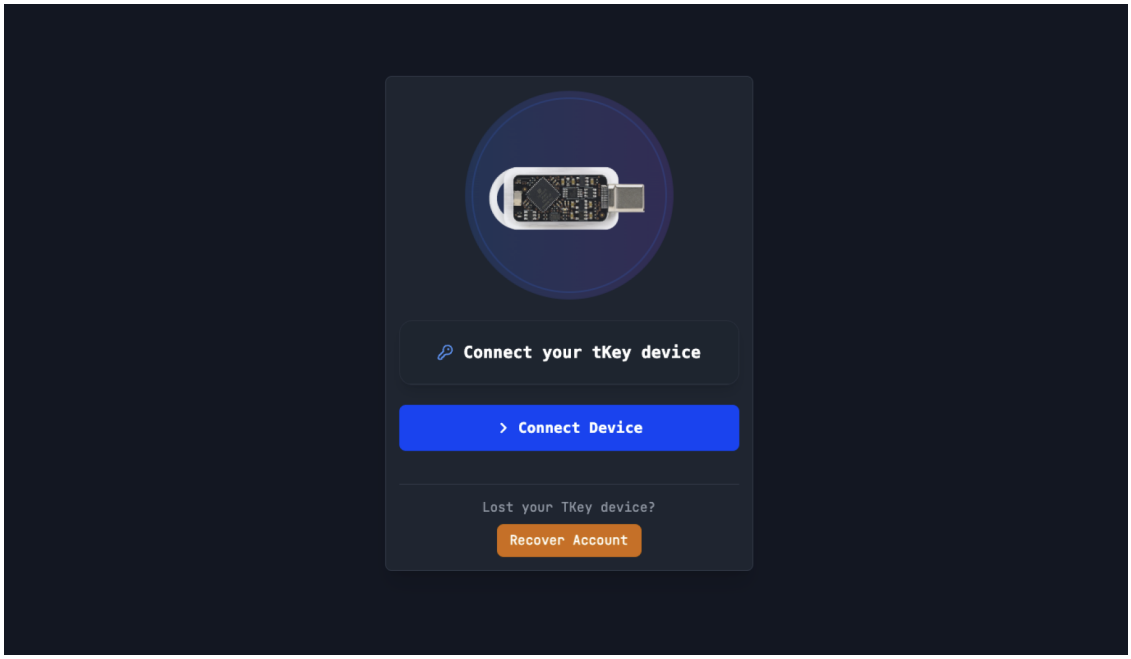


# Designing Passwordless Authentication with the Tillitis TKey for Secure Web Login

A user-friendly authentication method using cryptographic hardware

Bachelor's thesis in Computer science and engineering

Daner Faraj, Filip Kauffeldt, Ismail Sacic, Ken Phu, Lucas Häyhänen, Robin Karhu

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

Bachelor's thesis 2025

# Designing Passwordless Authentication with the Tillitis TKey for Secure Web Login

A user-friendly authentication method using cryptographic hardware

Daner Faraj
Filip Kauffeldt
Ismail Sacic
Ken Phu
Lucas Häyhänen
Robin Karhu

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

A user-friendly authentication method using cryptographic hardware
Daner Faraj  Filip Kauffeldt  Ismail Sacic  Ken Phu  Lucas Häyhänen  Robin Karhu



Supervisor (Handledare): Yasir Hussain, Doctoral Student - Computer and Network Systems, Computer Science and Engineering
Advisor: Romaric Duvignau, Head of Unit - Computer and Network Systems, Computer Science and Engineering
Examiners: Irum Inayat, Senior Lecturer, Interaction Design and Software Engineering, Computer Science and Engineering
Graded by teacher (Rättande lärare): Irum Inayat, Senior Lecturer, Interaction Design and Software Engineering, Computer Science and Engineering

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

Cover: Photo of the landing page created during this project.

Typeset in LaTeX
Gothenburg, Sweden 2025

# Abstract

The growing frequency of cyber threats, coupled with increasing regulatory demands, has intensified the need for secure user-friendly alternatives to traditional password-based authentication systems. This thesis presents a proof of concept implementation of a passwordless authentication solution using the Tillitis TKey, a cryptographic USB device designed for secure identity verification. The proposed solution uses a hardware based challenge-response authentication, implemented using a modern web stack, using a Svelte frontend and utilizing the Web Serial API for browser based hardware communication. To further increase security, and as an option for future hardware development, optional biometric authentication using facial recognition was introduced as a second factor. The project involved translating TKey Go libraries written by Tillitis into TypeScript, to simplify browser based execution, thereby enhancing usability for the development team. Although the system has certain limitations, such as restricted comparability with some browsers and operating systems, it demonstrates the feasibility and advantages of using of using hardware-based passwordless authentication on the web. The thesis also discusses technical trade-offs, ethical considerations that were made, outlines future work, including OAuth integration, secure key life-cycle management and improved usability. Another limitation of the project is lack of formal security audit, which was deemed to be outside the scope of this project. The Tillitis TKey is treated as a trusted component from the manufacturer, and the web application serves as a proof of concept rather than a production ready authentication service that is equipped to handle real world user data and account protection. Overall, the project provides a foundation for developing secure and privacy-conscious authentication systems as alternatives to traditional password-based authentication services.

# Sammanfattning

Den ökande frekvensen av cyberhot, i kombination med ökade regulatoriska krav, har intensifierat behovet av säkra användarvänliga alternativ till traditionella lösenordsbaserade autentiseringssystem. Denna avhandling presenterar en proof of concept-implementering av en lösenordslös autentiseringslösning med Tillitis TKey, en kryptografisk USB-enhet utformad för säker identitetsverifiering. Den föreslagna lösningen använder en hårdvarubaserad challenge-response-autentisering, implementerad med en modern webbstack, med en Svelte-frontend och med Web Serial API för webbläsarbaserad hårdvarukommunikation. För att ytterligare öka säkerheten, och som ett alternativ för framtida hårdvaruutveckling, infördes valfri biometrisk autentisering med ansiktsigenkänning som en andra faktor. Projektet omfattade översättning av TKey Go-bibliotek skrivna av Tillitis till TypeScript, för att förenkla webbläsarbaserad körning och därmed förbättra användbarheten för utvecklingsteamet. Även om systemet har vissa begränsningar, till exempel begränsad jämförbarhet med vissa webbläsare och operativsystem, visar det att det går att använda hårdvarubaserad lösenordslös autentisering på webben och att det finns fördelar med detta. Avhandlingen diskuterar också tekniska kompromisser, etiska överväganden som gjorts och beskriver framtida arbete, inklusive OAuth-integration, säker livscykelhantering av nycklar och förbättrad användbarhet. En annan begränsning i projektet är avsaknaden av en formell säkerhetsgranskning, som ansågs ligga utanför projektets omfattning. Tillitis TKey behandlas som en betrodd komponent från tillverkaren, och webbapplikationen fungerar som ett proof of concept snarare än en produktionsfärdig autentiseringstjänst som är utrustad för att hantera användardata och kontoskydd i den verkliga världen. Sammantaget ger projektet en grund för att utveckla säkra och integritetsmedvetna autentiseringssystem som alternativ till traditionella lösenordsbaserade autentiseringstjänster.

# Acknowledgements

We want to thank Tillitis for their support and for allowing us to develop this system with their hardware and software tools. We also thank our supervisor, Yasir Hussain, for his dedication to assisting during the development of our project, and we thank Chalmers University of Technology for giving us the opportunity to conduct this project.

Daner Faraj, Filip Kauffeldt, Ismail Sacic, Ken Phu, Lucas Häyhänen, Robin Karhu, Gothenburg, June 2025

# Contents

# List of Figures

# 1
# Introduction

This section outlines the aim and scope of the thesis. It defines the central objectives and the key questions. It also establishes a framework for understanding the motivations, expected outcomes and areas of focus.

## 1.1 Purpose

In recent years, the demand for secure authentication methods has increased significantly. This trend is expected to continue, driven in part by new regulations, including the NIS2 directive[1], which imposes stricter requirements on cybersecurity and identity management. Hardware-based authentication devices, such as the Tillitis TKey, present an alternative to traditional software-based password solutions. Tillitis has yet to provide a user-friendly, web-integrated authentication system that leverages the TKey's capabilities.

The purpose of this thesis is to explore new possibilities with the Tillitis TKey. Without any local software instillation, the project allows users to authenticate on websites using their TKey device. This is demonstrated by developing a TypeScript-based solution that allows TKey integration directly in the browser as a proof of concept.

The technical focus of the project is the development of a prototype system using TypeScript and the Svelte framework. This system will enable secure user registration and login through a challenge–response mechanism, where the user signs a server-issued challenge using the private key stored within the TKey. The server verifies this signature using the corresponding public key. This approach eliminates the need for one to store or manage passwords, thereby enhancing security and usability.

