Universität Siegen

Naturwissenschaftlich-Technische Fakultät

Master's Thesis
submitted for the degree of Master of Computer Science

Foundations of TOTP and Passkeys (with and without Hardware Security Tokens). User Awareness with an Interactive Web-Based Demonstrator.

Coaches	Prof. Bernhard Esslinger
	Dr. Martin Franz
First examiner	Prof. Bernhard Esslinger
Second examiner	Prof. Dr. Roland Wismüller

Author	Abdullah Abbas
Submitted	2025-12-15
Updated	2026-03-28

Kurzzusammenfassung

Das Ziel dieser Masterarbeit ist es, die Grundlagen sowie die praktische Bedeutung moderner Methoden der Zwei-Faktor-Authentifizierung aufzuzeigen. Dazu wird erläutert, was zeitbasierte Einmalpasswörter sowie hardwarebasierte und softwarebasierte Passkeys auszeichnet und welche Rolle sie heutzutage bei der Absicherung von Authentifizierungsprozessen spielen. Dabei wird auch auf die Herausforderungen hinsichtlich der Nutzung solcher Verfahren eingegangen, sowie auf die Schwierigkeit, die dahinterliegende Theorie verständlich zu erklären.

Ein wesentlicher Bestandteil dieser Arbeit ist die Konzeption und Implementierung eines interaktiven webbasierten Demonstrators für die Nutzeraufklärung. Der Demonstrator zeigt, wie zeitbasierte Einmalpasswort-Codes erzeugt und überprüft werden und wie passkey-basierte Authentifizierung in der Praxis funktioniert. In Zukunft soll der Demonstrator Teil von CrypTool-Online werden.

Die Zwei-Faktor-Authentifizierung macht sowohl in der Theorie als auch in der Praxis Fortschritte, was sich etwa in der Entwicklung interaktiver Lernwerkzeuge und praxisnaher Einrichtungsanleitungen zeigt. Allerdings schränken Herausforderungen wie die Komplexität moderner Authentifizierungsverfahren und das Fehlen klarer, allgemein verständlicher Lehrmaterialien die breite praktische Anwendung immer noch stark ein.

Abstract

The goal of this master's thesis is to examine the foundations and the practical relevance of modern two-factor authentication (2FA) methods. The focus lies on time-based one-time password (TOTP) as well as hardware-based and software-based passkeys, which are explained in terms of their underlying principles and their role in strengthening authentication processes. The challenges associated with the use of such authentication methods are also addressed, as well as the difficulty in explaining the theory behind them.

A key component of this work is the design and implementation of an interactive web-based demonstrator for user awareness. The demonstrator illustrates how TOTP codes are generated and verified, and how passkey-based authentication works in practice. In the future, the demonstrator will become part of CrypTool-Online (CTO).

User awareness of 2FA has improved through educational resources, as indicated by the development of interactive learning tools and practical setup guides. However, the complexity of authentication methods and lack of clear educational materials still limit widespread adoption.

Contents

1 Introduction

This chapter introduces the context and relevance of examining 2FA methods. It begins by presenting the background and motivation for this thesis, followed by a clear definition of the problem being addressed. The chapter then outlines the objectives of this thesis and concludes with an overview of its overall structure.

1.1 Background and motivation

This thesis examines modern authentication mechanisms, focusing on 2FA implementations using TOTP as the second factor, and on hardware-based and software-based passkeys, alongside the development of an interactive web-based demonstrator for enhancing user awareness. Authentication serves as the fundamental method for verifying user identity in digital systems, yet password-based authentication continues to dominate despite well-documented vulnerabilities. Over 80% of data breaches at organizations involve compromised or weak passwords [1]. This problem keeps occurring mainly because of user behavior patterns. Approximately 38% of users reuse the same password across different websites, while 21% modify existing passwords to create new ones with highly predictable patterns [2]. Approximately 24 billion passwords were exposed through data breaches in 2022, representing a 65% increase compared to exposures in 2020 [3]. Table 1 summarizes these critical password security statistics, where each row represents a specific security issue, with columns indicating the percentage or absolute number of affected users and the corresponding source reference.

Security Issue	Percentage/Number	Source
Data breaches involving weak and stolen passwords	80%	[1]
Users who reuse exact same password	38%	[2]
Users who modify existing passwords	21%	[2]
Passwords exposed in 2022	24 billion	[3]

Table 1: Password-related security statistics highlighting password vulnerabilities

2FA emerged as a promising solution to strengthen password-based systems because users must provide an extra form of authentication in addition to their password. The adoption of 2FA has grown since major service providers such as Google, GitHub, and various financial institutions began offering it as an option. However, as shown in Table 2 on the following page, adoption rates remain low. An empirical study from 2015 [4] examining

over 100,000 Google accounts found that no more than 6.4% of users had enabled 2FA, despite its availability. This limited user adoption continues despite the enhanced security benefits that 2FA provides, suggesting that factors beyond mere availability influence user decisions. Similar research [5] estimates Google's 2FA adoption between 6-10% even after years of availability and active promotion by the platform. Despite these low historical adoption rates, major service providers have moved toward making 2FA mandatory rather than optional. For example, GitHub began requiring all developers who contribute code on its platform to enable 2FA in a phased deployment starting in March 2023, with the aim of completing full adoption by the end of 2023 [6]. Recent data from the Bundesamt für Sicherheit in der Informationstechnik (BSI) [7] illustrates that 2FA usage among German internet users dropped from 42% in 2023 to approximately 34% in 2025 . This decline demonstrates the need for continuous awareness efforts to improve security practices [7].

Study	Year	Adoption Rate	Platform/Context
Petsas et al. [4]	2015	6.4%	Google (100,000+ accounts)
Oberheide [5]	2015	6–10%	Google
BSI [7]	2025	34%	German internet users

Table 2: Two-factor authentication adoption rates across different studies

The usability of authentication systems plays a critical role in their practical deployment. 2FA usability depends primarily on three factors: simplicity of use, cognitive demands, and perceived trust, as identified in a research examining 219 participants using three 2FA methods [8]. Several problem areas and specific issues that negatively affect human performance during 2FA configuration were identified in a usability evaluation of Google's 2FA setup process, confirming concerns about the expense of usability [9]. Similarly, the complexity the users face during configuring these authentication systems was highlighted in a two-week usability study [10] involving 72 participants testing five common 2FA methods that gathered both quantitative and qualitative data.

1.2 Problem statement

Despite the security advantages of 2FA, widespread adoption remains low due to several challenges. A fundamental problem is that many users do not clearly understand how these authentication methods work, which results in disabling 2FA. When users do attempt to configure 2FA, they often face major difficulties. Research on how people perform during Google's 2FA setup process found usability issues that reduce user success rates, confirming concerns that better security can sometimes make a system harder

to use [9]. These setup difficulties not only discourage user adoption but also result in poorly configured implementations that may frustrate users.

Existing educational resources on 2FA, particularly those explaining TOTP as a second factor, are considered insufficient (see Chapter 2.1). Current online tutorials and documentation provided by major platforms tend to be either too technical for average users or too simple to give a real understanding of how the 2FA mechanisms work. Furthermore, interactive tools that allow users to safely experiment with and explore authentication concepts remain scarce. Such interactive tools could greatly improve users' understanding and confidence when setting up 2FA.

In addition, if users set up 2FA incorrectly, they can end up locked out of their own accounts. This risk represents a major barrier to the adoption of 2FA because many users fear losing access to their important personal accounts such as email, banking, cloud storage, or work-related accounts. The setup process can also differ in difficulty depending on the 2FA method being used. For example, hardware tokens require users to manage a physical device and think about backup options.

1.3 Objectives and scope of the thesis

The main goal of this thesis is to explore modern 2FA methods, specifically TOTP-based 2FA and passkey-based authentication, and demonstrate how these methods provide an extra level of protection for personal accounts associated with online services. This is achieved through the development of an interactive web-based demonstrator that aims to help users gain an enhanced understanding of the fundamentals of 2FA. The web-based demonstrator supports both TOTP as a second factor and hardware-based and software-based passkeys.

1.4 Outline

Chapter 2 covers the related work and the history of different authentication methods. In addition, the chapter includes a review of existing educational resources on 2FA, followed by a brief history of authentication evolution.

Chapter 3 presents the theoretical foundations required to understand the construction of this thesis by covering the authentication principles and methods in detail. In addition, common attack vectors are covered in this chapter.

Chapter 4 presents the design and implementation of the interactive web-based demonstrator.

Chapter 5 presents the evaluations and results of the user study carried out after interacting with the demonstrator.

Chapter 6 concludes the thesis by summarizing the key findings, analyzing the results, and providing an outlook on potential future enhancements to the web-based demonstrator.

2 Related work and history

This chapter presents an overview of existing educational resources on 2FA. In addition, a brief history of authentication evolution is presented at the end of this chapter.

2.1 Existing 2FA educational resources

The educational materials available for 2FA include many different types of resources such as platform-specific guides, academic tutorials, and training programs. Major service providers like Google, Apple, and banks offer their own 2FA setup guides as part of their security documentation.

Google provides detailed documentation through its support pages covering setup procedures and verification methods [11]. The documentation explains different authentication methods such as Google prompts, authenticator apps using the TOTP standard, backup codes, and short message service (SMS) verification. Similarly, Apple offers guidance on 2FA through its official support pages [12], explaining the authentication process, trusted device management, and account recovery procedures.

In the banking sector, the National Institute of Standards and Technology (NIST) has published technical standards known as authenticator assurance level (AAL) for digital identity verification [13], which banks follow in the design process of their security systems. AAL includes three levels: AAL 1, which permits a single authentication factor such as passwords; AAL 2, which requires the combination of two different authentication factors like a password combined with an SMS code; and AAL 3, which relies on hardware-based security devices that are resistant to phishing attacks [13] (see Chapter 3.4.1 on page 32). Banks are required to develop educational programs for customers about security threats and safe authentication practices [14]. A research study showed that the value of these educational programs diminishes when banks fail to update them to mitigate emerging threats [14].

Research studies [9, 15] conducted about the accessibility and quality of current 2FA educational materials showed significant limitations. Most of the online platform guides focus on conciseness rather than comprehensiveness, resulting in insufficient documentation that fail to provide users with a clear explanation of the underlying security principles of 2FA. A systematic analysis conducted by Prümmer et al. [16] found that, although training programs provide users with knowledge of security, they have limited impact on user behavior. This gap between knowledge and behavior represents a major challenge to increase the adoption of 2FA.

The current state of 2FA educational resources can be characterized by several limitations. First, most available materials adopt either an overly technical approach that assumes prior knowledge, or an overly simplified approach that omits critical security considerations. Second, the lack of interactive tools remains a gap in existing educational resources. Finally, 2FA educational content is often spread out across different platforms and service providers, making it hard for users to understand. Most service providers focus only on their own implementation regarding 2FA. This approach does not give users knowledge that can be applied across different services.

2.2 Focused literature review

For the purpose of understanding how 2FA operates, it is necessary to analyze several types of sources, including technical standards that define how authentication systems operate.

Foundational technical standards The technical specifications that shape modern 2FA originate from the Internet Engineering Task Force (IETF), an international standards organization. HMAC-based one-time password (HOTP) was established as a foundational standard through RFC 4226, released in December 2005 [17]. RFC 6238 followed in May 2011, introducing the TOTP algorithm [18]. Both standards resulted from a collaborative work within the initiative for open authentication (OATH)[1] to develop specifications concerning system compatibility [18]. These RFC standards serve as authoritative references to follow when implementing one-time password (OTP) systems as they cover different requirements, including security considerations and deployment parameters that must be followed to create authentication systems compliant with these specifications [18].

For passwordless authentication, the primary literature comes from the World Wide Web Consortium (W3C) and the fast identity online (FIDO) alliance. The W3C published web authentication (WebAuthn) as an official recommendation in March 2019, with an updated level 2 version released in April 2021 [19]. The WebAuthn specification defines how web browsers and platforms can support public key-based authentication [19].

Regulatory and government guidelines In Germany, the BSI provides technical guidelines for government and business use. The BSI technical guideline TR-03107 defines requirements for digital identities and trusted services in online government contexts [20]. According to the website [21], organizations seeking approval for authentication systems must demonstrate compliance with TR-03107-1 and the related TR-03147 guideline.

[1] https://openauthentication.org

These government technical guidelines differ from technical standards by addressing both the technical implementation and the organizational procedures, including risk assessment frameworks, and assurance levels for security purposes. TR-03107 defines three trust levels that organizations must consider [20]. Such guidelines provide essential information to help users understand how authentication technologies are evaluated.

In the United States, similar guidelines are provided by the NIST. As mentioned in Chapter 2.1, the NIST technical standards establish authentication levels that influence how banks implement 2FA.

Literature gaps and research opportunities The existing literature provides strong technical foundations through standards documents that can help users improve their security knowledge. However, several gaps remain. First, most authentication education research focuses on general awareness rather than specifically addressing 2FA concepts. Second, the available literature on authentication consists of platform-specific documentation rather than educational materials. Finally, there is a lack of interactive educational tools that allow users to experiment with different 2FA methods.

These gaps in the literature highlight the need for interactive educational demonstrations that integrate standardized technical content with learning experiences.

2.3 Brief history of authentication evolution

This section examines how authentication systems progressed from basic passwords to 2FA approaches.

2.3.1 Password-only era

Massachusetts Institute of Technology (MIT) deployed the first computerized password mechanism in 1961 within the compatible time-sharing system (CTSS) [22]. The CTSS platform required a method allowing various researchers to use one machine while maintaining separate file storage [22, 23]. Fernando Corbató, who presented the platform, described how several terminals needed configuration for various users with individual private documents, making passwords for each person appear as a logical solution [23].

In 1966 [23], a graduate student at MIT Allan Scherr discovered a technique to output the master password document that allowed him to acquire the complete password access. This was the first password hack in history and it demonstrated the core limitation of single-factor authentication (SFA) relying on passwords only.

2.3.2 Hardware tokens emergence

RSA SecurID emerged as one of the most widely used hardware token platforms, controlling more than 70% of the 2FA marketplace by 2003 after manufacturing 25 million units [24]. These hardware tokens generated 6-digit token codes at 30-second intervals, which account holders entered together with their personal identification number (PIN)[2] to form single-use passcodes during OTP platform entry [26]. RSA devices employed time-based synchronization and did not require users to install software or maintain accounts, because it ran on a built-in power source [27]. The hardware token framework brought possession-oriented authentication concepts. Account holders were required to provide both a knowledge factor, i.e., their PIN, and a possession factor, i.e., the RSA SecurID hardware token.

2.3.3 SMS-based 2FA and mainstream adoption

The widespread use of mobile phones made it possible for authentication systems to send verification codes via SMS. Banks and online platforms used this method because most of their customers already had mobile phones. This approach removed the need to own unique devices, such as RSA SecurID hardware tokens.

Google launched 2-step verification (2SV) in 2010, providing multi-layered verification to consumer platforms. Google Authenticator was introduced in September 2010 as free mobile software supporting 2FA, enabling Google account users to generate TOTP codes for stronger account protection [28].

2.3.4 TOTP applications

TOTP apps such as Google Authenticator made 2FA using TOTP as the second factor widely accessible. These apps generate 6-digit confirmation codes known as TOTP codes that expire after 30 seconds, providing a high level of security for user accounts [28]. Services such as GitHub, Dropbox, and Amazon employ TOTP as the second factor in their authentication flows. Users can set up 2FA based on TOTP by accessing the authenticator app, i.e., Google Authenticator on their smartphone and scan the quick response (QR) code provided by the service [28]. The authenticator app then generates six-digit codes that refresh every 30 seconds.

[2] A PIN is typically a short numeric-only secret (usually 4-6 digits) used primarily for device unlock or personal verification [25].

TOTP apps offer several benefits over both hardware tokens and SMS-based authentication methods. While both RSA SecurID hardware tokens and TOTP generate OTP codes that expire after 30 seconds, TOTP's standardized approach, unlike RSA SecurID hardware tokens, allows compatibility across different platforms and services [18]. Users do not need to purchase special hardware tokens, as the generated TOTP codes work offline and multiple accounts for different services can be managed within a single authenticator app. Although these benefits contributed to the widespread adoption of TOTP as a second factor, 2FA using TOTP is still not fully resistant to phishing attacks, see Chapter 3.4.1 on page 32.

2.3.5 Modern era: FIDO standards and the password-less future

The FIDO alliance was established in 2012 by several major technology companies that aimed to reduce dependence on passwords as an authentication method and encourage the development of stronger authentication methods [29]. A few years later, the W3C cooperated with the FIDO alliance to develop common authentication standards known as fast identity online 2 (FIDO2) for browsers and operating systems [30].

Support for these authentication standards grew quickly. Google Chrome, Microsoft Edge, and Mozilla Firefox introduced FIDO2 support in 2018 [30]. In the same period, Yubico[3] released its first security key that supported the FIDO2 protocols, including WebAuthn and client to authenticator protocol (CTAP2), which made hardware-based and passwordless authentication accessible on various types of devices and enabled users to authenticate themselves to online services using biometrics, physical security keys, or device-bound credentials rather than passwords [31, 30]. A detailed technical analysis of FIDO2 is presented in Chapter 3.2.2 on page 21.

[3]Yubico `https://www.yubico.com/` is the company that designs and manufactures the YubiKey series of authentication hardware devices.

3 Theoretical foundations

This chapter provides a comprehensive theoretical background to understand the technologies, security principles, and vulnerabilities related to 2FA. It begins by introducing different authentication fundamentals and continues with detailed explanations of core 2FA methods, their security considerations, and common attack vectors.

3.1 Authentication fundamentals

Authentication mechanisms rely on three fundamental categories of factors, each addressing different aspects of user verification [32, 33, 34]:

1. **Knowledge factors:** Represent information that only the legitimate user should know. The most common example is the traditional password. Other knowledge-based factors include PIN or security questions [32, 33, 34].

2. **Possession factors:** Involve physical devices that the users must have in their control. These range from hardware security tokens and smart cards to mobile devices receiving OTP [32, 33, 34].

3. **Inherence factors:** Utilize biometric characteristics unique to each individual, such as fingerprints or facial recognition [32, 33, 34].

SFA relies only on a single authentication factor (knowledge, possession, or inherence). In most authentication systems, this means verifying the identity of the user through a password alone.

Multi-factor authentication (MFA) relies on two or more distinct factors.

2FA relies on exactly two distinct factors. For example, password (knowledge) and hardware token (possession). Every 2FA system is an MFA system.

The role of TOTP in 2FA systems TOTP is typically used as the second factor in 2FA systems, rather than providing complete authentication on its own. In a typical TOTP-based 2FA system, the first factor (knowledge) is the user's password, while the second factor (possession) is a TOTP code generated by an authenticator app on the user's device. While most TOTP implementations are deployed as 2FA (password + TOTP code), 2FA itself can be implemented using many other methods besides TOTP. For example, withdrawing money from automated teller machine (ATM) using a bank

card represents a 2FA system where the bank card itself (possession factor) and the PIN (knowledge factor) are required, yet no TOTP is involved.

Throughout this thesis, when referring to TOTP-based 2FA or TOTP as a second factor, we describe cases where TOTP codes serve as the possession factor in a 2FA system.

3.2 Core technologies used in 2FA systems

This section explains common authentication mechanisms frequently deployed in 2FA systems. The focus lies on TOTP used as the second factor, and on passkeys, which enable passwordless authentication method that can be implemented either through hardware tokens or integrated into the device's operating system.

3.2.1 Time-based one-time password (TOTP)

TOTP is a widely-used method for generating OTP that typically serves as the second factor in 2FA systems. TOTP builds upon the HOTP algorithm defined in RFC 4226 [17]. HOTP introduces a counter-based approach for generating OTP where both the client and server maintain a synchronized counter value. The client refers to the device or application that generates the OTP, while the server is the backend system responsible for verifying the OTP. The HOTP algorithm is shown in the Table 3, with the left column presenting the algorithm's inputs and the right column showing its output.

The HOTP Algorithm $\text{HOTP}(K, C) = \text{Truncate}(\text{HMAC-SHA-1}(K, C))$	
The ***inputs*** of the algorithm	The ***output*** of the algorithm
• K: the shared secret key between the client and server • C: an 8-byte counter value that increments with each authentication attempt	• HOTP: the generated one-time password obtained by truncating the HMAC-SHA-1 output

Table 3: Overview of the HOTP algorithm based on RFC 4226

The truncation process in the HOTP algorithm is a method for converting the 20-byte output of the HMAC-SHA-1 function[4] into an OTP [17]. The process works as follows:

1. The last byte of the HMAC result determines where the starting offset begins, it indicates where to start extracting a 4-byte segment from the HMAC output.

2. This 4-byte segment is interpreted as a 31-bit integer where the most significant bit is ignored to ensure a positive number.

3. Finally, the 31-bit integer obtained from the 4-byte segment is reduced modulo 10^d, where d represents the desired number of digits for the OTP. This produces the final HOTP value that is displayed to the user.

Building on the counter-based HOTP mechanism described above, RFC 6238 introduced a time-dependent variant defined as TOTP [18]. Instead of relying on an incrementing counter, TOTP derives its moving factor directly from the current Unix time [18]. The TOTP algorithm is defined in Table 4, with the left column presenting the algorithm's inputs and the right column showing its output.

The TOTP Algorithm $\text{TOTP}(K, T) = \text{HOTP}(K, T)$	
The ***inputs*** of the algorithm	The ***output*** of the algorithm
• K: the shared secret key between the client and server • T: a time-step counter	• TOTP: one-time password produced by HOTP(K, T)

Table 4: Overview of the TOTP algorithm based on RFC 6238

The value of the time-step counter T is computed from the current Unix time. Table 5 on the following page summarizes the parameters involved in computing the time-step counter T.

[4] HMAC-SHA-1 is a form of hash-based message authentication code (HMAC) that uses the SHA-1 cryptographic hash function along with a secret key to guarantee the integrity and authenticity of a message [35].

Computation of the Time-Step Counter T
$$T = \left\lfloor \frac{\text{UnixTime} - T_0}{X} \right\rfloor$$
• UnixTime: current Unix timestamp
• T_0: Unix epoch reference time (0 by default)
• X: time-step interval in seconds (30 seconds by default)
• $\lfloor \rfloor$: floor function that returns the greatest integer less than or equal to the value inside it (in this case the computed time-step counter value T)

Table 5: Parameters of the time-step counter T in TOTP

QR Code setup and shared secret key TOTP-based 2FA requires a shared secret key K that must be known to both the server and the client [18]. The server refers to the service provider, such as a website or online service requiring 2FA. The client refers to the user's authenticator application running on their mobile device, i.e., Google Authenticator. When users initially enable TOTP-based 2FA on their account associated with the service provider, the server must generate this shared secret key K and securely transmit it to the client to enable synchronized TOTP generation.

The QR code provisioning flow follows a standardized approach using the otpauth:// uniform resource identifier (URI) format, which is commonly encoded as a QR code for scanning using authenticator applications [36]. The URI encodes the shared secret key K and the associated parameters into a QR code for transmission from the server to client. The complete provisioning flow is illustrated in Figure 1 on page 19 and consists of the following steps:

1. The user enables TOTP-based 2FA for account authentication.

2. The server generates the shared secret key K using a cryptographically secure random number generator[5].

3. The server constructs an otpauth:// URI containing the Base32-encoded shared secret key K and configuration parameters.

[5]The random number generator produces unpredictable random values by collecting randomness (called entropy) from various sources in the system, such as hardware events, system timing, and environmental noise [37]. The generator ensures that the resulting key cannot be guessed or predicted.

4. During the account setup process, the server displays this URI as a QR code on the user's screen.

5. The user scans the QR code using a client-side authenticator application on the mobile device.

6. The client decodes the QR code, extracts the shared secret key K, and stores it securely in the device's storage. Both the server and the client share the same shared secret key K and can generate identical TOTP codes at synchronized time intervals.

