Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Faculty of Engineering, Department EE & CS
Course of Studies: IT-Security and Forensics

Master's Thesis

for the Attainment of the Academic Degree

Master of Engineering (M. Eng.)

Security Evaluation of Multi-Factor Authentication in Comparison with the Web Authentication API

Submitted by:	September 27, 2019
From:	Tim Brust born 03/31/1995 in Hamburg, Germany
Matriculation number:	246565
First supervisor:	Prof. Dr.-Ing. habil. Andreas Ahrens
Second supervisor:	Prof. Dr. rer. nat. Nils Gruschka

Purpose of this Thesis

The purpose of this master's thesis is an introduction to multi-factor authentication, as well as to the conventional methods of authentication (knowledge, possession, biometrics) including their technical functionality, web usability, and potential security threats and vulnerabilities.

The thesis will investigate whether the Web Authentication API is suitable as an alternative or possible supplement to existing multi-factor authentication methods. The question has to be answered to what extent the Web Authentication API can increase security and user comfort. The evaluation of the security of the Web Authentication API in comparison with other multi-factor authentication solutions plays a crucial role in this thesis.

Abstract

Internet users are at constant risk, given that data breaches happen nearly daily. When a breached password is re-used, it renders their whole digital identity in danger. To counter these threats, the user can deploy additional security measures, e.g., the multi-factor authentication. This master's thesis introduces and compares the multi-factor authentication solutions one-time passwords, smartcards, security keys, and the Universal Second Factor protocol with a focus on their security. Further, the Web Authentication API is explained and compared with the other multi-factor authentication solutions. The outcome of this thesis is that multi-factor authentication is subject to phishing attacks. It can be made phishing resistant, but it requires a change of the transportation medium or the usage of other authentication methods. Also, the Web Authentication API has the potential to replace passwords. However, it is not yet usable enough for the end consumer.

Keywords— authentication, multi-factor authentication, mfa, two-factor authentication, 2fa, fido, web authentication api, webauth, webauthn, web-authentication

Kurzfassung

Internetnutzer sind einem ständigen Risiko ausgesetzt, da Sicherheitsbrüche fast täglich auftreten. Wenn ein gehacktes Passwort wiederverwendet wird, stellt dies eine Gefahr für die gesamte digitale Identität des Nutzers dar. Um diesen Bedrohungen entgegenzuwirken, kann der Benutzer zusätzliche Sicherheitsmaßnahmen, z.B. die Multi-Faktor Authentifizierung, einsetzen. Diese Masterarbeit stellt die Multi-Faktor Authentifizierungen Einmalpasswörter, Smartcards, Sicherheitsschlüssel und das Universal Second Factor Protokoll mit Fokus auf deren Sicherheit vor und vergleicht diese. Weiterhin wird die Web Authentication API erläutert und mit den anderen Multi-Faktor-Authentifizierungen verglichen. Das Ergebnis dieser Arbeit ist, dass die Multi-Faktor Authentifizierung trotzdem Phishing-Angriffen ausgesetzt ist. Es erfordert andere Transportmechanismen oder Verfahren, um Resistenz gegenüber Phishing zu erreichen. Darüber hinaus hat die Web Authentication API das Potenzial, Passwörter zu ersetzen, ist aber für den Endverbraucher noch nicht ausreichend nutzbar.

Schlüsselwörter— authentifizierung, multi-faktor authentifizierung, mfa, zweifaktor authentifizierung, 2fa, fido, web authentication api, webauth, webauthn, web-authentication

Acknowledgments

Foremost, I would like to thank my supervisor Prof. Dr.-Ing. habil. Andreas Ahrens for the continued support throughout this thesis. The fast and valuable feedback and of course, the possibility to choose a topic of my choice, as well as the taught basics required for writing this thesis helped me along the way. Also, I have to express my thank-you to Prof. Dr. rer. nat. Nils Gruschka, especially for the practice in writing academic works that helped a lot while writing this thesis.

Moreover, I thank SinnerSchrader for backing me financially in such a way that I was able to focus my work on this thesis without worries. A personal thank-you is expressed to Ansgar Grapentin for keeping me motivated during this time and for proofreading this thesis.

Further, I would like to acknowledge the valuable feedback from my former co-worker Dr. Ingo Bente.

Additionally, I want to thank my fellow students, Gregor, Jasmina, and Jens for always answering my questions about the regularities en route to writing this thesis and for the comprehensive discussions in our study group.

Besides that, I would like to express my sincere gratitude to Caro for the countless hours of proofreading, providing useful feedback, and of course the continuous support, motivation, and understanding during this time.

Last but not least, I thank my parents for always supporting me and my decisions.

This accomplishment would not have been possible without you all – thank you.

Contents

1 Introduction

1.1 Problem Statement and Motivation

»Usernames and passwords are an idea that came out of 1970s mainframe architectures. They were not built for 2016.«[1]

Alex Stamos

Passwords in the way they are currently used, are not suited for the twenty-first-century, as Alex Stamos, the former Chief Security Officer (CSO) of Facebook and Yahoo!, stated. The secure handling of passwords is a problem for many users. Passwords are re-used between different websites and often shared across private and work environments. This renders the (private) user data, but also business secrets at high risk. If confidential business data is leaked or obtained by a competitor, it may have severe consequences for the respective company. These consequences have the potential to force the company into shutdown, such as the bitcoin marketplace Mt. Gox after a hack that resulted in bitcoin loss.[2]

To make things worse, very few people are using multi-factor authentication (MFA) and even fewer password managers in 2019. The majority of the users are either remembering their passwords or writing them down on a piece of paper – in cleartext.[3]

At the same time, the recorded amount of cybercrime cases is still increasing, and, for example, phishing remains a constant threat. While MFA can protect against threats such as brute force attacks or stolen credentials, some MFA solutions are still affected and vulnerable to phishing attacks. Besides that, short message service (SMS) traffic is not considered secure anymore, yet a lot of MFA solutions use

[1] See Col16.
[2] See Ros18, p. 43.
[3] See Kes18; See Fri19.

it. Nevertheless, the majority of the users are not using MFA at all, even if weak solutions can protect against automated attacks.[4]

To counter these negative trends, new application programming interfaces (APIs) are emerging, for example, the Web Authentication API. It is a standardized API supported in major web browsers such as Chrome, Firefox, or Edge. The Web Authentication API allows a secure registration, login, and two-factor authentication (2FA) – all without the generation, storage, and remembering of passwords by utilizing asymmetric cryptography. The private keys are stored, e.g., on external devices such as Universal Serial Bus (USB) sticks, but can be stored on built-in hardware, too. These are, for example, protected by a fingerprint sensor or dedicated chip designed for secure operations.

1.2 Goals of this Thesis

The goals of this thesis are an introduction to MFA and the different authentication factors such as »knowledge, possession, and biometrics«. This introduction includes the technical functionality, usability in web projects and web browsers, and their security threats alongside an introduction to the Web Authentication API. Those methods of authentication need to be mapped to actual implementations of authentication such as passwords, security keys, and fingerprint sensors, that need to be evaluated security-wise, too.

The Web Authentication API and its origin are being illustrated and technically in more depth explained. In this connection, the question has to be answered if the Web Authentication API can increase security, user comfort, and usability. In this regard, the potential security threats or vulnerabilities that Web Authentication API faces are discussed as well.

Finally, the thesis should answer the question if the Web Authentication API is ready to be used yet and whether it can replace passwords and existing MFA solutions or if it can be used in conjunction with passwords. Besides that, questions are taken into account and answered, such as:

- What are the risks of not using MFA?
- Why are weak passwords and password re-usage such a big issue?
- Is there protection against the weakest link, often being humans?

[4] See dim19; See Bun18, pp. 6–7; See Dot19, p. 58; See Doe+19, p. 2.

- If a user employs MFA, are there any threats, too?
- Are the architecture and algorithms of the used MFA solutions secure enough for usage in web projects and insecure connections?
- Is the Web Authentication API suitable, easy to use, and understandable for end-users?

1.3 Target Audience

The target audience of this thesis are technically experienced readers that have a good understanding and interest in data security and privacy. Additionally, the reader should have a basic knowledge about the functionality and mathematics behind algorithms such as Rivest–Shamir–Adleman (RSA), elliptic-curve cryptography (ECC), or symmetric and asymmetric key exchange (e.g., Diffie–Hellman key exchange). Moreover, the reader needs to be familiar with the underlying concept(s) and techniques of MFA.

Furthermore, the thesis is tailored to interested (web) developers. On the one hand, it shall introduce a new standardized Web API to them in detail. On the other hand, the thesis helps to understand the pros and cons of alternative registration, login, and MFA solutions using asymmetric cryptography, and whether the Web Authentication API suits their needs.

1.4 Delimitation of this Thesis

Existing formally verified and proven algorithms and concepts, as long as not required for the understanding of this thesis, are not explained in detail. It is not the goal of this thesis to perform complete cryptanalysis of existing MFA solutions, nor of the Web Authentication API. Instead, the thesis takes other factors, such as usability for the user, technical feasibility, and web browser support into account. Different, but adjacent, technologies such as OAuth (2.0), OpenID Connect or single sign-on (SSO) neither are a focus of this thesis. Additionally, the topic of authorization is not taken into account and is not of concern for this thesis.

1.5 Approach and Methodology

Initially, in Chapter 2, the reader is introduced into the basics of authentication. After that, in the following chapters, the areas single-factor authentication and MFA are explained. For example, their technical functionality is described, followed by an analysis regarding their security and protection against attacks such as phishing or man in the middle (MITM) attacks.

Hereupon, the Web Authentication API is introduced in Chapter 6 and described in detail. Technical functionality is a crucial aspect of this chapter. Additionally, it is explained against which attacks the Web Authentication API can offer protection. However, it is also asserted which security threats exist, too. As various proof of concepts (PoCs), libraries and full implementations in different programming languages exist, where suitable only example source code listings are used to highlight these analyses.

In Section 6.5, the Web Authentication API is compared with existing MFA solutions. Therefore, it is reviewed if the Web Authentication API can be used in conjunction or as a replacement for MFA.

Concluding, follows an evaluation based on the gained insights from the previous chapters with a summing-up, a conclusion, and outlook for further research and studies.

2 Basics of Authentication

2.1 Methods of Authentication

There are multiple different methods or forms, respectively, that can be used to authenticate a user against someone or something. Traditionally only knowledge, possession, and trait are considered the different forms of authentication. However, other sources also introduce or take new kinds into account, such as location- or time-based authentication. Therefore, this thesis accounts for them, too, and describes the forms in the following sections briefly. A detailed analysis of the security, especially potential threats and vulnerabilities, follows in Chapter 3.[5]

2.1.1 Knowledge

The most used method of authentication is knowledge, i.e., »something the user knows«. Commonly used in information technology (IT) are passwords. Other forms of knowledge are, for example, personal identification numbers (PINs), passphrases, secrets, recovery questions, or one-time passwords (OTPs). The PIN is an example of knowledge, used in, e.g., banking (ATM's, credit cards) or telephony (subscriber identity module (SIM)). The security relies on the fact that the knowledge is considered a secret that only the user knows. When compromised, it is relatively easy to replace the knowledge with a different secret the user knows. Unintentional side effects of replacing a password are that the users may have to replace the used knowledge everywhere in case of re-use.[6]

[5] See TW75, p. 299; See BB17, p. 140; See And08, p. 47; See ZKM12; See DRN17, p. 191.
[6] See Eck14, p. 467.

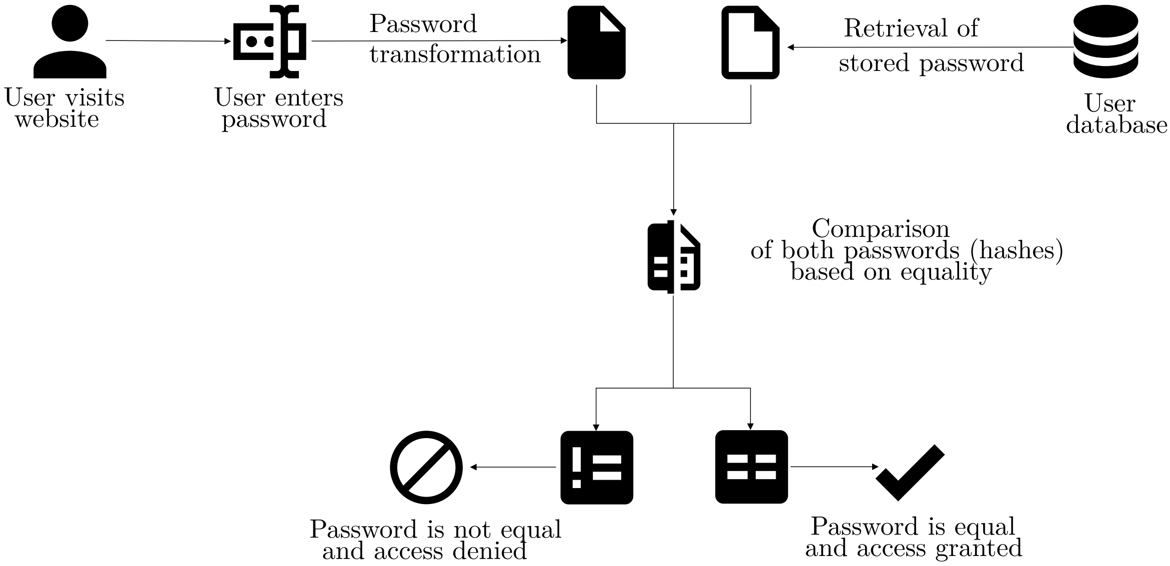

Figure 2.1: Exemplary, but simplified, authentication by knowledge flow[7]

Figure 2.1 shows a simplified authentication by knowledge flow. In this example, the user first visits a website and enters their password in the corresponding form fields. When the user submits the form, the transferred password is often transformed, e.g., hashed and salted. If the database contains an entry for the user, then the stored (hash of) the password is retrieved and compared to the entered one. Only if the hashes are identical, the login succeeds. Otherwise, it fails. The »access denied/cancel« and »checkmark« symbols are chosen since it cannot be verified by whom the authentication is made. It could be a genuine user or an imposter that gained access to the knowledge of the attacked user, in this case, their password.

