtual machines and a proxy allowed for precise interception of traffic, which was essential for PITM and replay attacks. This approach also limited the scope of attacks from the physical layer. Since the TKey was developed in a Go-based client in a controlled environment, certain hardware attacks, such as timing attacks, were beyond the scope of this simulation.

Another important angle is the distinction between hardware security and application-level security. Even if the TKey hardware is mathematically and physically "perfectly secure", because the private key never leaves the device and the signing process is isolated, the overall system security can still be compromised. It is therefore very

important for applications that implements the TKey as the authenticator to have good security measures to ensure that these attacks, and more, does not result in vulnerabilities within the application.

If the application using the TKey does not properly verify the returned signature or fails to use unique nonce, the effectiveness of the TKey security guarantees may be undermined. The security of the TKey does not automatically encrypt the communication channel between the client and the server. Therefore, the effectiveness of the TKey's security features depends on whether they are sufficient to protect the user when the surrounding software becomes the weakest link.

5.3 Security Evaluation

To make the security evaluation, the attacks performed in this project were chosen by researching high-level attack classes that serve as the basis for numerous other attacks. With this the attacks could be improved later on to target more specific vulnerabilities and delve deeper in exposing eventual weaknesses. Because of the time limit it was not realistic to make more attacks than the five performed in this project. These five attacks are therefore not specifically tailored to target a specific threat vector, but to give a baseline attack framework that can expose weaknesses and later be improved.

The results derived from the attacks performed in this project gives us that the evaluated attack scenarios did not expose compromise of the TKey cryptographic mechanisms, but the application using the TKey can contain weaknesses that can be exploited depending on the implementation of the TKey.

5.3.1 Person-In-The-Middle

From the results of the PITM attack, both the protocol metadata and the nonce hash were successfully extracted in readable text from the intercepted communication, as seen in Appendix A.1. This observation can indicate that the application may be susceptible to replay attacks if the protocol does not use new nonce each challenge, or if the application does not bind the nonce to a session. The communication is not protected by TLS or an equivalent integrity mechanism, which means that there is nothing to prevent an attacker from intercepting a valid response from the TKey and resending it later to authenticate against the host. If the protocol fails to implement strict sequence numbers or short validity windows, the security model may become practically ineffective against PITM scenarios. Under such conditions, replay or session-manipulation attacks could potentially allow an attacker to impersonate a legitimate user without directly compromising the underlying secrets.

Furthermore, a PITM position allows for traffic analysis. This opens up the possibility for further attack implementations. The fact that the device continued to function without security warnings confirms that the system lacks mechanisms to detect the latency introduced by a proxy, making these side-channel attacks difficult to detect.

5.3.2 Fuzzing Attack

Because the TKey worked throughout the attacks, it suggests the implementation includes a relatively robust input validation. Since both the null injection and the 1 MB zero injection triggered errors regarding nonce length, the system likely filters for size. That is important as it prevents malformed or oversized inputs from potentially compromising deeper system functions.

Similarly, the maximum integer injection attack failed to cause crashes or unexpected behavior. This indicates that the TKey is resilient to integer based boundary values, at least for the tested range.

While the TKey handled these cases well, it is important to note that this fuzzing testing was limited. The results provide a base of how TKey manages nonce values, it should not be considered exhaustive or advanced fuzzing testing.

5.3.3 Injection Attack

The injection attack attempted to insert additional commands through the nonce field. To succeed in that attack, the attacker would require the TKey to interpret parts of the nonce as separate framing instructions. However, as the TKey signed the complete nonce without interpreting any embedded commands, it is treating the input purely as payload data.

This suggests that the framing structure is separated from the payload handling in the implementation. The sign function appears to construct the message header before assigning the nonce as payload data. So when the TKey processes the request, the payload is handled as raw bytes. Because of this the injected command sequences inside the nonce are treated as ordinary data and included in the signature instead of being executed.

Using this approach reduces the risk of command injection through the signing interface. If not, an attacker could potentially manipulate the communication protocol or alter the behavior of the signing operation. The result indicates that the message parsing logic is relatively robust against this type of injection attempt.

Because this testing was limited to a small number of manually constructed payloads, it does not claim to be secure against more advanced attacks. They may still expose weaknesses that were not identified during this project.