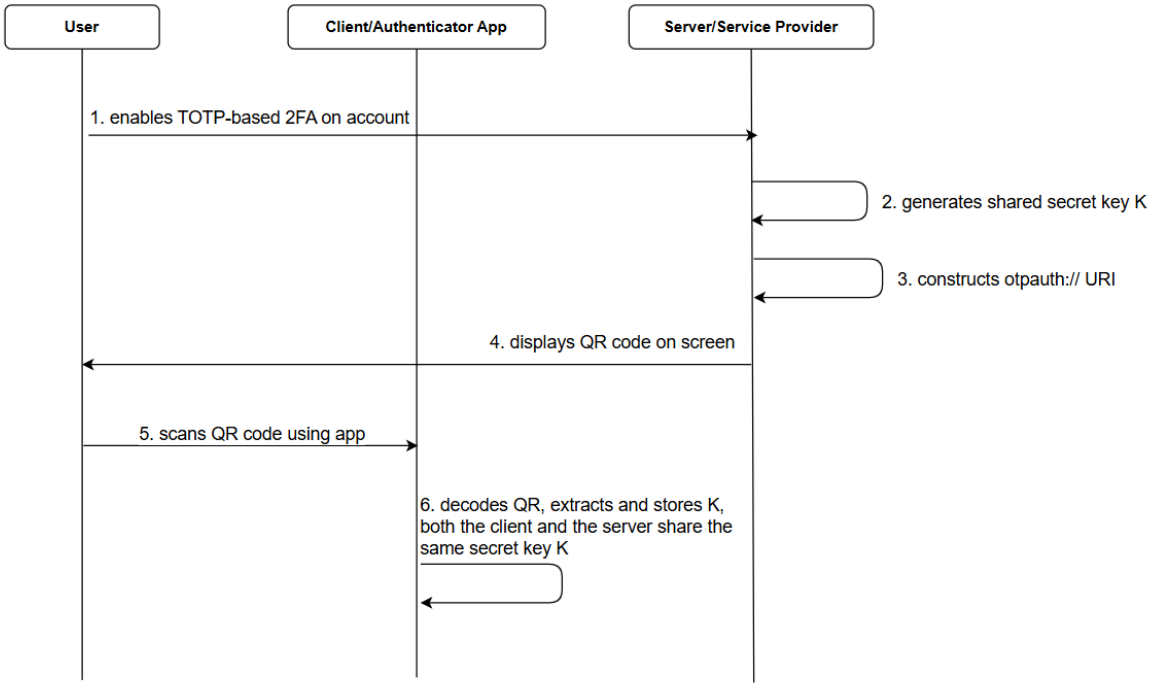

Figure 1: QR code provisioning flow showing the transmission of the shared secret key K from server to client

Time synchronization and clock drift tolerance Time synchronization between the client and server is essential for TOTP to work correctly. Network time protocol (NTP)[6]. is normally applied to maintain accurate system clocks [38]. Because clock alignment cannot be guaranteed, the TOTP standard allows a certain level of tolerance by accepting codes from nearby time intervals.

A TOTP code is valid only within a fixed period called a time-step counter. With the common 30-second time step, both the client device and the server generate a TOTP code based on the current 30-second block of time. However, if the client clock is slightly ahead or behind, the generated code may belong to a different window than the one the server expects. To reduce authentication failures caused by these minor time differences, many systems that implement TOTP verify both the code for the current[7] time window and codes from one or more adjacent[8] windows.

RFC 6238 recommends accepting at most one time step in either direction [18]. In practice many authentication systems that rely on 2FA based on TOTP, extend this tolerance to two or three previous time steps to better account for minor clock drift and network delays.

Conclusion By combining a shared secret key and a time-based counter, TOTP generates codes that are valid only for a short period, making them difficult for attackers to reuse. The shared secret key can be set up easily using QR codes, and allowing a small tolerance for clock differences ensures that TOTP-based 2FA systems work reliably.

[6] NTP is a standard protocol used to synchronize the clocks of computers and devices over a network, ensuring that all systems maintain accurate and consistent time [38].
[7] The 30-second period the server believes the authentication request occurs in.
[8] The 30-second windows directly before or after the current one.

3.2.2 Hardware-based passkeys

Hardware-based passkeys represent a transition from relying on shared secrets keys to employing asymmetric cryptography, known as public-key cryptography, for authentication. These physical devices implement the FIDO2 standard, which includes WebAuthn and CTAP2 core specifications developed by the FIDO Alliance and the W3C.

FIDO Alliance standards The FIDO Alliance aimed to develop authentication standards that eliminate the reliance on passwords by using public-key cryptography. This evolution progressed through two major phases:

1. **Universal 2nd Factor (U2F):** Released in 2014, established the foundation for hardware-based second-factor authentication. Universal 2nd factor (U2F) was designed specifically to enhance password-based authentication by requiring users to press a button on a hardware security token [39].

2. **FIDO2:** Represents the current generation, developed in collaboration with the W3C. It consists of two core specifications: the WebAuthn application programming interface (API), which defines browser-level interfaces for authentication, and the CTAP2, which specifies how authenticators communicate with client platforms [19, 30].

WebAuthn specification and public-key cryptography WebAuthn specifies a JavaScript (JS) API that allows web applications to perform public key-based authentication directly in the browser [19]. The specification defines three main entities:

1. Relying Party (RP): Represents the service or website.

2. Authenticator: Provides cryptographic capabilities, including key generation, digital signature, and user verification.

3. Client (browser): Handles communication between the relying party and the Authenticator.

Hardware authenticators used in WebAuthn commonly implement elliptic-curve–based public-key cryptography to provide a high level of security [19]. In practice, they typically rely on the elliptic curve digital signature algorithm (ECDSA) over the P-256 curve, a standardized 256-bit elliptic curve that defines the mathematical rules for generating secure key pairs. Each key pair includes a private key that is securely stored within the hardware component, and a corresponding public key, which can be safely transmitted to the relying party [19]. The fundamental principle is asymmetric cryptography: The

authenticator signs data using its private key, and the relying party verify these signatures using the stored public key, confirming possession of the legitimate authenticator without exposing the private key [19]. This cryptographic foundation enables WebAuthn to define two core workflows: The registration ceremony for establishing new passkey credentials and the authentication ceremony for verifying user identity.

Registration ceremony The registration ceremony, illustrated in Fig. 2 on the following page, follows a structured protocol [40, 19]:

1. The user initiates a registration request through the client/browser.

2. The client/browser forwards this request to the server with a "create passkey" command. The client/browser automatically includes the server's origin in the request. This origin consists of the scheme, domain, and port, for example https://example.com:123.

3. The server responds with configuration options.

4. The client forwards these configuration options to the authenticator.

5. The authenticator prompts the user to interact with the device, for example, by touching the security key.

6. After the user confirms the registration action, the authenticator generates a new asymmetric key pair. The private key is exclusively bound to the specific website origin that the client provides.

7. The private key remains securely stored within the authenticator and the public key is sent through the client/browser to the server.

8. The server stores the public key together with the corresponding origin and confirms successful registration back to the client. By binding the public key to the origin, phishing attacks are effectively prevented.

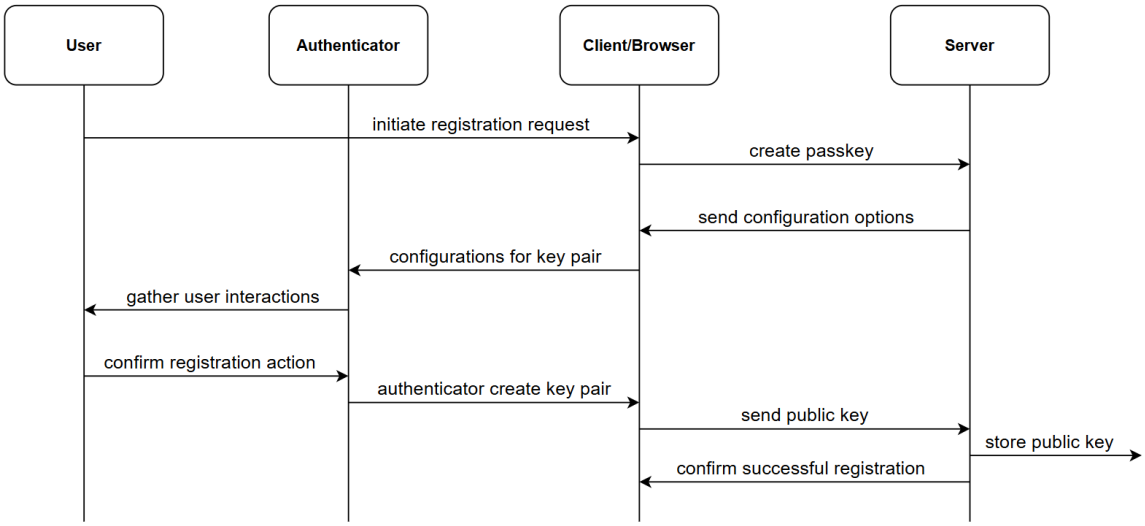

Figure 2: Passkey registration protocol flow

Authentication ceremony The authentication ceremony, illustrated in Fig. 3 on the following page, proceeds through the following steps [40, 19]:

1. The user initiates the login process.
2. The client/browser notifies the server with a “login via passkey” request.
3. The server generates a cryptographic challenge and stores it temporarily to enable later authentication.
4. The server sends this challenge back to the client/browser.
5. The client forwards the challenge to the authenticator.
6. The authenticator requests user interaction to confirm the login process.
7. The user confirms the authentication (through touching the security key, entering a PIN, or providing biometric verification).
8. Once verified, the authenticator signs the provided challenge using its private key.
9. The client receives the signed challenge and forwards it to the server.

10. The server verifies the challenge's signature using the stored public key and compares the result with the original challenge generated. If both are identical, the server confirms successful authentication.

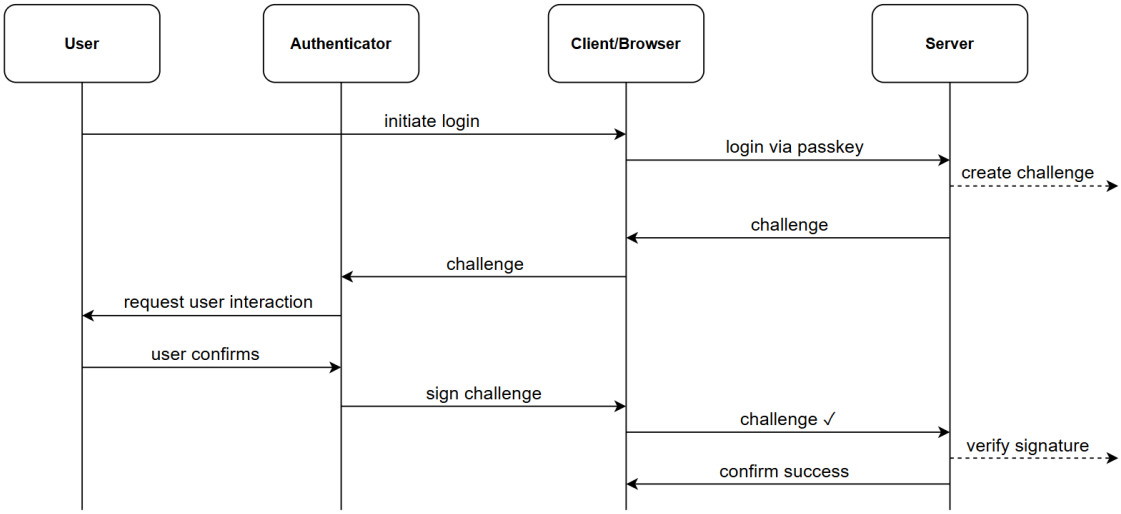

Figure 3: Passkey authentication protocol flow

In passkey authentication, the challenge-response protocol acts as the core security mechanism ensuring that every authentication operation is unique and used only once. When the user initiates an authentication request as shown in Fig. 3, the server generates a cryptographically secure random challenge, typically 16-32 bytes, obtained from a cryptographically secure pseudo-random number generator (CSPRNG)[9] as recommended by the W3C [19].

This challenge is unique for each authentication ceremony and is stored temporarily by the server to ensure that any response can later be verified against the originally issued value. Each authenticator credential has a unique ID, a short byte sequence generated by the authenticator during registration, which allows the server to identify which credential (i.e., key pair) was used when the user authenticates.

The challenge is embedded into the `clientDataJSON` object, which is constructed by the client platform. This object contains three essential fields: the operation type (i.e., `webauthn.get` for authentication), the base64url-encoded challenge[10], and the origin

[9] CSPRNG is a pseudo-random number generator designed to produce unpredictable values suitable for cryptographic use. See [37] for formal randomness requirements for security.

[10] The base64url-encoded challenge is just the server's random challenge converted into a text format that can be safely transmitted in JSON and URLs [19, 41].

[19]. The origin is derived by the browser after establishing a hypertext transfer protocol secure (HTTPS) connection with the server of the requested domain. During this step, the browser validates the server's transport layer security (TLS)[11] certificate to ensure that it is trusted and matches the requested domain. The browser verifies the certificate by validating the digital signature created by the issuing certificate authority[12] using its private key. If the certificate matches the requested domain, an encrypted communication channel with the server is established [42, 43].

The client hashes the `clientDataJSON` object using secure hash algorithm (SHA)-256[13] to produce the `clientDataHash`. By including both the challenge and the website's origin in the `clientDataHash`, WebAuthn ensures that the authenticator's response is bound to that particular website and authentication session. As a result, even if an attacker intercepts the signed data, it cannot be reused for another website or session.

When the authenticator receives the challenge, it builds the `authenticatorData` structure. This structure contains key security information: a SHA-256 hash of the relying party ID, flags indicating user presence and user verification, and a signature counter to detect cloned devices [19]. During authentication, the authenticator signs the `authenticatorData` and the hash of `clientDataJSON` using its private key.

The server verifies the authenticator's response by ensuring that [19]:

- The challenge in `clientDataJSON` matches the one originally sent by the server.
- The origin corresponds to the expected domain.
- The relying party ID hash matches the relying party's domain.
- The signature is valid using the stored public key.

[11] TLS is a cryptographic protocol that secures communication between a client and a server over a network [42].

[12] Certificate authority is a trusted organization responsible for validating the identity of domain owners and issuing certificates that bind a public key to a specific domain name [43].

[13] SHA-256 is a function that turns any input into a fixed 256-bit code. It is irreversible, meaning it is very hard to get the original input from the 256-bit code, and it produces different 256-bit codes for different inputs [44].

Security considerations for hardware-based passkeys Hardware-based passkeys provide security advantages that exceed those of passwords, since they rely on asymmetric cryptography and strict origin binding[14] as defined in the WebAuthn protocol. These mechanisms mitigate many of the attack vectors that affect traditional authentication systems:

1. **Phishing resistance:** WebAuthn guarantees that every authentication signature is tied to the real website a user is visiting. The authenticator includes both the website's origin and the relying party identifier in the data it signs, which prevents signatures from being created for fake or misleading domains. Even if a user is tricked into opening a phishing page, the authenticator will not generate a valid signature for the attacker's site. This behaviour is defined directly in the WebAuthn standard [19] and is a main reason why hardware authenticators such as the YubiKey provide strong protection against phishing attacks (see Chapter 3.4.1 on page 32).

2. **Elimination of shared secret keys:** Hardware-based passkeys eliminate the need for server-side storage of shared secret keys, as they rely on public-key cryptography. During passkey registration, only the public key is transmitted to the server, whereas the private key remains safely stored in the authenticator. This approach aligns with current security guidelines that support hardware-based asymmetric authentication to minimize the risks linked to storing passwords in authentication databases [19, 13].

3. **Protection against replay attacks:** Each authentication requires a unique challenge generated by the server using a CSPRNG. The authenticator signs this challenge along with contextual information, such as the website's origin, the relying party identifier, and the type of operation, which ensures the signature is valid only for that specific session. This prevents attackers from reusing old captured signatures. The challenge–response process is required by the WebAuthn protocol and follows NIST guidelines for high-assurance authentication [19, 13].

4. **Hardware-bound private key storage:** In hardware-based passkeys, private keys are generated and stored within dedicated secure hardware components[15] designed to prevent extraction and resist physical attacks. Hardware-based authenticators such as YubiKey employ these secure elements to ensure that the private keys remain protected. Guidance from major European authorities highlights the importance of secure hardware components for robust authentication [20].

[14] Origin binding is a mechanism used by FIDO2 and WebAuthn to ensure that a hardware authenticator, such as a YubiKey, will only respond to authentication requests from the website domain where the credential was originally registered [45].

[15] Secure hardware components are specialized hardware chips designed to store cryptographic keys and perform cryptographic operations securely, isolated from the host system. They are tamper-resistant, meaning they are built to resist physical attacks such as voltage or temperature manipulation [20, 19].

3.2.3 Software-based passkeys (platform-integrated)

Software-based passkeys provide the same general security benefits as hardware-based passkeys, but without requiring users to carry an external device. Instead, they rely on secure hardware components already embedded in modern operating systems. These platform-integrated solutions allow passkeys to be used more conveniently in everyday environments while still following the same WebAuthn and FIDO2 principles as hardware authenticators.

Platform-integrated authenticators across operating systems Most current devices include dedicated security hardware capable of generating and protecting the asymmetric key pairs used for passkeys. Operating systems use these hardware components to function as "platform-integrated authenticators":

1. **Windows Hello:** Uses facial recognition, fingerprints, or a device PIN to unlock credentials stored inside the computer's trusted platform module (TPM). The TPM generates and stores the asymmetric private key inside its secure enclave and only transmits the corresponding public key during registration. The private key never leaves the TPM in plaintext. Biometric data never leaves the device, and the Windows Hello PIN is used as a local unlock factor, not as an online password [46].

2. **Apple Touch ID and Face ID:** Rely on the secure enclave, a dedicated security processor inside Apple devices. The secure enclave generates and stores the asymmetric private key for each passkey credential and handles cryptographic operations without exposing the private key to the main operating system. Apple stores biometric information only as mathematical templates generated from the original fingerprint or facial scans; these representations never leave the secure enclave and never get uploaded to any server [47].

3. **Android biometric authentication:** Uses hardware-backed keystores inside the trusted execution environment (TEE). The TEE generates each passkey's asymmetric key pair and keeps the private key isolated from the main operating system. Biometric checks, such as fingerprint or facial recognition, occur through Android's biometric API and the private key used for passkeys remains protected inside secure hardware [48].

Across all mentioned platforms, the core principle is consistent: the device generates a WebAuthn asymmetric key pair inside a built-in secure hardware component, such as a TPM, secure enclave, or TEE. These secure hardware components are designed to isolate and protect cryptographic operations from the main operating system. The private key never leaves the hardware secure component, and local user verification such

as a PIN, fingerprint, or facial recognition is required before the private key is allowed to sign authentication challenges.

Synced passkeys and cloud synchronization A significant practical issue with device-bound passkeys is device loss or replacement. Synced passkeys address this issue through encrypted cloud synchronization:

1. **Apple's iCloud keychain:** Synchronizes passkeys using end-to-end encryption[16]. Before a passkey is uploaded to iCloud, its private key is encrypted on the device using a symmetric key stored in the device's secure hardware. Only this encrypted version of the private key is stored in iCloud. To use a synchronized passkey on a new device, users must sign in to their iCloud account and confirm their identity on another trusted Apple device. Apple also uses a protected recovery system designed to protect against attackers attempting a high volume of guesses during account recovery [50].

2. **Google password manager:** Provides a similar system for Android and Chrome users. Passkey private keys are encrypted on the device, and the encryption keys protecting them are linked to the user's device unlock method, such as a PIN or fingerprint. Google also protects recovery information and enforces strict limits on authentication attempts within its infrastructure to reduce the risk of unauthorized access [51].

3. **Cross-device authentication:** Passkeys can unlock one device even when they are stored securely on another device. For example, a passkey that is securely stored on a smartphone can be used for authentication on a laptop. The laptop and smartphone communicate to make sure they are near each other using Bluetooth, and a QR code sets up a secure connection between them. The smartphone performs all the private key operations, and the laptop only receives the information needed to complete the login. This process is made possible through the CTAP2 hybrid protocol, which defines how devices interact securely for cross-device authentication [52, 19].

WebAuthn specification for platform-integrated passkeys Platform-integrated authenticators follow the same registration and authentication protocols as hardware authenticators, using the identical WebAuthn API described earlier in Chapter 3.2.2 on page 21. The key technical difference between these two authentication methods is

[16] End-to-end encryption ensures that data is encrypted locally on the user's device and can only be decrypted on a secondary device device that the same user controls. No other party, including the service provider, can read the encrypted information while it is stored or transmitted. Apple applies this model to iCloud Keychain so that private keys remain inaccessible to Apple or any third party [49].

that hardware authenticators that store private keys securely in dedicated secure chips, while platform-integrated authenticators store private keys in built-in secure hardware such as secure enclaves. Applications request platform-integrated authenticators by setting the optional `authenticatorAttachment` parameter to `"platform"` instead of `"cross-platform"` [19].

A main difference is how platform-authenticator credentials are stored. Synced passkeys contain a user identifier and are kept directly inside the authenticator, allowing the device to find the correct credential without requiring the user to type a username. Because the authenticator already knows which passkey belongs to the user, it can display available credentials and start the authentication process immediately. This ensures fully passwordless login possible on platforms that support synchronized passkeys [19].

3.3 Attack vectors against password-only authentication

This section examines how relying on passwords as a SFA method is not enough, highlighting the vulnerability to common attack methods such as brute force and dictionary attacks.

3.3.1 Brute force attacks

A brute force attack is a technique in which attackers try to access systems illegally by testing different password combinations [53]. Brute force attacks target authentication systems in two fundamentally different ways as they can either attempt to break into live servers online or exploit stolen password databases offline.

Online attacks occur in real time against target systems, where attackers use automated software tools such as Hydra[17] to test passwords directly against live authentication services. However, these attacks are constrained by factors such as network latency and server-side defenses, including account lockout policies [54]. In April 2024 [55], Dell[18] experienced a significant data breach when attackers employed online brute force attacks to compromise the company's online portal. The attack revealed the personal data of more than 49 million customers between 2017 and 2024 [55].

Offline attacks, on the other hand, occur when attackers obtain password databases from previous data breaches and can test passwords locally on their own systems using tools

[17] https://en.wikipedia.org/wiki/Hydra_(software)
[18] https://www.dell.com/

such as Hashcat[19] or John the Ripper[20]. This allows them to bypass the factors that slow down online attacks and significantly increase the rate at which passwords can be tested [54, 56]. Modern GPUs have dramatically increased the effectiveness of offline brute force attacks. Eight linked NVIDIA RTX 4090 GPUs can test 200 billion eight-character password combinations in just 48 minutes [57]. This represents an increase in computational capability compared to traditional processing units, making even complex passwords vulnerable to such attack.

Although brute force attacks represent one of the oldest hacking techniques, their occurrence and complexity have continued to increase and still represent a major risk in modern digital security environment [53]. A study conducted in 2024 [58], indicated a 12% rise in brute force attacks compared to previous years, as attackers now use automated software tools to carry out large brute force attacks with very little human involvement [53].

How 2FA prevents brute force attacks 2FA provides strong protection against brute force attacks by introducing a second authentication factor. When organizations deploy 2FA, they eliminate the vulnerability of relying on a single authentication factor (password). An attacker who cracks the password still needs the second authentication factor to gain access. Successfully compromising the password alone is insufficient for authentication, since the second factor is still required [59, 60, 55]. The second authentication factor can take many forms: TOTP codes from smartphone authenticator apps, SMS codes, hardware security tokens, or biometric identifiers such as fingerprints and facial recognition [60]. Each type of second factor requires the attacker to compromise a completely different authentication component.

In TOTP-based 2FA systems, even if attackers using a brute force attack manage to guess the user's password, they would also need to compromise the shared secret key K (see Table 4 on page 17) stored on the user's authenticator device to generate valid TOTP codes.

According to Microsoft security studies [61], requiring multiple authentication factors significantly enhances account security, preventing over 99.9% of attacks that target compromised passwords. This high success rate occurs because attackers must overcome two separate authentication factors rather than just one [60]. As a result, implementing 2FA leads to a sharp drop in successful brute force attacks [61, 60].

[19] https://hashcat.net/wiki/
[20] https://en.wikipedia.org/wiki/John_the_Ripper

3.3.2 Dictionary attacks

A dictionary attack is a specialized brute force approach in which attackers attempt to access systems illegally by testing many possible passwords taken from lists of common words, phrases, or passwords stolen in previous data breaches [62]. Unlike brute force attacks that try every possible mix of characters, dictionary attacks use curated lists collected from previous data breaches, reflecting the types of passwords people often choose [62]. Because this approach focuses on commonly used passwords rather than all possible combinations, dictionary attacks are more efficient than pure brute force approaches [62].

Dictionary attacks use the same automated software tools as brute force attacks and also occur online against live systems or offline against stolen password databases. Dictionary attacks are effective because they exploit human behavior in creating passwords. Many users often pick passwords that simple to recall, which leads to predictable patterns [62]. Even when systems require more complex passwords, people often make simple and predictable adjustments rather than choosing something truly random. For example, users might replace the letter “e” with the number “3” [62]. According to research conducted in 2023 [63], more than 4 million users chose “123456” as their password, which can be guessed almost instantly. It also found that “123456”, “123456789”, “qwerty”, and “password” were the four most frequently used passwords [63].