2.1.2 Possession

Another form of authentication is the possession, i.e., »something the user (physically) has«. The most basic example is a key for a lock. Other forms are, for example, a bank or ID card that can use techniques such as radio-frequency identification (RFID), an onboard chip or magnetic stripes to store the information. In IT, security tokens are often used, which can be hardware (such as a YubiKey, an RSA SecurID or a smartcard) or software (for instance, a smartphone application) token. They can either be disconnected, connected (e.g., via USB or as a smartcard) or contactless (for example via near-field communication (NFC), Bluetooth Low Energy (BLE) or RFID). Sometimes these tokens contain a display that can

[7]Source: diagram by author.

show further information.[8]

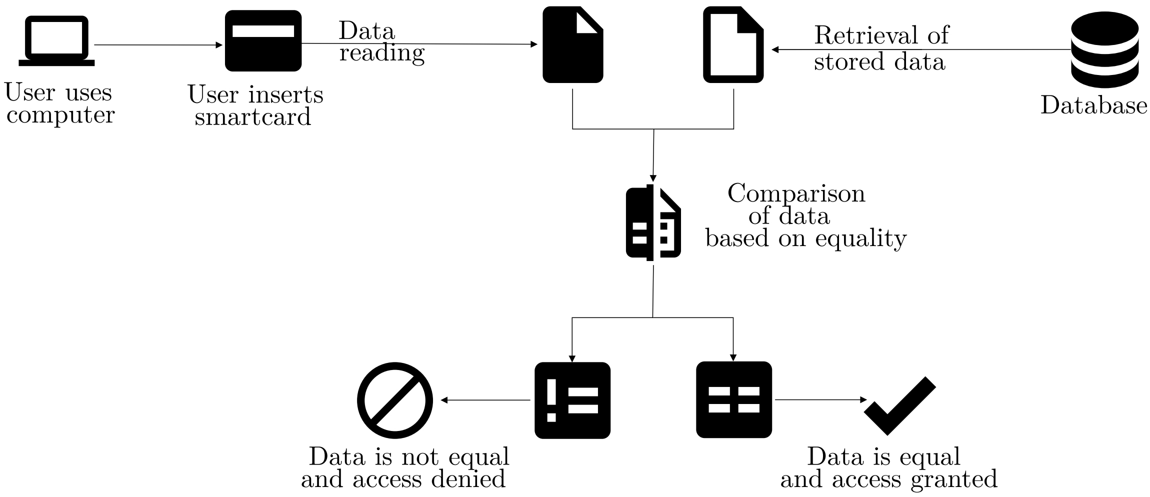

Figure 2.2: Exemplary, but simplified, authentication by possession flow[9]

Figure 2.2 shows an example of an authentication flow with a smartcard. First, the user inserts the given smartcard into their computer. The data is read subsequently. Contemporaneous the application or system reads the stored database entry and compares the data to the one stored on the smartcard. If the data is equal or matches, and the user is authorized, then the authentication succeeds. Again, any user can log on as long as they have the smartcard.

2.1.3 Biometrics

Besides the knowledge and possession factors, another one is biometrics. This factor is classified as »something the user is« and commonly includes the fingerprint, facial, or iris scan. In theory, many other characteristics, e.g., the gait, the ear, DNA, or even the human odor, can be a biometric factor.[10]
These intrinsic factors are sometimes referred to as traits or inherence, too.[11]

While it seems natural to authenticate a person with a biometric factor, it also comes with a couple of challenges. Both, the false rejection rate (FRR), i.e., the system rejects a user even though it is a legitimate user, and the false acceptance rate (FAR), i.e., an imposter is granted access, needs to be accounted for the usage.

[8] See Tod07, p. 24; See DLE19; See May17, pp. 8–11.
[9] Source: diagram by author
[10] See JRN11, pp. 30–34.
[11] See DRN17, p. 186.

Compared to knowledge and possession factors, the enrollment of the biometrics and the continuous update of the sample is more complicated and expensive.[12]

On the other hand, it is more complicated to steal, share, or copy this factor than the others – but it is also nearly impossible to replace compromised biometrics. The usability varies because of the quality of the used biometrics module, the chosen biometrics itself, and the availability of the biometrics.

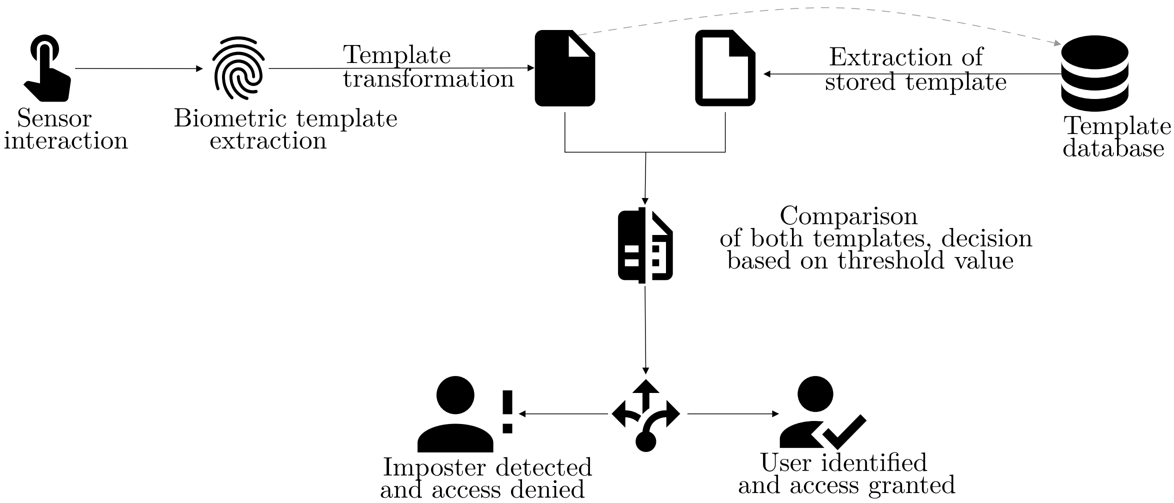

Figure 2.3: Exemplary, but simplified, authentication by biometrics flow[13]

Figure 2.3 shows an exemplary authentication flow using biometrics, in this case with a fingerprint. First, the user interacts with the sensor which reads the fingerprint and extracts the biometric template. Generally, the system or reader transforms the template into a more comparable format. For instance, fingerprints are scanned for minutiae and their direction. Simultaneously, the system retrieves the stored fingerprint or searches for it. The system now compares the stored probe to the fresh one. A threshold value determines how much difference is tolerable. This comparison result finally decides if the authentication attempt can proceed or if it has to be aborted, and access for the user is denied. If the authentication succeeds, the stored template can be updated in the database, as denoted by the dotted grey arrow.

[12]See JRN11, pp. 18–24; See Tod07, pp. 34–37.
[13]Source: diagram by author, based on JRN11, p. 11.

2.1.4 Further Methods of Authentication

While the mentioned authentication forms above are considered a standard in the literature, other forms exist, too. Those include, for example, the current position of the user. The location-based approach grants or denies access based on the current location. The location can either be physical (e.g., via Global Positioning System (GPS)) or digital with, e.g., an IP address.[14]

Another method of authentication is time-based authentication. A typical example is time-limited access to a banking safe, which can only be opened at specific times of the day. A time lock secures it. In IT, this form of authentication helps to protect against, for example, phishing attacks from abroad. The access is granted or denied based on the time and usual time routines where, for instance, a user logs typically on.[15]

Further methods of authentication are, for instance, social authentication, also referred to as »someone the user knows«. For example, Facebook uses this method to ensure that the authentication attempt is genuine by asking the user to identify a set of their friends. Of course, social authentication works in other scenarios, especially offline use cases, too. Besides these methods, »something the user does« is another form of authentication. Examples range from keystrokes to online shopping behavior.[16]

2.2 Processes of Authentication

The process of authentication can be done in three different manners that are explained in the following subsections. These are namely:

1. **active authentication**, where a user has to initiate the process
2. **passive authentication**, where the user does not need to interact with the system
3. **continuous authentication**, where a system continually monitors and authenticates the user

[14]See ZKM12; See Bis18, Chapter 13.9.
[15]See DRN17, p. 191.
[16]See Bra+06; See Sho14, pp. 278–279; See Shi+11; See Oud16.

A combination of active and passive authentication is also possible. For example, the biometric passport (»ePassport«) contains both active authentication and passive authentication with the help of an integrated RFID chip.[17]

2.2.1 Active Authentication

The most common process of authentication is active authentication. In this process of authentication, the user has to initiate the authentication. Instances for this process can be opening a website and entering the password in the form fields, pressing a button or placing the fingerprint on the corresponding sensor. The biometric passport authenticates against a reading device with an asymmetric challenge-response protocol. This security measure helps to identify cloned passports.[18]

2.2.2 Passive Authentication

In contrast to the active authentication process, in the passive authentication process, the user is authenticated without the need to take action on their part. Use cases of passive authentication are, for example, RFID chips that continuously send a signal in a short-range and can open a door when the user approaches it. Further examples can be the analysis of keystrokes or touch screen usage patterns. In comparison with active authentication, this process is more low-friction. The biometric passport provides a way to calculate the integrity and authenticity from a reading device to improve the protection against forgery.[19]

2.2.3 Continuous Authentication

Further, the process of continuous authentication exists, too. In this case, the user is continuously authenticated or monitored to ensure that it is still the initially authenticated user who is using the system. Authentication must happen in a non-intrusive way. Commonly used for continuous authentication are biometrics, such as fingerprints, facial recognition, or keystroke patterns.[20]

Unfortunately, the term active authentication is often used to describe continuous authentication, too. To avoid confusion, solely term continuous authentication is

[17]See Eck14, p. 545.
[18]See DZZ14, pp. 185–186; See Eck14, p. 545.
[19]See DZZ14, p. 186; See XZL14; See Eck14, p. 545.
[20]See DRN17, pp. 236–238; See Fri+17.

used to refer to this process of authentication, while any mentions of active authentication are a reference to the process described in subsection 2.2.1.

2.3 Attestation

A typical problem in authentication is the trustworthiness between two parties, usually a server and a client. Assuring and proving that an entity is trustworthy is called attestation. Trusted Platform Module (TPM) computing uses attestation, also called »Remote Attestation«, but it is also important in the context of the Universal Authentication Framework (UAF), Universal Second Factor (U2F) protocol, and the Web Authentication API. An essential aspect is to prove (»vouch for«) an entity while keeping the user and the users' data private. This form of attestation is called Direct Anonymous Attestation (DAA), which can use ECC to achieve this goal, too.[21]

2.4 Challenge-Response Authentication

Challenge-response authentication is a further method of authentication by knowledge. Instead of transmitting the knowledge, the client answers challenges sent by the server. This proves that the client knows the shared secret. Both symmetric and asymmetric cryptosystems can be used. A basic symmetric approach is the following:

0. **requirement**: The server and client both know the same secret key K
1. the server generates a unique challenge c (e.g., a random number) and sends it to the client
2. the client computes the keyed hash $resp_c = hash(c, K)$
3. the server compares its computation of $resp_s = hash(c, K)$ to the received $resp_c$

To achieve mutual authentication, the client can also send a unique challenge to the server, which in turn generates the keyed hash and sends it back to the client for verification. A typical challenge-response protocol is Fiat-Shamir.[22]

[21] See Fen+17; See May17, p. 501; See Cok+11, p. 4; See Cel+17, p. 100; See CPS10, p. 226.
[22] See Was17, Chapter 13.6; See Eck14, pp. 489–491.

2.5 Zero-Knowledge Protocol

Zero-knowledge protocols or proofs are special variants of the challenge-response authentication where two participants want to prove the knowledge of a secret without disclosing the secret or parts of it to the other or third-parties. The verifying party asks the participant to solve a challenge that can only be answered correctly by knowing the secret. To decrease the chance that an attacker guessed correctly, the challenges are repeated multiple times. An example implementation is the Feige–Fiat–Shamir identification scheme. A more sophisticated variant is the zero-knowledge password proof (ZKPP) which is an interactive zero-knowledge proof. It is standardized in the Institute of Electrical and Electronics Engineers (IEEE) standard IEEE 1363.2. Using ZKPP protects against, e.g., guessing and dictionary attacks.[23]

2.6 Wording Differences between Multi-Factor, Multi-Step, Authentication, and Verification

Three different terms are used in the authentication environment. Single-factor authentication describes the authentication of a user with one of the described authentication methods. Two-factor authentication (2FA) described the process with two distinct methods of authentication involved in the authentication process, while multi-factor authentication (MFA) is an abstraction of this term that enables the usage of 2-n different methods of authentication.[24]

Authorities such as the Federal Office for Information Security (BSI), the European Union (EU), or the National Information Assurance Glossary also use the term »strong authentication« as a synonym for MFA, although strong authentication often differently defined.[25]

The naming of the chosen authentication or verification methods by companies is often confusing and difficult to understand. The terms used by companies vary from two-factor authentication (2FA), often just calling it 2FA, to two-step-verification, sometimes written as 2-Step Verification, too.[26]

[23] See Eck14, p. 492; See BKW14, Chapter 28.3.7; See FB17, pp. 769–770; See FFS88.
[24] See DRN17, pp. 186–188.
[25] See Nat18, p. 47; See Sic19, p. 11.
[26] See Sup19a; See Sup19b; See Pla; See Goob; See Mic19.