A key aspect of the thesis is the exploration of user authorization through hardware-based identity verification. In addition, the project investigates the integration of biometric data, specifically facial recognition, as an optional secondary security layer. This includes analyzing technical feasibility and assessing security benefits. By encrypting password data using biometric input, even compromised credentials can remain protected by a unique, user-specific biometric key. This layered approach reduces the risk of unauthorized access and strengthens system security.

Finally, the thesis aims to evaluate the current limitations of the TKey device and propose potential improvements for future development. It also seeks to provide practical recommendations for implementing, passwordless authentication solutions in broader web and IT security contexts, thus contributing to industrial practices.

## 1.2 Objectives

The main challenge of this project is integrating the TKey supplied by Tillitis into a web application, ensuring users can register and authenticate securely. This involves exploring effective cryptographic methods for user identification and applying them in a practical environment.

The project is divided into three key tasks/objectives:

- **Implementation of the Authentication System:** This task involves developing the core authentication system using public key cryptography and selecting an appropriate backend technology for the server-side logic. User information should be stored in a database.

- **Client Application Development:** Creating a browser-compatible interface that can communicate with the TKey. The objective is to enable this communication by integrating the necessary API functionality into the frontend, either through WebAssembly or a TypeScript-based implementation.

- **Integration and Evaluation:** This task involves evaluating the performance and usability of each implementation strategy. Additionally, it focuses on integrating hardware communication protocols, such as WebUSB or WebHID, to ensure secure communication with the TKey hardware device.

## 1.3 Limitations

This subsection will detail the limitations that had to imposed on the project. These limitations result from project scope, time constraints, and limited hardware access. The project aims to demonstrate a proof of concept for passwordless authentication using the Tillitis TKey rather than to deliver a production-ready solution. Several features, e.g. full platform support, formal security reviews, and regulatory compliance, are beyond the scope. The focus has been on prototyping, exploring potential integrations and showcasing functionality that Tillitis may consider for future development.

L1. **Limited Cross-Platform Compatibility Testing**
The prototype will only be tested in one controlled environment, chromium-based namely Chrome or Microsoft Edge and not across all browsers, operating systems or mobile devices. The objective is to optimize the

debugging and validation process. Due to time and resource constraints, testing the prototype on multiple platforms falls outside of the scope of this project.

L2. **No Formal Security Audit**
The system will not undergo an independent security review, third-party security tests, or official certification. A complete security audit would require more experience and resources than is available for this bachelor's project.

L3. **Limited Hardware Tampering Analysis**
Physical attacks on the TKey device, e.g. side-channel attacks or hardware reverse-engineering, will not be addressed. The TKey is considered a trusted component in this project. Physical security is beyond the scope of software development and remains relevant to hardware security research.

L4. **No Compliance with Regulatory Standards**
Legal or regulatory compliance, e.g. GDPR, will not be addressed. This project is a technical proof-of-concept, not a production-ready system. Ensuring compliance requires specialized legal and bureaucratic efforts, a task that is outside the academic goal of demonstrating TKey-based authentication.

L5. **Simplified Key Management**
Advanced key lifecycle features, such as key rotation and revocation, will not be fully implemented. The project will focus on registration and authentication workflows. Implementing comprehensive key management would expand the scope of the project and is suited for future work.

L6. **Basic Error Handling and Recovery**
Robust error recovery mechanisms, as in the case of fallback authentication for lost TKeys, will not be prioritized. The prototype focuses on the ideal use case of TKey-based authentication, with simplified error handling to focus development on the main objectives to ensure timely completion.

L7. **No Performance Optimization for Scalability**
Stress testing, load balancing, or optimization for high-traffic scenarios will not be conducted. The prototype is designed for a limited user base (bachelor students) to validate functionality. Scalability is relevant for production systems and falls outside the scope of this prototype.

L8. **Simplified Facial Recognition Implementation**
The facial recognition component uses a model and stores only a single image per user, resulting in limited accuracy. This implementation primarily serves to represent facial data numerically, providing a proof of concept. The intention is for Tillitis to evaluate whether investing in writable memory and an advanced facial representation model is worthwhile for future development.

# 2
# Background

This section provides an overview of the foundational concepts and technical underpinnings relevant to the project. It aims to establish the theoretical and technical context for understanding processes involved in the proposed solution. The discussion is divided into two subsections the theoretical background, which covers the conceptual principles and the technical background, detailing the specific technologies and implementations used.

## 2.1 Theoretical background

To understand the project's core mechanisms, it is essential to first explain the foundational concepts utilized by the TKey, as well as the client and server systems. This section will detail the concepts behind public key cryptography, the use of external hardware as authentication and the Tkey.

### 2.1.1 Public key cryptography

Public key cryptography, also referred to as asymmetric cryptography, is a type of encryption method that utilizes public key cryptosystems to generate a pair of keys. These two keys are the private key and public key which are used to encrypt and decrypt messages respectively. The reliability of public key cryptography is dependent on keeping the private key a secret while the public key can be openly distributed without compromising security. Before the utility of public key cryptography can be discussed, it is necessary to explain how the Tillitis TKey generates its key pair[2].

### 2.1.2 Key Generation

To generate a public/private keypair, TKey applies a Key Derivation Function (KDF) that uses three main components, User Supplied Secret (USS), Unique Device Secret (UDS) and a device app (can be a cryptographic function)[3]. The TKey's KDF utilizes SHA-512, a one-way cryptographic hash functions to translate the supplied data to a unique sequence to enhance the security of the KDF. The UDS has a unique sequence on each TKey that, with the USS, generates a key pair with the private key being stored exclusively on the TKey device and the public key being supplied to the client. Equation 2.1 depicts the complete KDF for the TKey.

$$Hash(UDS, Hash(DeviceApp), USS) = CDI(compounddeviceID) \tag{2.1}$$

### 2.1.3 Utilization of public key cryptography

Public key cryptography is primarily used for authentication and confidentiality. Authentication entails the insurance that the information being received and decrypted by one party is sent by the correct sender with the corresponding private key.

Figure 2.1 illustrates an authentication protocol that utilizes public key encryption. In this example, B wants to deduce whether A is the holder of a specific private key or not. B starts by sending a specific message to A that is also stored by B for later validation. A (who is in possession of the private key) encrypts the message and sends it back to B. The receiver B can, after receiving the encrypted message, decrypt it using their corresponding public key and compare the message that was sent to the one that was decrypted. It is important to note that only a recipient possessing the corresponding public key can decrypt the message and validate the authenticity of A.

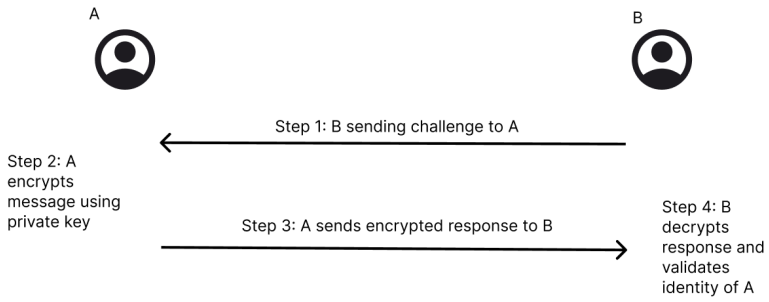


**Figure 2.1:** Authentication using public and private key pair

This project utilizes the method explained above to validate the authenticity of each TKey. The server acts as person B and the TKey as person A. This ensures that no individual without the specific TKey linked to an account can access the holders information stored on the server.

## 2.2 Technical Background

This section outlines the key technologies, frameworks and tools used in developing the passwordless authentication system, focusing on both the hardware and software components involved.

### 2.2.1 TKey

The Tilitis TKey is a security focused USB device designed to serve as a secure hardware alternative to traditional password-authentication. It is designed around open-source principles and can securely load and run attestation enabled applications. The main part of the TKey is a RISC V based SoC (System on Chip). The TKey device uses a WCH microcontroller based on the 32-bit RISC-V instruction set. It provides the computational resources necessary to run both the firmware and

user-loaded applications used for authentication and validation, which are loaded at different stages of the login process. The TKey connects to a computer via USB-C and is used for both power and communication. The USB interface is the primary I/O channel for loading programs and interacting with the host machine. A single RGB LED provides visual feedback to the user through different colors and blink rates. It indicates the state of the device, such as boot status, program loading and secure operation modes. The TKey includes a touch-sensitive area that the user can interact with to confirm sensitive actions, such as granting access or signing requests. The TKey indicates this via the LED which blinks green when it is waiting for human interaction. This provides an additional layer of physical user consent and enhances the security of the TKey[4].