In October 2013 [64], Adobe[21] experienced a critical data breach that revealed around 153 million user accounts. Security analysis [64] showed that attackers were able to access user accounts by employing offline dictionary attack on weak and simple passwords. The most common passwords among the exposed user accounts were “123456”, “123456789”, and “password” [64]. This data breach leaked a large amount of sensitive user data and harmed Adobe’s reputation, showing just how dangerous dictionary attacks can be when users rely on weak passwords [64].

2FA prevents dictionary attacks through the same mechanism described in Chapter 3.3.1 for brute force attacks, ensuring that even if attackers successfully guess the correct password from their dictionary lists, they still cannot authenticate without the additional authentication factor.

3.4 Attack vectors against 2FA

In the following section, different types of attacks against 2FA systems are discussed, including real-time phishing attacks and hardware-based attacks targeting devices such as YubiKey.

[21] https://www.adobe.com/

3.4.1 Real-time phishing attacks (man-in-the-middle)

Real-time phishing attacks represent an advanced threat that can bypass 2FA deployments that rely on SMS codes or TOTP as the second factor. These attacks use a man-in-the-middle technique to intercept the victim's authentication credentials and relay them in real time to the legitimate service.

How real-time phishing attacks work The attack mechanism uses a reverse proxy server positioned between the victim and the legitimate server. When the victim tries to access what looks like a legitimate login page, they are actually connecting to an attacker-controlled server that relays all communication and data to the legitimate server, and the responses from the legitimate server are likewise intercepted and sent back to the victim [65, 66, 67]. The attack sequence proceeds through several steps [67] as shown in Fig. 4.

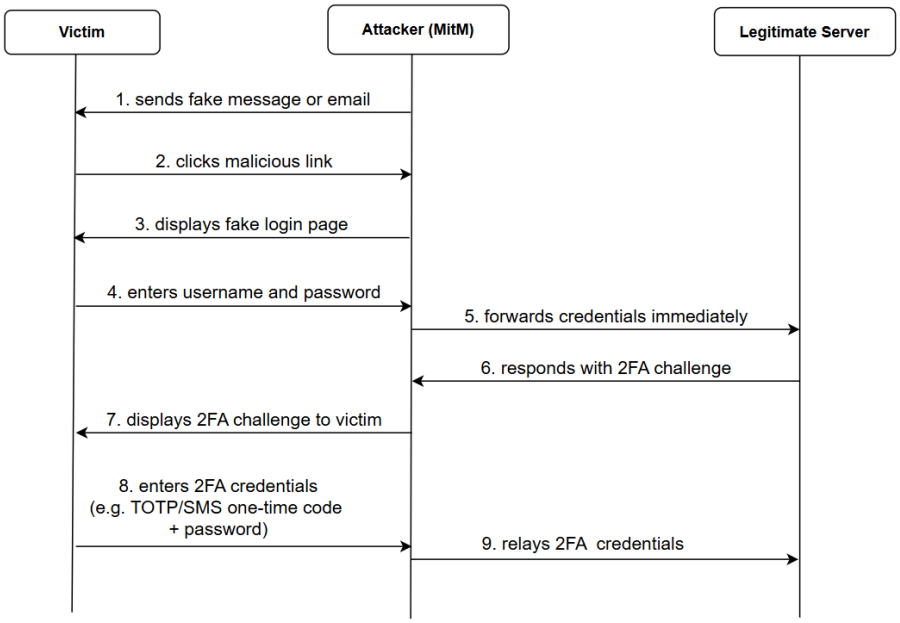

Figure 4: Phishing attack sequence protocol

The Tycoon 2FA phishing platform[22] illustrates how large-scale real-time phishing attacks are carried out in practice. This phishing platform aimed primarily at compromising Microsoft 365 and Gmail accounts [66]. Access to its platform was sold through private

[22] https://www.proofpoint.com/us/blog/email-and-cloud-threats/tycoon-2fa-phishing-kit-mfa-bypass

Telegram channels, with prices beginning at about $120 for a ten-day subscription, which made it affordable and usable even for attackers with limited technical experience [66]. The technical design of the platform goes far beyond simple credential theft, as it included several anti-detection features that helped in avoiding security monitoring. These features included bot filtering through security challenges and traffic filtering to block automated scanning tools [66].

Vulnerability of SMS and TOTP to real-time phishing attacks SMS-based and TOTP-based 2FA deployments are weak against real-time phishing attacks because they cannot distinguish whether a login request comes from the real website or a phishing one. When a user enters an SMS code or TOTP code in combination with a password, the authentication service cannot verify if this authentication request is legitimate or being forwarded by an attacker, and because these codes are time-based and expire quickly, they provide limited protection if forwarded immediately to the legitimate service. In the case of TOTP-based 2FA systems, as mentioned in Chapter 3.2.1 on page 17, the codes that TOTP generates, based on a shared secret key and the current Unix time, remain valid regardless of which website or service is requesting them. In the case of SMS, the one-time code which is sent to the user's mobile phone can also be immediately forwarded to any site, including phishing sites. As a result, both TOTP and SMS codes offer only limited protection against real-time phishing attacks.

FIDO2 resistance to real-time phishing attacks FIDO2-based passkeys provide strong protection against phishing attacks through origin binding. When a user attempts to access their online account on a website, the authenticator verifies the website's domain and will only complete authentication if it matches the domain of the website where the device was originally registered. In this context, credentials refer to the asymmetric cryptographic keys generated by the authenticator for that specific website. During authentication, the server generates a challenge which is forwarded by the client to the authenticator together with the origin and the relying party identifier derived from the requesting website. The authenticator verifies that the relying party identifier matches the domain associated with the stored credential. If the domains match, the authenticator signs the challenge using its private key and returns the signed response to the client. The client forwards this signed assertion to the server, which verifies the signature using the stored public key. If a user is tricked into visiting a phishing website with a different domain, such as examp1e.com instead of example.com, the origin and the relying party identifier will not match the domain associated with the registered credential. In this case, the authenticator refuses to generate a valid signature, preventing the authentication process from completing [68]. This mechanism ensures FIDO2-based passkeys highly effective at preventing phishing attacks, providing a level of protection that traditional time-based codes cannot achieve [68]. Fig. 5 on the following page illustrates the origin

and relying party identifier verification process used in passkey authentication to prevent phishing attacks.

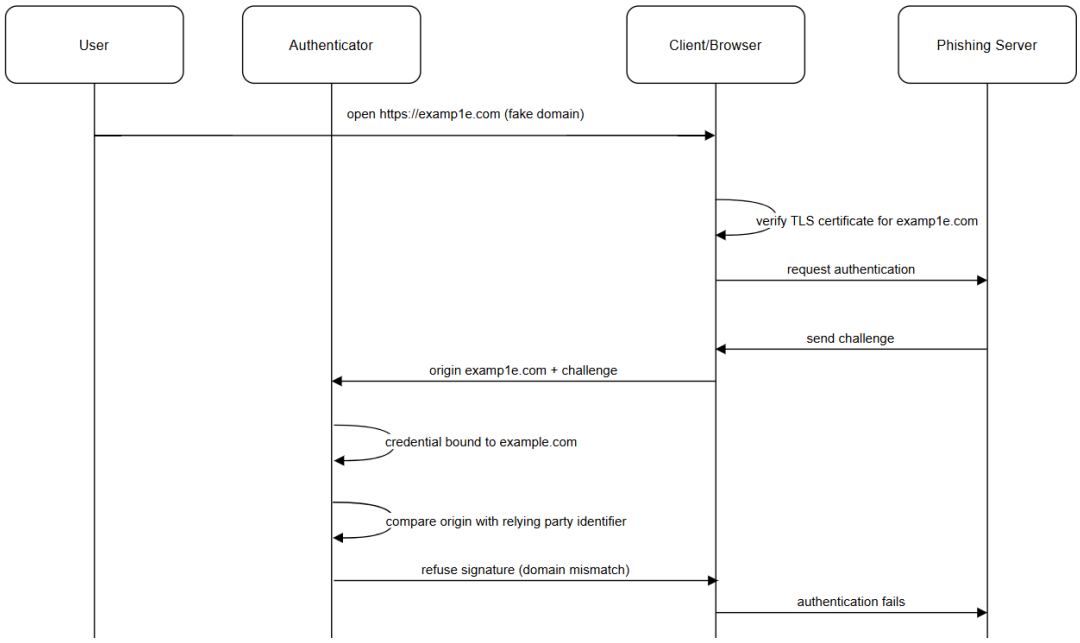

Figure 5: Domain mismatch detection in passkey authentication

3.4.2 Hardware token attacks (YubiKey)

Hardware security tokens such as YubiKey and Nitrokey[23] are widely regarded as highly secure authentication devices because they are tamper-resistant and support the modern FIDO2 security standard, which provides strong cryptographic protection. However, even these devices can be vulnerable to attacks. This section discusses a notable security weakness found in YubiKey devices and explains why Nitrokey devices are not affected by this vulnerability.

The EUCLEAK side-channel attack In September 2024, security researchers [69, 70] disclosed a critical security vulnerability known as EUCLEAK that affects YubiKey 5 series devices. EUCLEAK is a side-channel[24] attack that exploits a vulnerability in the Infineon cryptographic library used by these devices [70, 69]. The EUCLEAK vulnerability exploits timing variations in the modular inversion operation within Infineon's ECDSA implementation. Through electromagnetic emissions analysis, attackers can extract sensitive data during cryptographic computation [70]. Successful exploitation of the EUCLEAK vulnerability requires certain conditions and capabilities to be met [70]:

1. **Physical access**: Attacker must obtain physical access of the target authentication device to position electromagnetic probes near to the Infineon security microcontroller.

2. **Specialized equipment**: The attack requires electromagnetic probes (such as Langer ICR HH 500-6[25]) and oscilloscopes with sufficient sampling rates (5-10 GSa/s).

3. **Technical proficiency**: Successful exploitation demands expertise in cryptographic implementation analysis, side-channel attack methodology, and signal processing techniques.

4. **Temporal requirements**: The attack includes two phases:

    - Online phase: Approximately 5-10 minutes for electromagnetic signal capture during 40-200 authentication operations.

    - Offline phase: Between 1-24 hours for signal analysis and cryptographic key reconstruction.

[23] https://www.nitrokey.com/
[24] https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Kryptografie/Seitenkanalresistenz/seitenkanalresistenz_node.html
[25] https://www.langer-emv.de/en/product/near-field-microprobes-icr-hh-h-field/26/icr-hh500-6-near-field-microprobe-2-mhz-to-6-ghz/108

The EUCLEAK vulnerability affects YubiKey 5 series devices manufactured before firmware version 5.7 [70, 69, 71]. Yubico has acknowledged the vulnerability but noted that firmware updates cannot be applied to existing devices due to the hardware nature of the secure element [69, 71]. To address this issue, Yubico demonstrated responsive security practices by transitioning to an independent cryptographic library in firmware version 5.7, thereby eliminating dependence on the vulnerable Infineon implementation [71].

Nitrokey devices, another hardware security token manufacturer, are not impacted by the EUCLEAK vulnerability because they do not use Infineon security chips, which were the source of the side-channel vulnerability [72]. Instead, Nitrokeys rely on alternative security elements, such as those from NXP[26], and their architecture emphasizes firmware-updatable components [72]. This design allows security issues to be addressed through software updates, unlike the affected Infineon chips, where the vulnerability could not be patched. Additionally, Nitrokey's open-source firmware and, where applicable, open hardware designs enable independent review and community auditing, enhancing transparency and reducing the risk of undiscovered vulnerabilities [72]. Together, these factors ensure that Nitrokeys maintain a high level of security even in the presence of hardware-specific vulnerabilities affecting other devices.

Conclusion The EUCLEAK vulnerability demonstrates that even highly secure hardware tokens like YubiKey can be vulnerable to side-channel attacks. Exploiting this vulnerability in practice, however, requires physical access, specialized equipment, and advanced technical skills.

[26] https://www.nxp.com/

4 Design and implementation

This chapter describes the design and implementation of the 2FA web-based application integrated into CTO[27]. The goal of the web-based application is to educate users about modern authentication methods, including TOTP-based 2FA and passkeys.

4.1 Requirements analysis

Before any implementation work, both functional and non-functional requirements were gathered.

4.1.1 Functional requirements

1. **TOTP generation**: Users must generate a shared secret, scan it via a QR code using their mobile device into an authenticator app, i.e., Google authenticator and observe the real-time generation and verification of TOTP. The TOTP generation tab must explain the cryptographic operations and highlight the truncation process. TOTP verification should occur in the browser and display either a success or an error message.

2. **Attack simulation**: The web-based application must simulate both time-drift attacks and phishing/replay attacks, to raise awareness of TOTP's weaknesses. Users should be able to adjust the client's clock and observe authentication failures or successes, and follow a step-by-step timeline of a phishing attack.

3. **Passkey registration**: Users must understand and interact with the registration process of hardware-based and software-based passkeys. The registration flow must show the challenge generation, user verification, key-pair creation, signature generation, and storage of the public key on the server.

4. **Passkey authentication**: Once the users are done with the passkey registration process, they should be able to authenticate the registered passkey with a chosen credential. The simulation must reflect the WebAuthn protocol.

5. **Recovery planning**: The web-based application must educate users about recovery strategies and backup options for different types of 2FA, such as the loss of a phone with a TOTP authenticator app, a hardware token, or a device with software-based passkeys. In addition, it provides prevention tips, describes available recovery

[27] CrypTool-Online `https://www.cryptool.org/` offers free tools for testing and learning cryptography.

options, and includes a comparison table that summarizes the relative difficulty of each recovery method.

6. **Self-assessment and summary**: A multiple-choice quiz should allow users to test their understanding of 2FA concepts and security principles. After the quiz, a summary table comparing TOTP, hardware passkeys, and software passkeys must be shown. The quiz evaluation must remain local to the browser and not shared with third parties.

4.1.2 Non-functional requirements

1. **Browser compatibility**: The web-based application must run on all modern browsers on desktop and mobile platforms.

2. **Security and privacy**: All cryptographic operations, including hashing and random number generation, occur locally in the browser. The quiz evaluation is also performed locally to avoid storing user responses on the server.

3. **Integration with CTO**: The design of the application should follow CTO's design guidelines and integrate into it's web portal. The content should be available in English and German languages.

4.2 Technical architecture

The web-based application is built as a modular `React`[28] application and divided into six tabs as shown in Fig. 7 on page 39. Each tab corresponds to a dedicated `React` component. Upon visiting the start page, the application looks like Fig. 6 on page 39. Users are provided with a brief introduction about the transition from password-only authentication to 2FA methods. Cryptographic operations such as Base32 encoding, HMAC-SHA1 computation, and random number generation are implemented in JS. All user interface elements are rendered using `Chakra-UI`.[29]

[28] https://react.dev/
[29] https://chakra-ui.com/

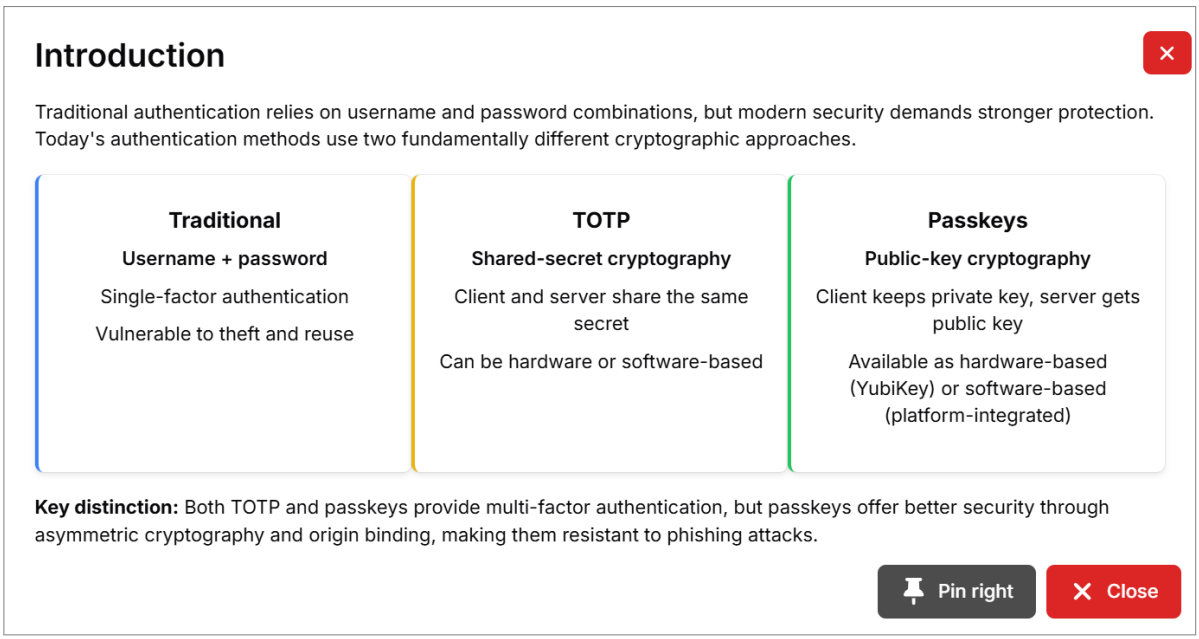

Figure 6: The landing page of the application

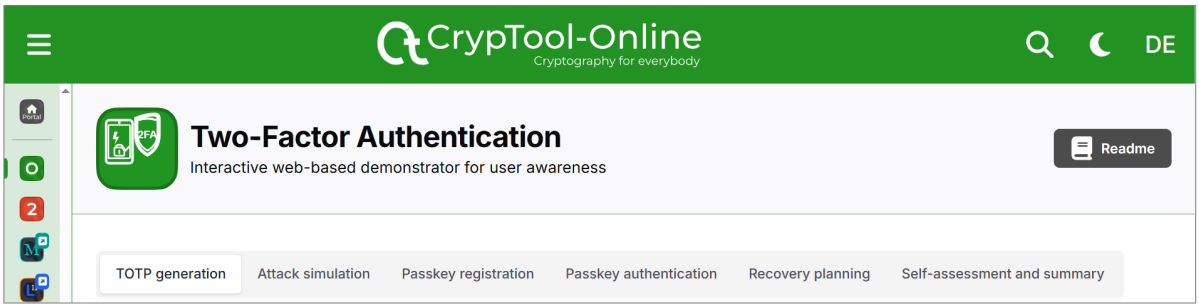

Figure 7: Overview of the tab structure in the 2FA web-based demonstrator

4.3 Core implementations

In the following section, each tab of the application is described. Code excerpts are shown in listings and the corresponding user interfaces are displayed in figures. Section 4.3.1 first outlines the helper functions and internal logic that support the implementation of all tabs.

4.3.1 Implementation details and helper functions

In the current implementation, the web-based demonstrator provides three different cryptographic workflows: TOTP generation, passkey registration, and passkey authentication. The TOTP generation tab uses real cryptographic functions to create and verify the TOTP codes. In contrast, the passkey registration and authentication tabs do not perform real asymmetric cryptographic operations. Instead, they follow the steps of the WebAuthn process but rely on helper functions that create example values for the challenge, the public key, the private key, and the signature. This approach allows the web-based demonstrator to show the overall procedure without exposing or handling actual private keys.

Passkey registration flow The registration sequence is handled by several supporting functions. The utility function `generateRandomString(length)` creates random alphanumeric strings and is used for all placeholder values in the simulation. The function `generateMockData()` uses this helper to create a 32-character challenge and a sample public key by combining a fixed ECDSA header ("MFkwEwYHKoZIzj0C…") with a 40-character random string. It also produces a sample signature by attaching the fixed prefix "MEUCIQDxHHDx4Ku1…" to another 60 random characters. Together with a credential identifier and timestamp, these values simulate what a real WebAuthn authenticator would return after creating a key pair and signing the server challenge with the private key.

User interaction is managed through `handleMethodSelection()`, which stores the chosen authentication method and resets earlier results. The function `getFlowSteps()` prepares the list of registration steps shown in the interface. The central function `handleStartRegistration()` begins the simulated registration by calling `generateMockData()`, storing the created values, and forwarding the new credential to the parent component through `onRegisterCredential()` after a short delay. The helper `getRegistrationResultText()` selects the success message that matches the chosen authentication method. Together, these functions form the implementation of the simulated passkey registration flow, as shown in Listing 1 on page 42.

Passkey authentication flow The authentication sequence is handled by several supporting functions. The function `generateMockAuthData()` creates a new 32-character challenge and a sample signature by combining the fixed prefix "MEUCIQDyHHEx5Lu2…" with 60 random characters, and it also records a timestamp. When the user chooses a credential, `getSelectedCredential()` retrieves it from the stored list. The description of the user action, such as touching a hardware key or using a biometric method, is provided by `getUserActionText()`, while `getSuccessMessage()` prepares the message shown when the simulation finishes.

The main workflow is controlled by `handleStartAuthentication()`, which checks whether a method and a credential were selected, stores the generated challenge and signature, and marks the authentication as complete. The helper `resetAuthentication()` clears the previous state and allows repeated demonstrations. Although the values are placeholders, the sequence of steps reflects the structure of a real WebAuthn authentication, where the server sends a challenge, the client signs it with its private key, and the server checks the signature with the stored public key. The implementation of the simulated passkey authentication flow can be seen in Listing 2 on page 45.