One could argue that the different authentication factors can be reduced to a single one, e.g., that an OTP is »something the user knows« since it relies on a secret that *could, in theory,* be memorized, too, but practically is not memorizable.
In this case, the term MFA or 2FA is technically incorrect, since it is instead a multi-step authentication because the same factor is used multiple times. However, it has to be noted that using the same authentication factor multiple times is weaker than using different authentication factors.[27]

The (user) verification, especially the verification of access permissions, is a part of the authentication process. Because of this, for the remainder of this thesis, the subtle differences between verification and authentication are not relevant, and the term MFA is used throughout.

2.7 FIDO's Universal Authentication Framework

2.7.1 FIDO Alliance

The Fast IDentity Online (FIDO) alliance is an open industry association founded in July 2012 that launched publicly in February 2013. Companies such as PayPal, Lenovo, and Infineon founded the FIDO alliance. Currently, the alliance has more than 260 members, including, e.g., Google, Amazon, Yubico, Samsung, Microsoft, VISA, or MasterCard. The goal of the FIDO alliance is to develop new authentication protocols and standards in order to enhance and simplify the user experience of MFA and to reduce the supersaturated usage of passwords. The FIDO alliance developed the specifications UAF and U2F.[28]

Another goal of the FIDO alliance is user privacy. As all specifications are based on public-key cryptography, this goal is easily achieved as each key-pair is unique for every registration the user performs and not shared with third parties. Because of the public-key cryptography, no link between the same user on different websites exist. Further, a relying party (RP) is only allowed to access the public-key credentials that are associated with the origin of the RP. Besides, one of the core principles is that biometric data never leaves the local authenticator and that no action is performed without the users' consent. Besides that, no authenticator device is uniquely identifiable, but only on a manufacturer or production-batch level.[29]

[27] See Gri17, p. 117.
[28] See FIb; See Eck14, p. 583; See Sch19a, p. 17.
[29] See FI14, pp. 6–7.

2.7.2 Universal Authentication Framework

The Universal Authentication Framework (UAF) is FIDO's solution for a passwordless experience and standardized in the International Telecommunication Union (ITU) recommendation ITU-T X.1277. It uses local and native device authentication, such as biometrics, to authorize the user. UAF does not feature 2FA but is instead designed as a direct replacement for the login with passwords. It is based on public-key cryptography with the use of challenge-response authentication to prevent replay attacks. The goal of the UAF is to provide a generic API that enables interoperability and unified user experience between different operating systems and clients.[30]

It features three key components, the *UAF client*, the *UAF server*, and the *UAF authenticators*. Besides that, the alliance offers a centralized metadata service. The communication between the client and the authenticator is performed via the UAF Authenticator-Specific Module (ASM), which offers a standardized API for the client to access and detect the different authenticators. Each authenticator is identified by the Authenticator Attestation ID (AAID), a unique model ID that comprises the vendor and model ID. The FIDO alliance centrally assigns and manages the vendor ID. Further, at manufacturing a private attestation key is inserted into the authenticator which can and will not change.[31]

Figure 2.4 on the next page shows the UAF architecture, where the user device is responsible for the communication between the authenticator, ASM, FIDO client, and the corresponding web browser or (mobile) app. The web browser is also called the user agent. The RP, commonly being a web server, and FIDO server are responsible for secure communication over the UAF protocol between the user device and RP. Further duties of the RP are authenticator validation and user authentication. The metadata service updates the database of approved, genuine, and certified authenticators that the FIDO uses for authenticator validation. The protocol defines four different uses cases, which are explained in more detail below:[32]

1. registration of the authenticator
2. user authentication
3. transaction confirmation

[30]See SM18, p. 249; See DRN17, pp. 197–198; See ITU18a.
[31]See LJ15, p. 145; See LT17, p. 8.
[32]See LT17, p. 4.

4. de-registration of the authenticator

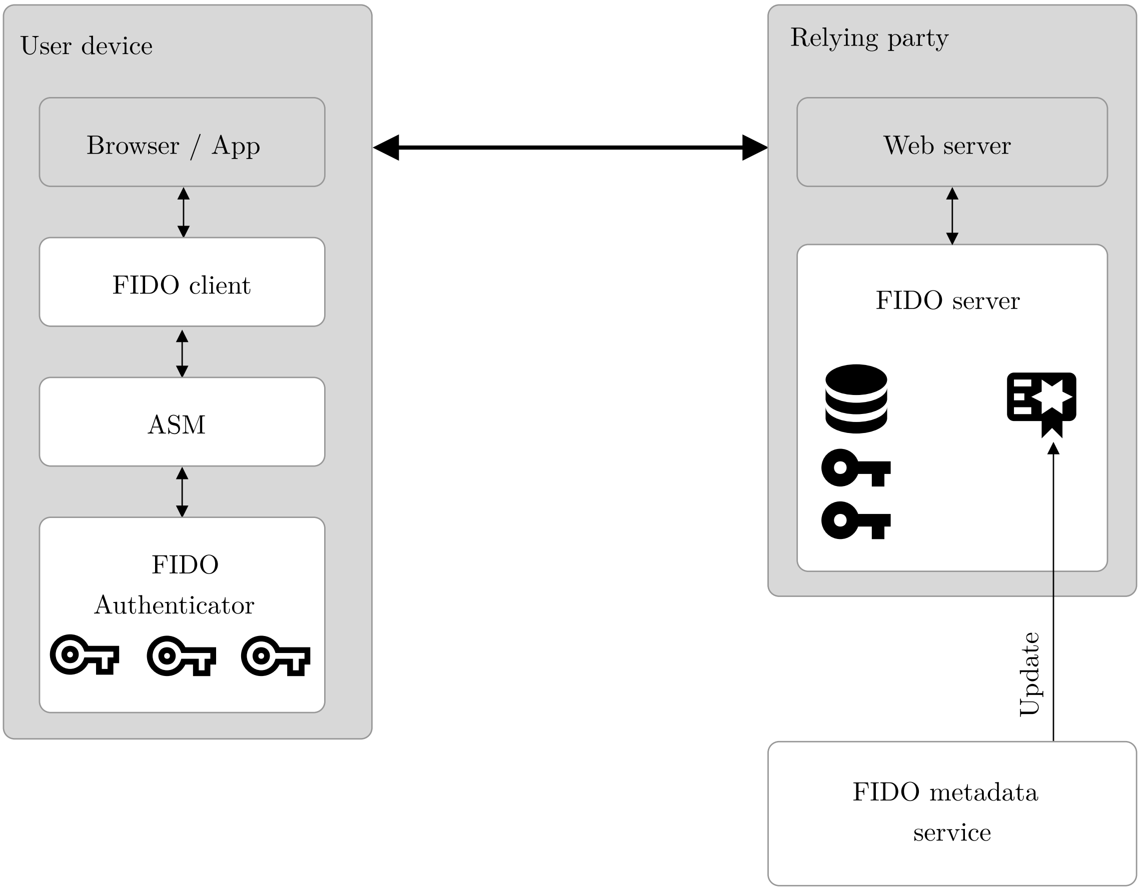

Figure 2.4: UAF architecture overview[33]

Registration

The registration process contains different steps. A FIDO server generates a policy object which contains allowed and disallowed authenticators. This data is sent together with the *server challenge*, *username*, *AppID*, and *FacetID* to the client. An AppID describes the RP origin, for example, »https://auth.timbrust.de«. Since the created credentials are subject to the same-origin policy, other (sub-)domains are not allowed to access the credentials. With the FacetIDs, a relying party can specific further subdomains that are allowed to access the credential. An example of a valid FacetID is »https://admin.timbrust.de«, where »https://auth.wings.de« is an invalid FacetID because the origin is different.[34]

[33] Source: diagram by author, based on Mac+17, p. 4.
[34] See Pan+17, pp. 131–132; See LT17, pp. 17–19; See Lin17a, pp. 3–4.

The client checks that the given AppID matches the requested server and processes the registration. In the first place, the *final challenge parameter (FCP)* is generated by hashing the server challenge, AppID, FacetID, and Transport Layer Security (TLS) data. Subsequently, the ASM computes the *KHAccessToken*, an access control mechanism to prevent unauthorized access to the authenticator. It comprises the AppID, *ASMToken* (a randomly generated and maintained secret by the ASM), *PersonalID* (a unique ID for each operating system (OS) user account), and the *CallerID* (the assigned ID of the OS for the FIDO client).[35]

Once the FCP and KHAccessToken are generated, the client sends the hashed FCP, KHAccessToken, and username to the authenticator. After receiving the data, the authenticator presents the data to the user (e.g., the AppID in a display) and performs user verification. When the user has been verified, and they approved the request, the authenticator generates a new key-pair and stores the data as the *key handle* in its secure storage. The key handle consists of the *public key*, KHAccessToken hash, and username. Also, it can be wrapped, i.e., encrypted in a way that only the client, ASM, and authenticator can decrypt it again. It has to be noted though, that the exact generation of the key handle is explicitly not specified, i.e., it varies among the vendors of UAF authenticators.[36]

After that, a *Key Registration Data (KRD)* object is sent back to the client. It contains the AAID, a signature counter, a registration counter, the hashed FCP, the public key, the key handle, and the attestation certificate of the authenticator. Further, a signature over the values AAID, hashed FCP, counters, and the public key is signed by the private attestation key of the authenticator.[37]

Finally, when the FIDO server receives the registration request (KRD and signature) from the user agent back, it can cryptographically verify the data by checking the sent signature. Additionally, the RP can evaluate the attestation certificate, the AAID of the authenticator, and the hash of the FCP. Ultimately, the server stores the public key in the database.[38]

Authentication

The authentication process is similar to the registration flow. When a user initiates the authentication, the RP sends the same payload as in the registration process.

[35]See Pan+17, pp. 131–132; See LT17, pp. 17–19.
[36]See LK17b, pp. 9, 16–17.
[37]See Ang18, pp. 12–13; See LT17, p. 22; See LK17a, p. 17.
[38]See HZ16, pp. 192–193; See LT17, p. 23.

The FIDO client again determines the correct authenticator based on the received server policy and the sent AppID. The FCP and its hash are generated in the same way as in the registration process. Further, the key handle and KHAccessToken are retrieved from the RP database and sent to the authenticator.[39]

When the authenticator receives the key handle, KHAccessToken, and the hash of the FCP, this data can be verified by the authenticator. If it matches, the user has been verified, and they approved the authentication request, the corresponding private key is retrieved from the key handle, and the signature counter increased. The authenticator sends the hashed FCP, the counter and a number used once (nonce) back to the client. Additionally, a signature signed by the private key consisting of the hashed FCP, nonce, and counter is sent back. In return, the client forwards the data to the RP. Finally, the RP can cryptographically verify the sent data by the signature and proceed with the authentication.[40]

Transaction Confirmation

Confirming a transaction is a special use case of the authentication process. The only difference between the regular authentication and the transaction confirmation is the additional t*ransaction text* the FIDO server sends to the client. This feature enables the UAF protocol to not only authenticate a user but also to let the user confirm certain transactions. A transaction text can be displayed on the authenticator display to show the user details about the transaction. However, the specifications list the authenticator display as optional. In case of the absence of a display, the ASM can offer the display functions as a software solution.[41]

De-registration

In contrast to the authentication and registration process, the de-registration process of authenticators is done without user verification. The server or client can initiate the process. The necessary information required for the authenticator is the AppID and optionally the specific credential, identified by the KeyID. The FIDO client and ASM send the required data to the authenticator. To ensure the genuineness of the request, the client checks that the AppID matches the origin of the request.[42]

[39]See Pan+17, pp. 132–133.
[40]See LK17a, pp. 20–21; See Ang18, p. 15.
[41]See Mac+17, p. 4; See SM18, p. 251.
[42]See LT17, p. 31; See Mac+17, p. 7.

2.8 Summary

This chapter established the basis of authentication, most importantly, the different methods of authentication. These include the knowledge, often being passwords, the possession of, e.g., security tokens and the secret to generate an OTP. Further, biometrics is of importance for the remainder of this thesis.

Additionally, the concepts of attestation, i.e., proving or vouching for an entity and the challenge-response authentication, have been introduced. This concept is of concern for the U2F and Web Authentication API. Moreover, the FIDO alliance with the first building block, the UAF has been explained.

Building upon these conceptions of single-factor authentication, the next chapter analyses their security in more depth, while Chapter 4 describes the combination of different authentication methods to achieve a MFA.

3 Security of Single-Factor Authentication

3.1 Threats Independent of the Authentication Method

Besides threats that affect specific methods of authentication, there are authentication independent threats, such as the enrollment or the transmission of the authentication data. The following sections take these threats into account, too.