5.3.4 Replay Attack

The results from the replay attack shows that the verify on the application level appeared resistant to replay attempts during the conducted experiments. The nonce disables the reuse of a verified signature by making each authentication request use a fresh nonce, making previously valid signatures invalid for future requests. Right now, the nonce is completely random, making it possible, but highly improbable that the next challenge uses the same nonce.

The replay attack does not concern the security within the TKey. The interruption in communication in the second challenge part of the attack, as seen in the figure 4.3 shows that the TKey is never affected and nothing in the input is ever changed.

Therefore this attack only works on the application layer and suggests that the implementation of the TKey should use cryptographically secure procedures within the challenge/response with the TKey.

5.3.5 Nonce Reuse Attack

The computational hardness obtained by the usage of 32-byte long nonce shown in the results makes it improbable for the nonce reuse attack to ever succeed and results in the TKey and the authentication flow to be secure.

The Tillitis TKey in theory prevents these forms of automated attacks through the touch sensor, making authentication a non-automatic procedure. But through the mitm-proxy, the flow of communication was broken and made it so the request arriving at the mitm-proxy did not send the flow forward to the TKey, but instead forced a response to the web client with a saved, once verified signature (see Figure 4.4). Therefore the nonce reuse attack never affects the TKey directly, but targets the application layer of the ecosystem.

As stated in the results of the Nonce reuse attack in subsection 4.2.5, the computer used in the execution of the attack made about 10,000 nonce reuse attempts per minute. This means that, if the Tillitis TKey use truly random 32-byte long numbers, there is a probability of $1 \div 2^{256}$ for one forged signature to match the randomly generated signature. In theory this means that 2^{256} challenges needs to be made in order to get at least one to verify. For this to happen it would take about $2^{256} \div 11,000 \simeq 1.053 \times 10^{73}$ minutes, which would be about 2.20×10^{67} years. This is an unfathomably long time and is computationally infeasible under the evaluated conditions.

Overall, the nonce reuse attack demonstrates that the ecosystem in this project correctly implements challenge-response mechanisms, ensuring that nonce reuse does not compromise the security of the TKey ecosystem.

5.4 Proposed Countermeasures

The results of the security evaluation shows that the identified ecosystem level weaknesses are not necessarily the TKey itself, but the surrounding ecosystems communication and applications. Based on the attack vectors and observations during testing, there are several viable countermeasures and mitigation to improve the ecosystems security.

One of the most important improvements is the communication between the TKey and the server, it needs to be encrypted. During the PITM attack metadata and values were in plain text, demonstrating insufficient protection. By implementing TLS or similar encryption the intercepted traffic would be much harder to analyze or reuse. Additional protection could be session validation with session IDs and timestamp validation. This would reduce the risk of a replay attack by ensuring received data becomes invalid after one use.

The replay attack and nonce reuse attack also demonstrated that it is important to have proper nonce handling within the authentication flow. Nonce should always be generated using cryptographically secure random values and should only

remain valid for a very limited amount of time. Maintaining temporary records of previously used nonce could improve protection against replayed authentication attempts. During fuzzing and injection attacks, the TKey appeared to handle malformed input robustly. The payloads tested did not cause crashes or unexpected behavior. Despite these positive results, the tests were limited in scope. Therefore, to prevent future and more advanced exploits and improved resilience, continued input validation and strict boundary checking are important during development.

From this project, it is clear that the security of the TKey cannot be evaluated independently from the surrounding application. Even if the TKey remains cryptographically secure, insecure implementation choices can still compromise the authentication setup. Because of this, applications integrating the TKey should implement strict server-side signature verification, minimize unnecessary software complexity, and regularly check security. Reducing the number of third-party dependencies and applying the principle of least privilege could further decrease the attack surface surrounding the authentication flow.

5.5 Knowledge and Future Work

Our findings provide a better understanding of the security of the Tillitis ecosystem. However, it is important to understand that our assessment is highly dependent on the specific time frame and prior knowledge of the group. For this project, a significant amount of time was devoted to understanding the research field in order to provide the necessary expertise. More knowledge within the field of vulnerability assessment and penetration testing could have given the project an experienced starting point and brought the research further. With more experience, theoretical vulnerabilities within the TKey may have been found and attacks could have been implemented and tailored specifically to target said vulnerabilities. Constructing targeted attacks could be more likely to prove weaknesses and assess flaws within the TKey.

The scope of this project does not cover all possible breaches of TKey security. Vulnerabilities that were deemed infeasible or secure were not explored and could potentially be exploited, given more time or complex implementations.