TOTP generation and cryptographic operations The TOTP generation tab performs real cryptographic operations. The `TOTPUtils` object provides `base32Decode()` and `base32Encode()` for converting Base32 strings to and from byte arrays, `sha1()` for computing SHA-1 message digests, and `hmacSha1()` for creating HMAC-SHA1 values using a secret key and message. The function `generateTOTP(secret, timeStep)` combines these operations by decoding the shared secret, calculating the time-based counter, applying HMAC-SHA1, performing the standard TOTP truncation, and producing the final six-digit code. The function `generateRandomSecret()` uses the browser's secure random number generator to create a 20-byte secret that is then encoded in Base32. At the interface level, `handleGenerateNewSecret()` creates a new Base32 secret string, updates the shared secret in the demo, and resets the verification state. The function `handleVerifyTOTP()` compares the user input to the TOTP value currently generated by the application. These cryptographic operations are shown in Listing 3 on page 47.

```
// Generate mock data
const generateMockData = () => {
  const generateRandomString = (length) => {
    const chars =
      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    let result = ""
    for (let i = 0; i < length; i++) {
      result += chars.charAt(Math.floor(Math.random() * chars.length))
    }
    return result
  }

  return {
    challenge: generateRandomString(32),
    publicKey:
      "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE" + generateRandomString(40),
    signature: "MEUCIQDxHHDx4Ku1" + generateRandomString(60),
    credentialId: selectedMethod + "_" + generateRandomString(8),
    timestamp: new Date().toISOString(),
  }
}

// Return the step definitions based on the current method
const getFlowSteps = () => {
  switch (selectedMethod) {
    case "yubikey":
      return [
        {
          title: translation.challengeGenTitle,
          description: translation.challengeGenDesc,
          secret: translation.challengeGenSecret,
        },
        {
          title: translation.userPresenceTitle,
          description: translation.userPresenceDesc,
          secret: translation.userPresenceSecret,
        },
        {
          title: translation.keyPairGenTitle,
          description: translation.keyPairGenDescHardware,
          secret: translation.keyPairGenSecret,
        },
        {
          title: translation.signatureCreationTitle,
          description: translation.signatureCreationDesc,
          secret: translation.signatureCreationSecret,
        },
        {
          title: translation.keyStorageTitle,
          description: translation.keyStorageDescHardware,
          secret: translation.keyStorageSecret,
        },
```

```
        ]

      default: // passkey
        return [
          {
            title: translation.challengeGenTitle,
            description: translation.challengeGenDesc,
            secret: translation.challengeGenSecret,
          },
          {
            title: translation.userAuthTitle,
            description: translation.userAuthDesc,
            secret: translation.userAuthSecret,
          },
          {
            title: translation.keyPairGenTitle,
            description: translation.keyPairGenDescSoftware,
            secret: translation.keyPairGenSecret,
          },
          {
            title: translation.signatureCreationTitle,
            description: translation.signatureCreationDesc,
            secret: translation.signatureCreationSecret,
          },
          {
            title: translation.storageSyncTitle,
            description: translation.storageSyncDesc,
            secret: translation.storageSyncSecret,
          },
        ]
    }
  }

  // Handle switching between methods -- resets stateful outputs
  const handleMethodSelection = (method) => {
    setSelectedMethod(method)
    setRegistrationComplete(false)
    setRegistrationData({})
  }

  // Begin the mock "registration" flow
  const handleStartRegistration = () => {
    setIsRegistering(true)
    const mockData = generateMockData()
    setRegistrationData(mockData)

    // Simulate async work
    setTimeout(() => {
      setIsRegistering(false)
      setRegistrationComplete(true)

      // Prepare a consumable object for parent state (if provided)
```

```
      const newCredential = {
        id: mockData.credentialId,
        method: selectedMethod,
        publicKey: mockData.publicKey,
        label: `${registrationMethods[selectedMethod].title}:
  ${mockData.credentialId}`,
        timestamp: mockData.timestamp,
      }
      onRegisterCredential(newCredential)
    }, 2000)
  }

  const getRegistrationResultText = () => {
    switch (selectedMethod) {
      case "yubikey":
        return {
          authTitle: translation.step2TouchTitle,
          authDesc: translation.step2TouchDesc,
          keyLocation: translation.privateKeyHardware,
          successMsg: translation.registrationCompleteHardware,
        }
      case "passkey":
        return {
          authTitle: translation.step2FaceIdTitle,
          authDesc: translation.step2FaceIdDesc,
          keyLocation: translation.privateKeySoftware,
          successMsg: translation.registrationCompleteSoftware,
        }
    }
  }
```

Listing 1: Implementation code of the simulated passkey registration flow

```
// Generates mock values for authentication
const generateMockAuthData = () => {
  const generateRandomString = (length) => {
    const chars =
      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    let result = ""
    for (let i = 0; i < length; i++)
      result += chars.charAt(Math.floor(Math.random() * chars.length))
    return result
  }
  return {
    challenge: generateRandomString(32),
    signature: "MEUCIQDyHHEx5Lu2" + generateRandomString(60),
    timestamp: new Date().toISOString(),
  }
}

// Looks up the full credential object from the selected id
const getSelectedCredential = () => {
  return registeredCredentials.find((cred) => cred.id === selectedCredential)
}

// Returns titles/descriptions for the user-action step based on method
const getUserActionText = () => {
  switch (selectedAuthMethod) {
    case "yubikey":
      return {
        title: translation.authStep3TouchTitle,
        description: translation.authStep3TouchDesc,
      }
    case "passkey":
    default:
      return {
        title: translation.authStep3BiometricTitle,
        description: translation.authStep3BiometricDesc,
      }
  }
}

// Success message varies slightly per method
const getSuccessMessage = () => {
  switch (selectedAuthMethod) {
    case "yubikey":
      return translation.authSuccessHardware
    case "passkey":
    default:
      return translation.authSuccessSoftware
  }
}

// Starts the simulated authentication
const handleStartAuthentication = () => {
```

```
    if (!selectedAuthMethod || !selectedCredential) {
      alert("Please select both authentication method and credential")
      return
    }
    setIsAuthenticating(true)
    const mockData = generateMockAuthData()
    setAuthData(mockData)

    // Simulate async server/device work to show the timeline
    setTimeout(() => {
      setIsAuthenticating(false)
      setAuthComplete(true)
    }, 3000)
  }

  // Resets the flow output when the method/credential changes
  const resetAuthentication = () => {
    setAuthComplete(false)
    setAuthData({})
    setIsAuthenticating(false)
  }
```

Listing 2: Implementation code of the simulated passkey authentication flow

```
const TOTPUtils = {
  // Base32 decoding: converts Base32-encoded secret to bytes
  base32Decode(encoded) {
    const alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";
    let bits = "";
    let result = [];

    for (let char of encoded.replace(/=/g, "")) {
      const val = alphabet.indexOf(char.toUpperCase());
      bits += val.toString(2).padStart(5, "0");
    }

    for (let i = 0; i + 8 <= bits.length; i += 8) {
      result.push(parseInt(bits.substring(i, i + 8), 2));
    }

    return new Uint8Array(result);
  },

  // Base32 encoding: converts bytes to Base32-encoded string
  base32Encode(data) {
    const alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";
    let bits = "";
    let encoded = "";

    for (let byte of data) {
      bits += byte.toString(2).padStart(8, "0");
    }

    for (let i = 0; i < bits.length; i += 5) {
      const chunk = bits.substring(i, i + 5);
      const index = parseInt(chunk.padEnd(5, "0"), 2);
      encoded += alphabet[index];
    }

    while (encoded.length % 8 !== 0) {
      encoded += "=";
    }

    return encoded;
  },

  // SHA-1 implementation
  sha1(data) {
    const msg = new Uint8Array(data);
    const ml = msg.length * 8;

    let withOne = new Uint8Array(msg.length + 1);
    withOne.set(msg);
    withOne[msg.length] = 0x80;

    let zeroBytes =
```

```
      (56 - (withOne.length % 64) + 64) % 64;
    let padded = new Uint8Array(withOne.length + zeroBytes + 8);
    padded.set(withOne);

    let mlBits = new Uint8Array(8);
    for (let i = 7; i >= 0; i--) {
      mlBits[i] = ml & 0xff;
      ml >>>= 8;
    }
    padded.set(mlBits, padded.length - 8);

    let h0 = 0x67452301;
    let h1 = 0xefcdab89;
    let h2 = 0x98badcfe;
    let h3 = 0x10325476;
    let h4 = 0xc3d2e1f0;

    for (let i = 0; i < padded.length; i += 64) {
      let w = new Uint32Array(80);
      for (let j = 0; j < 16; j++) {
        w[j] =
          (padded[i + j * 4] << 24) |
          (padded[i + j * 4 + 1] << 16) |
          (padded[i + j * 4 + 2] << 8) |
          padded[i + j * 4 + 3];
      }
      for (let j = 16; j < 80; j++) {
        w[j] = ((w[j - 3] ^ w[j - 8] ^ w[j - 14] ^ w[j - 16]) << 1) |
               ((w[j - 3] ^ w[j - 8] ^ w[j - 14] ^ w[j - 16]) >>> 31);
      }

      let a = h0, b = h1, c = h2, d = h3, e = h4;

      for (let j = 0; j < 80; j++) {
        let f, k;
        if (j < 20) {
          f = (b & c) | (~b & d); k = 0x5a827999;
        } else if (j < 40) {
          f = b ^ c ^ d; k = 0x6ed9eba1;
        } else if (j < 60) {
          f = (b & c) | (b & d) | (c & d); k = 0x8f1bbcdc;
        } else {
          f = b ^ c ^ d; k = 0xca62c1d6;
        }

        const temp =
          (((a << 5) | (a >>> 27)) + f + e + k + w[j]) >>> 0;
        e = d >>> 0;
        d = c >>> 0;
        c = ((b << 30) | (b >>> 2)) >>> 0;
        b = a >>> 0;
        a = temp >>> 0;
```

```
105       }
106
107       h0 = (h0 + a) >>> 0;
108       h1 = (h1 + b) >>> 0;
109       h2 = (h2 + c) >>> 0;
110       h3 = (h3 + d) >>> 0;
111       h4 = (h4 + e) >>> 0;
112     }
113
114     return new Uint8Array([
115       h0 >>> 24, (h0 >>> 16) & 0xff, (h0 >>> 8) & 0xff, h0 & 0xff,
116       h1 >>> 24, (h1 >>> 16) & 0xff, (h1 >>> 8) & 0xff, h1 & 0xff,
117       h2 >>> 24, (h2 >>> 16) & 0xff, (h2 >>> 8) & 0xff, h2 & 0xff,
118       h3 >>> 24, (h3 >>> 16) & 0xff, (h3 >>> 8) & 0xff, h3 & 0xff,
119       h4 >>> 24, (h4 >>> 16) & 0xff, (h4 >>> 8) & 0xff, h4 & 0xff,
120     ]);
121   },
122
123   // HMAC-SHA1 used for TOTP generation
124   hmacSha1(key, message) {
125     const blockSize = 64;
126     let k = key;
127     if (k.length > blockSize) {
128       k = this.sha1(k);
129     }
130     if (k.length < blockSize) {
131       const tmp = new Uint8Array(blockSize);
132       tmp.set(k);
133       k = tmp;
134     }
135
136     let oKeyPad = new Uint8Array(blockSize);
137     let iKeyPad = new Uint8Array(blockSize);
138
139     for (let i = 0; i < blockSize; i++) {
140       oKeyPad[i] = k[i] ^ 0x5c;
141       iKeyPad[i] = k[i] ^ 0x36;
142     }
143
144     const innerHash = this.sha1(
145       new Uint8Array([...iKeyPad, ...message])
146     );
147     return this.sha1(new Uint8Array([...oKeyPad, ...innerHash]));
148   },
149
150   // Main TOTP generation function
151   generateTOTP(secret, timeStep = null) {
152     const decodedSecret = this.base32Decode(secret);
153     const time = timeStep ?? Math.floor(Date.now() / 30000);
154
155     const counter = new Uint8Array(8);
156     for (let i = 7; i >= 0; i--) {
```

```
157       counter[i] = time & 0xff;
158       time >>>= 8;
159     }
160
161     const hmac = this.hmacSha1(decodedSecret, counter);
162
163     const offset = hmac[hmac.length - 1] & 0x0f;
164     const binary =
165       ((hmac[offset] & 0x7f) << 24) |
166       (hmac[offset + 1] << 16) |
167       (hmac[offset + 2] << 8) |
168       hmac[offset + 3];
169
170     return binary % 1000000;
171   },
172
173   // Generates a new random shared secret
174   generateRandomSecret() {
175     const bytes = new Uint8Array(20);
176     window.crypto.getRandomValues(bytes);
177     return this.base32Encode(bytes);
178   },
179 };
```

Listing 3: Implementation code of the TOTP cryptographic operations

4.3.2 TOTP generation

The TOTP generation tab introduces TOTP as a method for generating OTP used as the second factor in 2FA systems and allows users to experiment with the underlying algorithm. Fig. 8 demonstrates the introductory layout of the TOTP generation page. Two information cards compare the secrets stored on both the client and server. Additionally, an example view of Google authenticator app is shown, illustrating multiple TOTP codes for various services. A formula box presents the TOTP algorithm and its truncation process, explaining how TOTP codes are derived from the shared secret and current time as shown in Fig. 9 on the following page.

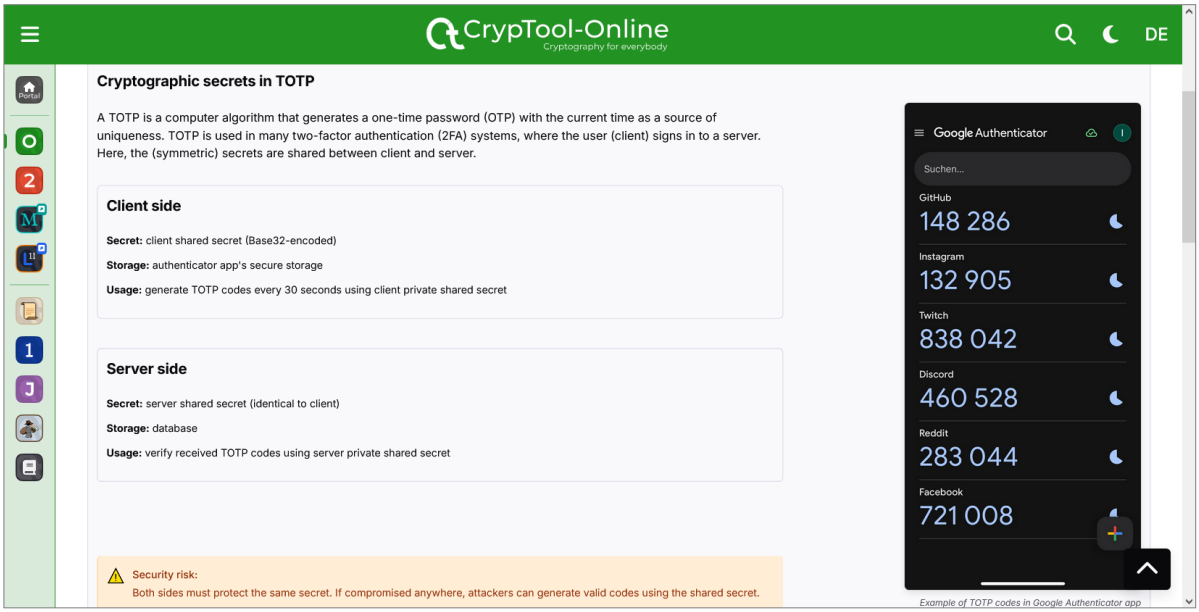

Figure 8: Overview of TOTP

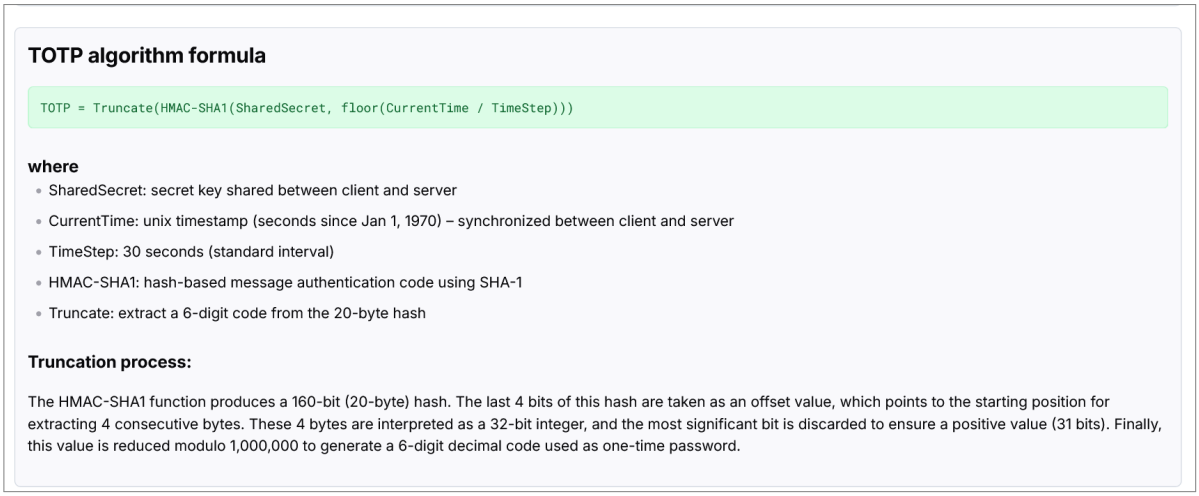

Figure 9: TOTP algorithm formula and truncation process

The interactive part of the TOTP generation tab contains the following elements (see Fig. 10 on the following page):

- **Shared secret field**: The current Base32-encoded shared secret is displayed in a text box. Users can manually copy it to an authenticator app, i.e., Google authenticator or generate a new secret by clicking the *Generate new secret* button.
- **QR code**: A QR code encodes the `otpauth://` URI for the shared secret. Users are instructed to scan it with Google authenticator app. The QR code relies on the Base32 decoding.
- **Live TOTP code**: The demonstrator uses an interval timer that updates the countdown every second and recomputes the TOTP code whenever a new 30-second window begins. The code updates automatically as time advances.
- **Verification field**: Users can type the 6-digit TOTP code displayed in their authenticator app. Clicking the *Verify TOTP* button compares the entered value with the code computed on the server. If the values match, a green success alert appears; otherwise, a red error alert indicates that the verification failed.

As explained earlier in Chapter 4.3.1, the core implementation of these operations is provided in the `TOTPUtils` helper object, shown in Listing 3 on page 47.

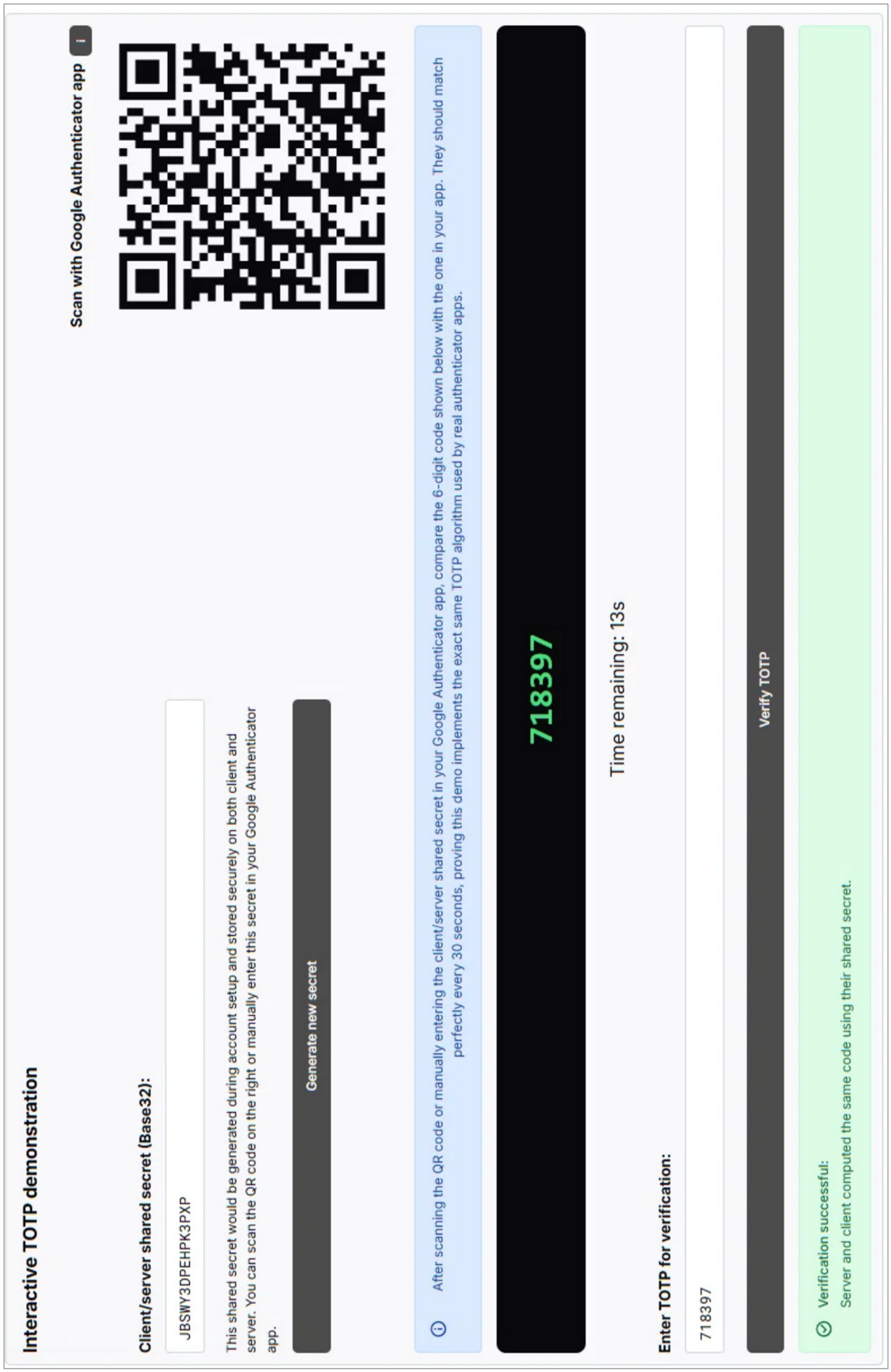

Figure 10: Interactive TOTP demonstration interface

4.3.3 Attack simulation

The attack simulation tab highlights the limitations of TOTP-based 2FA. Fig. 11 provides an overview of the main vulnerabilities. The tab further offers two interactive attack scenarios: time drift attack and phishing/replay attack, which illustrate the practical implications of these vulnerabilities.

Figure 11: Vulnerabilities when TOTP is used as the second factor in 2FA

Time drift attack In this scenario, users adjust the client clock shift via a stepper input. A positive value means the client clock is ahead of the server, a negative value means it is behind. When the user clicks the *Simulate time drift attack* button, the system computes whether the TOTP code would still be accepted. If the absolute drift exceeds the typical tolerance of ±60 seconds, the simulation shows a red error alert explaining that the authentication failed. Otherwise, it displays a green success alert with the drift value and a note that most servers allow a ±30–60 second window. Both cases are illustrated in Fig. 12 and Fig. 13 on the following page.

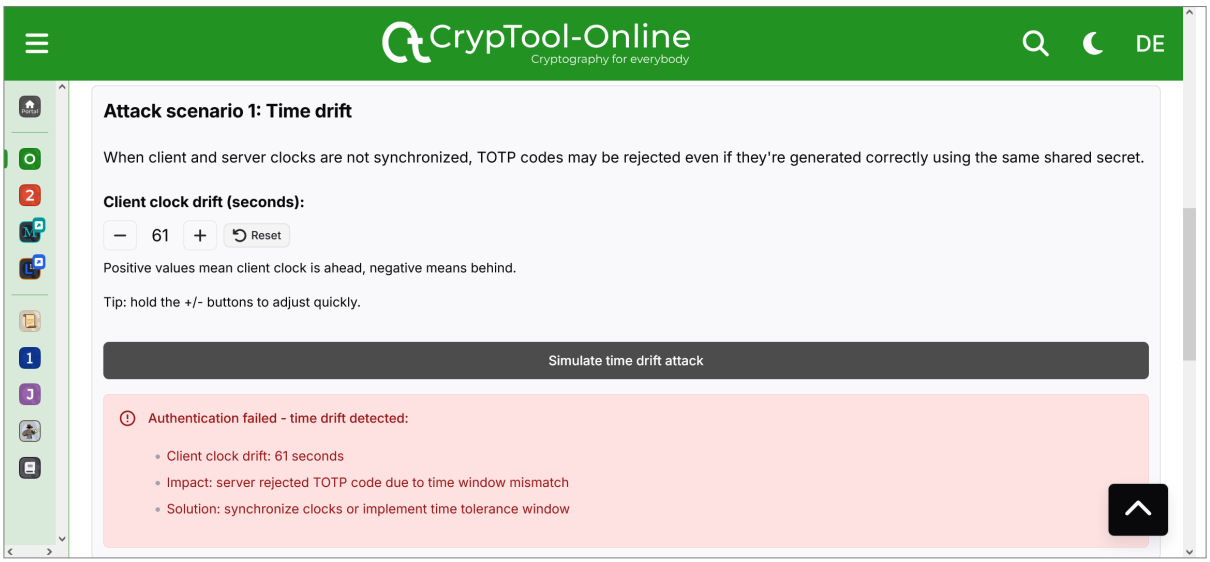

Figure 12: Time drift attack simulation 1

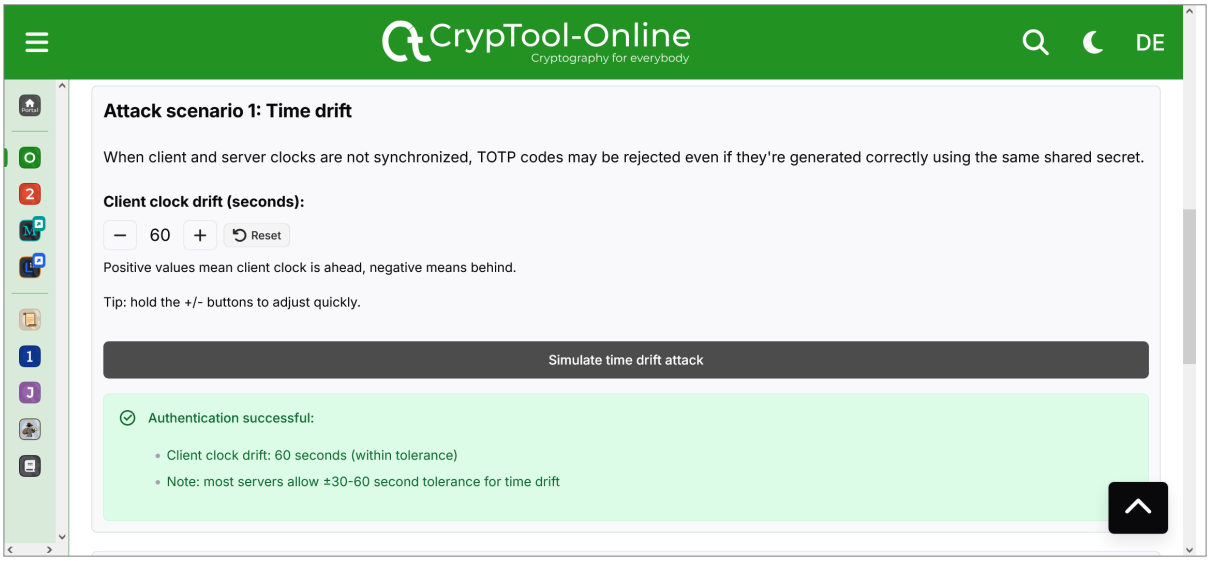

Figure 13: Time drift attack simulation 2

Phishing and replay attack This scenario illustrates how phishing attacks can manipulate TOTP codes. A timeline describes how an attacker gains access to valid TOTP codes and replays them to the legitimate server. *Simulate phishing attack* button produces an error alert summarizing that the attacker successfully replayed the stolen code and that TOTP provide no origin binding. Fig. 14 illustrates the interface.