3.1.1 Initialization

A more general threat is the initialization of the authentication, which is also referred to as registration or enrollment. The user has to make sure that no attacker can intercept or copy the required enrollment data. For instance, if malware compromises a user's computer and installs a keylogger, then an entered password is no longer a secret and therefore compromised. A computer virus could also intercept a USB connection from a security key, both when registering the device and while using it.[43]

Furthermore, the user needs to make sure that his enrollment process is not observed from, e.g., a surveillance camera, a hacked webcam, or a colleague from behind. With the recent rise of the Internet of Things (IoT) devices, for instance, internet-connected security cameras, the risk of observation is increased. Mobile phones are subject to trojans, and viruses, too. Because of this, the risk that, for instance, the camera is intercepted and a scanned Quick Response (QR) code that contains enrollment data for a time-based one-time password (TOTP) is sent to an attacker, is increased, too.[44]

3.1.2 Transmission

Further, the chosen transmission channel is an important fact to consider. Entering a password on an unencrypted website that uses only Hypertext Transfer Protocol

[43] See ULC19, p. 61.
[44] See Mul+13, pp. 152–153; See Dmi+14a, pp. 371–375.

(HTTP) enables network sniffing because the password is transmitted in cleartext and therefore accessible for everyone on the same network. For example, public Wi-Fi hotspots are a lucrative target, especially when the user is accepting custom Secure Sockets Layer (SSL) certificates. This enables an attacker to perform a MITM attack and even steal the passwords that are sent via an encrypted channel.[45]

The threat also applies to other authentication methods, too. A manipulated USB or smartcard port could copy the data on a security key or smartcard. Alternatively, a tampered sensor can capture the fingerprint of a user. SMS traffic is at high risk of being intercepted or eavesdropped. For example, the transmission of PINs or transaction authentication numbers (TANs)). Further, unencrypted e-mail traffic containing, e.g., temporary passwords or TOTPs is at risk.[46]

3.2 Knowledge

»Passwords are both the bane and the foundation of [...] security«,[47] yet the most used authentication method remains knowledge. In particular, passwords are used in IT. While it seems the simplest method to use, it also comes with many downsides, too. The service providers expect the user to remember the knowledge. Nevertheless, the human brain has difficulties remembering a unique and secure password, PIN, or secret questions for every different account the user has registered. The average amount of different internet accounts a user has is ten or more, not including, e.g., credit card PINs.[48]

Because of this fact, the user often does a couple of insecure things:

(a) using the same secret knowledge for multiple accounts or variations of the same knowledge[49]

(b) using something easy to guess or knowledge that is tied to a personal object, such as birthdays or names[50]

[45] See She+19, p. 518.
[46] See GB17, p. 103; See Dot19, p. 58; See May17, p. 6.
[47] Har05, p. 206.
[48] See Las18, pp. 7, 9; See Ras12, p. 424.
[49] See You18, p. 8; See Löw16, p. 14; See Las18, p. 7.
[50] See Fri19; See And08, p. 34.

(c) writing down the username and passwords, for instance, on a piece of paper that is easily accessible for others, storing PINs, e.g., in the briefcase or saving an unencrypted file on their computer or smartphone[51]

This enables an attacker to steal the login credentials of a user without much effort. Written down post-it notes enable any physical attacker to take the credentials. It might be captured by a camera or a colleague looking over the shoulder, too. Leaving an unencrypted file on the computer or mobile devices enables computer viruses and trojans to send the file to an attacker.[52]

When using a weak password, an attacker might be able to guess the chosen password. Writing down the banking PIN and storing it in the same briefcase as the credit card even annuls the 2FA example of possession and knowledge.[53]

Figure 3.1 shows a representative study of password re-usage in the United States in 2018 conducted by YouGov. In the survey, over 70% of all participants answered that they at least re-use some of their passwords for different accounts. Only 20% of the participants use a unique password for every service. The survey is further classified into age and gender.

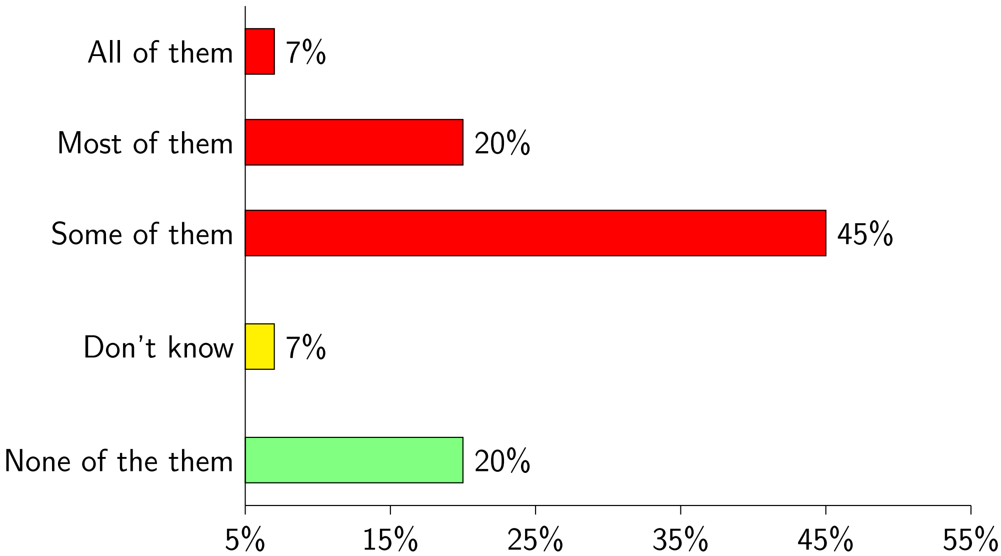

Figure 3.1: Percentage of online accounts sharing the same password in the United States in 2018[54]

[51]See Fri19; See You19, p. 6.
[52]See Kis19, Chapter 4.1.
[53]See Swe08, p. xxi.
[54]Source: You18, p. 8.

While there is only a marginal difference between the genders, the survey is showing that the password re-usage rate in the age group 18 to 34 is 79% in total, which is weakening the potential argument that younger people tend to be more aware of the risks of stolen credentials and therefore use more complex and unique passwords. Other surveys strengthen the observation that millennials are re-using passwords more often compared to other age groups.[55]

Regarding the security of recovery and secret questions, it must be noted that these might even decrease security. Relatives and friends can answer common examples of questions such as »the first pet name, first car model, middle name of a parent, or the city where your parents met«. This enables a malicious insider attack. Some questions might be answerable by employing a social engineering attack, too.
Besides that, data can even be gathered by using, e.g., data mining of publicly available data sources and reports. Ironically, it is more secure to answer security questions wrong than honest and correct. When possible and allowed by the service, custom security questions should be used. Additionally, it is not uncommon to be able to guess the partner's password.[56]

Unfortunately, in history, it was thought that a forced change of passwords increases security. Because of this, a lot of enterprises, policies, and standards still contain sections regarding the enforced password rotation. However, studies show that security is not increased by forcing the user to change their password on a regular basis. Of course, this does not mean that the user should not change their password in case of a potential data breach.[57]

Further, especially true for passwords, it is not known to the user what the service provider does in order to protect the security of the passwords. As security breaches happen nearly daily, it is crucial to protect the password of the user. For instance, if the passwords are stored in a database, they can be:

(a) unencrypted (worst case)

(b) hashed but not salted

(c) hashed and salted

(d) hashed but not peppered

(e) hashed and peppered

[55] See Kes18, p. 10; See You18, p. 8; See Las18, p. 11; See Tho+17, p. 1429.
[56] See Las18, p. 11; See Bra+06, p. 169; See Bon+15; See Rab08, pp. 5–6; See SBE09, p. 386.
[57] See Sic16, p. 1520; See Gra+17, p. 14; See Sch16; See And08, p. 34.

(f) hashed, salted, and peppered

(g) encrypted

It is pretty evident that unencrypted passwords in a database render the most significant threat, especially when re-used. Along with the e-mail or username, an attacker can probably use the stolen credentials for other accounts, too. In case of an e-mail provider breach, they can re-issue a new password for other websites with the »forgotten password« mechanism.[58]

Even if the password is hashed, but not salted, it renders the credentials at risk. Hashing algorithms such as Message Digest (MD) version 5 or Secure Hash Algorithm (SHA)-1 might be vulnerable to a vulnerability in the future, given that their collision resistance has already been attacked successfully. However, if a weak password is used, the hash might have been reversed already. Possessing the hashed password list enables the attacker to execute a brute force attack in order to reverse as many hashes as possible. A rainbow table attack, a dictionary attack, or just searching it in databases that contain a billion of reversed hash values is another attack.[59]

Obtaining a password hash is often enough for an attacker to gain access to further user accounts. They can reverse easy hashes and then automatically try to gain access with these credentials on other websites, too. This form of attack is called »credential stuffing« and a subform of the brute-force attack.[60]

Higher protection of the password can be achieved by using a unique »salt« for each password. The salt is a fixed-length cryptographically strong random value that is concatenated with the actual password before hashing it. Salting a password serves two purposes. Firstly, it decreases the risk of a successful rainbow or brute force attack dramatically because the hashes changed for known passwords. Secondly, it does not reveal users who have chosen the same password. The salt itself does not need to be encrypted or obfuscated since its purpose is to harden the brute-force, dictionary, and rainbow attacks and decrease their chance of success.[61]

Another technique to harden the password hashes is the use of a »pepper«. In contrast to the salt, the pepper is treated as a secret and not stored in the database. The pepper is not uniquely generated for each user account, but instead a fixed

[58]See Sho14, p. 277.
[59]See Tho+17, p. 1425; See Ras12, pp. 427–430; See And08, pp. 56–57.
[60]See Hun17; See Tho+19, p. 1565; See Zab19, Chapter 5.5.
[61]See LM16, pp. 32–34; See BB17, pp. 130–131; See Gra+17, p. 15.

string or a string from a fixed set. When the latter is chosen, the server needs to generate the hash with each possible pepper value and then compare it with the stored hash when authenticating a user. An example of the effects the salt and pepper have to the hash is shown in Table 3.1. The beginning of identical hashes is marked bold. The table shows that just relying on pepper is not sufficient to hide users that share the same password.[62]

Both techniques can be combined, but each of them independently strengthens the generated hash of a potential weak password by increasing its length and complexity. Therefore, these techniques reduce the risk of a collision.

user	salt	pepper	password	resulting SHA-1 hash
tim			Wings	**daaf17**ba041ff1a2184a2b024a9f83442a7ca3ee
caro			Wings	**daaf17**ba041ff1a2184a2b024a9f83442a7ca3ee
tim	c261012e		Wings	2e35d46e345fd77317e5473586f15d681e89b9a3
caro	5f40720d		Wings	ee229d4f4c8f3a9137f98e7f8b5d46f26d9c9b8d
tim		18e6c63a	Wings	**2e6536**c7a16feaaca34b6b83a311a0880ad0f80e
caro		18e6c63a	Wings	**2e6536**c7a16feaaca34b6b83a311a0880ad0f80e
tim	c261012e	18e6c63a	Wings	7d707f1b6dd8f811fabd17e311e01d35015ce9cd
caro	5f40720d	18e6c63a	Wings	33c1b9d955d6d0b7f420871907e822ccbe708249

Table 3.1: Example password SHA-1 hashes with and without salt and pepper

3.3 Possession

The primary threats of authentication by possession are that it is not tied to the user itself and can be lost or even worse stolen by an attacker. Besides that, possession factors can be shared between multiple users, allowing attacks such as a malicious insider attack. Often the possession factors are not protected itself, e.g., a keycard to open a door can be used by an attacker, too.

Another usage implication is that the possession must be carried with the user and can be forgotten. This makes the authentication impossible if no access to

[62] See LM16, pp. 33–35; See Gra+17, p. 15; See Man96, p. 173.

the possession is possible, and no backup or different authentication methods are available. A different threat is that possession can be damaged or destroyed. For example, carrying security keys on a keyring exposes them to damage by a fall or liquids.[63]

Especially possessions that use wireless transmissions, such as BLE, NFC, or RFID, can be copied even over some distances. For instance, an attacker can copy credit cards in crowded places, such as trains or buses.[64]

Compared with knowledge, a replacement is more costly, complicated, and time-consuming (e.g., when a passport is lost or stolen). If, for example, a whole algorithm is broken, such as the first generation of RSA SecurID or some YubiKey models that happened to be vulnerable, it can cause severe problems depending on the number of keys that need to be replaced.[65]

3.4 Biometrics

In contrast to possession and knowledge, the biometric trait cannot be stolen easily, but it can be copied, e.g., the fingerprint from a high-resolution photograph. Alternatively, copies of face models can be used to circumvent face recognition systems. In the recent past, researchers could copy both German Chancellor's Angela Merkel's iris and the fingerprint of Ursula von der Leyen, the now elected President of the European Commission, from high-resolution photographs. It must be taken into account though that especially the so-called latent fingerprints are nearly left everywhere, i.e., the security of biometrics heavily relies on the chosen biometric trait.[66]

Further implications are that the biometric characteristics can change over time or be temporarily unavailable because of injuries. While some injuries can heal over time, others, especially scars, can permanently change the biometric trait and therefore render it unusable. Also, each time the user authenticates with biometrics, a new sample of the trait is gathered and compared to the stored one. Because the recent probe is never 100% identical compared to the stored one (»intra-user variants«), a threshold needs to be defined, which allows or denies the authentication attempt. Setting the threshold to a too low value increases the risk of the FAR, while a too

[63] See Sho14, pp. 263–264.
[64] See KSM14, pp. 2–3.
[65] See DRN17, p. 18; See BLP05, pp. 364–265; See Wes19b.
[66] See FKH14, pp. 7–8; See FSS18, pp. 7–8; See Mar13, p. e199; See Kre14.

high value decreases the usability and increases the FRR.[67]
Traits such as facial recognition must also be usable with, e.g., different amounts of facial hair, hairstyles, or with and without glasses.[68]

Another significant threat is data privacy and security. Over 50% of the users fear of data usage, both legitimate and abusive, and collection of their biometrics. Nevertheless, the majority of the user states that biometrics are the most secure authentication compared to, e.g., passwords and PINs. It is crucial that the stored biometric probe is not accessible by third parties nor shared with them. For example, a theft of a smartphone should not mean theft of the biometrics, e.g., fingerprint or facial scan, too.[69]

However, the primary threat remains the difficulty to replace a compromised biometric template. A password or a security key can be changed or replaced. However, for instance, a fingerprint cannot be altered, changed, or replaced since it remains the same for the whole lifespan of a person. To counter this threat, it is advised to use, for example, only a hash of the fingerprint and not to store the *image* of the fingerprint itself.[70]

Moreover, it is necessary to respect the quality and availability of the sensor. If a sensor is damaged, too cheap, or the surface is, for example, dirty, then the authentication and especially the usability suffers.[71]

3.5 Further Methods

A high threat of location-based authentication is the spoofing of the actual location by an attacker. An attacker can choose different attacks, such as spoofing the source IP address that tries to access a system. Another form of spoofing is GPS spoofing, where an attacker modifies the actual GPS by broadcasting false information. Further, the Caller ID spoofing technique can be used with Voice over Internet Protocol (VoIP) to disguise the location. Besides these techniques, the most common variant remains the usage of a virtual private network (VPN) or Domain Name System (DNS) proxy to hide the genuine location.[72]

[67] See JRN11, pp. 13–17, 52.
[68] See JRN11, p. 98.
[69] See Kes18, p. 8.
[70] See Sho14, p. 266.
[71] See Tod07, p. 37.
[72] See Har05, pp. 138–145; See Yua05, Chapter 4.5.3; See Eck14, pp. 115–116, 133.