### 2.2.2 TKeyclient

The TKeyclient Go package is responsible for establishing a connection with the TKey, retrieving device information and most importantly, uploading binaries for execution. The primary focus was on the binary upload functionality, which is required for enabling further interactions with TKey. One key binary is the signer binary, which is essential for cryptographic operations.

### 2.2.3 TKeySigner

The TKeySigner Go package provides an interface for sending requests to the signer application on the TKey. It provides functionality for retrieving the public key and signing messages. The ability to obtain a public key allows external systems to verify the identity of the signer, while signing messages ensures secure digital signatures.

### 2.2.4 Svelte

Svelte is a modern, compile time JavaScript framework for building reactive user interfaces. Unlike traditional frameworks, Svelte shifts work into a build step, generating highly optimized imperative code with no runtime virtual DOM. Svelte was chosen for its simplicity, small bundle size and straightforward reactivity model. The frontend, built entirely in Svelte, handles the UI states for device connection, registration, login flows, and session management[5].

### 2.2.5 Supabase

Supabase is an open source Firebase alternative that provides a suite of backend services including a PostgreSQL database, authentication system, storage solutions and real-time subscriptions. It offers a comprehensive set of APIs and tools for developers building web and mobile applications[6].

### 2.2.6 Web Serial API

The Web Serial API provides browser access to serial devices like USB-CDC, HID in serial mode. through a standardized JavaScript interface. By using the Web Serial

API, the web application can enumerate, open, read from and write to the TKey without any native helper applications. Framing and packet validation layers were implemented on top of the raw serial stream to match the TKey's communication protocol[7].

### 2.2.7 SHA-512

SHA-512 is part of the SHA-2 family of cryptographic hash functions, producing a 512-bit digest. The project uses SHA-512 to hash server-generated challenges before sending them to the TKey for signing, ensuring fixed-length, collision-resistant inputs. On the server side, the same hash is verified after signature validation to guarantee data integrity[8].

### 2.2.8 Face Recognition

The face_recognition Python library is used [9], which serves as a wrapper around dlib's deep learning models for face detection and recognition. This library provides a interface for detecting faces, extracting facial features and comparing identities using face encodings.

The face recognition process begins by loading an image and detecting the locations of faces using a Histogram of Oriented Gradients HOG-based model, which is the default detection method in this implementation. The library also supports a convolutional neural network CNN-based model. Once a face is detected, the library extracts a 128-dimensional face encoding, a numerical vector that uniquely represents the facial features of an individual. These encodings are generated by passing the detected facial region through a pre-trained deep convolutional neural network trained on a dataset of labeled facial images.

Each face encoding is a fixed-length vector that captures the spatial structure of a person's face. Face comparison is performed by calculating the Euclidean distance between two such vectors, where a smaller distance indicates greater similarity. To authenticate a user during login, the face encoding of the captured image is compared to previously stored encodings, and a match is determined based on a predefined threshold.

### 2.2.9 WebAssembly

WebAssembly, commonly abbreviated as Wasm, is a low level binary instruction format designed to run in web browsers alongside JavaScript with near-native performance[10][11]. Several high level programming languages, such as Go, C#, Rust, and C/C++, provide Wasm compilation targets. This allows applications written in these languages benefit from the performance of the languages compared to JavaScript, while maintaining the portability and accessibility on web applications. The existing libraries for the Tillitis TKey are implemented in Go, and are theoretically able to run in a web browser by compiling them to Wasm, without requiring major modification to the original code.

# 3
# Implementation

This chapter outlines the practical steps taken to develop the final product, detailing the tools and design choices that shaped its implementation. Each section covers key aspects of the development process, from initial setup to final refinements, providing a structured overview of how theoretical concepts were translated into a working system.

## 3.1 System Overview

The developed system implements a web-based authentication solution using the TKey hardware device. The architecture comprises four main components: the TKey device, a frontend web application developed using Svelte, a backend server responsible for managing authentication logic, and an external database hosted by Supabase. The backend incorporates a dedicated biometric authentication module implemented as a separate Flask server, enabling biometric verification to operate as an optional extension of the core authentication process. The frontend communicates with the TKey via the Web Serial API, while tRPC is used for structured communication between the frontend and backend. User information and authentication records are stored in Supabase. The system architecture and interactions between these components are illustrated in Figure 3.1.

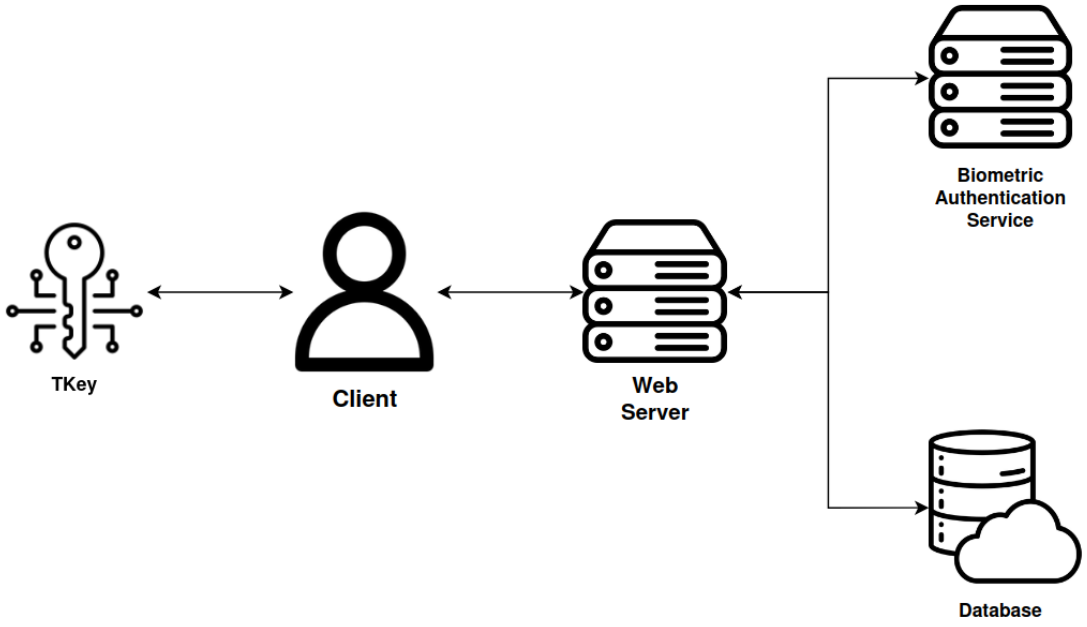


**Figure 3.1:** An overview of the system architecture

### 3.1.1 Subsystem

The system is divided into distinct subsystems, each with defined responsibilities. Figure 3.2 illustrates the specific tasks assigned to each subsystem. The TKey device is responsible for performing cryptographic operations, including as key pair generation and challenge signing. The frontend application, developed using Svelte, handles user interaction, renders static content, and establishes communication with the TKey device through the Web Serial API. It also interacts with the backend using tRPC, enabling structured and type-safe communication.

The backend server is responsible for core authentication workflows, including account creation, session management, and account recovery. It also validates signatures produced by the TKey device. Furthermore, the backend integrates with a biometric authentication service, implemented as a separate Flask server. This service handles is decoupled from the primary authentication flow for modularity and ease of development.

User related data, such as public keys, authorized sessions, and recovery codes, is stored in a Supabase hosted database. The structure of the data, including table definitions and their relationships, is presented in Figure 3.3.