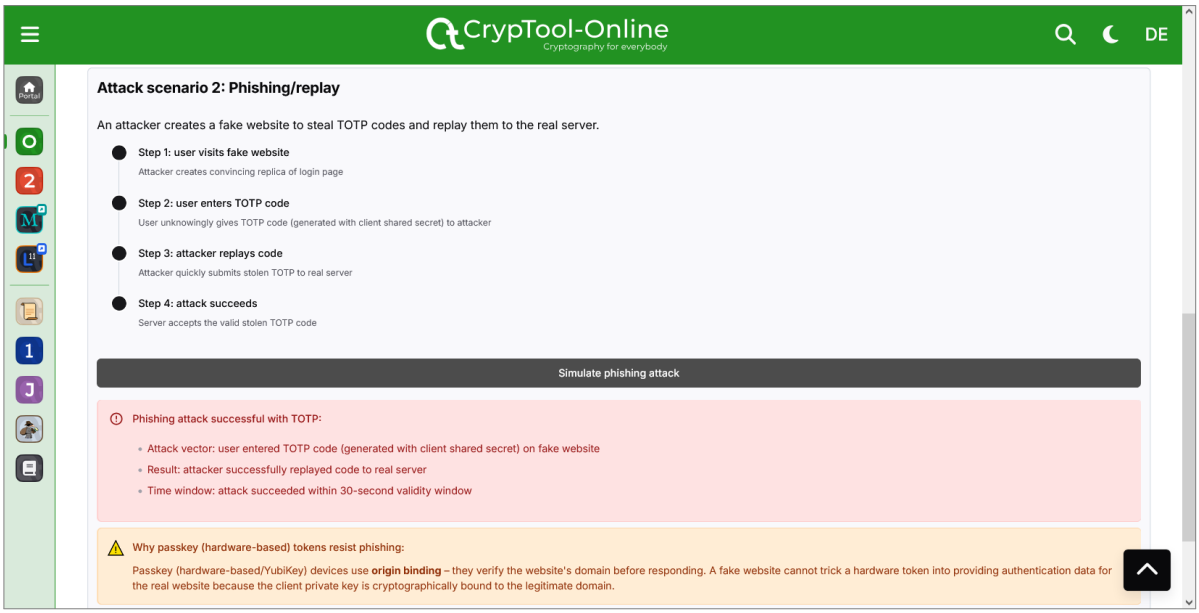

Figure 14: Phishing attack simulation

4.3.4 Passkey registration

The passkey registration tab is designed to familiarize users with the cryptographic foundations and practical workflow of registering a passkey. The tab contrasts hardware-based (YubiKey) and software-based (platform-integrated) authentication methods, explains how asymmetric key pairs are created and stored, and provides an interactive simulation of each stage during the passkey registration process.

Cryptographic secrets in passkeys In contrast to TOTP, passkeys rely on asymmetric cryptography rather than a shared secret. The upper section of the passkey registration tab first introduces this concept in a descriptive paragraph. Below this explanation, two cards labelled *Client side* and *Server side* illustrate where the private and public keys are stored. This can be seen in Fig. 15.

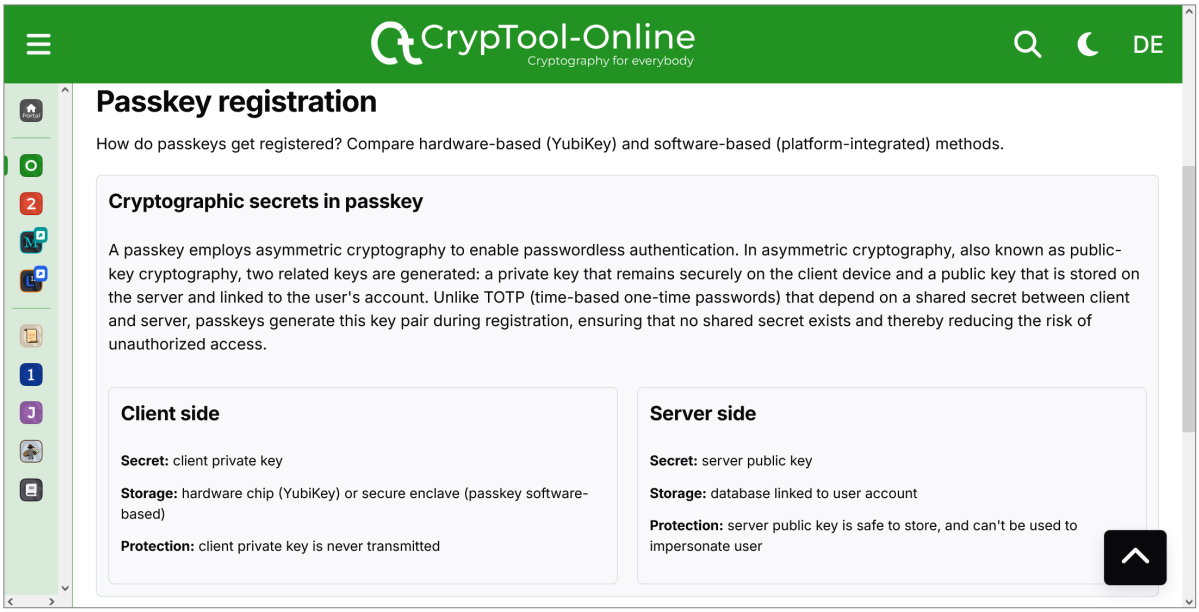

Figure 15: Cryptographic secrets in passkeys

Registration simulation The registration simulation guides the user through the passkey registration process. After selecting a YubiKey (hardware-based) or a platform passkey (software-based) via a drop-down menu, a timeline displays the following steps:

1. **Challenge generation:** The server generates a random challenge and sends it to the client.

2. **User verification:** Depending on the registration method selected, the user is prompted to either touch the YubiKey (physical presence), use biometric authentication (Face ID or fingerprint), or enter a PIN on a platform authenticator.

3. **Key pair generation:** The client device generates a new key pair. In the case of hardware tokens, this generation occurs within a tamper-resistant secure chip, while software passkeys use the operating system's secure enclave.

4. **Signature creation:** The client private key is used to sign the challenge. The resulting signature proves possession of the private key without exposing it.

5. **Key storage:** The passkey registration process is completed when the server stores the credential ID and public key for future authentication attempts.

The user interface for the passkey registration process is shown in Fig. 16 on page 59 and in Fig. 17 on page 60, while the corresponding core implementation, explained earlier in Chapter 4.3.1, is provided in Listing 1 on page 42.

Registration output Once the registration process is completed, the user interface shown in Fig. 18 on page 61 and in Fig. 19 on page 62 presents detailed information about the generated key pair, including the credential ID and public key, which will be used during the authentication process described on page 63.

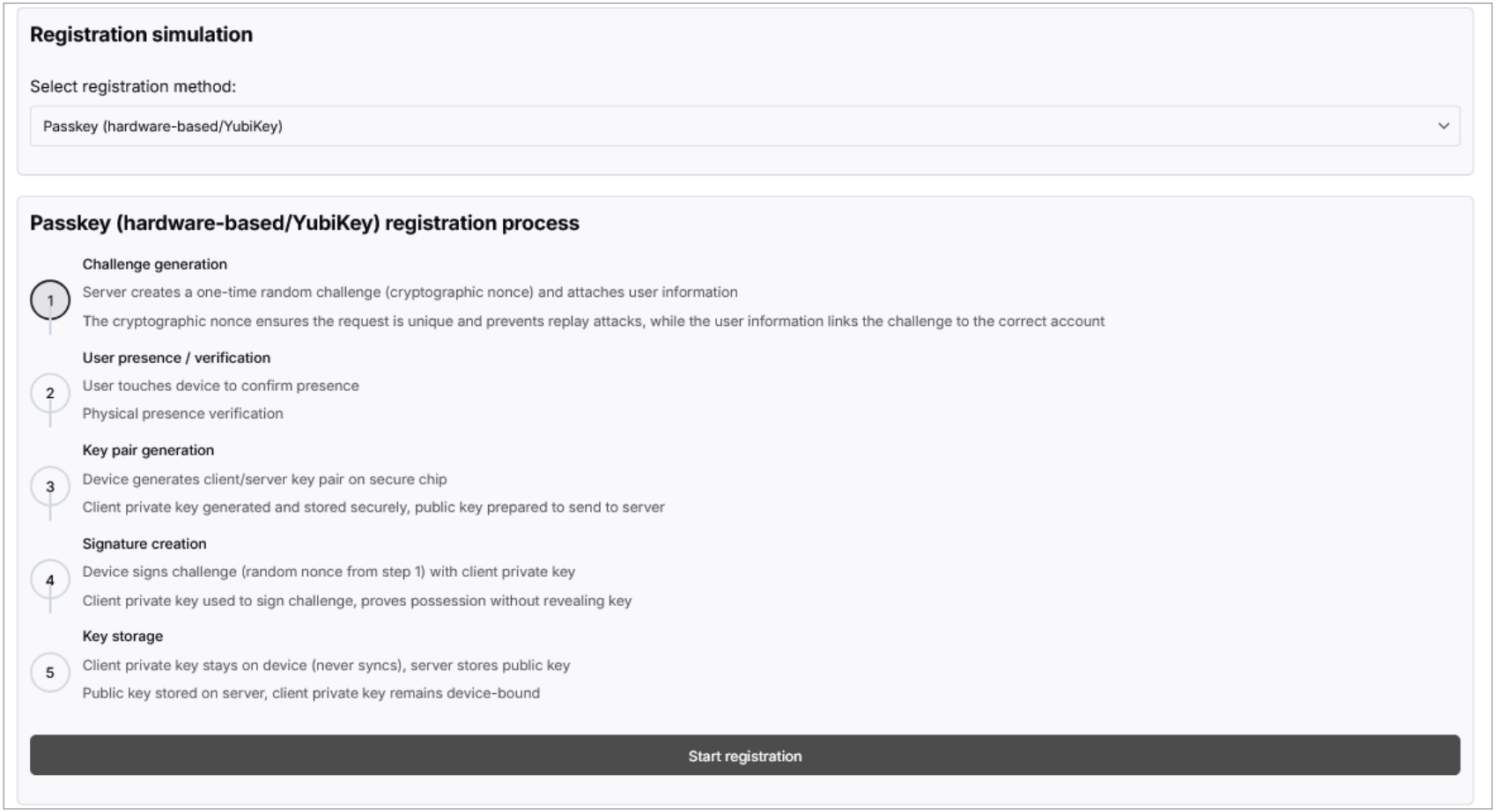

Figure 16: Hardware-based passkey registration process

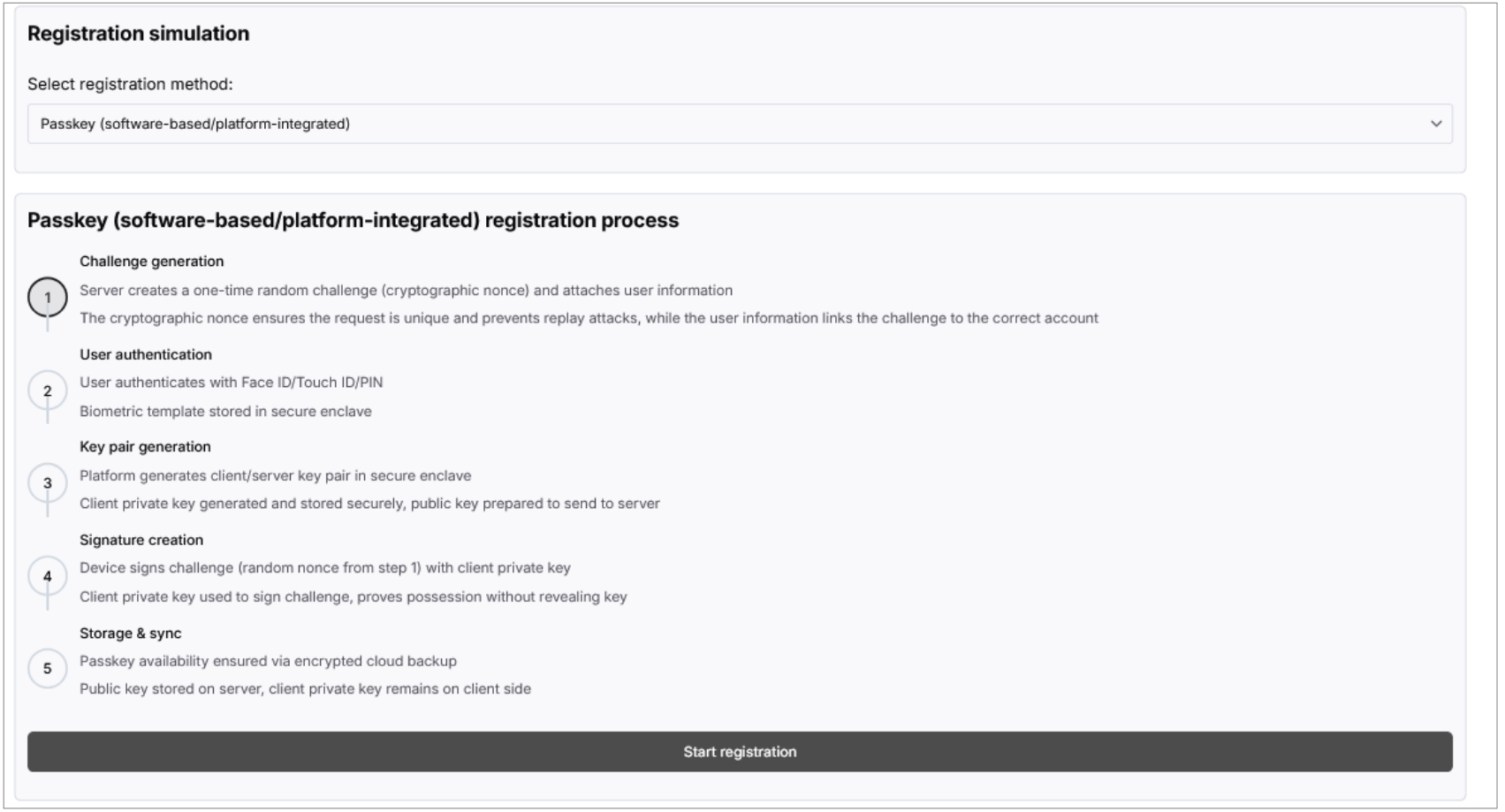

Figure 17: Software-based passkey registration process

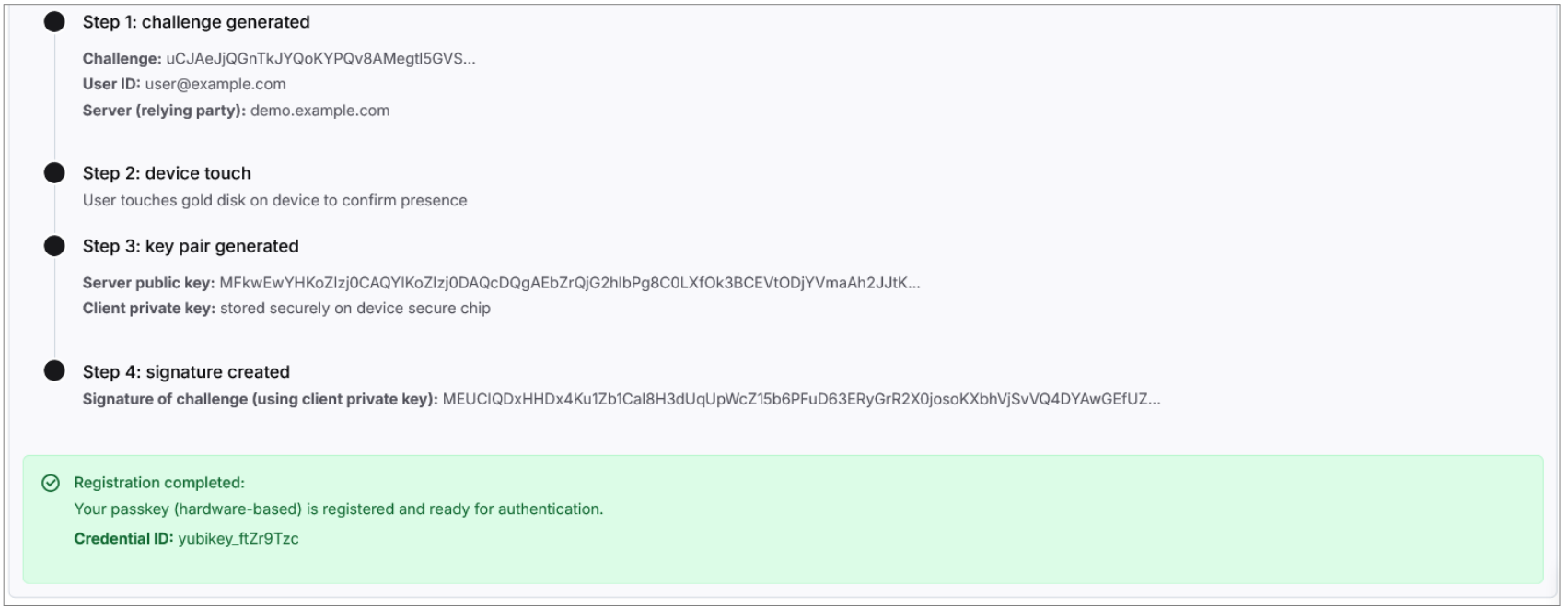

Figure 18: Hardware-based passkey credential details

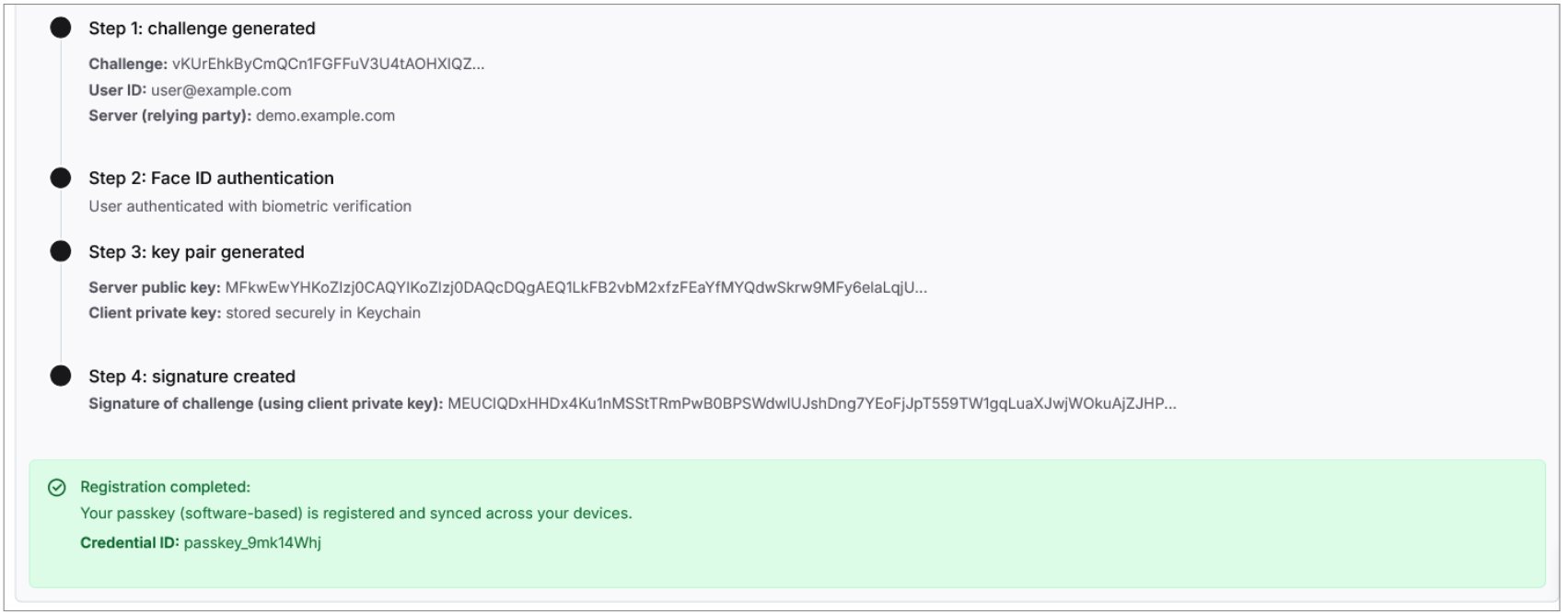

Figure 19: Software-based passkey credential details

4.3.5 Passkey authentication

The passkey authentication tab demonstrates how the previously registered passkeys (see page 57) are used to authenticate users via the WebAuthn protocol. The tab compares hardware-based and software-based authentication methods across several aspects, as shown in Fig. 20, and provides an interactive simulation of each stage of the authentication process.

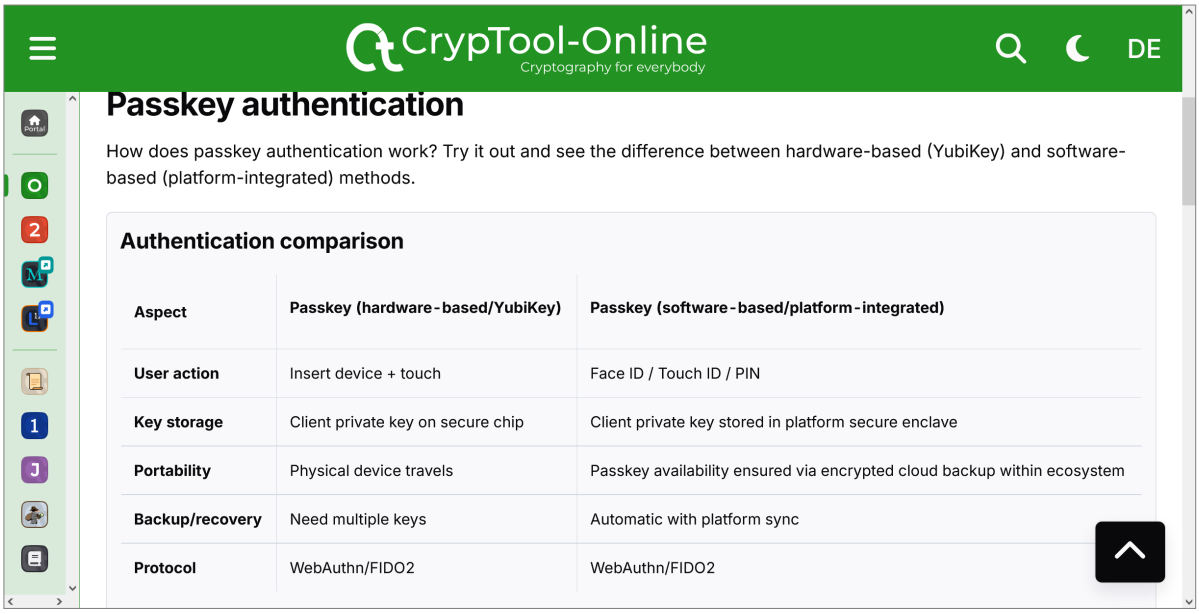

Figure 20: Comparison of hardware and software passkey authentication

WebAuthn authentication protocol A diagram summarizes the five stages of the WebAuthn authentication protocol: establishing a secure TLS connection, sending the challenge and credential IDs, verifying the user, generating the signature, and verifying it on the server. This can be seen in more details in Fig. 21 on the following page.