For time-based authentication, an attacker could use attacks against the Network Time Protocol (NTP) in order to either gain access to the verification system or to modify the synchronized time in order to allow the login attempt to succeed.[73]

[73]See Mal+15.

4 Multi-Factor Authentication

In this chapter, different MFA solutions are described in detail. It describes the process of using two (2FA) or more (MFA) distinct authentication methods for the user authentication, e.g., the password (knowledge) and a security token (possession).

Since this thesis focuses on the internet and web technologies, it is always assumed that the first factor is knowledge-based, i.e., in the majority of the use cases, a password. Therefore, further knowledge-based authentication methods are not taken into account in this chapter.

4.1 Motivation for the Usage of Multi-Factor Authentication

The motivation for the usage of MFA is derived from the previous chapter. The chapter showed that multiple security threats exist, which are independent of the specific authentication method. These make user accounts, for instance, vulnerable to theft, impersonation, or phishing. Also, the previous chapter outlined the threats of password re-usage and weak passwords that have the potential to lead to subsequent attacks with, e.g., credential stuffing.

In order to decrease or even eliminate these threats, the user needs to deploy additional security measures that are explained in this chapter. The topics of OTPs, smartcards, security tokens, and the U2F protocol are presented in this chapter as possible solutions towards the security threats.

Further, new formalities even require the usage of MFA, such as the new version of the Payment Services Directive (PSD), EU Directive 2015/2366, coming into full effect in September 2019.[74]

[74]See Noc18, p. 10.

4.2 Transmission of Information

A key aspect to take into account is the chosen transmission channel for the second or different (multi) factor. Out-of-band (OOB) authentication describes the transportation of information on another channel or network than the current one. While, e.g., the *standard* transmission of information on websites happens via HTTP, an example of OOB authentication is a phone call or SMS to send the second factor.

This technique helps to reduce the risks of eavesdropping drastically since an attacker needs to have control over two (or more) distinct communication channels. Of course, the chosen OOB channel should protect against eavesdropping, i.e., be secure or encrypted.

The increased security only takes effect if the different factor is not transmitted over the first channel because the first channel might be intercepted. For instance, OOB provides no benefits when the user enters the OTP received via a different channel on a phishing website along with their password.[75]

4.3 One-Time Passwords

A widely used method to achieve MFA is OTPs. These belong to the category of possession because of a shared secret between the client and the server. Both parties possess it to verify or generate the OTP.

To fully understand how the OTP works, first, the basics and origins, especially the underlying message authentication code (MAC), have to be explained. In the following subsection, the required algorithms are shortly described. Hereupon in subsection 4.3.3 and subsection 4.3.4 two variants of OTPs, namely HMAC-based one-time password (HOTP) and time-based one-time password (TOTP), are introduced. Both extend the keyed-hash message authentication code (HMAC) algorithm.

4.3.1 Message Authentication Code

The message authentication code (MAC) is a generated *code* (hash), i.e., some sort of information to protect and ensure the integrity of a message. Integrity, besides confidentially and availability, is one of the main concepts of IT security. The MAC

[75] See Gra+17, p. 17; See Ras12, p. 441; See BB17, p. 140; See GB17, p. 106.

is built using two parameters, a secret key that both parties know and the message itself. The algorithm generates a checksum that the sender can send accompanied by the message. Upon retrieval of the message, the recipient calculates the checksum (MAC) themselves. If it differs from the received MAC, then the message has been manipulated, or the transmission was faulty. Technically, the MAC can be generated with, e.g., cryptographic-hash functions, such as HMAC, or using block ciphers, such as cipher block chaining message authentication code (CBC-MAC) or Data Encryption Standard (DES).[76]

The MAC is standardized in different norms from various institutions, for example, the National Institute of Standards and Technology (NIST) Federal Information Processing Standard Publication (FIPS) 198-1, the BSI technical guideline TR-02102-1 (»Cryptographic Mechanisms: Recommendations and Key Length«), or the International Organization for Standardization (ISO) norm ISO/IEC 9797-1 and ISO/IEC 9797-2.[77]

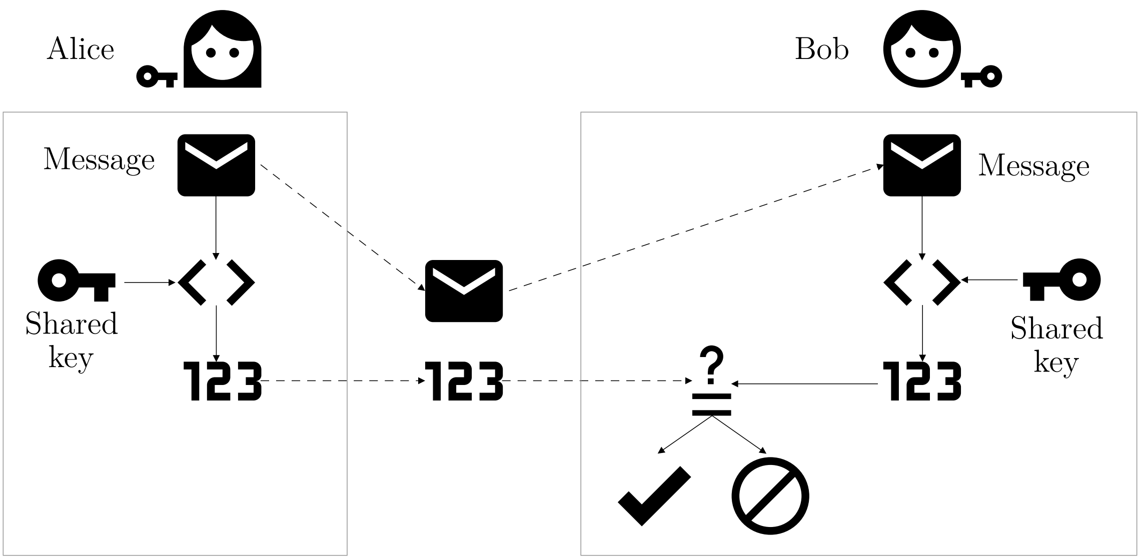

Figure 4.1: Message authentication code used to protect a sent message[78]

Figure 4.1 shows the MAC in use between Alice and Bob. Both Alice and Bob exchange a secret key only they know via a secure channel. Alice now sends a message to Bob. In order to secure the message integrity, she uses an algorithm that takes both message and the secret key as inputs and computes the cryptographic hash of the message, the MAC. She transmits both the message and the MAC to

[76]See Pan12, p. 565; See And08, pp. 163–168; See Eck14, pp. 391–393.
[77]See ST08; See Inf19a; See ISO11a; See ISO11b.
[78]Source: diagram by author

Bob. If the message is not confidential, it is also possible to choose an insecure transmission channel. Bob is now able to calculate the MAC himself by using the same algorithm, key, and the received message from Alice.

If his computation of the MAC matches the one sent by Alice, then the integrity and authenticity of the message are given. Otherwise, the message might have been intercepted and manipulated.[79]

Mathematically, the MAC is defined as:

$$mac = MAC(M, K)$$

Where M is the input message, MAC the used MAC function, K the shared secret key, and mac the resulting message authentication code.

Sometimes the MAC is also called Message Integrity Code (MIC) in order to avoid confusion with the media access control (MAC) address layer used in network protocols. Additionally, the MIC, without the use of a shared secret key, but rather only a hash function, does not prove authenticity. An attacker can modify the message and re-generate the MIC of the modified message with a hash function.[80]

Further, while the MAC provides authenticity regarding the origin of the data and the data integrity, it does not provide any authenticity regarding the content of the data. For example, mobile code is not detected by the MAC, as long as the MAC belongs to the sent message. This implication has to be taken into account when using the MAC to authenticate and evaluate the trustworthiness of received messages. The encrypted traffic is increasing, but at the same time, the encrypted malware traffic is, too.[81]

4.3.2 HMAC

The keyed-hash message authentication code (HMAC) extends the MAC and is standardized in Request For Comments (RFC) 2104 and NIST's standard FIPS 198-1. It allows the usage of any cryptographic hash function, such as SHA family, MDs algorithms, bcrypt, or whirlpool. Because of the black-box design of the HMAC, the easy replacement of the used cryptographic hash function is possible. Besides

[79] See PP11, p. 320.
[80] See Tod07, pp. 60–62.
[81] See Wel15, p. 100.

authentication, the HMAC is, e.g., used in TLS and JSON Web Tokens (JWTs) to ensure data authenticity and integrity.[82]

Mathematically, the HMAC is defined as:

$$HMAC(K, m) = H((K' \oplus opad), H((K' \oplus ipad), m))$$

Where K is the shared secret key and *K'* is the result by appending zeroes to the key K until it reaches a full block size (B) defined by the hash function. The inner padding *ipad* is constructed by repeating the byte *0x36* B-times, and *opad* is the outer padding constructed by repeating the byte *0x5C* B-times.

Naively one could think that the HMAC is constructed by just hashing the secret key with the message. In order to increase the security and to protect against a probable collision of the hash functions, the algorithm design is slightly different and shown in the next figure.

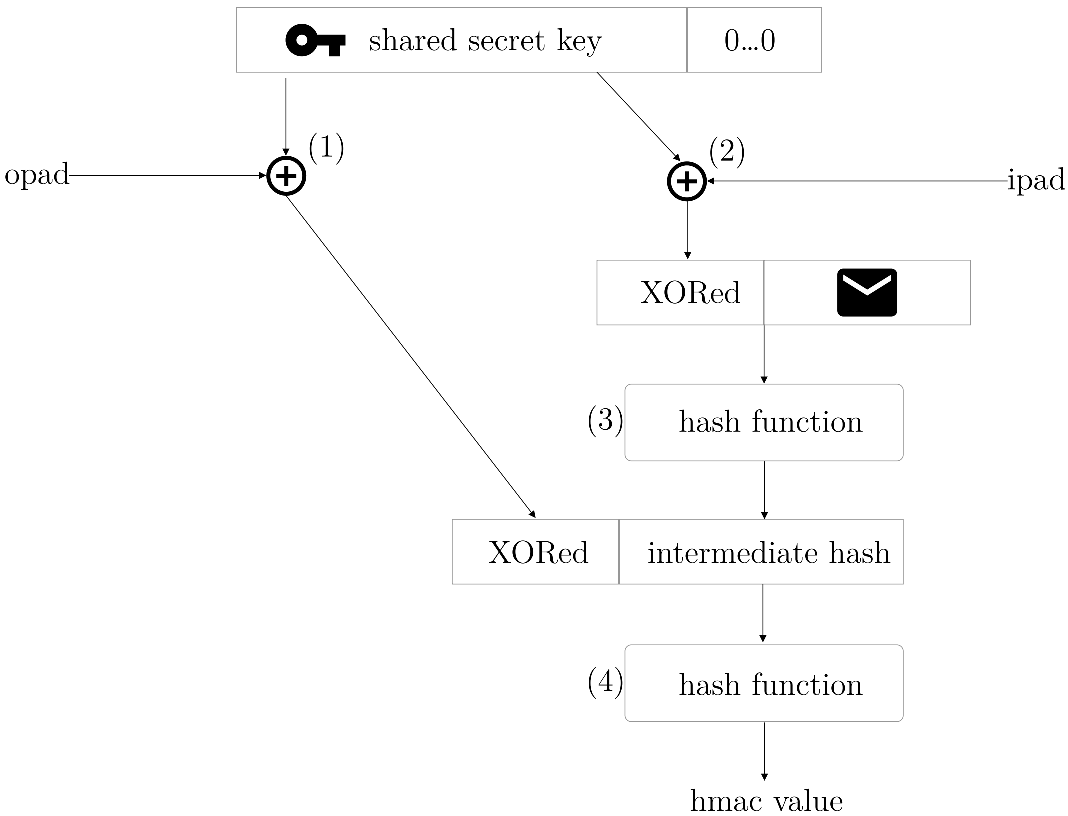

Figure 4.2: Visualization of the HMAC algorithm[83]

[82]See KBC97; See ST08; See DR08, p. 14; See JBS15, p. 8; See TC11, pp. 3–4.
[83]Source: diagram by author, based on Eck14, p. 395.