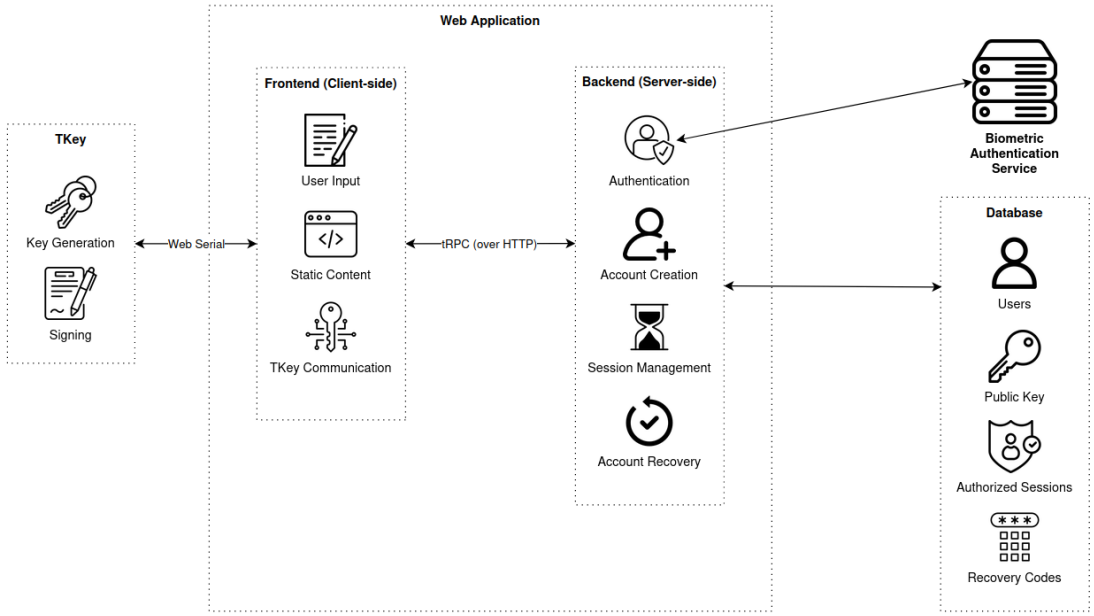


**Figure 3.2:** A subsystem view over the different components of the system

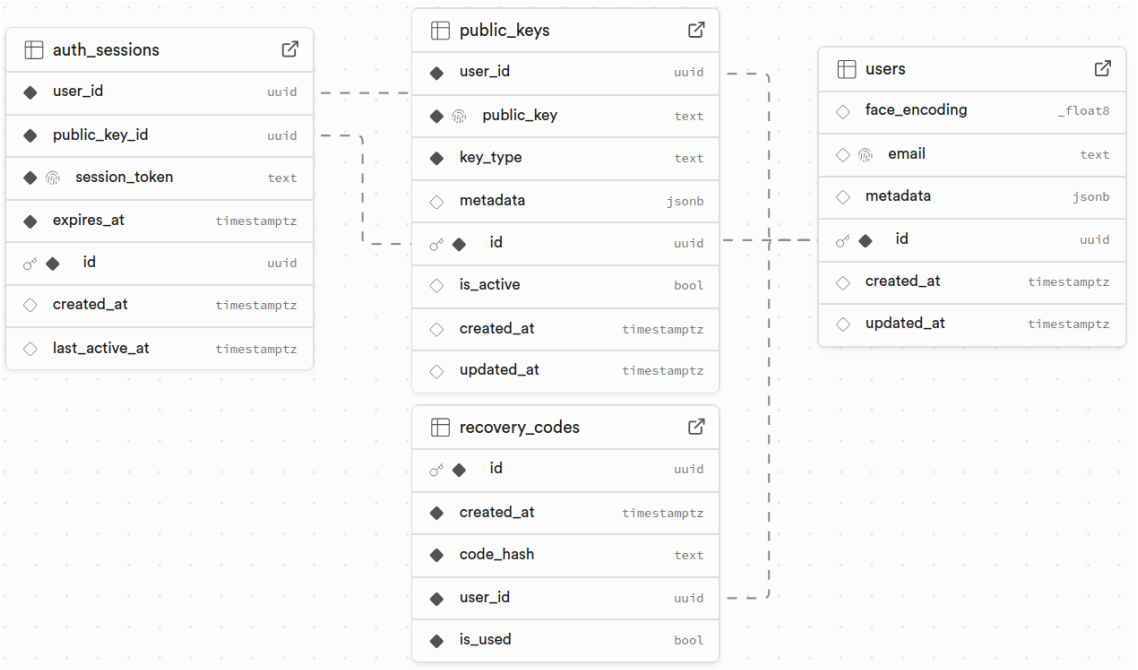


**Figure 3.3:** An database diagram

## 3.2 Setting up communication with TKey through a Web App

To enable communication with the TKey device via a web application, the Web Serial API was utilized to establish a interface between the browser and

the hardware. This browser-native approach eliminates the need for additional drivers or middleware, ensuring compatibility across platforms that support standard web technologies. The connection process is initiated by a call to requestPort(), which opens a system dialog for selecting a compatible serial device. Upon selection, the application calls open() with a specified configuration for baud rate, data bits, stop bits, and parity, to initialize the connection which is implemented in Listing A.1 [12]. This setup allows direct serial communication within the web environment, forming the basis for interaction with the TKey device.

As part of the communication layer, the protocol used by the TKey device was implemented to support structured and reliable data exchange. The protocol involved three primary operations: constructing frames, writing frames, and reading frames. All available commands along with their identifiers, lengths, and associated endpoints were defined according to device specification. To construct a frame, a buffer was initialized with a size based on the command's length, and the appropriate header fields were set using the command's values which can be seen Listing A.2. Communication with the device was handled through the Web Serial API using the write() as can be seen in Listing A.3 and read() methods. Reading a response followed a similar process: a buffer was created, values were read, and the resulting data was validated by comparing it with the expected values defined in the command specification as can be seen in Listing A.4 [13].

The decision to utilize the Web Serial API restricts compatibility to Chromium-based browsers, including Chrome, Edge, and Opera, which currently support the required functionality. This limitation was considered acceptable due to the significant market share held by Chromium-based browsers [14].

## 3.3 Conversion of Go Packages

The protocol functionality was implemented in the web application by porting core logic from two Go libraries developed by Tilitis: TKeyClient and TKeySigner. These libraries contained the essential components for device communication and request handling. The logic was translated into TypeScript to preserve the structure and behavior of the original implementation.

Each protocol command was encapsulated as a standalone function in TypeScript. These functions replicate the behavior of their Go counterparts, following the same structure, response handling, and binary parsing logic as seen in Listing A.5.

The TKeyClient module in TypeScript was developed to support binary data transfer. This included implementing data chunking, error detection, and acknowledgment mechanisms. Buffer handling was adapted to use JavaScript's ArrayBuffer and TypedArray types to align with the browser's memory model.

The TKeySigner module was implemented to enable cryptographic operations within the browser. The signer logic from the Go implementation was

translated into TypeScript while retaining the original API structure. This ensured compatibility with the existing interface and allowed the signing functionality to be executed entirely in the browser.

Although both modules are part of the same protocol implementation, they address different parts of the system. TKeyClient targets the system-level device endpoint and handles operations such as data transfer as shown in Listing A.6. TKeySigner is an application uploaded to the device and uses a separate endpoint for receiving commands as shown in Listing A.7. Commands are directed to either the firmware or the application based on the intended operation.

## 3.4 Authentication

As a proof of concept for passwordless authentication using the TKey, this section outlines the implementation of core authentication features to demonstrate its capabilities in a web-based environment. The focus was on creating a secure yet user-friendly flow, anchored around the use of cryptographic challenge-response via the TKey device. To complement this, biometric authentication using facial recognition was explored as an optional second factor, highlighting how such features could be layered onto a hardware-backed identity system. Together, these implementations serve to showcase how the TKey can replace traditional password-based login flows.

### 3.4.1 Challenge-Response Authentication

With the ability to sign messages using the TKeySigner, the team proceeded to implement challenge-based authentication. On the server side, a challenge generator creates a random string of fixed length, hashes it, and sends it to the client. The client forwards this challenge to the TKey, which signs it. The signed response is then returned to the server. Validation occurs by verifying the signature using the public key associated with the TKey. This mechanism ensures that only the legitimate holder of the TKey can respond to the challenge.

### 3.4.2 Biometric Authentication

To further enhance security, biometric authentication was explored. Although options like fingerprint recognition were considered, it was decided to opt for facial recognition due to broader accessibility. Most laptops already include cameras, making this a practical choice for users.

Facial recognition was not the project's primary focus, so an existing Python library was integrated for this functionality. All biometric data is transmitted over HTTPS and processed transiently, with raw images deleted after encoding. While this introduced a second server into the system, the tradeoff was deemed acceptable due to the ease of implementation and clear modular separation of concerns.

## 3.5 Designing and Implementing the Frontend

This section describes the overall approach to designing and building the frontend of the application. It includes the main style and layout choices, as well as the technologies used to create a clean and functional user interface. It also briefly outline how the frontend was implemented to provide a smooth and user-friendly experience.

- **Design Philosophy and Style Guide**
  The aim was to showcase the solution the without overwhelming the user. The interface features rounded corners, a simple and intuitive layout, and a color scheme inspired by Tilitis, fostering visual cohesion and brand familiarity. These stylistic choices were established in Figma and guided all subsequent frontend development.

- **Landing Page and TKey Integration**
  The first point of interaction in the application is the landing page, which immediately prompts the user to connect their TKey. This approach anchors the TKey as the central authentication device and sets a clear, guided flow for the user. Only after a successful TKey connection are users presented with the options to either register a new account or log in to an existing one. This sequencing minimizes complexity and avoids redundant steps for users who do not yet have a connected TKey.

- **Registration and Login Pages**
  After TKey authentication, users are given access to either register or log in. Both views follow the visual and structural patterns defined in the design phase, maintaining consistency with the landing page. These interfaces were developed to reflect the finalized Figma design and are tightly integrated with the backend logic for authentication and session handling.