Basic command injection was considered infeasible in our current setup. So more advanced techniques such as protocol-aware fuzzing could uncover vulnerabilities about how the TKey parses complex or nested commands. Because this project excluded hardware-level attacks, such as side-channel or timing attacks, to focus on logical communication. Future research could utilize the testing environment to measure processing times to see if any information leaks from the internal cryptographic operations.

Additionally, with the improvement of quantum computing and the large amount of research in progress, valid future work would be to make a security evaluation on eventual weaknesses that can be exploited if quantum computers were to be available to malicious parties.

The environment developed in this project was designed to be reproducible and modular. While it fits our tests it can be further expanded to allow for a variety of attacks. The current environment uses three virtual machines to simulate a

network-based flow. Future iterations could scale this to simulate more complex network topologies, such as multiple concurrent users or compromised relay servers. That could be used to test the system's robustness under high load or multi-vector attacks.

The environment could also be modified to work with both software simulations and physical hardware analysis. That would allow a more complete evaluation of the entire chain of authentication.

Finally, future work could include a more detailed study of the interaction between the TKey and a wider variety of host applications. As the security of the TKey depends on the full chain, including client software and protocol logic. By analyzing different third-party implementations of the TKey libraries it would clarify which security properties hold across the ecosystem. This could eventually lead to the development of a standardized security certification process for applications integrating the Tillitis TKey.

5.6 Conclusion

The purpose of this project was to perform a structured security evaluation of the Tillitis TKey ecosystem. This project aimed to investigate possible weaknesses within the TKey ecosystem. To do the security evaluation a controlled environment was constructed, potential threats on the protocol- and communication-level were evaluated, and an analysis on how secure the authentication flow is during simulated attacks were developed.

To test the attacks this project successfully constructed a modular and controlled environment. The attacks tested were Person-in-the-Middle, Replay, Injection, Fuzzing, and Nonce reuse attacks. With these attacks, the project showed that the TKey itself appears to have strong cryptographic functions and protocol separation principles. The performed attacks did not expose the private key nor bypass the cryptographic processes.

However, the results showed that the security of the TKey was not limited to the hardware. The communication channel and the application using the TKey were critical parts of the security. During the Person-in-the-Middle attack, metadata and hashed values could be intercepted in plaintext. So even when the hardware is secure, insecure communication can cause vulnerabilities. The replay attack also showed that correct nonce handling and session validation is important.

The fuzzing and injection attacks showed that the TKey handles malformed input relatively robustly. However, the attacks performed in this project were limited in scope and complexity. The results of this project should therefore be interpreted as an initial basis for security analysis and as a reproducible testing environment for future research. It is not proof that the TKey ecosystem is entirely secure from these attacks.

The project overall achieved its primary goal of evaluating the security of the TKey ecosystem. It shows that hardware-based authentication systems are only as secure as the ecosystem they operate in. Even if the cryptographic hardware remains secure, the authentication process can still be compromised.

Bibliography

[1] M. I. M. Yusop, N. H. Kamarudin, N. H. S. Suhaimi, and M. K. Hasan, "Advancing passwordless authentication: A systematic review of methods, challenges, and future directions for secure user identity," *IEEE Access*, vol. 13, pp. 13 919–13 943, 2025. DOI: 10.1109/ACCESS.2025.3528960. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.3528960.

[2] Tillitis. "Tkey," Accessed: Feb. 9, 2026. [Online]. Available: https://www.tillitis.se/products/tkey/.

[3] Tillitis. "Threat model," Accessed: Feb. 3, 2026. [Online]. Available: https://github.com/tillitis/tillitis-key1/blob/79920ee0bf2722b8bfd432d097e4eb19546972c9/doc/threat_model/threat_model.md.

[4] Tillitis. "Tillitis security bulletin 240115-1," Accessed: Feb. 9, 2026. [Online]. Available: https://bugbounty.tillitis.se/security-bulletins/tillitis-security-bulletin-240115-1/.

[5] National Institute of Standards and Technology, "Digital identity guidelines: Authentication and lifecycle management," U.S. Department of Commerce, Special Publication NIST SP 800-63B-4, 2024. DOI: 10.6028/NIST.SP.800-63b-4. Accessed: Mar. 24, 2026. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63B-4.pdf.