The exclusive or (XOR) operation is performed on the key, *opad* (1) and *ipad* (2), respectively, instead. Besides that, the hash function is invoked twice, first on the result of the XOR operation on *K'* and the *ipad* (3) with the message and then again on the final result of the concatenation of (1), (2) and (3). Figure 4.2 shows the intermediate steps in order to generate the HMAC.

One of the key aspects of the HMAC is that the efficiency of the original hash function is maintained and not altered by wrapping it in the HMAC algorithm. The security of the MAC relies on the used cryptographic hash function and the strength of the secret key, for example, length and chosen alphabet. The best-known attacks against HMAC remain the brute force and birthday attack. Section 5.2 performs further security analysis.[84]

4.3.3 Counter-based

The HMAC-based one-time password (HOTP) is an extension and truncation of the HMAC which is standardized in the RFC 4226. It is a joint effort between the Internet Engineering Task Force (IETF) and the Initiative for Open Authentication (OATH). In contrast to the HMAC, it is not an algorithm for message authentication and integrity, but instead an algorithm for the generation of OTPs. The security relies on the fact that a »moving factor«, i.e., in this case, a counter is used to generate passwords that are only valid once. Alternatively, the HMAC-based one-time password (HOTP) is also referred to as event-based, and the secret key is called the seed. The length of the numeric OTP is configurable, and the defined minimum is six digits. The standard only defines HMAC-SHA-1 as the cryptographic hash function to use. However, it is also possible to replace the cryptographic hash function due to the black-box design, although the implementation will not comply with the RFC anymore.[85]

The HOTP is mathematically defined as:

$$HOTP(K, C) = truncate(HMAC(K, C))\ mod\ 10^d$$

Where K is the secret key, C is a counter value, and *truncate* the function to truncate the result of the HMAC dynamically. The result is then transformed via the modulo operation into decimal numbers (mod 10^d, where d is the number of

[84] See Bis18, Chapter 10.4.1; See Sta17, p. 398; See BCK96, pp. 3, 10–13; See PO95.
[85] See MRa+05; See Sta15, Chapter 3.

digits to generate). The *truncate* function is the core of the HOTP and explained below:

1. At first, the dynamic truncation extracts the least four significant bits as an offset from the 20 bytes long HMAC-SHA-1 result, i.e., from the byte 20.
2. Extract the next 31 bits from the offset position in order to generate a 4-bytes long string. The most significant bit is skipped in order to avoid issues such as modulo operations on negative numbers caused by varying computation results based on implementation differences.

Due to its design, there are a couple of limitations to the HOTP. The counter used between the parties can become out of synchronization, requiring further efforts to re-synchronize. Since the server only increases the counter on successful authentication, the out of sync scenario can occur when the client incremented the counter on, e.g., a failure, too. The server and client can become synchronized again by generating the next OTP by increasing the counter (look-ahead window) in order to verify if this OTP matches.
Another method for re-synchronization is the sending of multiple future values. It is vital to limit the look-ahead window to decrease the attack surface. Further, the server should throttle the authentication attempts in order to counter brute-force attacks. Section 5.2 does further analysis.[86]

Additionally, the HOTP allows bidirectional (mutual) authentication, i.e., the user can authenticate the server if it sends the next OTP value that the client then can validate. HOTPs are commonly used in physical security keys, such as YubiKeys, but are also present in software solutions, e.g., in the Google Authenticator.[87]

4.3.4 Time-based

The time-based one-time password (TOTP) is again an extension of the HOTP and is time-based instead of counter-based. It is a joint effort of the IETF and the OATH, too, resulting in standardization in RFC 6238.[88]

Mathematically the TOTP is defined equal to the HOTP:

$$TOTP(K, T) = truncate(HMAC(K, T))\ mod\ 10^d$$

[86]See SM18, p. 236; See Bis18, Chapter 13.5.1.
[87]See HS17, p. 716; See MRa+05, p. 14.
[88]See MRa+11.

The only difference is that the counter is substituted by T, where T is a computed time value derived from the reference date (*T0*). The default reference date is the Unix epoch time (1st January 1970). Instead of increasing the counter manually or on an event, a time-step value (X) in seconds is used to increase the counter value. The default defined in the RFC is 30 seconds.

Formally more correct, T can be described as:

$$T = \frac{(Current\,Unix\,time - T0)}{X}$$

In contrast to the HOTP definition, RFC 6238 explicitly defines the use of other cryptographic hash algorithms such as SHA-256 or SHA-512. Besides the introduced security considerations and usability implications in subsection 4.3.3, such as throttling and synchronization, an essential aspect of the TOTP to take into account is the configured time-step. While a greater time-step size increases the usability of the user, it also expands the attack window. Also, the user has to wait a long time until a new OTP is generated in case a fresh one is required. If a user or attacker sends the same OTP in the same time-step window, the server must not accept the same value after a successful authentication but instead wait until the next time-step window.[89]

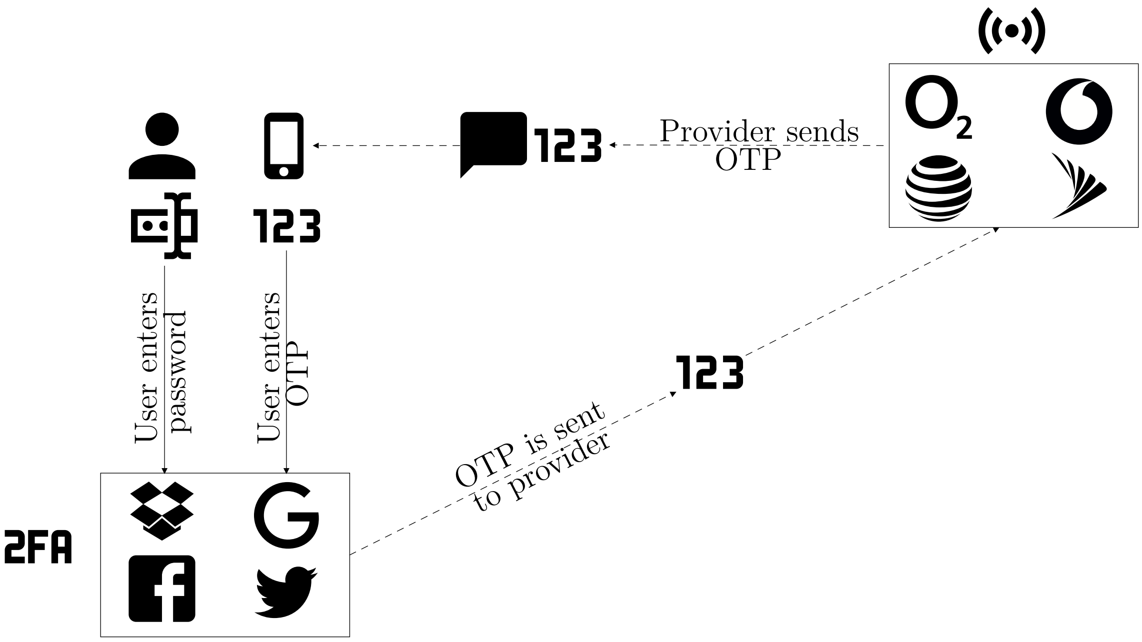

Figure 4.3: Exemplary MFA flow[90]

[89]See MRa+11, p. 6.
[90]Source: diagram by author

Figure 4.3 shows an example of an authentication flow using TOTPs. In this scenario, the user tries to log in to a service that uses 2FA. After entering their password (knowledge as the first factor), they either

(a) use, e.g., a smartphone app or hardware token to generate the TOTP.

(b) receive the TOTP from the service, e.g., via a text message, e-mail, or phone call (the figure shows an SMS).

Once the user has obtained the OTP (possession as the second factor), they can enter it at the login screen and send it to the server, the RP. The service can now validate the OTP while respecting the look-ahead window and allow the user authentication.

4.3.5 Yubico OTP

In contrast to the open standards the TOTP and HOTP, the Swedish company Yubico developed a proprietary OTP protocol, too. It is available for all their manufactured and sold YubiKeys. The generated OTP is a 44-characters long string which is constructed by using Advanced Encryption Standard (AES) with 128-bit. It is encoded into 32 hexadecimal characters using a modified hexadecimal (»modhex«) encoding, yielding a 22-byte value. Each YubiKey contains a unique public ID of 6-bytes that is optionally prepended to the OTP. The OTP is 16-bytes long, which is exactly the block size of the AES 128-bit algorithm.[91]

The constructed OTP can be divided into the following logical groups:

$$\underbrace{4645652b4551}_{\text{private uid}}\quad \underbrace{5561}_{\text{usage counter}}\quad \overbrace{4c5f44}^{\text{timestamp}}\quad \underbrace{65}_{\text{session usage counter}}\quad \overbrace{4752}^{\text{pseudo-random}}\quad \underbrace{6545}_{\text{CRC}}$$

Where the 48-bit *unique private/secret ID* is stored in the YubiKey configuration and can be changed (write-only). Further, the OTP consists of a non-volatile 16-bit *usage counter*, a 24-bit *timestamp* value, set to a random value after startup and increased by an 8-Hz clock, and the *session usage counter*. Besides that, the OTP contains an 8-bit volatile counter that is initialized with zero after power-up and then increased by one each OTP generation, the *pseudo-random number* of 16-bits

[91] KS12; See Jac16, pp. 84–86.

and a cyclic redundancy check (CRC) *checksum* of 16-bits for the fields. Finally, the generated OTP is encrypted with the per-device unique AES-128 key.[92]

The authentication server can either be used as a service from Yubico, as only they know the pre-configured AES key of each YubiKey. This renders their central key server a lucrative target for criminals though since it is a centralized place of all AES keys. Alternatively, the validation server software is available as a self-hosted solution in different programming languages. This requires changing the AES key of the YubiKeys in order to save the shared key on the server, too, since Yubico will not give access to the pre-configured AES keys.[93]

4.4 Smartcards

Smartcards, sometimes called chip cards or integrated circuit cards (ICCs), too, are physical plastic cards, often the size of a credit card. They contain an internal chip for user authentication. The chip is either exposed or can be accessed contactless. Typical examples are SIM cards, credit cards, Common Access Cards (CACs) used by the United States (US) Department of Defense, or identity cards issued by authorities. In IT, smartcards can also store certificates and are used for computer log on. The smartcard differs from a regular storage card by having a microprocessor and an erasable programmable read-only memory (EPROM) or electrically erasable programmable read-only memory (EEPROM). It is defined in the ISO standard 7816, which also defines different sizes of smartcards. The NIST standard FIPS 201-2 defines the usage of smartcards for Personal Identity Verification (PIV) for federal employees.[94]

Figure 4.4 on the next page shows the typical architecture of a smartcard chip. The read-only memory (ROM) contains the OS of the smartcard, whereas the random-access memory (RAM) is used for temporary storage. The application storage uses the EEPROM. Some smartcards also contain a second processor (co-processor) for cryptographic operations.

[92] See Yub15, pp. 8–9, 33–34; See ORP13, pp. 209–210.
[93] See Yub12, pp. 8–9.
[94] See Eck14, pp. 525–527; See ISO11c; See May17, pp. 6–9; See ST13.

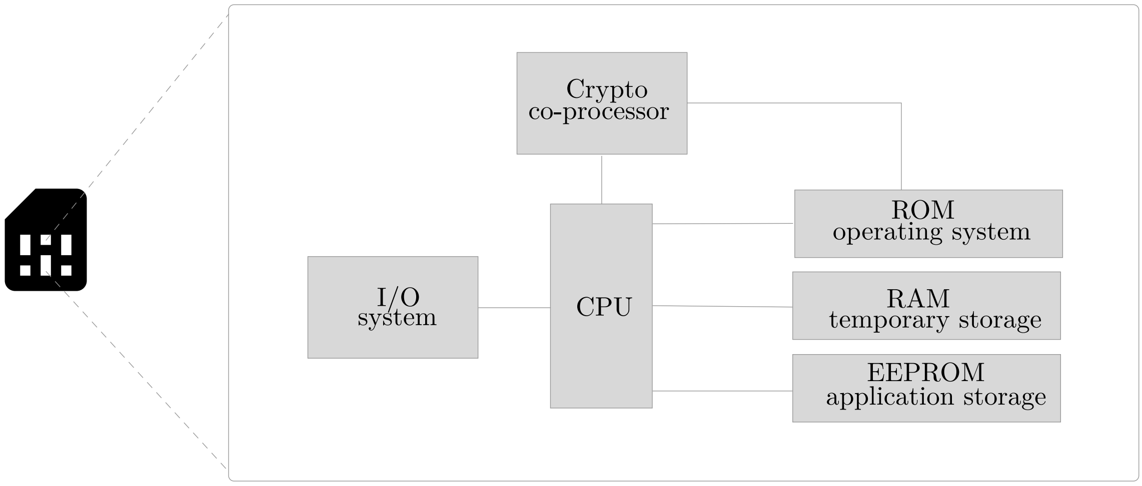

Figure 4.4: Typical smartcard architecture[95]

Security-wise an essential requirement is that an attacker cannot access the private data on the internal chip, i.e., that the smartcard is tamper-resistant. This is, e.g., achieved by physically covering the central processing unit (CPU), RAM, and EEPROM with a shield. The data stored on the smartcard can itself protected by using a PIN or biometrics, such as a fingerprint, to access the data.[96]