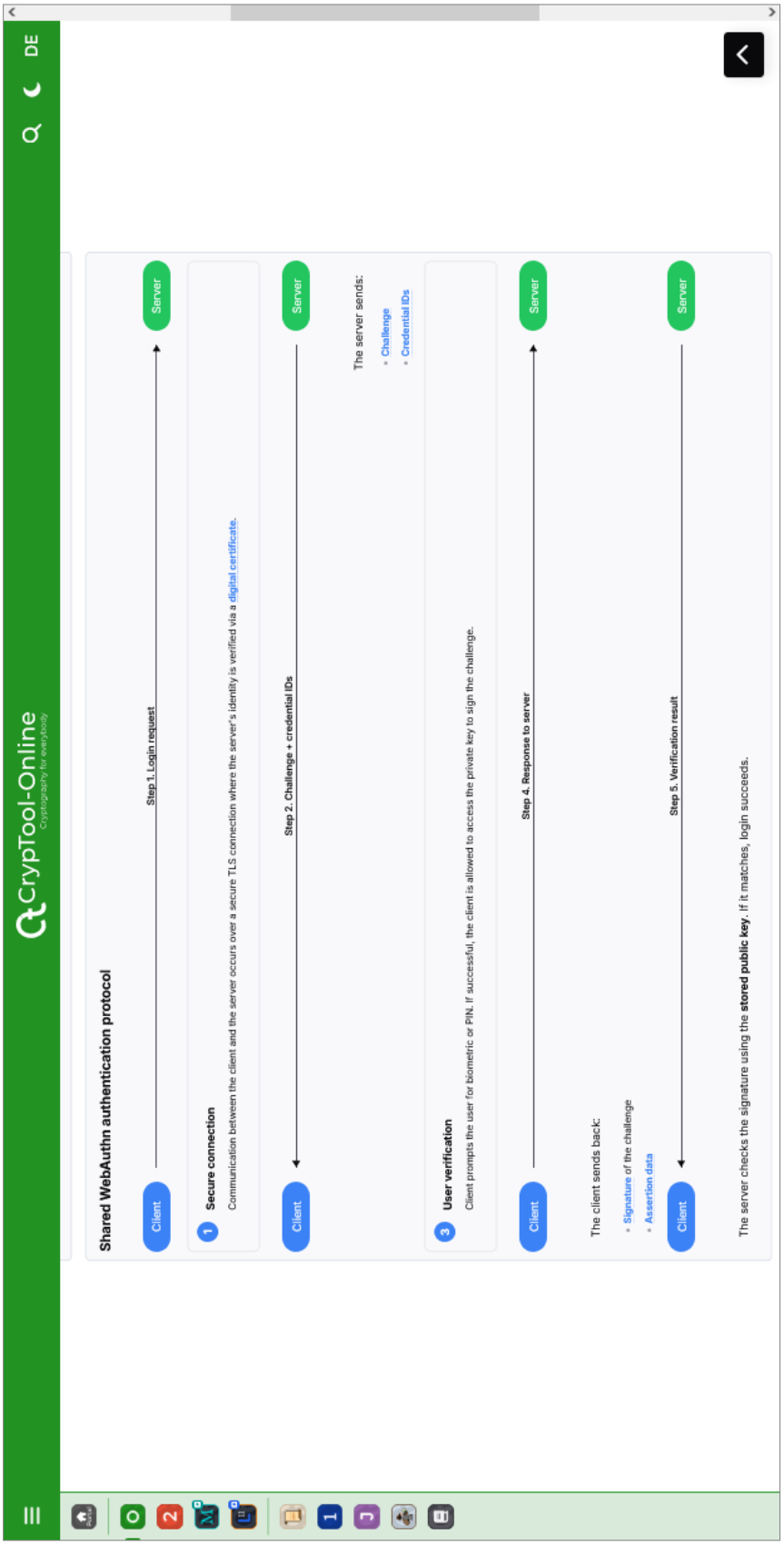

Figure 21: Visualization of the WebAuthn protocol

Authentication simulation The interactive authentication simulator allows users to choose an authentication method and select a registered credential, as shown in Figs. 23–24 on page 67. Upon clicking the *Start authentication* button the following sequence is displayed:

1. **Login request sent:** User clicks the *Login with security key* button.

2. **Server challenge:** The server generates and sends a random challenge (cryptographic nonce) to prove the authentication is current and prevent replay attacks.

3. **User verification:** The client prompts the user to prove presence, either via biometric authentication for software-based passkeys or by touching the device for hardware-based passkeys.

4. **Device signs challenge:** The client device signs the challenge with its private key; only the device holding the private key can generate this signature.

5. **Server verifies signature:** The signed challenge is returned to the server, which verifies it using the stored public key. If the signature matches, authentication succeeds.

The user interfaces in Fig. 25 on page 68 and Fig. 26 on page 69 show the detailed authentication steps, while the corresponding core implementation, explained earlier in Chapter 4.3.1, is provided in Listing 2 on page 45.

Security advantages At the bottom of the passkey authentication tab a list summarizes why passkey authentication is more secure than traditional authentication methods. Both hardware-based and software-based passkeys share important properties:

1. The client's private key is never transmitted.

2. Each challenge is unique and time-bound.

3. Origin binding prevents phishing on fake sites.

4. Authentication relies on strong, proven public-key cryptography.

5. Both hardware-based and software-based passkey authentication methods follow the WebAuthn standard.

An illustrative screenshot of these advantages is provided in Fig. 22 on the following page.

Figure 22: Shared security advantages of hardware-based and software-based passkeys

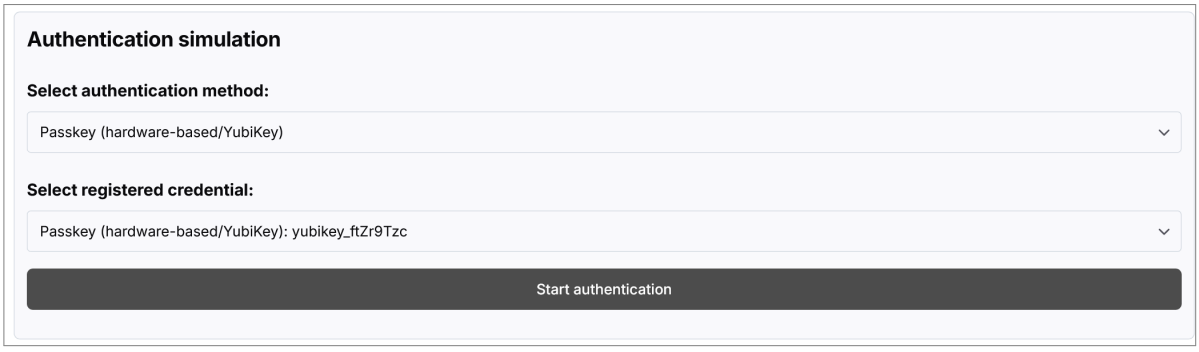

Figure 23: Authentication interface for hardware-based passkeys (YubiKey): method and credential selection

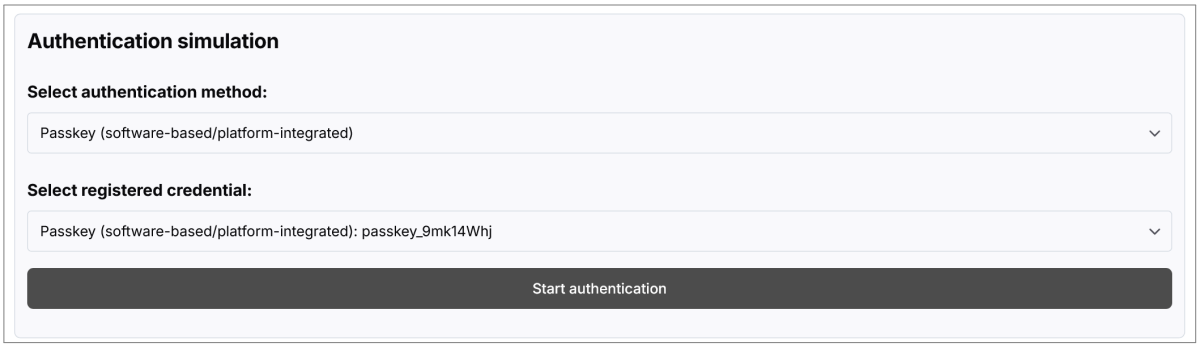

Figure 24: Authentication interface for software-based passkeys: method and credential selection

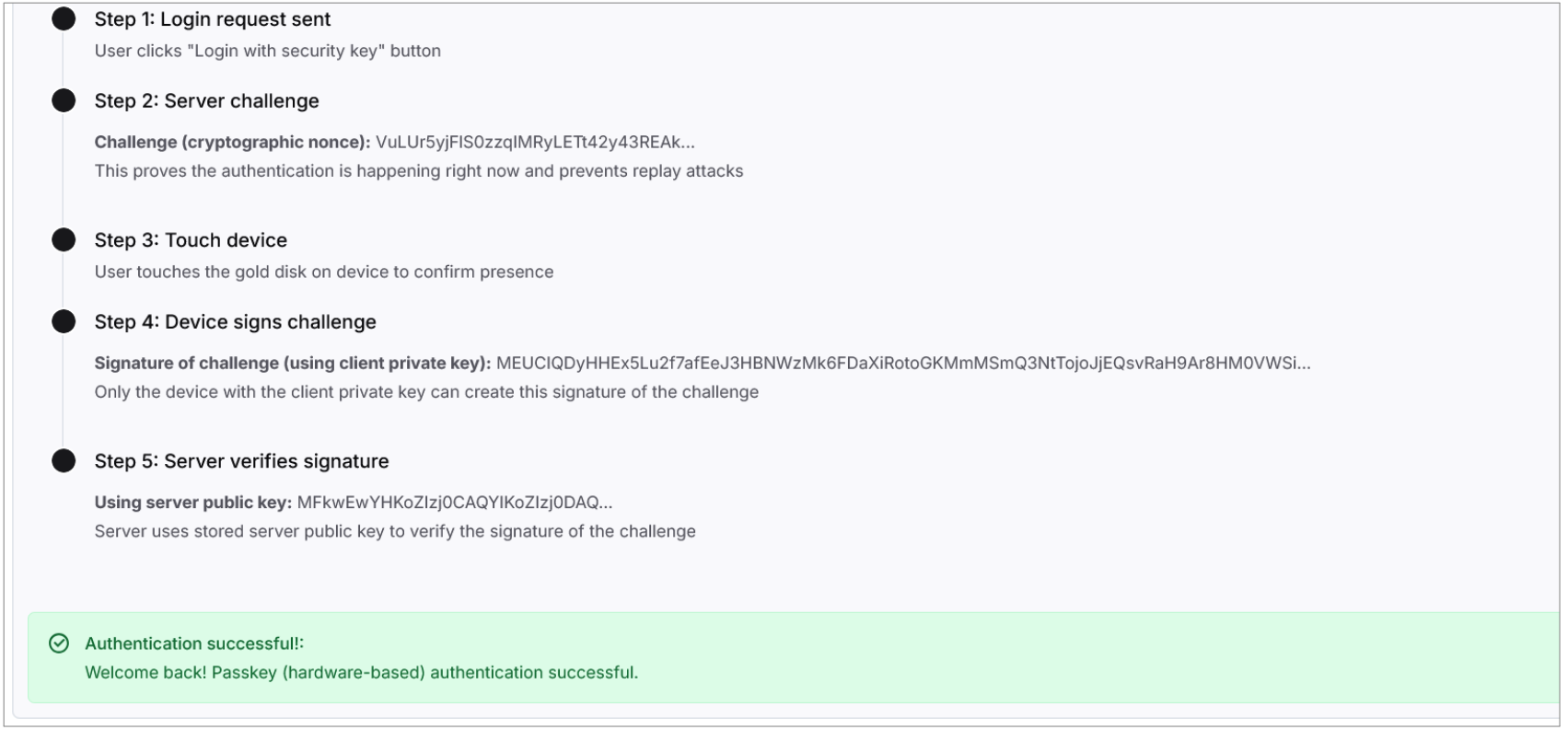

Figure 25: Authentication interface for hardware-based passkeys (YubiKey): authentication steps

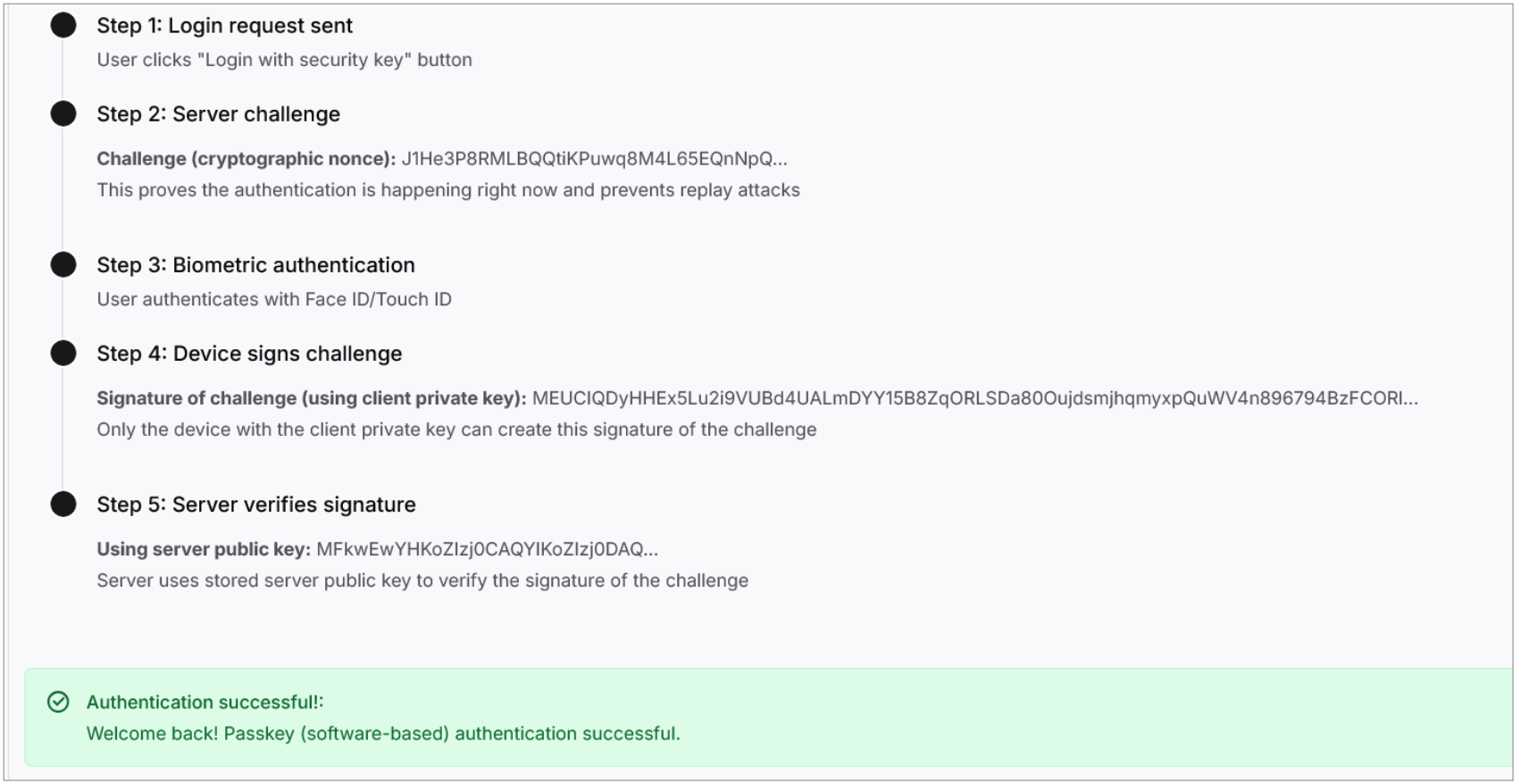

Figure 26: Authentication interface for software-based platform passkeys: authentication steps

4.3.6 Recovery planning

Recovery planning is crucial in 2FA, as it ensures that users can regain access if their authentication device is lost. Without a proper recovery strategy, losing a phone or hardware token can permanently lock users out of their accounts. 2FA methods employ different approaches when the authentication device is lost. TOTP uses symmetric shared secrets which cannot be recovered if lost. Hardware-based passkeys store private keys on physical devices that are gone forever if the device is lost. Software-based passkeys sync across devices and offer better recovery options. Understanding these differences is essential for effective backup planning, which is why the recovery planning tab presents three different scenarios:

1. **Phone lost (TOTP app):** Losing the phone results in the loss of the TOTP client's private shared secret, which cannot be recovered. Prevention measure include saving backup codes offline and registering a second device with the same secret. Recovery options are limited to using backup codes or a secondary device.

2. **Hardware token lost:** Losing the hardware token results in the loss of the private key stored on that token. Prevention measures includes registering multiple hardware tokens, storing them separately and keeping backup codes. Recovery is possible using a secondary hardware token; the private keys on lost tokens cannot be restored.

3. **Software-based device lost:** Losing a device with platform-integrated passkeys results in the loss of the device-specific private keys and the associated biometric unlock capability. Prevention is to register multiple devices and enable platform sync across these devices. Recovery options include using synced passkeys on another device or cross-device authentication.

Following the scenarios presented in Figs. 27–28 on the next page and in Fig. 29 on page 72, a comparison table summarizes the recovery difficulty and backup options together with the best-practice recommendations for TOTP, hardware-based and software-based passkeys. This can be seen in Fig. 30 on page 72.

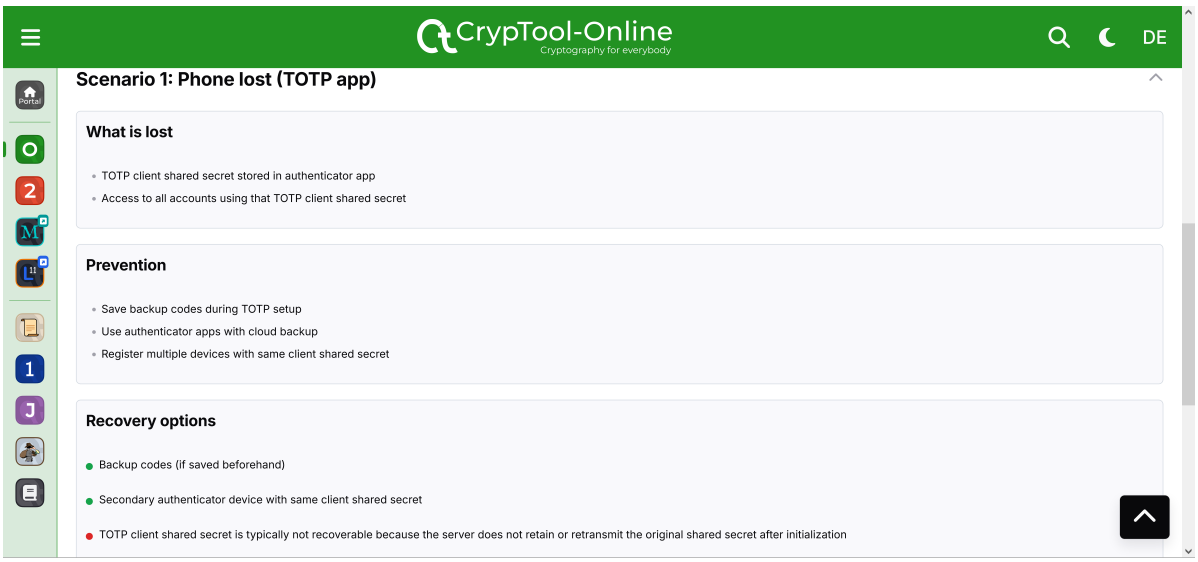

Figure 27: Phone lost scenario

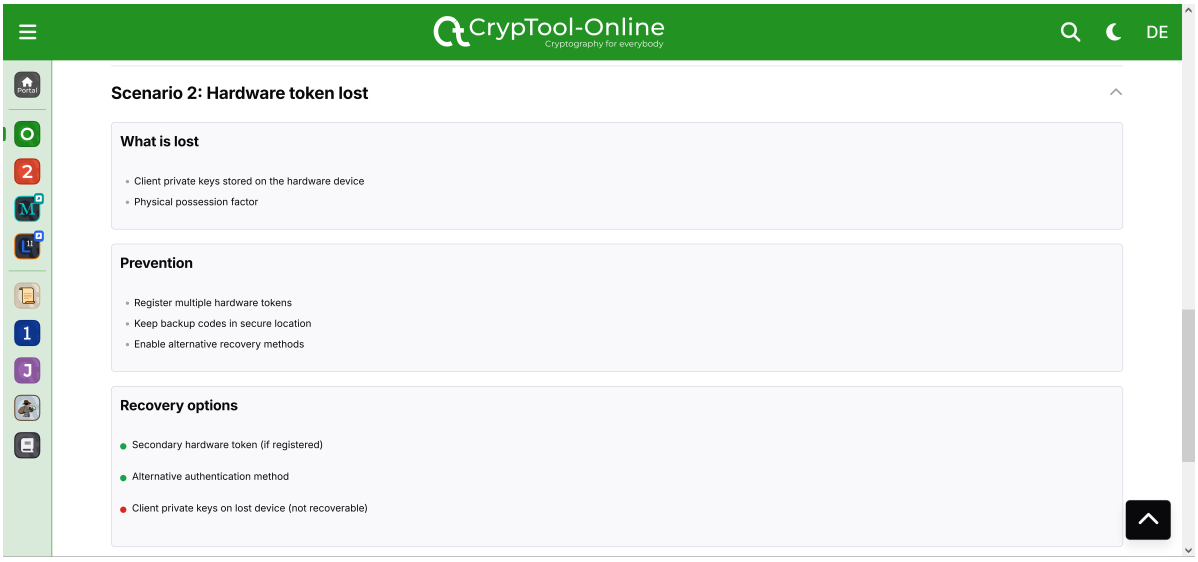

Figure 28: Hardware token lost scenario

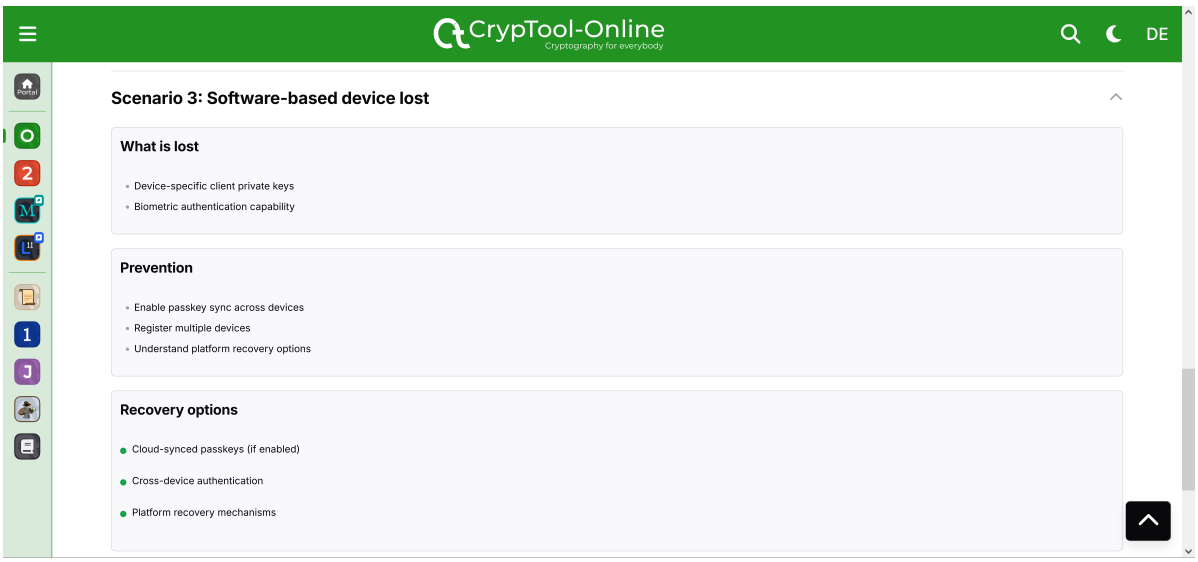

Figure 29: Software-based device lost scenario

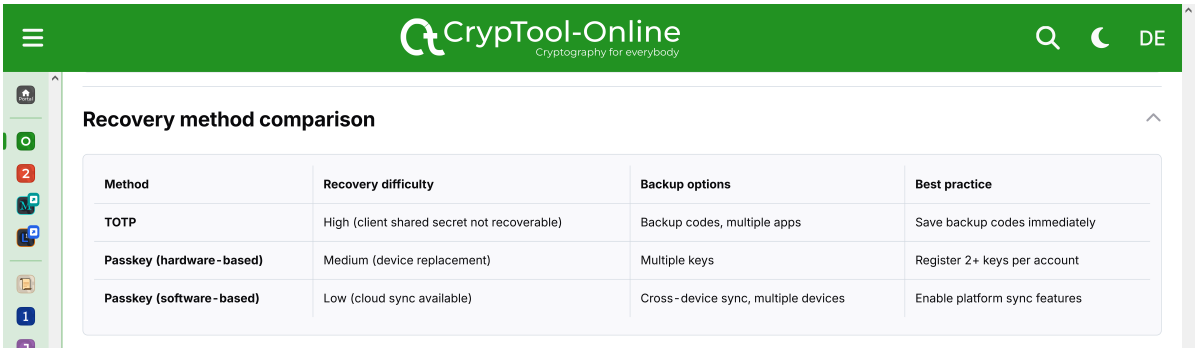

Figure 30: Recovery method comparison

4.3.7 Self-assessment and summary

The self-assessment and summary tab allows users to check their understanding of 2FA concepts and security principles. It opens with a short privacy notice stating that "This quiz is evaluated entirely in your browser. No answers are sent to any server—your responses remain private".