[6] FIDO Alliance. "Passkeys (multi-device fido credentials)," Accessed: Mar. 24, 2026. [Online]. Available: https://fidoalliance.org/passkeys/.

[7] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 4th. Pearson, 2017, ISBN: 978-0134794105.

[8] D. M'Raihi, M. Machani, M. Pei, and J. Rydkog, "Totp: Time-based one-time password algorithm," IETF, Tech. Rep. RFC 6238, 2011.

[9] ARM Limited, "Arm security technology: Building a secure system using trustzone technology," ARM Limited, Tech. Rep. PRD29-GENC-009492, 2009.

[10] GlobalPlatform, *Tee system architecture*, Available: https://globalplatform.org/specs-library/tee-system-architecture/, 2019.

[11] Trusted Computing Group, "Trusted platform module library specification, family 2.0, part 1: Architecture," Trusted Computing Group (TCG), Standard, version Version 185, 2026. Accessed: Mar. 30, 2026. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-library-specification/.

[12] Tillitis. "Tkey introduction," Accessed: May 15, 2026. [Online]. Available: https://dev.tillitis.se/intro/#measured-boot--secrets.

[13] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, *Nonce-disrespecting adversaries: Practical forgery attacks on gcm in tls*, Cryptology

ePrint Archive, Paper 2016/475, 2016. Accessed: May 16, 2026. [Online]. Available: https://eprint.iacr.org/2016/475.

[14] MITRE. "Cwe-323: Reusing a nonce, key pair in encryption," Accessed: May 16, 2026. [Online]. Available: https://cwe.mitre.org/data/definitions/323.html.

[15] OWASP Foundation. "Fuzzing," Accessed: May 4, 2026. [Online]. Available: https://owasp.org/www-community/Fuzzing.

[16] IBM. "What is ethical hacking?" Accessed: May 18, 2026. [Online]. Available: https://www.ibm.com/think/topics/ethical-hacking.

[17] FIRST.org. "Common vulnerability scoring system sig," Accessed: Feb. 4, 2026. [Online]. Available: https://www.first.org/cvss/.

[18] U. Jakob, F. Kraemer, F. Kraus, and T. Lengauer, "Applying ethics in the handling of dual use research: The case of germany," *Research Ethics*, vol. 21, no. 2, pp. 228–244, 2025. DOI: 10.1177/17470161241261044.

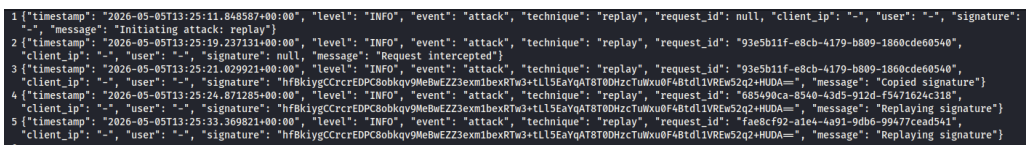
[19] M. Celestin and N. Vanitha, "Ethical hacking demystified: How 'good' hackers keep us safe," *International Journal of Multidisciplinary Research and Modern Education*, vol. 1, no. 1, pp. 721–727, 2015. [Online]. Available: https://www.researchgate.net/profile/Prof-Celestin/publication/385620131_ETHICAL_HACKING_DEMYSTIFIED_HOW_%27GOOD%27_HACKERS_KEEP_US_SAFE/links/672cb2e3ecbbde716b5ecdaa/ETHICAL-HACKING-DEMYSTIFIED-HOW-GOOD-HACKERS-KEEP-US-SAFE.pdf.