- **User page and Session Management**
  While the primary focus of this project has been on TKey and server authentication, it was deemed necessary to implement a page that is displayed following a successful login. The completed signed in page consists of three views navigated to using a classic side menu. These pages are Profile, TKey and documents. The first two are self explanatory with profile displaying the profiles username and the TKey page displaying the public key associated with its private key, app name and version. The last page named Documents is just an example of what the server could contain ( notes, documents and sensitive information).

  During the development of these pages, it became evident that access control was necessary to ensure that only authenticated users could view the content displayed on the TKey and Profile pages. To address this requirement, a

session management system was implemented as a secure and scalable solution.

Upon successful authentication, the user is issued a session token that is uniquely tied to their active client session. This token is then used to navigate the user to their designated profile page. Prior to rendering any sensitive account information, a validation function is executed. This function verifies the legitimacy of the session by comparing the provided token with the corresponding token stored in the database. If the tokens match, the session is deemed valid, and the page proceeds to display the user's account details.

## 3.6 Database Integration

Supabase was selected as the backend service for data management. This choice was primarily motivated by its native TypeScript support and the availability of an SDK, which allowed development to proceed without the need to configure or host supporting infrastructure. This simplified the development workflow. After it was set up it required only a .env file with the SUPABASE_URL and SUPABASE_KEY to connect. While relying on a third-party hosted service such as Supabase introduces dependency risks, it was considered an acceptable tradeoff given the efficiency and simplicity it provided during development.

## 3.7 Client and Server Communication

The communication between the client and server was implemented using tRPC, a TypeScript RPC framework. RPC was chosen over traditional REST endpoints to enable tighter coupling between client and server logic. Although this approach is not ideal for large-scale systems due to its reduced flexibility and strong interdependency, it aligns well with the goals for this proof-of-concept project. One advantage of using tRPC is that it enforces type safety across the stack, reducing runtime errors and enabling more reliable communication between client and server.
This combination of Supabase and tRPC provided a type-safe development with minimal configuration, supporting ease of collaboration.

## 3.8 Account Recovery

A common challenge in passwordless or multi factor authentication is account recovery in the event of a loss of the hardware factor, which is the TKey in this case. Traditional methods such as BankID or SMS were considered but excluded to maintain accessibility for all users, including those without access to a phone or national ID system. Additionally, not using these methods increases the user privacy by not coupling a users account to their person or any other identifiable information, both allowing the users to stay anonymous and reducing the risk of personal information being compromised. Instead, recovery codes were implemented, a simple yet secure fallback mechanism that aligns with the goal of

minimizing dependencies on external systems as well keeping the implementation simple.

During registration, users are issued five unique, randomly generated recovery codes. These codes are stored securely in the database using strong hashing and are never exposed after initial generation. Users are instructed to save these codes in a secure location, such as a password manager.

If access to the TKey is lost, a user may use one of their recovery codes to authenticate. Each code is single-use and immediately invalidated upon successful authentication. After a recovery code is used, the user is prompted to register a new TKey, which replaces the old one to ensure that access cannot be regained using the compromised or lost key.

This approach balances usability and security. It allows users to regain access without relying on external services while maintaining a high level of protection against unauthorized access. Additionally, since recovery codes are generated and managed internally, the developers retain full control over the recovery process, reducing attack surfaces introduced by third-party dependencies.

# 4
# Results

This section presents the outcomes of the implementation, detailing the functional components of the password-less authentication system, performance benchmarks, and usability observations

## 4.1 TKey communication and authentication

Communication between the TKey and the server was implemented using the Web Serial API. The TKeyClient and TKeySigner Go packages were successfully ported to TypeScript. The core functionality of the TKeyClient and TKeySigner was implemented into their own classes, resulting in a modular and maintainable codebase.

## 4.2 Frontend implementation

A minimalist user interface was implemented using Svelte. A landing page was created, prompting users to connect their TKey before proceeding to registration or login. Both the registration and login interfaces follow the same minimalist design principles. During login, the user must connect their TKey to a USB-C port on their computer. After choosing the correct port in the user's browser, an information page detailing some information about the specific key is displayed. Next, the user chooses whether to log in or register (Creating a new account). During log-in, the user has to enter their email address associated with their TKey device, and will then have to sign the message sent from the server by interacting with the touch sensor on the TKey. Subsequent steps after initiating login differ based on whether the user has registered with biometric authentication or not. If the user has registered using biometric authentication, the computer takes a picture of the user and performs a check comparing the biometric encoding from the taken picture to the encoding saved in the database. If the email, TKey and biometric encoding (if the user has a biometric encoding) exist and match in the database connected to the server, the user gets successfully logged in and is redirected to their profile page. Figure 4.1 depicts the journey the user must take during login without biometric authentication.

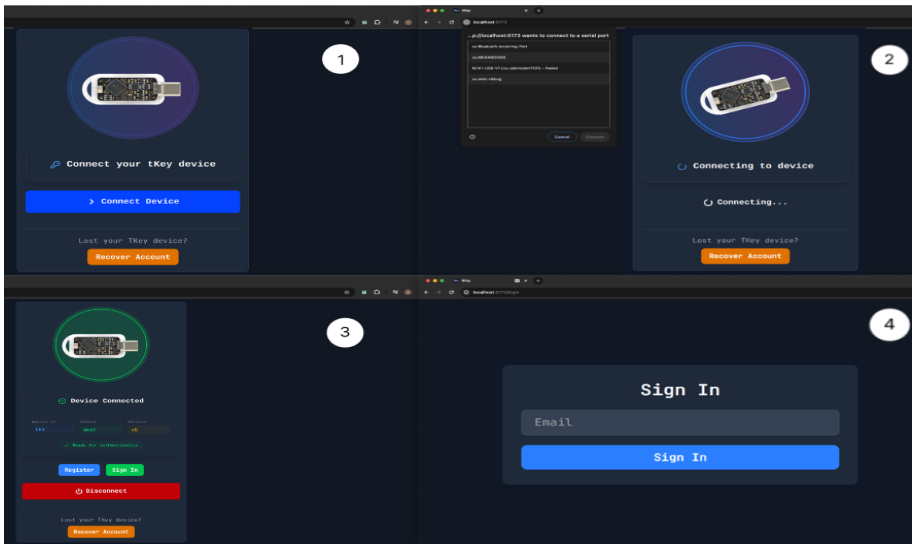


**Figure 4.1:** Description of sign in steps

During registration of a new TKey and email, the user follows the same steps of connecting to the TKey as during login. Instead of choosing the login option, the user pushes the register button and is prompted to enter an email address and take a picture for biometric authentication. After successfully taking a picture, entering a valid email-address and interacting with the TKey, an account is created and the user can then log in using their entered credentials. Figure 4.2 depicts the steps needed to successfully register a new account.

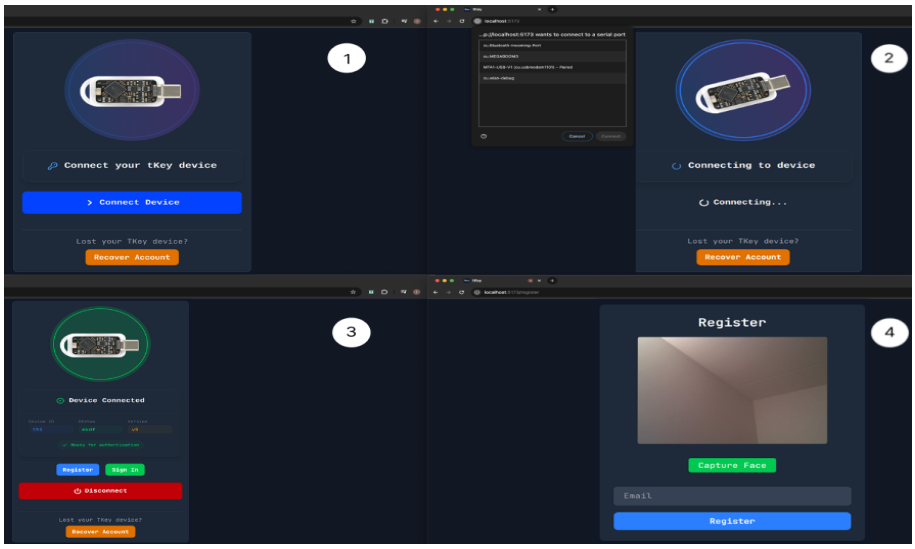


**Figure 4.2:** Description of register steps

## 4.3 Backend and Database Integration

The backend and database implementations achieved the following functionalities. Supabase stored and managed user data including public keys, facial encodings,

recovery codes, and user emails. Furthermore, forntend-backend communication was established and is operational through the tRPC framework. Supporting the necessary data exchange between the client and server. Finally, the account recovery feature, which utilizes hashed, one time recovery codes to ensure a secure recovery process.