The knowledge assessment quiz consists of ten multiple-choice questions covering cryptographic method differences. Each question has four possible answers, and the user must select one answer per question before submitting the quiz. A progress bar indicates how many questions have been completed. When the user clicks the *Submit quiz* button, the quiz module validates that all questions have been answered and then calculates the score. Feedback is presented through statistical elements showing the number of correct responses, the overall percentage, and a qualitative assessment such as "Excellent", "Good progress", or "Needs improvement". Below the score, the quiz module lists the topics that require further review based on the user's incorrect answers. A *Retake quiz* button allows users to reset all results and complete the assessment again, enabling repeated practice of the underlying concepts. The user interface is shown in Figs. 31–32 on the following page and in Figs. 33–34 on page 75.

Once the quiz is completed, a 2FA methods summary table compares TOTP, hardware passkeys, and software passkeys across several aspects. Fig 35 on page 76 presents the table. Finally, the tab concludes with a short explanation of why these authentication methods are considered secure. This can be seen in Fig 36 on page 76.

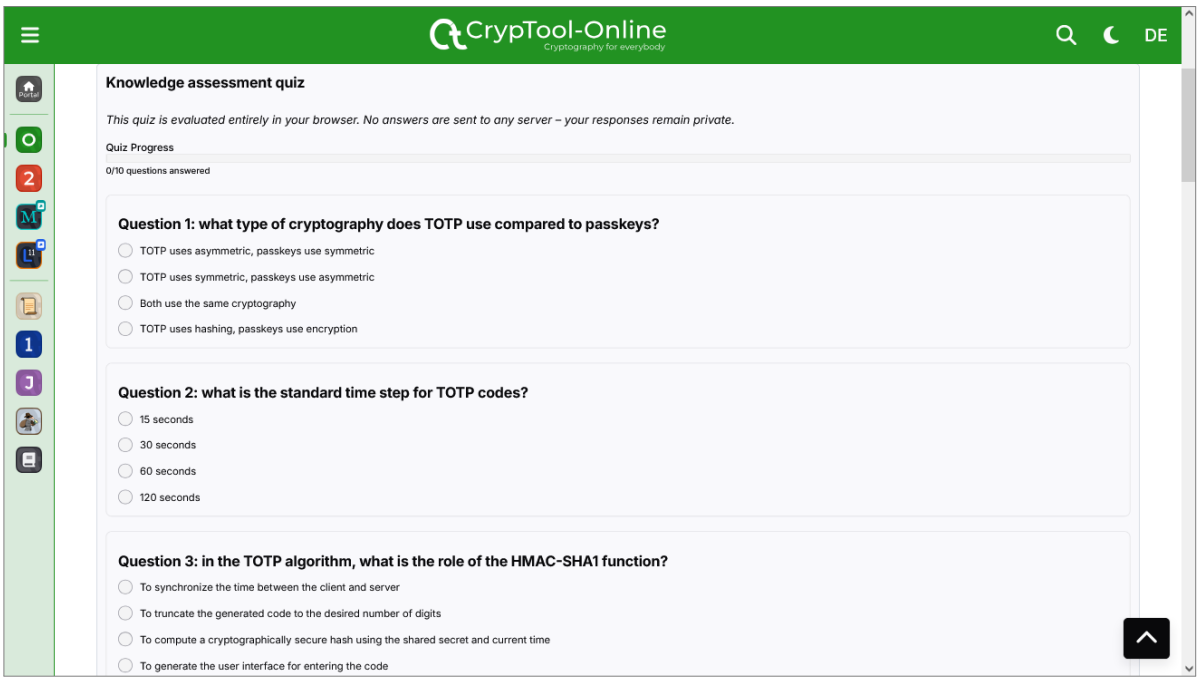

Figure 31: Knowledge assessment quiz interface 1

Figure 32: Knowledge assessment quiz interface 2

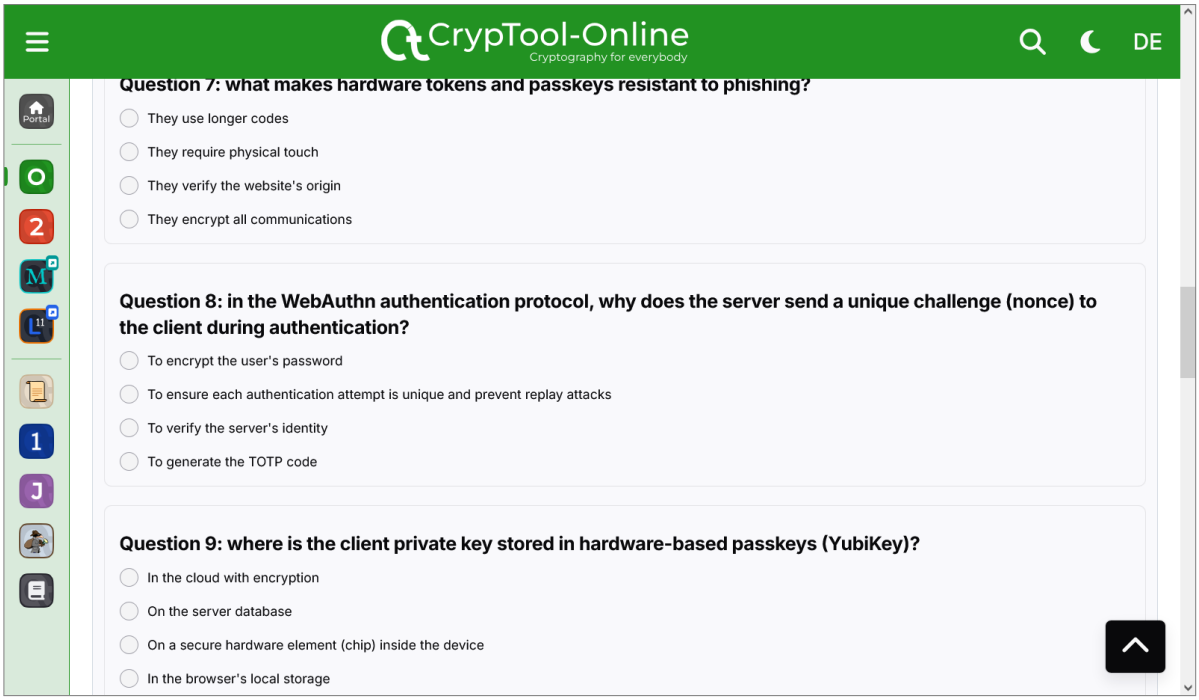

Figure 33: Knowledge assessment quiz interface 3

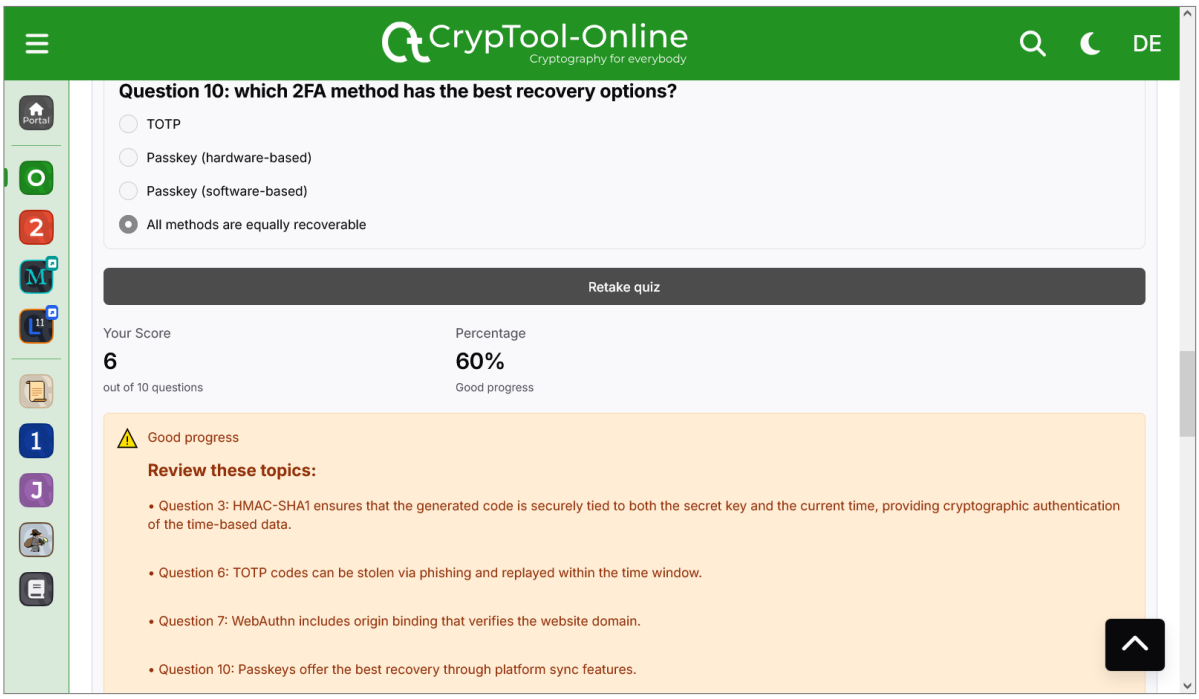

Figure 34: Knowledge assessment quiz interface 4

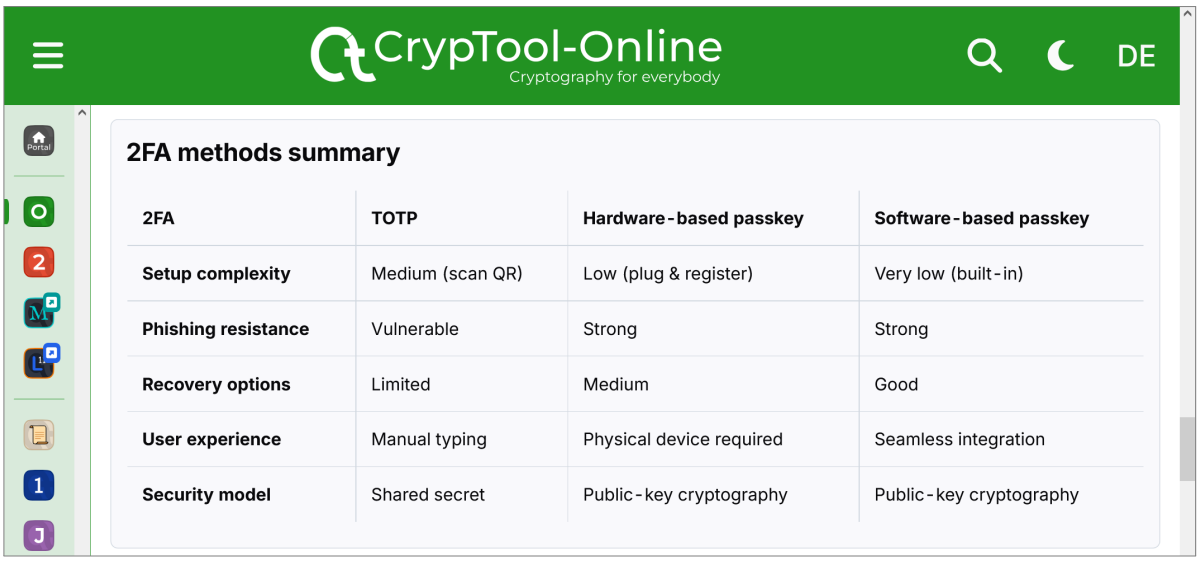

Figure 35: Summary table comparing 2FA methods

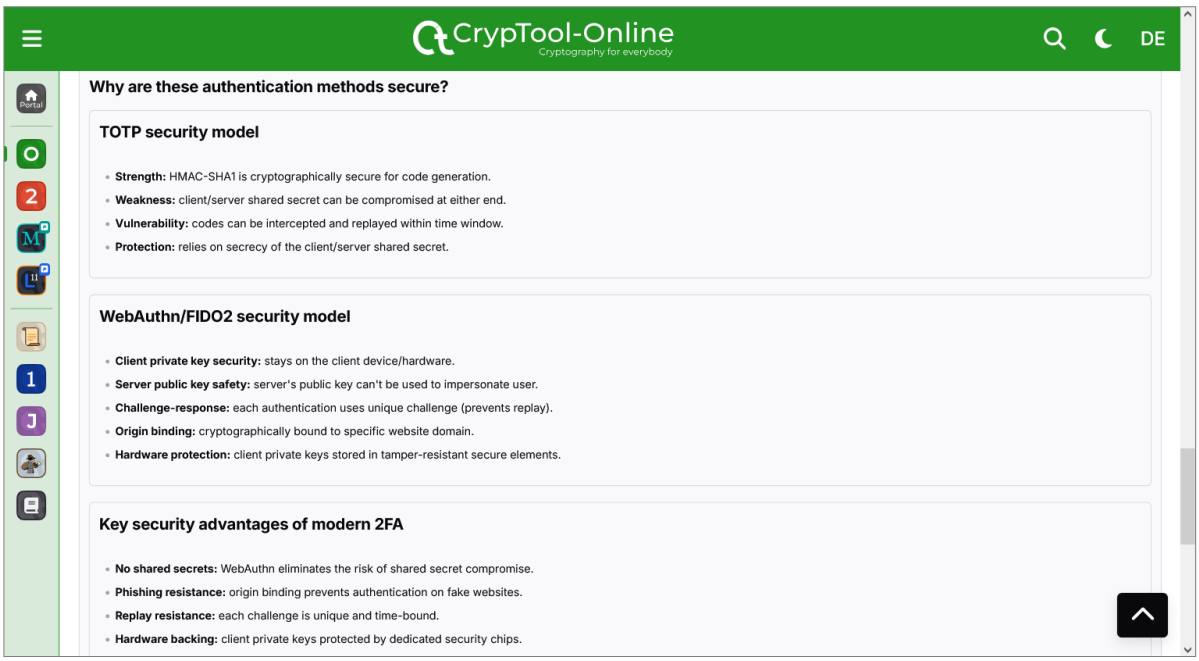

Figure 36: Why TOTP and passkeys are secure: overview of their underlying security models

4.4 Integration into CrypTool-Online

The web-based application can be accessed at `https://cryptool-org-gamma.vercel.app/en/cto/2fa/` and will be integrated as a plugin into the CTO website. CTO is built around a plugin architecture where each application resides in its own folder under `/ctoapps`. The following steps were followed during the integration phase:

1. **Repository fork and setup**: A fork of the CTO repository was created to develop the plugin locally. Development was carried out in Visual Studio Code[30] using `Next.js`, `React`, `Webpack`, `Markdown-X` and `Chakra-UI`.

2. **Plugin folder and configuration**: A new folder `/ctoapps/2fa` was created containing all `React` components and readme files, see Fig. 37 on the following page. The `ctoapps.json` file was updated to include the plugin's metadata, see Fig. 38 on page 79. This file controls how the plugin appears on the CTO homepage.

3. **Integration and testing**: All programming was carried out using Visual Studio Code (see Fig. 39 on page 80). The application was tested locally and submitted to the main repository through a pull request.

[30] `https://code.visualstudio.com/`

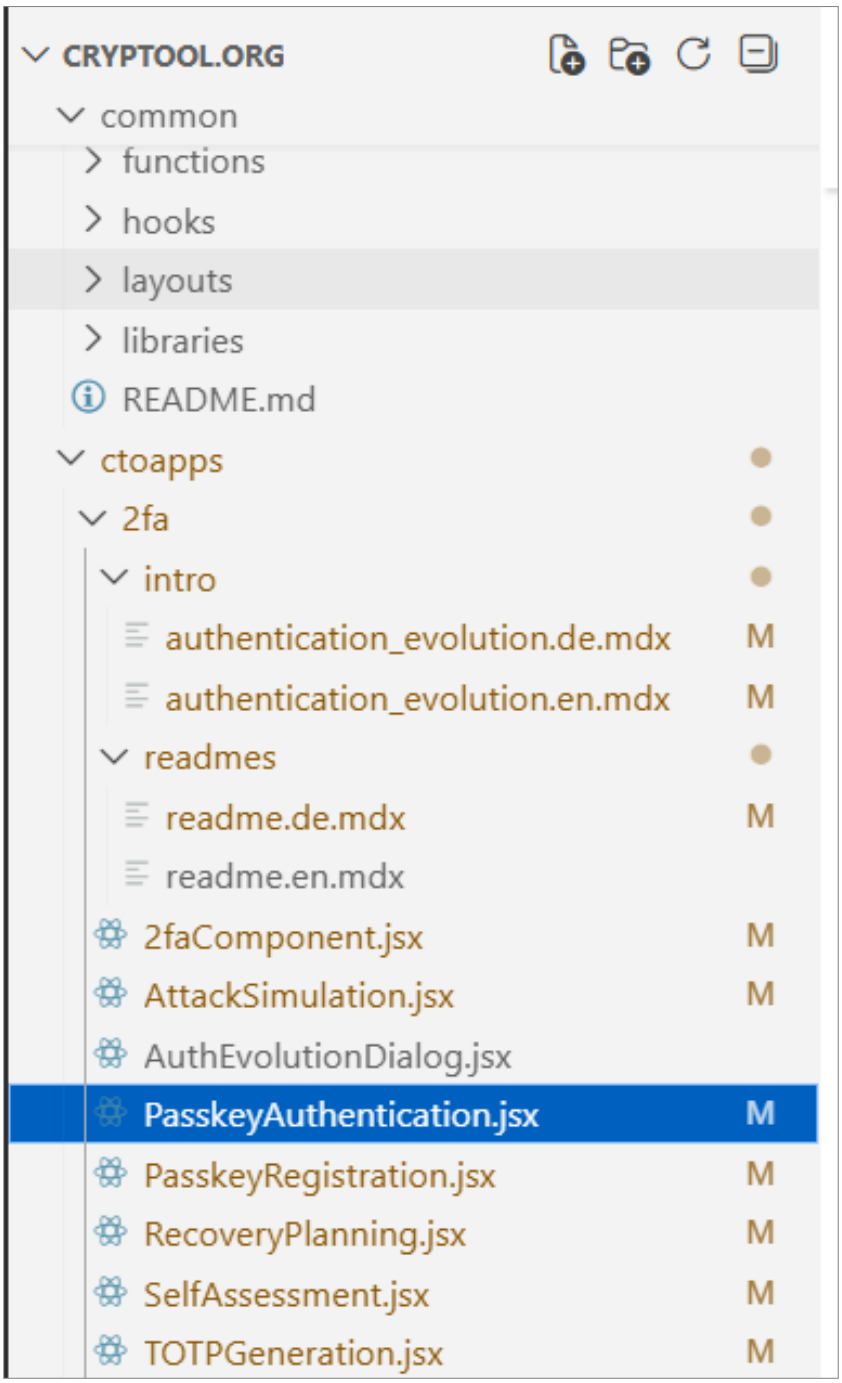

Figure 37: Project structure

Figure 38: Ctoapps.json file

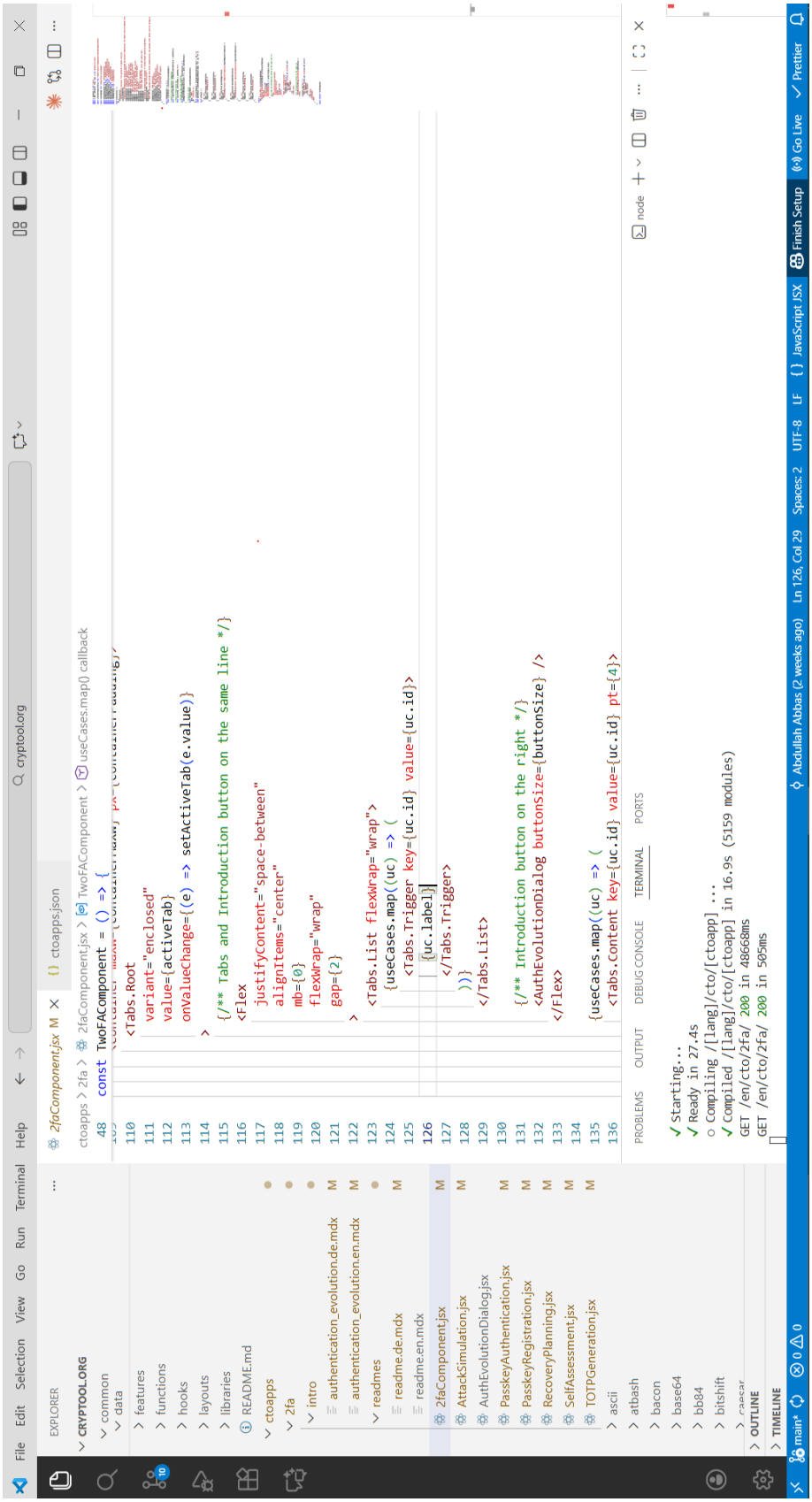

Figure 39: Visual Studio Code editor displaying the main 2FA React component

5 Evaluation and results

This chapter presents the results of the user study conducted to evaluate the usability, clarity, and educational effectiveness of the 2FA web-based demonstrator. The evaluation was carried out using an online questionnaire created with Google Forms. A total of 14 participants completed the survey after interacting with the demonstrator.

5.1 Methodology

Participants were asked to explore the full functionality of the demonstrator and familiarize themselves with its different components before completing the survey. Afterwards, participants completed a questionnaire that measured usability, conceptual understanding, and their overall impressions. All questions were answered on a five-point Likert scale[31], where 1 represented strong disagreement and 5 represented strong agreement.

The original Google Form used in this study was accessible at:

https://docs.google.com/forms/d/e/
1FAIpQLSdFpeaLH9snJ930CRdDgfpgg4UEqMdBIcVs66bH1zzsdX-_Ig/viewform

The form is now closed and can no longer be viewed in full. However, the complete analytical overview, which contains all charts and response distributions generated by Google Forms, remains publicly accessible and can be viewed at:

https://docs.google.com/forms/d/e/
1FAIpQLSdFpeaLH9snJ930CRdDgfpgg4UEqMdBIcVs66bH1zzsdX-_Ig/viewanalytics

A summary of the most relevant results is presented in the following section, while the charts remain available online for further examination.