In the aspect of usability, the smartcard always requires dedicated hardware, either external or built-in. A smartcard reader is necessary to use the smartcard as an authentication method. While especially enterprise notebooks contain a smartcard slot, e.g., mobile phones do not. The chip card interface device (CCID) protocol defines a USB protocol, so it is at least possible to use a USB card reader. Additionally, smartcards with an embedded Java Card Virtual Machine (JCVM) allow the execution of Java application and servlets, further opening the development possibilities for smartcard applications.[97]

4.5 Security Tokens

Besides smartcards, further MFA solutions with possession as an additional factor are security tokens or keys. These security tokens exist as pure hardware solutions, as well as software-based solutions. The minimum security requirements for the cryptographic modules are defined in, e.g., the NIST FIPS 140-3 standard. This

[95]Source: diagram by author, based on Fer15, p. 33; Tun17, p. 228.
[96]See Tod07, p. 34; See Tun17, p. 228.
[97]See MA17, p. 65; See Eck14, p. 539.

section introduces the well-known security tokens »RSA SecurID« and »YubiKeys«. Typically security tokens either store a private key used in public-key cryptography or the shared secret in order to generate or validate OTPs.[98]

4.5.1 RSA SecurID

The RSA SecurID exists in several variants, both as hardware and as software tokens. First hardware revisions used a 64-bit proprietary protocol called »SecurID hash function«. Hardware keys newer than 2003 use the standardized 128-bit RSA algorithm in order to generate OTPs. Newer revisions also feature a USB port that allows the device to store custom certificates, i.e., making it a smartcard device, too. Other form-factors, such as credit card-sized variants, exist, too. Each token contains a burned in seed and a random key that was generated while manufacturing the device. Since this seed needs to be known to validate the OTP, the RSA SecurID server needs to be used. The default time for the OTP time-step value is 60 seconds, but this can be configured to, e.g., 30 seconds. The SecureID tokens are battery powered and small enough to be carried on the keyring. The SecurID can itself be protected by a PIN that is required to generate the OTP.[99]

Mobile applications for iOS, BlackBerry OS, BlackBerry 10, Windows Phone and Android exist, too, offering support for a soft-token based solution. Desktop applications for macOS and Windows are available, too.[100]

4.5.2 YubiKey

Besides a proprietary OTP algorithm, the company Yubico is best known for its physical security tokens, the YubiKey. A variety of tokens exist, ranging from different USB-A and USB-C variants, NFC-capable tokens to lightning connectors for the usage with iOS. Besides different connectivity, various form factors are available, too. For example, Yubico offers very tiny tokens that can remain in the USB port permanently. All tokens, except the »Security Key« series, support Yubico's OTP algorithm, HOTP, TOTP, and U2F, as well as static passwords and OpenPGP. The »FIPS series« is FIPS 140-2 certified, i.e., their cryptographic modules are approved by the US government and are usable for PIV.[101]

[98] See ST19; See BKW14, Chapter 28.4.3.
[99] See Eck14, pp. 479–480; See Han+07, p. 296.
[100] See WPR16, pp. 3–6; See LB10, p. 49.
[101] See HS17, p. 716; See Jac16, p. 83; See Jac19, p. 109.

4.5.3 Software Tokens

While already touched briefly in the previous subsections, the software or soft tokens are security tokens completely available as software, either as, e.g., a smartphone, mobile phone or desktop application. In contrast to hardware tokens, the software tokens are more easily copyable. Software tokens, especially smartphone applications, can itself be protected by a password or biometric factor. Software tokens have the advantage of using device APIs such as push notifications. Some of them even allow an »authentication by push notifications«method, where a user needs to tap on an incoming push notification to confirm the authentication. Other software tokens do not generate an OTP but instead allow the user to approve or deny the authentication request. Software tokens were available before the smartphone era, too, by using, e.g., Java MIDlets for regular mobile phones that were capable of using the Wireless Application Protocol (WAP).[102]

4.6 Universal Second Factor

The Universal Second Factor (U2F) is the second open standard developed by the FIDO alliance before the Web Authentication API. It explicitly defines a second factor for the password-based login flow. As the UAF, it is backed by public-key cryptography, too. The main contributors are Google and Yubico, both being alliance members. The *strong second factor* can be either connected or disconnected, e.g., built-in hardware or for instance, a USB token, NFC-capable device, or a standalone BLE dongle. Besides that, the U2F protocol only specifies USB-human interface device (HID) devices (internal or external), NFC, Bluetooth, and the low energy variant BLE, as possible transport protocols. The protocol defines two layers:[103]

1. the first layer defines the cryptographic basics of the protocol
2. the second layer defines the communication between the user's authenticator and the first layer over the chosen transport protocol (such as USB, NFC, or BLE)

[102] See HS17, p. 717; See MS14, p. 111; See ULC19, p. 60; See DRN17, pp. 222–223; See HJT07, p. 3.
[103] See Sri+17, p. 4; See BBL17, p. 4.

The U2F protocol relies on a web browser that is U2F-capable, a web server that supports U2F protocol, and the authenticator called the U2F token. Two different operations are defined by the specification the *registration* and *authentication*. Authentication is performed by generating a signature. A notable difference to the UAF protocol is the absence of a de-registration request. The message frame defined by the standard is based on the ISO standard for smartcards (ISO-7816) application protocol data unit (APDU).[104]

Because U2F relies on the web in contrast to the UAF protocol, web browser support has to be taken into account. Due to the fact that U2F is superseded by FIDO2, the web browser support of U2F is not of interest to this thesis. However, it is essential to notice that the U2F was never standardized and remains an experimental API with less web browser support.[105]

Moreover, U2F has been renamed to Client-to-Authenticator Protocol (CTAP)-1 since the release of FIDO2 to avoid confusion and questions whether U2F as the CTAP can be used for the Web Authentication API.[106]

Registration

A requirement of the registration process of a U2F token is that the user already is registered on the RP, the web server. The registration process is similar to the introduced process of the UAF protocol and also displayed in Figure 4.5. At first, the server generates a *challenge* for the client and sends it along with the *username* and its *AppID* to the client, in this case the web browser. The payload also contains the desired *version* of the U2F protocol and the already *registered keys*, if any. The client can verify that the AppID matches the origin it is communicating with.[107]

Further, the challenge parameters are constructed by hashing *client data*, i.e., the challenge, AppID, and *typ*. The typ always has the value *navigator.id.finishEnrollment* for a registration process. This data, along with the hash of the AppID, is sent to the U2F token. After that, the token optionally verifies the presence of the user and generates a new key-pair over the NIST elliptic-curve (EC) P-256 and stores it with the username in its database. The token sends the *registration data* consisting of the public key, i.e., an uncompressed point on an EC and the key handle, which can be

[104]See RKM16, p. 3; See BEL17, p. 3.
[105]See Sch19b, p. 31.
[106]See Bra+19, p. 4.
[107]See BBL17, pp. 4–5; See Lan+17, p. 431.

wrapped, (encrypted), back to the client. Additionally, the attestation certificate of the token and an Elliptic Curve Direct Anonymous Attestation (ECDSA) signature over the hashed AppID, hashed challenge, key handle, and public key are sent back to the client.[108]

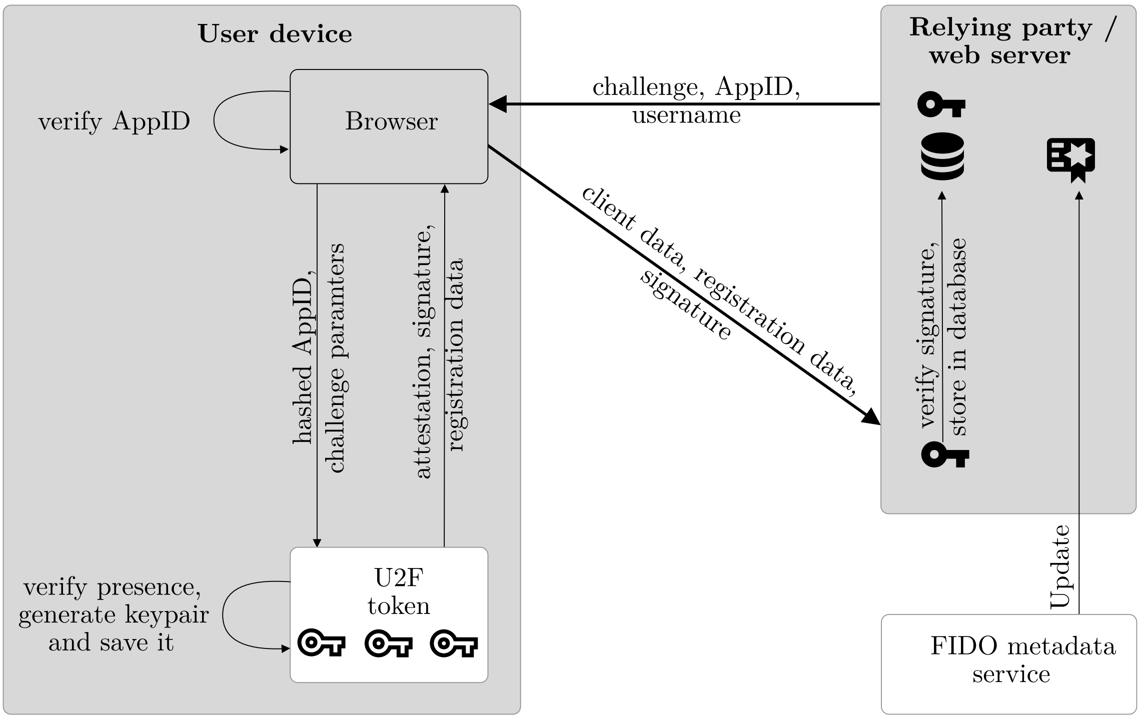

Figure 4.5: U2F registration process[109]

Finally, the client forwards the registration and client data to the RP, which can cryptographically verify the data. The RP can verify the sent signature with the associated public key to check the provided challenge. Additionally, the RP can test the correctness of the token with the public key of the attestation certificate, which is provided from the FIDO metadata service.[110]

The following Listing 4.1 shows the high-level JavaScript (JS) API registration process.

[108] See BEL17, pp. 4–5; See PRW18, p. 70.
[109] Source: diagram by author, based on PRW18, p. 69; Lan+17, p. 428.
[110] See RKM16, p. 3.

Listing 4.1: Example U2F registration request

```
const registerRequest = {
  challenge: 'Wings2019', // usually a random string
  version: 'U2F_V2' // where V2 refers to protocol version 1.2
  appId: 'https://timbrust.de'
};

const registeredKeys = [];

u2f.register('https://timbrust.de', [registerRequest],
↪  registeredKeys, (response) => {
  console.log(response)
});
```

The challenge value in the *registerRequest* usually is a random challenge and base64 encoded, but for demonstration purposes, a plain text string is used instead. In a real-world scenario, the *registerRequest* object is generated by the RP and sent to the client, and the *u2f* JS object is called by the client. Passing in a list of already registered keys with the RP avoids the duplicate registration of a user with the RP.[111]

The received response is displayed in Listing 4.2 and contains the already explained *registrationData* and *clientData*, both being the base64 encoded. For better readability, the strings clientData and registrationData are trimmed. Also, the clientData is decoded to show which data it contains. The client always returns an *errorCode* OK (0) indicating successful registration. Other error codes include a bad request (2), unsupported configuration (3), ineligible device (4), timeout (5), or an other error (1). Further, the clientData consists of *typ*, as well as the challenge of the RP and the origin.[112]

[111]See BBL17, p. 3; See Lan+17, p. 430.
[112]See BBL17, p. 7; See BEL17, p. 8.