A

Appendix - Attack logs


```
[ATTACK FILTER: pitm]
{
  "timestamp": "2026-04-22T08:10:49.484154+00:00",
  "level": "INFO",
  "event": "attack",
  "technique": "pitm",
  "request_id": null,
  "client_ip": "172.20.0.20",
  "user": "PITM",
  "message": "HTTP POST http://172.20.0.30:7070/challenge -> Status: 200"
}
{
  "timestamp": "2026-04-22T08:10:49.486560+00:00",
  "level": "DEBUG",
  "event": "attack",
  "technique": "pitm",
  "request_id": null,
  "client_ip": "172.20.0.20",
  "user": "PITM",
  "message": "DATA: {\"from\":\"\",\"data\":\"dfQwE5BjDfOSk+qmóbhOIpOncYgBh3AHZI8u2CdETNo=\",\"type\":4,\"publicKey\":null,\"verified"
}
{
  "timestamp": "2026-04-22T08:11:05.808192+00:00",
  "level": "INFO",
  "event": "attack",
  "technique": "pitm",
  "request_id": null,
  "client_ip": "172.20.0.20",
  "user": "PITM",
  "message": "HTTP POST http://172.20.0.30:7070/challenge -> Status: 200"
}
{
  "timestamp": "2026-04-22T08:11:05.808519+00:00",
  "level": "DEBUG",
  "event": "attack",
  "technique": "pitm",
  "request_id": null,
  "client_ip": "172.20.0.20",
  "user": "PITM",
  "message": "DATA: {\"from\":\"\",\"data\":\"/MnZJIIkefFTMzxN/edrV5IhmtdG6cRjb3BpoYbq8eY=\",\"type\":4,\"publicKey\":null,\"verified"
}
```


Figure A.1: Captured HTTP communication during PITM attack




```
1 {"timestamp": "2026-05-05T13:25:11.848587+00:00", "level": "INFO", "event": "attack", "technique": "replay", "request_id": null, "client_ip": "-", "user": "-", "signature":
  "-", "message": "Initiating attack: replay"}
2 {"timestamp": "2026-05-05T13:25:19.237131+00:00", "level": "INFO", "event": "attack", "technique": "replay", "request_id": "93e5b11f-e8cb-4179-b809-1860cde60540",
  "client_ip": "-", "user": "-", "signature": null, "message": "Request intercepted"}
3 {"timestamp": "2026-05-05T13:25:21.029921+00:00", "level": "INFO", "event": "attack", "technique": "replay", "request_id": "93e5b11f-e8cb-4179-b809-1860cde60540",
  "client_ip": "-", "user": "-", "signature": "hfBkiygCCrcrEDPC8obkqv9MeBwEZZ3exm1bexRTw3+tLl5EaYqAT8T0DHzcTuWxu0F4Btdl1VREw52q2+HUDA==", "message": "Copied signature"}
4 {"timestamp": "2026-05-05T13:25:24.871285+00:00", "level": "INFO", "event": "attack", "technique": "replay", "request_id": "685490ca-8540-43d5-912d-f5471624c318",
  "client_ip": "-", "user": "-", "signature": "hfBkiygCCrcrEDPC8obkqv9MeBwEZZ3exm1bexRTw3+tLl5EaYqAT8T0DHzcTuWxu0F4Btdl1VREw52q2+HUDA==", "message": "Replaying signature"}
5 {"timestamp": "2026-05-05T13:25:33.369821+00:00", "level": "INFO", "event": "attack", "technique": "replay", "request_id": "fae8cf92-a1e4-4a91-9db6-99477cead541",
  "client_ip": "-", "user": "-", "signature": "hfBkiygCCrcrEDPC8obkqv9MeBwEZZ3exm1bexRTw3+tLl5EaYqAT8T0DHzcTuWxu0F4Btdl1VREw52q2+HUDA==", "message": "Replaying signature"}
```


Figure A.2: Captured logs from the replay attack

[FULL ROW VIEW]

REQUEST_ID	ATTACK_TIME	SIGN_TIME	VERIFY_TIME	ATTACK	NONCE	SIZE	SIGNATURE	ERROR	FLAG	VERIFIED
	-	11:37:48.385	11:37:48.386	-	dd9c5858...	32	a8357ef2...	-	-	true
d828fb20...	11:38:03.080	11:38:04.325	11:38:04.329	commandInjection	002008	3	38d37421...	-	-	false
f859b895...	11:38:07.924	11:38:09.414	11:38:09.417	commandInjection	ff2008	3	e1041a7d...	-	-	false
3eab7cee...	11:38:08.704	11:38:09.553	11:38:09.556	commandInjection	002008	3	38d37421...	-	-	false
42727aa6...	11:38:11.019	11:38:12.203	11:38:12.205	commandInjection	ff2008	3	e1041a7d...	-	-	false