5.2 Results

The participant group consisted of individuals with different levels of experience with 2FA. Some participants reported using 2FA regularly, others only occasionally and few participants had no prior experience at all. This variation ensured that both experienced

[31] https://en.wikipedia.org/wiki/Likert_scale

and inexperienced users were represented in the evaluation. The group also included participants from technical backgrounds such as computer science and IT as well as individuals from engineering and several non-technical fields. This diversity allowed the demonstrator to be assessed from multiple perspectives and provided a balanced view of its accessibility and clarity.

Across all usability-related questions, participants consistently reported positive experiences. The demonstrator was described as easy to use, with 50% selecting the highest rating and 35.7% selecting the second highest rating. Navigation between modules was rated as intuitive by 85.7% of participants, and the design and layout were considered clear and consistent with equally positive ratings. In addition, 78.6% participants (those who selected ratings 4 or 5 in Table 6) stated that they would recommend the web-based application to others who are learning about authentication.

Participants also reported clear learning gains. A majority indicated a better understanding of how TOTP works as a second factor in 2FA systems, increased awareness of the impact of time drift on authentication and the ability to explain how passkeys differ from TOTP codes. These results suggest that the visual and interactive explanations in the demonstrator successfully clarified the underlying authentication concepts.

A summary of all Likert scale responses is provided in Table 6. The values under each rating (from 1 to 5) add up to the total number of participants (14) for each question. The average column shows the weighted mean response for each question. For each rating i (from 1 to 5), let n_i be the number of participants who selected that rating. The average is then calculated as: $(1 \times n_1 + 2 \times n_2 + 3 \times n_3 + 4 \times n_4 + 5 \times n_5)/14$. The percentage column indicates the percentage of participants who selected ratings 4 or 5.

Question	1	2	3	4	5	Average	Percentage
Application easy to use	0	0	2	5	7	4.36	85.7%
Navigation intuitive	0	0	2	5	7	4.36	85.7%
Design clear	0	0	2	5	7	4.36	85.7%
Confident using tool	0	0	3	4	7	4.29	78.6%
Recommend tool	0	0	3	4	7	4.29	78.6%
Understand TOTP	0	0	3	5	6	4.21	78.6%
Understand time drift	0	1	3	4	6	4.07	71.4%
Explain passkeys vs codes	0	0	3	4	7	4.29	78.6%

Table 6: Summary of Likert scale responses from the evaluation survey

5.3 Summary of findings

The evaluation results show that the demonstrator achieved its educational objectives. Users found the demonstrator intuitive, well structured and informative. The mixture of technical and non-technical participants confirms that the demonstrator is accessible to users with different backgrounds. Reported learning improvements, combined with the consistently positive usability ratings, suggest that the interactive components such as the TOTP visualization and the simulations of the passkey registration and authentication processes effectively supported users in understanding modern authentication mechanisms.

The availability of the complete response statistics online further strengthens the transparency of the evaluation and allows readers to review the original response distributions alongside the summarized results presented in this chapter.

6 Conclusion

This chapter summarizes how the objectives of this thesis were achieved and provides an outlook on potential future enhancements to the web-based demonstrator.

6.1 Achievement of goals

The web-based demonstrator developed in this thesis successfully fulfilled the functional and non-functional requirements described in Chapter 4.1. The necessary features were defined, including TOTP generation and verification, passkey registration and authentication, attack simulations, recovery planning, and self-assessment and summary. Chapter 4.3 explained the implementation of these features, showing how the modular design of the application supports both user interaction and cryptographic operations.

All the goals of this thesis described earlier in Chapter 1.3 were successfully met. The web-based demonstrator presented the theoretical foundations of TOTP-based 2FA and passkey-based authentication in a clear and comprehensible manner, linked these concepts to practical examples, and provided users an interactive experience, helping to bridge the gap between knowledge and behavior. Through its modular design, the application helped users understand both the underlying cryptographic operations and their use in real authentication processes. Regular meetings during development allowed early feedback to improve the user interface and incorporate new ideas and suggestions into the application.

The implementation itself proved to be stable and flexible. The modular design of the implementation allows flexible and fast handling of all features and also allows easy adjustments for future requirements. The browser-only execution model ensured secure operation without external dependencies. This established a reliable basis for continued use and development within CTO.

Additionally, the evaluation presented in Chapter 5 showed that the application effectively supports learning about modern 2FA methods. As described in Chapter 5.2, the usability results indicated that the chosen design was effective. Participants rated navigation, structure, and clarity very highly, and most found the application straightforward to use. Overall, the results showed that the demonstrator met its objectives as it provides a solid foundation for educating users about secure 2FA methods.

6.2 Outlook

There are several smaller enhancements that could improve the existing functionality of the application. These include better visualization of the cryptographic operations and steps for TOTP and passkeys. In addition to these smaller improvements, more advanced extensions could further increase the educational value of the application. For example, integrating real WebAuthn signatures into the passkey modules would allow users to interact with actual hardware tokens or platform authenticators, providing a more realistic experience. An optional server-side component could also enhance challenge validation while keeping the application compatible with the local-first approach.

Another potential extension is the inclusion of additional attack simulations. These could show manipulated QR codes during TOTP setup, hardware token vulnerabilities, or more detailed phishing scenarios. Combined with the existing interactive features, these additions would help users understand attacks and protection mechanisms even more effectively.

Future evaluations could rely on task-based assessments rather than solely on subjective ratings. Asking participants to complete specific activities would allow for a more precise measurement of learning outcomes. Broader studies with larger and more diverse participant groups would strengthen the evidence for the demonstrator's effectiveness.

In summary, the application offers a strong basis for explaining modern 2FA methods in a simple and understandable way. With further technical and educational improvements, it can continue to help many users understand and use secure authentication methods.

Bibliography

[1] JumpCloud. *50+ Password Statistics & Trends to Know in 2024*. JumpCloud Blog. 2024. URL: https://jumpcloud.com/blog/password-statistics-trends (visited on 07/01/2025).

[2] Chun Wang, Steve T.K. Jan, Hang Hu, Douglas Bossart, and Gang Wang. "The Next Domino to Fall: Empirical Analysis of User Passwords across Online Services". In: *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. CODASPY '18. Tempe, AZ, USA: Association for Computing Machinery, 2018, pp. 196–203. ISBN: 9781450356329. DOI: 10.1145/3176258.3176332. URL: https://doi.org/10.1145/3176258.3176332.

[3] Naveen Kumar. *35 Password Statistics 2025 - Data Breaches & Industry Report*. DemandSage. 2025. URL: https://www.demandsage.com/password-statistics/ (visited on 07/01/2025).

[4] Thanasis Petsas, Giorgos Tsirantonakis, Elias Athanasopoulos, and Sotiris Ioannidis. "Two-Factor Authentication: Is The World Ready? Quantifying 2FA Adoption". In: *Proceedings of the eighth european workshop on system security*. New York, NY, USA: Association for Computing Machinery, 2015. ISBN: 9781450334792. URL: https://doi.org/10.1145/2751323.2751327.

[5] Jon Oberheide. *Estimating Google's Two-Factor (2SV) Adoption with Pen, Paper, and Poor Math*. Jon Oberheide's Blog. 2015. URL: https://jon.oberheide.org/blog/2015/05/15/estimating-googles-two-factor-2sv-adoption/ (visited on 07/15/2025).

[6] Laura Paine and Hirsch Singhal. *Raising The Bar for Software Security: GitHub 2FA Begins March 13*. GitHub Blog. URL: https://github.blog/news-insights/product-news/raising-the-bar-for-software-security-github-2fa-begins-march-13/ (visited on 12/10/2025).

[7] Bundesamt für Sicherheit in der Informationstechnik. *Befragung zur Cybersicherheit 2025*. June 2025. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Digitalbarometer/CyMon-ProPK-BSI_2025_Kurzbericht.pdf (visited on 12/10/2025).

[8] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Greg Norcie. "A Comparative Usability Study of Two-Factor Authentication". In: *arXiv e-prints* (2013), arXiv–1309. URL: https://doi.org/10.48550/arXiv.1309.5344.

[9] Shivam Pandey, Tewodros Taffese, Michelle Huang, and Michael D Byrne. "Human Performance in Google's Two-factor Authentication Setup Process". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 63. 1. SAGE Publications Sage CA: Los Angeles, CA. 2019, pp. 2221–2225. URL: https://doi.org/10.1177/1071181319631348.

[10] Ken Reese et al. “A Usability Study of Five Two-Factor Authentication Methods”. In: *Fifteenth symposium on usable privacy and security (SOUPS 2019)*. 2019, pp. 357–370. URL: https://www.usenix.org/system/files/soups2019-reese.pdf.

[11] Google. *Turn on 2-Step Verification*. Google Account Help. URL: https://support.google.com/accounts/answer/185839 (visited on 07/15/2025).

[12] Apple Inc. *Two-Factor Authentication for Apple Account*. Apple Support. 2025. URL: https://support.apple.com/en-us/102660 (visited on 07/16/2025).

[13] Paul Grassi et al. *Digital Identity Guidelines: Authentication and Lifecycle Management*. 2017. DOI: https://doi.org/10.6028/NIST.SP.800-63b.

[14] Federal Financial Institutions Examination Council. *Authentication and Access to Financial Institution Services and Systems*. Tech. rep. 2021. URL: https://www.ffiec.gov/sites/default/files/media/press-releases/2021/authentication-and-access-to-financial-institution-services-and-systems.pdf (visited on 08/01/2025).

[15] Claudia Ziegler Acemyan, Philip Kortum, Jeffrey Xiong, and Dan S Wallach. “2FA Might Be Secure, But It’s Not Usable: A Summative Usability Assessment of Google’s Two-factor Authentication (2FA) Methods”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 62. 1. SAGE Publications Sage CA: Los Angeles, CA. 2018, pp. 1141–1145. URL: https://doi.org/10.1177/1541931218621262.

[16] Julia Prümmer, Tommy van Steen, and Bibi van den Berg. “A Systematic Review of Current Cybersecurity Training Methods”. In: *Computers & Security* 136 (2024), p. 103585. URL: https://www.sciencedirect.com/science/article/pii/S0167404823004959.

[17] David M’Raihi, Frank Hoornaert, David Naccache, Mihir Bellare, and Ohad Ranen. *HOTP: An HMAC-Based One-Time Password Algorithm*. RFC 4226. Dec. 2005. DOI: 10.17487/RFC4226. URL: https://www.rfc-editor.org/info/rfc4226.

[18] David M’Raihi, Johan Rydell, Mingliang Pei, and Salah Machani. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. May 2011. DOI: 10.17487/RFC6238. URL: https://www.rfc-editor.org/info/rfc6238.

[19] World Wide Web Consortium (W3C). *Web Authentication: An API for accessing Public Key Credentials - Level 2*. W3C Recommendation. Apr. 2021. URL: https://www.w3.org/TR/webauthn-2/ (visited on 09/15/2025).

[20] Bundesamt für Sicherheit in der Informationstechnik. *Technical Guideline TR-03107: Electronic Identities and Trust Services in E-Government*. Technical Guideline. BSI. URL: https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03107/TR-03107_node.html (visited on 08/01/2025).

[21] Federal Ministry of the Interior (Germany). *Approval of Private Providers of Identification and Authentication Solutions for Interoperable User Accounts*. URL: https://www.personalausweisportal.de/Webs/PA/EN/business/approval-procedure-for-user-accounts/approval-procedure-for-user-accounts-node.html (visited on 08/01/2025).

[22] Wikipedia. *Password*. URL: https://en.wikipedia.org/wiki/Password (visited on 07/20/2025).

[23] Fusionauth. *The first password to be hacked*. URL: https://fusionauth.io/blog/password-history (visited on 07/21/2025).

[24] Wikipedia. *RSA SecurID*. URL: https://en.wikipedia.org/wiki/RSA_SecurID (visited on 08/10/2025).

[25] Wikipedia. *Personal identification number*. URL: https://en.wikipedia.org/wiki/Personal_identification_number (visited on 08/13/2025).

[26] Lawrence Livermore National Laboratory. *RSA Token Frequently Asked Questions (FAQs)*. URL: https://access.llnl.gov/otp/cgi-bin/faq.cgi (visited on 08/10/2025).

[27] CDW. *RSA SecurID SID700 - Hardware Token*. URL: https://www.cdw.com/product/rsa-securid-700-3-year-10-pack/1035335 (visited on 08/10/2025).

[28] Wikipedia. *Google Authenticator*. URL: https://en.wikipedia.org/wiki/Google_Authenticator (visited on 08/12/2025).

[29] StrongDM. *The Definitive Guide to FIDO2 Web Authentication*. URL: https://www.strongdm.com/blog/fido2 (visited on 08/13/2025).

[30] Beyond Identity. *FIDO2 vs. WebAuthn: What's the Difference?* URL: https://www.beyondidentity.com/resource/fido2-vs-webauthn-whats-the-difference (visited on 08/13/2025).

[31] Wikipedia. *YubiKey*. URL: https://en.wikipedia.org/wiki/YubiKey (visited on 08/13/2025).

[32] Aleksandr Ometov et al. "Multi-Factor Authentication: A Survey". In: *Cryptography* 2.1 (2018). DOI: 10.3390/cryptography2010001. URL: https://www.mdpi.com/2410-387X/2/1/1.

[33] Harini Narasimhan and T. Padmanabhan. "2CAuth: A New Two Factor Authentication Scheme Using QR-Code". In: *International Journal of Engineering and Technology* 5 (Apr. 2013), pp. 1087–1094. URL: https://www.researchgate.net/publication/286870446_2CAuth_A_New_Two_Factor_Authentication_Scheme_Using_QR-Code.

[34] Laura Vegh. "Cyber-Physical Systems Security through Multi-Factor Authentication and Data Analytics". In: *2018 IEEE international conference on industrial technology (ICIT)*. 2018, pp. 1369–1374. DOI: 10.1109/ICIT.2018.8352379. URL: https://ieeexplore.ieee.org/document/8352379.

[35] Wikipedia. *SHA-1*. URL: https://en.wikipedia.org/wiki/SHA-1 (visited on 08/15/2025).

[36] Google. *Key Uri Format*. Google Authenticator Wiki. URL: https://github.com/google/google-authenticator/wiki/Key-Uri-Format (visited on 08/15/2025).

[37] Donald E. Eastlake 3rd, Steve Crocker, and Jeffrey I. Schiller. *Randomness Requirements for Security*. RFC 4086. June 2005. DOI: 10.17487/RFC4086. URL: https://www.rfc-editor.org/info/rfc4086.

[38] Wikipedia. *Network Time Protocol*. URL: https://en.wikipedia.org/wiki/Network_Time_Protocol (visited on 08/17/2025).

[39] FIDO Alliance. *FIDO U2F Overview*. Technical Specification. 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html (visited on 08/25/2025).

[40] Heise Online. *Funktionsweise von Passkeys im Detail erklärt*. URL: https://www.heise.de/-9659204 (visited on 11/05/2025).

[41] Simon Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. Oct. 2006. DOI: 10.17487/RFC4648. URL: https://www.rfc-editor.org/info/rfc4648.

[42] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[43] Sharon Boeyen et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. May 2008. DOI: 10.17487/RFC5280. URL: https://www.rfc-editor.org/info/rfc5280.

[44] National Institute of Standards and Technology (NIST). *Secure Hash Standard (SHS)*. FIPS PUB 180-4. 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (visited on 08/21/2025).

[45] Yubico. *What is FIDO U2F?* URL: https://www.yubico.com/authentication-standards/fido-u2f-standard/ (visited on 08/30/2025).

[46] Microsoft Corporation. *Windows Hello for Business Overview*. URL: https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/ (visited on 09/03/2025).

[47] Apple Inc. *Apple Platform Security*. URL: https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web (visited on 09/03/2025).

[48] Google for Developers. *Android Keystore System*. URL: https://developer.android.com/privacy-and-security/keystore (visited on 09/05/2025).

[49] Wikipedia. *End-to-end encryption*. URL: https://en.wikipedia.org/wiki/End-to-end_encryption (visited on 09/10/2025).

[50] Apple Inc. *About the security of passkeys*. URL: https://support.apple.com/en-us/102195 (visited on 09/03/2025).

[51] Google Security Blog. *Security of Passkeys in the Google Password Manager*. URL: https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html (visited on 09/05/2025).

[52] World Wide Web Consortium (W3C). *Web Authentication: An API for accessing Public Key Credentials - Level 1*. W3C Recommendation. Mar. 2019. URL: https://www.w3.org/TR/webauthn-1/ (visited on 08/27/2025).

[53] ET Edge Insights. *Combatting Brute Force Attacks in 2024: Attacks persist despite cyber security advancements*. URL: https://etedge-insights.com/technology/cyber-security/combatting-brute-force-attacks-in-2024/ (visited on 09/13/2025).

[54] Alexandra Jonker and Tom Krantz. *What Is a Brute Force Attack?* IBM Think. URL: https://www.ibm.com/think/topics/brute-force-attack (visited on 12/10/2025).

[55] Huntress. *What is a Brute Force Attack? A Guide for IT Security Professionals*. URL: https://www.huntress.com/cybersecurity-education/cybersecurity-101/topic/what-is-a-brute-force-attack (visited on 09/13/2025).

[56] Triaxiom Security. *What's the Difference Between Offline and Online Password Attacks?* URL: https://www.triaxiomsecurity.com/whats-the-difference-between-offline-and-online-password-attacks/ (visited on 12/10/2025).

[57] Wikipedia. *Brute-force attack*. URL: https://en.wikipedia.org/wiki/Brute-force_attack (visited on 09/13/2025).

[58] Security Magazine. *There Was a 12% Increase in Brute Force Cyberattack Techniques in 2024*. 2024. URL: https://www.securitymagazine.com/articles/101081-there-was-a-12-increase-in-brute-force-cyberattack-techniques-in-2024 (visited on 09/15/2025).

[59] OneLogin. *What Type of Attacks Does Multi-Factor Authentication Prevent?* URL: https://www.onelogin.com/learn/mfa-types-of-cyber-attacks (visited on 10/01/2025).

[60] Eftsure. *Two-Factor Authentication Statistics: First Line of Defence*. URL: https://www.eftsure.com/statistics/two-factor-authentication-statistics/ (visited on 09/15/2025).

[61] *One Simple Action You Can Take to Prevent 99.9 Percent of Attacks on Your Accounts*. Microsoft Security Blog. 2019. URL: https://www.microsoft.com/en-us/security/blog/2019/08/20/one-simple-action-you-can-take-to-prevent-99-9-percent-of-account-attacks/ (visited on 09/15/2025).

[62] Brenda Buckman. *What Is a Dictionary Attack?* Huntress. 2025. URL: https://www.huntress.com/cybersecurity-101/topic/what-is-dictionary-attack (visited on 10/01/2025).

[63] Mariusz Michalowski. *70+ Password Statistics for 2025*. Spacelift. 2025. URL: https://spacelift.io/blog/password-statistics (visited on 10/01/2025).

[64] SubRosa. *Real-World Examples of Dictionary Attacks and Their Impacts*. URL: https://www.subrosacyber.com/en/blog/real-world-dictionary-attack-examples (visited on 10/01/2025).

[65] Lukas Reiter. *2-Factor-Phishing – the 'Man-in-the-Middle' Attack*. InfoGuard. 2021. URL: https://www.infoguard.ch/en/blog/2-factor-phishing-the-man-in-the-middle-attack (visited on 12/04/2025).

[66] Garrett Guinivan Laura Hamel and Chris Dawson. *Unmasking Tycoon 2FA: A Stealthy Phishing Kit Used to Bypass Microsoft 365 and Google MFA*. Proofpoint. May 2024. URL: https://www.proofpoint.com/us/blog/email-and-cloud-threats/tycoon-2fa-phishing-kit-mfa-bypass (visited on 12/04/2025).

[67] Dana Vioreanu. *MiTM Phishing Attacks that Bypass 2FA Are on The Rise*. CyberGhost. URL: https://www.cyberghostvpn.com/privacyhub/mitm-phishing-attacks-bypass-2fa/ (visited on 12/04/2025).

[68] *Is a Hardware Based 2FA More Resistant to Phishing than SMS or TOTP?* URL: https://security.stackexchange.com/questions/261349/is-a-hardware-based-2fa-more-resistant-to-phishing-than-sms-or-totp (visited on 12/05/2025).

[69] Dirk Knop. *Yubikey-Cloning-Angriff: Offenbar möglich, aber nicht trivial*. Heise Online. Sept. 2024. URL: https://www.heise.de/news/Yubikey-Cloning-Angriff-Offenbar-moeglich-aber-nicht-trivial-9856972.html (visited on 11/30/2025).

[70] Thomas Roche. *EUCLEAK: Side-Channel Attack on the YubiKey 5 Series*. Tech. rep. NinjaLab, Sept. 2024. URL: https://ninjalab.io/eucleak/ (visited on 11/30/2025).

[71] Yubico. *Security Advisory YSA-2024-03*. 2024. URL: https://www.yubico.com/support/security-advisories/ysa-2024-03/ (visited on 11/30/2025).

[72] Nitrokey. *Nitrokeys Offer Investment Security Without Infineon's Security Vulnerability*. 2024. URL: https://www.nitrokey.com/news/2024/nitrokeys-offer-investment-security-without-infineons-security-vulnerability (visited on 11/30/2025).

List of Abbreviations

2FA	two-factor authentication
2SV	2-step verification
AAL	authenticator assurance level
API	application programming interface
ATM	automated teller machine
BSI	Bundesamt für Sicherheit in der Informationstechnik
CTAP2	client to authenticator protocol
CTO	CrypTool-Online
CTSS	compatible time-sharing system
CSPRNG	cryptographically secure pseudo-random number generator
ECDSA	elliptic curve digital signature algorithm
FIDO	fast identity online
FIDO2	fast identity online 2
HMAC	hash-based message authentication code
HOTP	HMAC-based one-time password
HTTPS	hypertext transfer protocol secure
IETF	Internet Engineering Task Force
JS	JavaScript
MIT	Massachusetts Institute of Technology
MFA	multi-factor authentication
NIST	National Institute of Standards and Technology
NTP	network time protocol
OTP	one-time password
OATH	initiative for open authentication
PIN	personal identification number
QR	quick response
SFA	single-factor authentication
SHA	secure hash algorithm
SMS	short message service
TLS	transport layer security
TEE	trusted execution environment
TPM	trusted platform module
TOTP	time-based one-time password
U2F	universal 2nd factor
URI	uniform resource identifier
W3C	World Wide Web Consortium
WebAuthn	web authentication

List of Tables

List of Figures

Acknowledgments

For the realization of this thesis, I would like to begin by expressing my deepest gratitude to my mother for her continuous support throughout my entire learning journey. Her encouragement has been a constant source of strength and motivation.

I would also like to sincerely thank Professor Bernhard Esslinger for giving me the opportunity to work on this topic and for his valuable feedback. He was always available to support me throughout the development of this thesis. Furthermore, I would like to thank Dr. Martin Franz for his guidance and for always being available to answer my questions.

ChatGPT was utilized to refine the clarity and flow of the writing and to assist with translation. The tool was used solely for improving grammar, phrasing, formulation, and translation.

Eigenständigkeitserklärung

Ich versichere, dass ich die schriftliche Ausarbeitung selbständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach (inkl. Übersetzungen) anderen Werken entnommen sind, habe ich in jedem einzelnen Fall unter genauer Angabe der Quelle (einschließlich des World Wide Web sowie generativer KI und anderer elektronischer Datensammlungen) deutlich als Entlehnung kenntlich gemacht. Dies gilt auch für angefügte Zeichnungen, bildliche Darstellungen, Skizzen und dergleichen. Ich nehme zur Kenntnis, dass die nachgewiesene Unterlassung der Herkunftsangabe als versuchte Täuschung gewertet wird.

Ich bestätige, dass die elektronische Version inhaltlich mit der gedruckten Version übereinstimmt.

I certify that I have prepared this written work independently and have not used any aids other than those specified. I have clearly identified all passages that are taken from other works in terms of wording or meaning (including translations) as borrowed material in each individual case, stating the exact source (including the World Wide Web as well as generative AI and other electronic data collections). This also applies to attached drawings, pictorial representations, sketches and the like. I acknowledge that the proven omission of the indication of origin will be regarded as attempted deception.

I confirm that the content of the electronic version is the same as the printed version.

(Datum/Date)	(Unterschrift/Signature)

Inhalt des USB-Sticks

- Masterarbeit als PDF
- Ergebnisse der Evaluation als PDF
- Quellcode der Applikation als ZIP