## 4.4 Biometric Authentication

The biometric authentication utilizes a client-server architecture with a dedicated Flask microservice handling facial recognition processing. The system architecture separates concerns between frontend capture and backend processing. A Python microservice exposes two REST(Representational State Transfer) endpoints, `/capture-face` for enrollment during registration and `/verify-face` for authentication during login. This separation moreover isolates computationally processing from the main application server.

The implemented tolerance threshold of 0.4 for facial matching represented an optimal balance between security and usability. Testing with lower threshold values (0.1-0.2) resulted in excessive false negatives, while higher values (0.5-1.0) increased false positive rates.

The system was designed to automatically detect users enrolled with biometric authentication and presents the appropriate authentication interface. The error handling implementation provides appropriate fallback mechanisms when biometric authentication fails.

Further analysis confirmed that facial images are processed transiently. The backend deletes original images immediately after encoding, with only mathematical encodings stored in the database.

# 5
# Discussion

This section critically examines the outcomes of the implementation of a password-less authentication system using the Tillitis TKey hardware device. This section analyses the technical decisions made during development, evaluating how well the initial objectives were achieved, discuss the security implications and user experience considerations, compare the approach in this project with alternative authentication methods, and identify areas for future improvement. This discussion provides a comprehensive assessment of both the strengths and limitations of the implementation while positioning this work within the broader context of modern authentication systems. The analysis draws on both technical findings and relevant theoretical perspectives in the field of secure authentication.

## 5.1 Evaluation of Objectives

The team successfully implemented a back-end authentication server using Supabase and tRPC to handle TKey communication, user registration and challenge response validation. This demonstrates the viability of server-side management of TKey authentication, which is crucial for maintaining security and control over user data. TypeScript streamlined integration with the Svelte front-end. It also ensured type safety. This decision proved beneficial for development speed and code maintainability, allowing the team to focus on the core authentication logic. However, this approach introduced a dependency on third-party services like Supabase, while simplifying database management, may not be ideal for production environments requiring full control over data governance and potentially introducing long-term cost considerations.

The initial objective was to leverage the existing Tillitis Go packages by compiling them to WebAssembly for direct browser interaction with the Tkey. For this to be feasible in a web application the Go code would have to run in the client's web browser. This is achievable by compiling the libraries to WebAssembly and loading it with JavaScript. However, as WebAssembly lacks the standardized USB communication, the connection would have to be established using JavaScript. Although initially planned, WebAssembly integration was abandoned due to limitations in USB communication and Go runtime support. Consequently, porting both the TKeyClient and TKeySigner to TypeScript 3.3 enabled native browser execution. While this allowed for a more efficient development process and maintained feature parity, it introduced a critical limitation. Reliance on the Web Serial API restricted

compatibility to Chromium based browsers, particularly Chrome and Edge. This trade off prioritizes immediate functionality and development efficiency over universal accessibility, a gap that future work should address to ensure a broader user base can utilize the system.

Furthermore, the system incorporates a basic implementation of biometric authentication using facial recognition as a secondary authentication factor. While preliminary, this demonstrates the potential for multi factor authentication methods to enhance security. Moreover, it highlights the need for careful consideration of privacy concerns and the complexities involved in securely managing biometric data.

## 5.2 User Experience and Adoption Considerations

Passwordless authentication is designed to streamline the login process and reduce reliance on traditional password-based systems. In contrast to methods that require users to create, remember, and manage complex passwords, the implemented system has the potential to improve both security and usability. By eliminating password management, it also mitigates common vulnerabilities, such as susceptibility to phishing attacks.

As noted in Section 5.5, the reliance on the Web Serial API limits browser compatibility, which directly impacts accessibility. Users who rely on unsupported browsers are currently excluded from using the system.

Several factors can influence the adoption of the passwordless authentication system developed in this project. The requirement for a hardware device the Tillitis TKey introduces a cost and a potential learning curve for users. Users must be willing to purchase and learn how to use the TKey, which may be a deterrent.

## 5.3 Comparison with Software-based Authentication Methods

The proposed authentication system that integrates Tilitis TKey and biometric authentication offers a secure user authentication approach. Compared to existing solutions such as Microsoft Authenticator, this system demonstrates both strengths and limitations.

One primary strength is the inclusion of Tilitis TKey, it enhances the security of the system by introducing a hardware-based authentication factor. Hardware tokens like TKey are more resistant to common attack vectors such as phishing and MITM (man-in-the-middle) attacks[15], which often exploit weaknesses in software-based authentication systems. By requiring both the possession of the TKey device and the successful verification of biometric data the system achieves a multi-factor

authentication (MFA) framework. This combination of factors increases the security threshold for potential attackers as they would need to compromise both the physical token and the user's biometric data to gain unauthorized access. Biometric authentication ties access to unique physical traits, making it harder to replicate or share than passwords or TOTP-based (time-based one-time password) methods. This reduces the risk of credential theft or misuse as biometric data is not easily transferable.

Another notable advantage of this solution is its potential to decentralize sensitive data. While facial encodings are currently stored in a database, integrating writable memory into the TKey would allow biometric data to be stored locally or securely hashed on the device, significantly reducing the risk of large-scale data breaches.

Despite these strengths, the system also presents limitations when compared to existing solutions like Microsoft Authenticator. One significant drawback is its reliance on hardware. The requirement for users to possess a physical TKey device introduces a dependency that does not exist in software-only solutions. While Microsoft Authenticator operates on smartphones, which are already widely owned and used, the TKey device represents an additional item that users must carry, maintain, and safeguard. The loss, theft, or damage of the TKey device could result in users being locked out of their accounts, necessitating the implementation of desired recovery mechanisms to address such scenarios. While biometric authentication is generally secure, it is not without limitations. Certain users, such as individuals with disabilities or facial injuries, may find these systems inaccessible. Software-based solutions offer broader inclusivity. Moreover, since this system relies on standard image-based facial recognition rather than more advanced IR-based methods used in smartphones, it may be vulnerable to spoofing if an attacker can mimic the appearance of the TKey owner.

Cost and scalability also represent potential challenges for the proposed system. Hardware-based solutions like TKey are inherently more expensive to deploy and scale compared to software-only systems. Software-based authenticators, which are freely available on smartphones, offer a lower entry barrier. The additional expense of procuring and distributing TKey devices may limit the adoption, particularly in resource-constrained environments.

In conclusion, the proposed authentication system enhances security by combining biometrics with hardware-based protection. However, its dependence on hardware, potential biometric limitations and issues of cost and accessibility present notable challenges. The TKey also introduces a single point of failure, where a loss of the device may disrupt access and necessitate recovery mechanisms. This could introduce new security risks. Careful evaluation of these limitations is essential to determining the system's appropriateness for specific applications.

## 5.4 Comparison with existing web authentication standards

The current implementation of the TKey hardware, along with the solution developed in this project does not follow any existing standards for hardware based passwordless authentication, or web based multi factor authentication. Notably it does not conform to FIDO2 or the WebAuthn APIs. These standards are maintained by the FIDO Alliance and the W3C and are widely supported by browsers and operating systems. WebAuthn provides a JavaScript API that allows websites to interact with authenticators for secure login using public key cryptography[16]. FIDO2 also includes the Client to Authenticator Protocol (CTAP), which defines how external authenticators communicate with clients like browsers or operating systems[17].

If the TKey was to adopt these standards in the future with the release of a new version of the hardware, it could become compatible with most major browsers and operating systems. This would likely require change to the existing TKey firmware. For example, the TKey would need to support CTAP2 so that it can exchange messages with a web browser in a way that conforms to the WebAuthn specification. This would possibly include the ability for secrets to be generated and stored on the device, rather being generated each time as is currently the case. Additionally, it could include the ability to have integrated biometric scanners, such as a fingerprint scanner integrated into the device, with data stored locally and securely.

Were these standards to be implemented, the TKey could become a feasible option for use in current applications supporting these standards, making the possible market for the device larger. Notably, the TKey promotes openness through its fully open source hardware and firmware, setting it apart from other existing options following these standards. One example of a device implementing these is the YubiKey series of authenticators from Yubico[18]. While these devices do support the standards mentioned and are able to be widely used in web applications, both the hardware and firmware of the devices are closed source. In contrast, the TKey allows full access to its firmware and hardware schematics. This openness allows users and organizations to audit the devices used for authentication, allowing for a possible higher level of trust. If implemented, these standards would allow the TKey device to offer a balance between standards compliance and interoperability, and an openness that allows for high levels of trust.

## 5.5 Limitations and future improvements

As previously mentioned in section 3.2, there is currently a limited support for different browsers in the web application due to the project relying on the Web

Serial API. This means that only around 80% of all desktop users' browsers are supported as of writing [14]. Future improvements of the product would include support for other major browsers using other engines, like Safari and Firefox. This could be achieved by either using a different method for communication than the WebSerial API, or the API gaining support in other browsers.