Figure A.6: Captured logs from the command injection attack

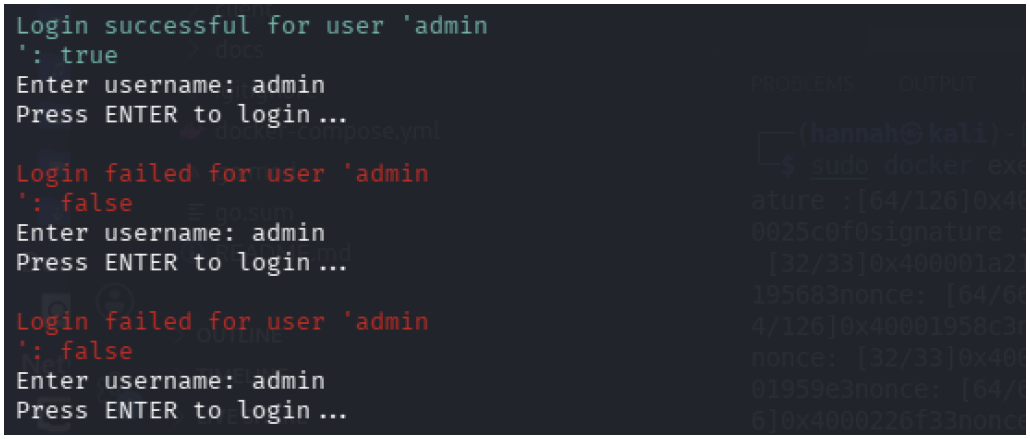
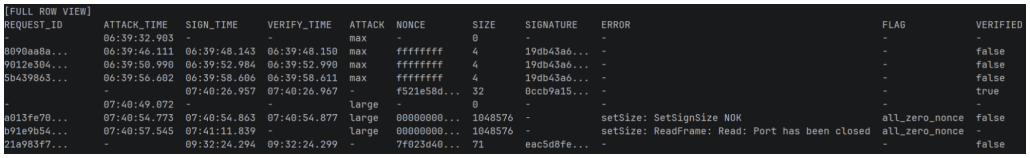


Figure A.3: Captured terminal output from the replay attack



[FULL ROW VIEW]

REQUEST_ID	ATTACK_TIME	SIGN_TIME	VERIFY_TIME	ATTACK	NONCE	SIZE	SIGNATURE	ERROR	FLAG	VERIFIED
-	06:39:32.903	-	-	max	-	0	-	-	-	-
8090aa8a...	06:39:46.111	06:39:48.143	06:39:48.150	max	ffffffff	4	19db43a6...	-	-	false
9012e304...	06:39:50.990	06:39:52.984	06:39:52.990	max	ffffffff	4	19db43a6...	-	-	false
5b439863...	06:39:56.602	06:39:58.606	06:39:58.611	max	ffffffff	4	19db43a6...	-	-	false
	-	07:40:26.957	07:40:26.967	-	f521e58d...	32	0ccb9a15...	-	-	true
-	07:40:49.072	-	-	large	-	0	-	-	-	-
a013fe70...	07:40:54.773	07:40:54.863	07:40:54.877	large	00000000...	1048576	-	setSize: SetSignSize NOK	all_zero_nonce	false
b91e9b54...	07:40:57.545	07:41:11.839	-	large	00000000...	1048576	-	setSize: ReadFrame: Read: Port has been closed	all_zero_nonce	-
21a983f7...	-	09:32:24.294	09:32:24.299	-	7f023d40...	71	eac5d8fe...	-	-	false

Figure A.4: Captured logs from the fuzzing attacks

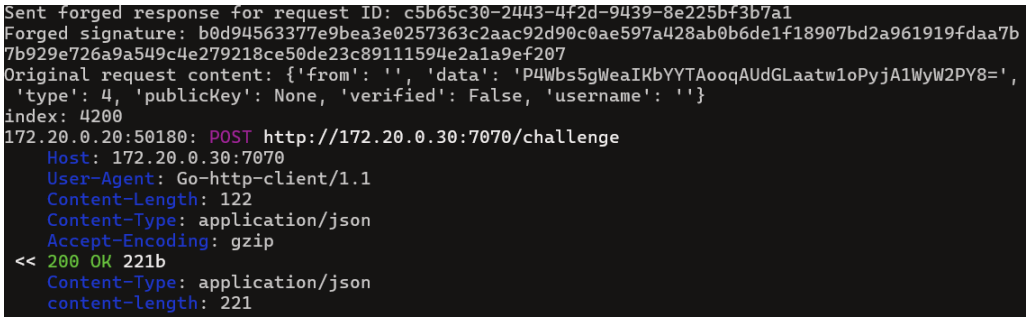


Figure A.5: An example of one capture from the nonce reuse logs