Another limitation of the project is lack of formal security audit, which was deemed to be outside the scope of this project. The Tillitis TKey is treated as a trusted component from the manufacturer, and the web application serves as a proof of concept rather than a production ready authentication service that is equipped to handle real world user data and account protection.

Another limitation of the current application is the operating system compatibility. The current version works well on Linux and Windows, but there are issues with the application on macOS where the hardware communication does not work as expected. This is likely due to the way this operating system handles USB connections, and further investigation is needed to solve the issue.

There is additionally a lack of user key management features in the web application. Currently it is not possible for a user to easily add additional keys for the account or easily remove unused ones. This is something that should be considered for further development of the application.

## 5.6 Future Work

This section outlines the potential directions for future development and enhancements of the implemented system.

### 5.6.1 Writable Memory on TKey Hardware

The current Tillitis TKey does not allow any data on the device to be modified or stored once its firmware is loaded. Introducing a small, secure non-volatile memory region would enable new features without requiring a complete redesign. For example, the device could:

- Store multiple pairs of cryptographic keys and switch among them as needed.
- Maintain a monotonic counter of signing operations to detect unusual or duplicated use.
- Record a short audit log of recent operations so that users or administrators can review activity offline.

To support these capabilities, the USB command set would need extensions for authenticated read/write operations, each gated by a user presence check. On the hardware side, a dedicated secure chip (for example, a small flash partition or an external secure element) would be reserved and protected by the bootloader. On

the host side, client tools must be able to upload initial data to this memory during provisioning and later retrieve it for backup or auditing purposes [19].

### 5.6.2 Advanced key management

With writable memory in place it would be possible to manage key life cycles directly on the TKey. A new key could be generated on the device and marked as the active key while the old key remains until it is no longer needed. Secure backup of key data could be done by exporting encrypted snapshots of the writable area so that a token can be replaced without losing access. It would also be possible to add a revocation flag so that the server can refuse tokens that have been marked invalid. All of these ideas depend on having a safe way to write to the protected storage area on the device [19].

### 5.6.3 Device level policies

Another future direction is to allow each TKey to carry its own security settings. A user might wish to require a personal code each time the device is used or to limit the number of operations per session. These settings would be stored in the writable area and enforced by the device firmware before any cryptographic action is performed. Client applications could then check the policy and avoid asking for a code again if the user has already unlocked the device during the same session. This approach could lead to new ideas such as timed unlock periods or approvals for specific websites without changing the entire firmware [19].

### 5.6.4 Secure rollback protection

To prevent an attacker from loading an older version of the firmware with a known weakness it would help to store a version counter in the writable memory. The device firmware would refuse to run if this counter is not at the expected level and would only allow an update signed by a trusted key to move to a lower version number. This measure would block attacks during the update process and ensure that only approved code ever runs on the device [19].

### 5.6.5 More secure registration flow

The current account registration flow contains a few possible security vulnerabilities that could possibly be used as attack surfaces for malicious actors, as well as risking bugs in the registration process, preventing users from effectively creating accounts.

The current way of registering users does not use any atomic database operations, like transactions in PostgreSQL. Where nothing is added to the database before all operations have succeeded [20]. This is because of the limitations of the database provider used. The registration flow consists of multiple steps, such as adding the user with an email, registering a public key generated from the TKey, adding face encoding for biometric authentication, and adding one-time recovery codes for the user in case of a lost key. All of the steps can fail in different ways,

and means that the registration should not complete.

In the current implementation of the application, the database is updated with information after each step, leaving it in an incomplete state during the registration process. In the event of a failure, there is a risk that the database is permanently left in an unwanted state. To mitigate these risks, the program currently keeps track of any changes made and tries to roll them back in case of any potential errors.

Although the current solution appears to work in the limited testing that has been done, there remains a risk that certain rollback scenarios may not be handled correctly or that some errors may not be properly detected. This implementation represents a potential point of fragility, especially as the application evolves and additional features are introduced. These risks could have been mitigated by adopting a more suitable database solution, which would reduce the likelihood of future bugs and improve the overall robustness of the system.

A potential security vulnerability could arise with the user registration and other account related activities, e.g. account recovery, not being atomic. During these operations, the database is momentarily in a state that is not what is intended, potentially leaving the account vulnerable to exploits depending on the action.

### 5.6.6 Standalone authentication service

The current application serves as a good proof of concept that it is very possible to have a service using hardware based passwordless authentication running directly in a web browser. Although a good demonstration of the capabilities of today's web browsers, it is limited in every day usability for what would be considered a normal web user.

To further develop this, the application could become a standalone OAuth 2 service provider that can be used in other applications [21]. This would allow more users to use hardware keys either as a second factor, or for entirely passwordless authentication in more parts of their every day internet activities, thus making the project more useful than in it's current state. The service would allow users to use the same account and hardware seamlessly across many sites. Without give any unnecessary personal details or passwords to the sites they make accounts for and use. This would in addition to reducing the risk of passwords being leaked, allow users to stay more private by minimizing the exposure of personal data to different parties and thus increasing online privacy and possibly reducing the risk of personal information being leaked.

To facilitate the development of a standalone OAuth 2 service based on the applications developed in this project, several architectural changes would be required. The findings in this project with regards to the feasibility of safely communicating with the hardware keys through a web browser are valuable

could be used for further development. As earlier mentioned, the project has made architectural choices that prioritize rapid development, which can limit the scalability. A library for UI components that use the key takeaways from the front end part of the application could be developed to allow other applications to use in their front ends. A new separate application without a tight coupling to a UI would be developed to handle secure user registrations, authentication, and account management.

### 5.6.7 Password manager

This thesis investigates the potential of replacing traditional password-based authentication with alternative, more secure methods. Despite advancements in passwordless authentication, the majority of widely used applications still rely on passwords and do not support any type of hardware-based passwordless authentication. To increase the security, users should not re-use passwords but instead use a new unique and strong password for every account. For this to be possible, some people use password managers to generate and store the unique passwords. Several password managers themselves use a password to protect access, which has the same drawbacks as discussed in this thesis.

An option for this project moving forward would be to further develop the application into a password manager that stores the users' unique passwords and protects those by using either passwordless authentication with a secure hardware key, or with the hardware key as an additional factor to a password.

Many parts of the development process would be the same as for creating a standalone authentication service, but with a focus on storing passwords in a secure manner, and developing new clients as browser extensions and mobile apps for users to access their passwords.

## 5.7 Ethical aspects

This section addresses the ethical considerations associated with the design, implementation and deployment of the proposed system.

### 5.7.1 Privacy of biometric data

Biometric data such as face models are considered sensitive personal information under the General Data Protection Regulation. It is important to store only the numerical model rather than raw images so that if the system is breached no actual pictures are revealed. Even encoded models must be encrypted and protected with strict access controls to comply with privacy rules [22].

### 5.7.2 Informed user consent

Before collecting any biometric data, the system should explain the purpose and intended use of the data. One must give permission and be able to withdraw and must be able to withdraw consent at any time. The interface should allow removal of biometric data on request and confirm to the user when data has been deleted [22].

### 5.7.3 Accessibility and fairness

Not all users have a camera or wish to use face recognition. Alternative methods, such as personal codes or recovery procedures, must be provided to ensure that users are not permanently locked out of the system. Biometric methods can also work differently for people with different skin tones or ages so the system needs regular testing and updates to keep error rates low for all groups [22].

### 5.7.4 Data minimization and retention

Only the data that is strictly necessary should be collected and it should be kept only as long as it is useful. For example biometric models could be deleted after a period of inactivity unless the user opts in for longer storage. Device logs should be erased once they have served their audit or support purpose [22].

Since the TKey design and software are open source anyone can inspect them for backdoors. However any writable memory or update path must also be governed by public signing keys and reproducible builds. This transparency helps users verify that the code running on their device is genuine and has not been tampered with [19].

### 5.7.5 Platform support

As Stated earlier, the client-server application does not support macOS, which raises ethical considerations related to accessibility and inclusivity. While this limitation is primarily due to the time constraints of this project, it nonetheless excludes a segment of users who rely on Apple devices. This may disproportionately affect users who prefer macOS for accessibility features and workflows. From an ethical standpoint, such exclusion can be seen as a form of digital inequality if not explicitly stated why the support is missing. It may also be precived as implicit bias in design decisions that prioritizes development convenience over user diversity.

# 6
# Conclusion

The implemented system successfully demonstrates the feasibility of a secure, passwordless authentication method using the Tillitis TKey. Combining public key cryptography, browser based hardware communication, and biometric verification, demonstrates the feasibility of replacing traditional password-based systems in controlled environments.

The prototype confirms that challenge response authentication via TKey is both technically viable and practically usable in a web based environment. Moreover, the integration of facial recognition as a second authentication factor further strengthens security without adding significant complexity for users.

Despite some limitations including restricted browser support, lack of formal security audits, and no current support for macOS the results provide a solid foundation for future development.

This project contributes valuable insights for both academic research and industrial applications. Hardware backed authentication systems like this have potential applications in industries such as online banking, healthcare systems, government services, and secure corporate environments.

Future work should focus on expanding browser and operating system compatibility. Conducting formal security assessments, improving key management capabilities like key rotation and revocation and Refining the user experience for broader public adoption.

Overall, the project validates the concept of using modern hardware security devices to create safer and more accessible web authentication solutions.

# Bibliography


[1] Swedish Civil Contingencies Agency (MSB). (2025) The nis directive. Accessed: 2025-04-08. [Online]. Available: https://www.msb.se/sv/amnesomraden/informationssakerhet-cybersakerhet-och-sakra-kommunikationer/krav-och-regler-inom-informationssakerhet-och-cybersakerhet/nis-direktivet/

[2] TechTarget Contributor. (2025) Public key. Accessed: 2025-04-29. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/public-key

[3] T. AB, "Key derivation function overview," Presentation provided by Tillitis AB, 2025, internal presentation describing the public/private key derivation used in the TKey.

[4] Tillitis_AB. (2025) Tkey product page. Accessed: 2025-04-29. [Online]. Available: https://tillitis.se/products/tkey/

[5] Svelte contributors. (2025) Svelte: Web development for the rest of us. Accessed: 2025-04-29. [Online]. Available: https://svelte.dev/

[6] Supabase Inc. (2025) Supabase | the open source firebase alternative. Accessed: 2025-04-29. [Online]. Available: https://supabase.com

[7] MDN Web Docs contributors. (2024) Web serial api. Last modified Dec 2, 2024; Accessed Apr 29, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API

[8] National Institute of Standards and Technology, *Federal Information Processing Standard (FIPS) 180-4: Secure Hash Standard*, 2015, accessed: 2025-04-29. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf

[9] A. Geitgey, *Face Recognition*, 2017, accessed: 2025-04-29. [Online]. Available: https://face-recognition.readthedocs.io/en/latest/readme.html

[10] W3 WebAssembly Community Group. (2025) Webassembly. Accessed: 2025-04-30. [Online]. Available: https://webassembly.org/

[11] IETF. (2025) Webassembly | mdn. Accessed: 2025-04-30. [Online]. Available: https://developer.mozilla.org/en-US/docs/WebAssembly

[12] Tillitis AB. (2024) Tkey hardware design. Accessed: 2025-05-19. [Online]. Available: https://dev.tillitis.se/hw/

[13] ——. (2024) Protocols | tkey developer handbook. Accessed: 2025-05-19. [Online]. Available: https://dev.tillitis.se/protocol/

[14] Statcounter. (2025) Desktop browser market share worldwide. Accessed: 2025-04-29. [Online]. Available: https://gs.statcounter.com/browser-market-share/desktop/worldwide

[15] Imperva. (2025) What is mitm (man in the middle) attack. Accessed: 2025-04-29. [Online]. Available: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

[16] W3C. (2021) Web authentication: An api for accessing public key credentials - level 2. Accessed: 2025-05-19. [Online]. Available: https://www.w3.org/TR/webauthn-2/
[17] FIDO Alliance. (2025) Fido2 - fido alliance. Accessed: 2025-05-19. [Online]. Available: https://fidoalliance.org/fido2/
[18] W3C. (2025) Discover yubikey 5 | authentication for secure login | yubico]. Accessed: 2025-05-19. [Online]. Available: https://www.yubico.com/products/yubikey-5-overview/
[19] Tillitis AB. (2023) On tkey key generation. Accessed: 2025-04-29. [Online]. Available: https://tillitis.se/blog/2023/03/31/on-tkey-key-generation/
[20] PostgreSQL. (2025) Postgresql: Documentation: 17: 3.4. transactions. Accessed: 2025-04-29. [Online]. Available: https://www.postgresql.org/docs/current/tutorial-transactions.html
[21] IETF. (2025) Web authorization protocol (oauth). Accessed: 2025-04-29. [Online]. Available: https://datatracker.ietf.org/wg/oauth/about/
[22] European Union. (2016) General data protection regulation (gdpr). Accessed: 2025-04-29. [Online]. Available: https://gdpr-info.eu/

# A

# Code

```
async connect(): Promise<void> {
    try {
      this.port = await navigator.serial.requestPort();
      if (!this.port) throw new Error('Failed to get
      serial port');
      await this.port.open({
        baudRate: 62500,
        dataBits: 8,
        stopBits: 1,
        parity: 'none'
      });
      this.isConnected = true;
    } catch (error) {
      console.error('Failed to connect to device:', error);
      throw error; // Re-throw to propagate the error
    }
}
```

**Listing A.1:** Function `connect` for initiating connection with TKey

```
public async write(data: Uint8Array): Promise<void> {
    if (!this.port?.writable) throw new Error('Port not
    writable');

    const writer = this.port.writable.getWriter();
    try {
      await writer.write(data);
    } finally {
      writer.releaseLock();
    }
}
```

**Listing A.3:** Function `write` for writing frames to TKey

```
public createFrame(command: Command, id: number):
Uint8Array {
    if (id > 3) throw new Error('Frame ID must be 0..3');

    // Calculate total frame size: header byte + command
    length
    const frameSize = 1 + getCmdLength(command.cmdLen);
    const frame = new Uint8Array(frameSize);

    // Construct header byte:
    // Bit 7: Reserved (0)
    // Bits 6-5: Frame ID
    // Bits 4-3: Endpoint
    // Bit 2: Response NOK (0 for commands)
    // Bits 1-0: Command length
    frame[0] = ((id & 3) << 5) |
      ((command.endpoint & 3) << 3) |
      (command.cmdLen & 0x03);

    // Set command code
    frame[1] = command.code;

    return frame;
}
```

**Listing A.2:** Function `createFrame` for creating a frame for a command

```
const FWCommands = {
    GetNameVersion: {
        code: 0x01,
        name: 'cmdGetNameVersion',
        cmdLen: CmdLen.CmdLen1,
        endpoint: Endpoint.DestFW
    },
    RspGetNameVersion: {
        code: 0x02,
        name: 'rspGetNameVersion',
        cmdLen: CmdLen.CmdLen32,
        endpoint: Endpoint.DestFW
    },
    ...
```

**Listing A.6:** Structure of `TKeyClient` Firmware Commands

```
function readFrame(expectedResponse, expectedId):
    if port is not readable:
        throw Error

    buffer = []
    set timeout for 10 seconds

    while true:
        chunk = read from port
        if chunk is empty:
            break

        append chunk to buffer

        if buffer.length < 1:
            continue

        header = buffer[0]
        length = 1 + getCmdLength(header & 0x03)

        if buffer.length < length:
            continue

        id       = (header >> 5) & 0x03
        endpoint = (header >> 3) & 0x03
        isNOK    = (header >> 2) & 0x01
        code     = buffer[1]

        if isNOK:
            throw Error("NOK response")

        if id != expectedId:
            throw Error("Unexpected ID")

        if endpoint != expectedResponse.endpoint:
            throw Error("Unexpected endpoint")

        if code != expectedResponse.code:
            throw Error("Unexpected response code")

        cancel timeout
        return buffer

    on timeout:
        throw Error("Read timeout")
```

**Listing A.4:** Psuedocode of function `readFrame` for reading the response frames sent by TKey

```
  async getNameVersion(): Promise<{ name0: string; name1:
  string; version: number }> {
    const id = 2;
    const frame =
    this.createFrame(FWCommands.GetNameVersion, id);

    await this.write(frame);
    const response = await
    this.readFrame(FWCommands.RspGetNameVersion, id);

    // Parse name version response (matches Go
    implementation)
    const decoder = new TextDecoder();
    const name0 = decoder.decode(response.slice(2,
    6)).replace(/\0/g, '');
    const name1 = decoder.decode(response.slice(6,
    10)).replace(/\0/g, '');
    const version = new DataView(response.buffer,
    response.byteOffset + 10, 4).getUint32(0, true);

    return { name0, name1, version };
  }
```

**Listing A.5:** Example of an encapsulated firmware command ported from the TKeyClient Go package

```
const AppCommands = {
    GetPubkey: {
        code: 0x01,
        name: 'cmdGetPubkey',
        cmdLen: CmdLen.CmdLen1,
        endpoint: Endpoint.DestApp
    },
    RspGetPubkey: {
        code: 0x02,
        name: 'rspGetPubkey',
        cmdLen: CmdLen.CmdLen128,
        endpoint: Endpoint.DestApp
    },
    ...
```

**Listing A.7:** Structure of `TKeySigner` Application Commands