Listing 4.2: Example U2F registration response

```
const response = {
  clientData: 'eyJjaGFsbGVuZ2UiOiJXaW5nczIwMTkiLCJvcmlnaW4i
  ↪  [...]', // further data is omitted for readability
  errorCode: 0,
  registrationData: '...', // omitted for readability
  version: 'U2F_V2'
};

// btoa() decoded clientData yields
const decodedClientData = {
  challenge: 'Wings2019',
  origin: 'https://timbrust.de',
  typ: 'navigator.id.finishEnrollment'
};
```

Authentication

The authentication process involves signing a challenge from RP with the corresponding private key. This enables the RP to verify the response with the saved public key cryptographically. Figure 4.6 shows the procedure, too. The RP begins by sending a random *challenge*, the *key handle* associated with the user, and the *AppID* to the client. As in the registration phase, the client can compare the received AppID to the origin it communicates with and verifies that no phishing attempt is in progress.[113]

Afterward, the challenge parameters are constructed by hashing the challenge, AppID, and *typ*. The typ is always set to *navigator.id.getAssertion* for an authentication. This data is sent along with the hashed AppID to the U2F token. There is no difference in this process compared to the registration.[114]

Upon reception, the U2F token verifies the presence of the user and retrieves the stored key-pair associated with the key handle. It sends the counter, that is increased

[113] See RKM16, p. 3; See BBL17, p. 6.
[114] See BEL17, p. 6.

by each usage, and the ECDSA signature over the values user presence, counter, challenge parameters and the hashed AppID back to the client.[115]

The client forwards the signature data, client data, and key handle to the RP, which in return can verify the signature data with the stored public key.[116]

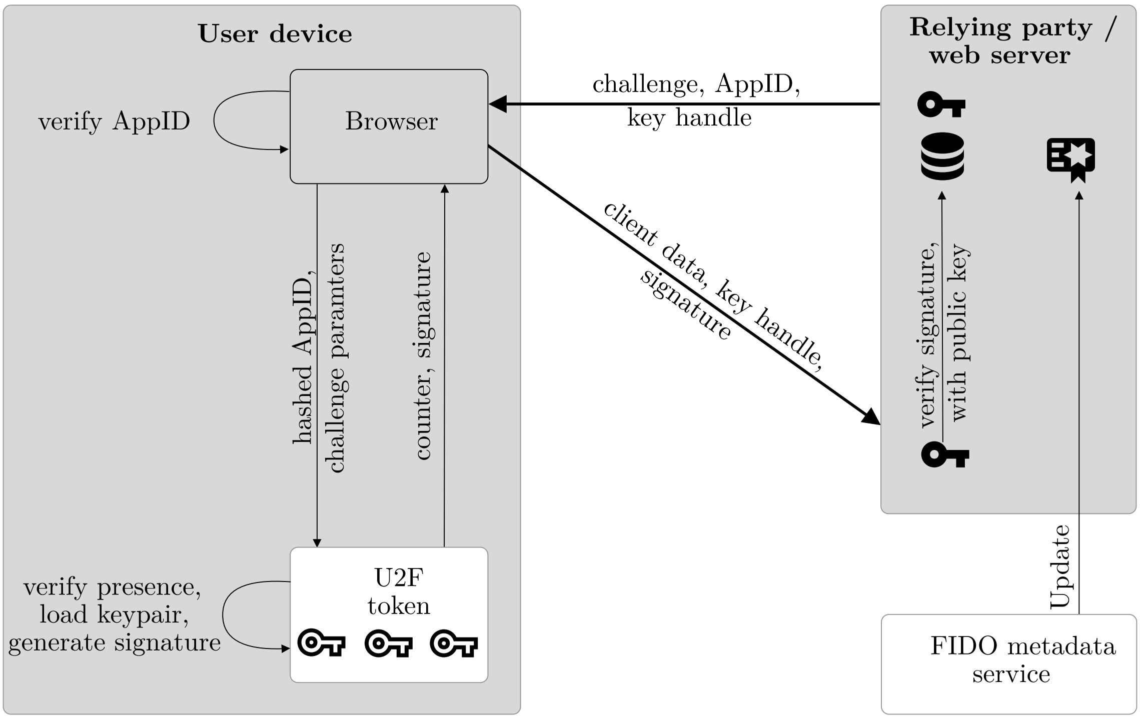

Figure 4.6: U2F authentication process[117]

Listing 4.3 shows an example of a high-level JS API signing process. The RP sends the associated key handle, the protocol version, AppID, and the authentication challenge to the client. Listing 4.4 shows the generated response by the U2F token. The response object and decoded client data are also shown in the listing. The clientData is also be generated beforehand by the client and identical to the decodedClientData object.[118]

The client forwards the response to the RP. It consists of the client data, not its hash, the error status, as well as the key handle and the signature data, signed with the private key.

[115]Lan+17, p. 431; See BEL17, p. 7.
[116]See LM16, p. 118.
[117]Source: diagram by author, based on PRW18, p. 70; Lan+17, p. 428.
[118]See BBL17, p. 3.

Listing 4.3: Example U2F authentication request

```
const registeredKey = {
  keyHandle: '_WFf5BJ1dwtSCFzfWHoqKUhc9M3Hi0Tv58LAtPz0qM6B3A
  ↪  [...]', // further data is omitted for readability
  version: 'U2F_V2'
};

u2f.sign('https://timbrust.de', 'Wings2019Auth',
↪  [registeredKey], (response) => {
    console.log(response)
  }
);
```

Listing 4.4: Example U2F authentication response

```
const response = {
  clientData: 'eyJjaGFsbGVuZ2UiOiJXaW5nczIwMTlBdXRoIiwib3JpZ2H
  ↪  [...]', // further data is omitted for readability
  errorCode: 0,
  keyHandle: 'WFf5BJ1dwtSCFzfWHoqKUhc9M3Hi0Tv58LAtPz0qM6B3A-iT
  ↪  [...]', // further data is omitted for readability
  signatureData: 'AQAAAhIwRQIhAK7xli8pV2cc8TKTOYMcdiz-ZuNVes
  ↪  [...]', // further data is omitted for readability
};

// btoa() decoded clientData yields
const decodedClientData = {
  challenge: 'Wings2019Auth',
  origin: 'https://timbrust.de',
  typ: 'navigator.id.getAssertion'
};
```

5 Security of Multi-Factor Authentication

5.1 Introduction

This chapter analyses the introduced MFA solutions in regards to their security aspects, ranging from algorithms and transportation threats, past vulnerabilities to implementation pitfalls.

Figure 5.1 on the next page shows the already in Figure 4.3 explained MFA flow using a TOTP. In this figure, the scenario is expanded by a phishing attack. The user visits a phishing copy of the website they want to use and does not notice this. They enter their password which is intercepted by an attacker and then forwarded to the legitimate service. Because the user knows that this site is using MFA, they provide the TOTP to the phishing site, too.

This allows the interceptor to steal both the password (knowledge as the first factor) and TOTP (possession as the second factor). In turn, an attacker can now successfully log in to the victim's account and effectively bypassing the MFA solution. This scenario does not explicitly exploit a vulnerability, but instead use the conceptual weakness of a MFA solution, since it is not phishing resistant.

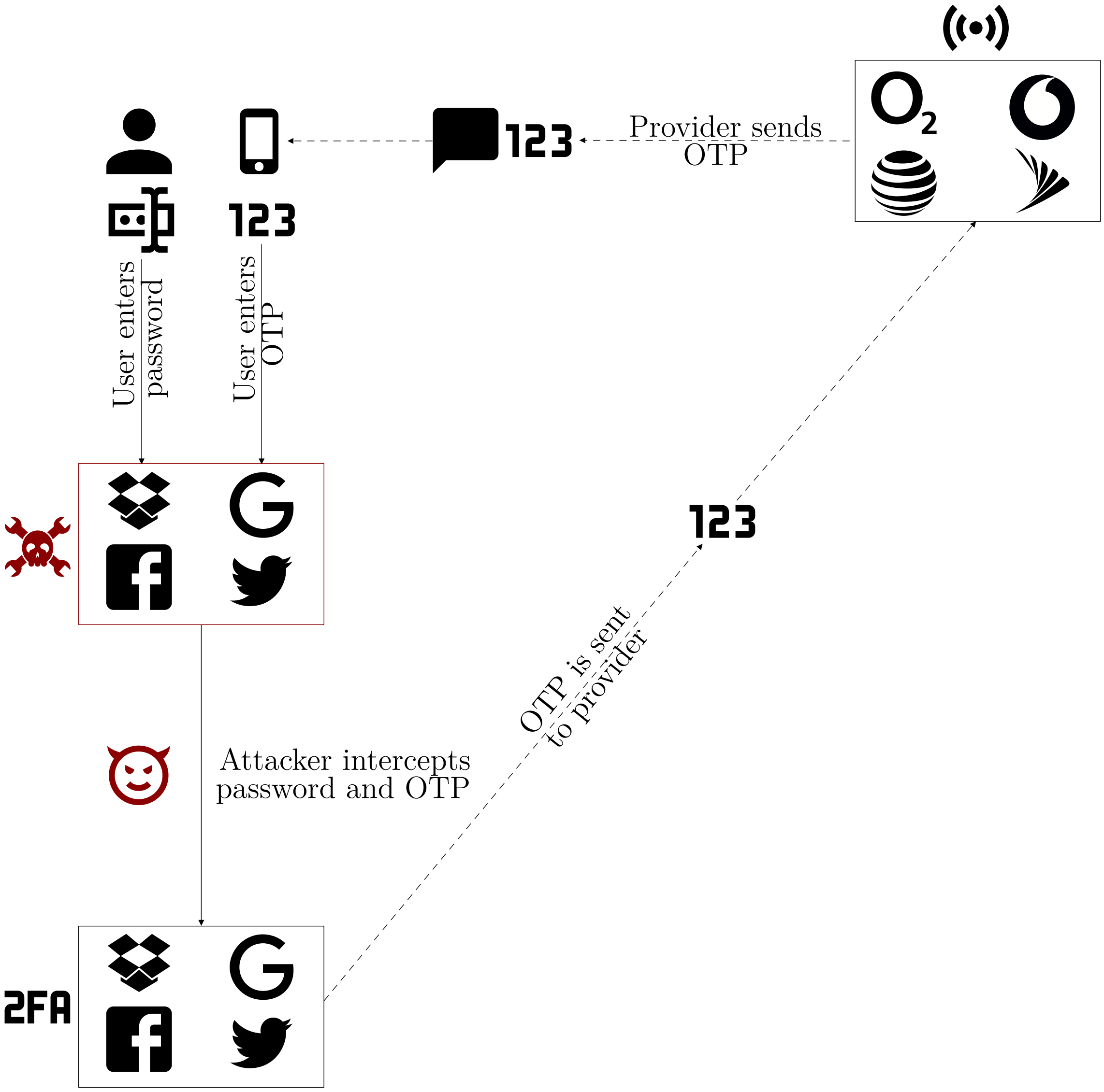

Figure 5.1: Exemplary phishing of an OTP with 2FA enabled[119]

5.2 One-Time Passwords

In this section, the security of both HOTP and TOTP is being analyzed by taking their specifications into account, mainly the underlying hash functions and synchronization features. However, also the generation and transmission of the OTP, which is not part of the specification are considered.

[119]Source: diagram by author

5.2.1 Algorithm

As both the HOTP and the TOTP are based on the HMAC specification, the underlying algorithm needs to be evaluated first. The vital factor is the chosen cryptographic hash algorithm. Mostly SHA-1 is used since it is the default defined in the RFC.[120]

Given that both SHA-1 and MD5 are considered insecure, one has to ask if they are still considered secure in the OTP context. Because of the truncation and limited character set, the collision resistance of the chosen cryptographic hash algorithm is not of importance for the security of the OTP generation. Therefore, using the MD5 and SHA-1 algorithms do not expose a threat. Besides, both the RFC and the BSI still list these algorithms as secure for HMAC after a consideration of the collision attacks.[121]

It is more important to implement the algorithm correctly than replacing the used hash function. In the past, e.g., Google did not issue OTP values with a leading zero. Besides that, the defined minimum length of the OTP values is six digits. Meanwhile, the RFC supports up to ten. However, nearly no service provider uses more than six digits. This decreases the OTP entropy and strengthens the brute-force attack.[122]

Further, for example, the online gaming platform Steam uses a different alphabet and character length. These divergences show that not all implementing parties follow the recommendations of the RFC. Moreover, the user cannot verify that the algorithms are correctly implemented. Because HOTP and TOTP rely on a shared secret, it is crucial that both the server and client store the secret in a secure manner.[123]

A theoretical vulnerability is to use the look-ahead window feature. It enables an attacker to use a token that is much longer valid than it should be. The larger the look-ahead window period is, the bigger the time-frame an attack has brute-force, or phish is, too. Also, it is essential that the OTP is invalidated after a successful use or when the time in the look-ahead window has passed.[124]

[120] See MRa+11, p. 3.
[121] See Ste+17; See Inf19a, p. 18; See TC11, p. 2; See Eck14, p. 395.
[122] See Dmi+14a, p. 369.
[123] See Xia18, pp. 6–7; See MRa+05, pp. 11–13.
[124] See Dmi+14a, p. 369; See MRa+05, p. 11.

In addition, the OTPs are subject to a brute-force attack. The server must throttle the number of tries a user can make to counter this attack.[125]

The main threat, however, remains the vulnerability to phishing because RFCs do not specify any requirements or recommendations on how to verify the origin a user communicates with. The phishing threat is examined in more depth in the following subsection 5.2.2.

5.2.2 Transportation and Generation

Given that the fact that the weaknesses of both SHA-1 and MD5 do not expose a threat to the use of OTP. Instead, the transmission and secure generation of the OTPs pose a challenge. This section considers the transportation mediums SMS, e-mail, and the threats in regards to a generation by smartphone apps.

SMS

The most significant advantage of SMS as a transportation medium is every mobile, ranging from an old Nokia 3310 to a new iPhone 11 Pro, is capable of receiving SMS. All major mobile phone OSs come with an SMS application pre-installed, so no external apps are required. The first SMS was sent in 1994, and while the SMS traffic is decreasing, there were 9 billion messages sent solely in Germany in 2018.[126]

Although there are some significant advantages with SMS transportation, such as the easy to use factor and the fact that the user does not need to know the OTP secret, it also comes with many downsides. Besides the cost aspect of SMS traffic, both for the sender and potentially for the receiver due to roaming fees, the current state of SMS traffic is considered insecure.[127]

The SMS traffic relies on the Signalling System No. 7 (SS7) network, which was developed in the 1970s. It has multiple security flaws that allows an attacker to eavesdrop or modify the in- and out-coming traffic.[128]

Figure 5.2 shows the described MFA flow of Figure 4.3 using TOTP. In this scenario, the attacker is able to phish the TOTP designated for the user. The figure shows that

[125]See MRa+11, p. 6; See MRa+05, pp. 21–22; See SM18, p. 240.
[126]See Alp12, pp. 2–3; See Bun19, p. 57.
[127]See Isl+18, p. 167.
[128]See Wel17, pp. 17–18; See HO17, pp. 3–4; See Puz17, pp. 40, 46.

the attacker uses an exploit in the SS7 network. This allows them to intercept all incoming SMSs. With, e.g., a phished password, the attacker bypasses the enabled 2FA without the user's knowledge.

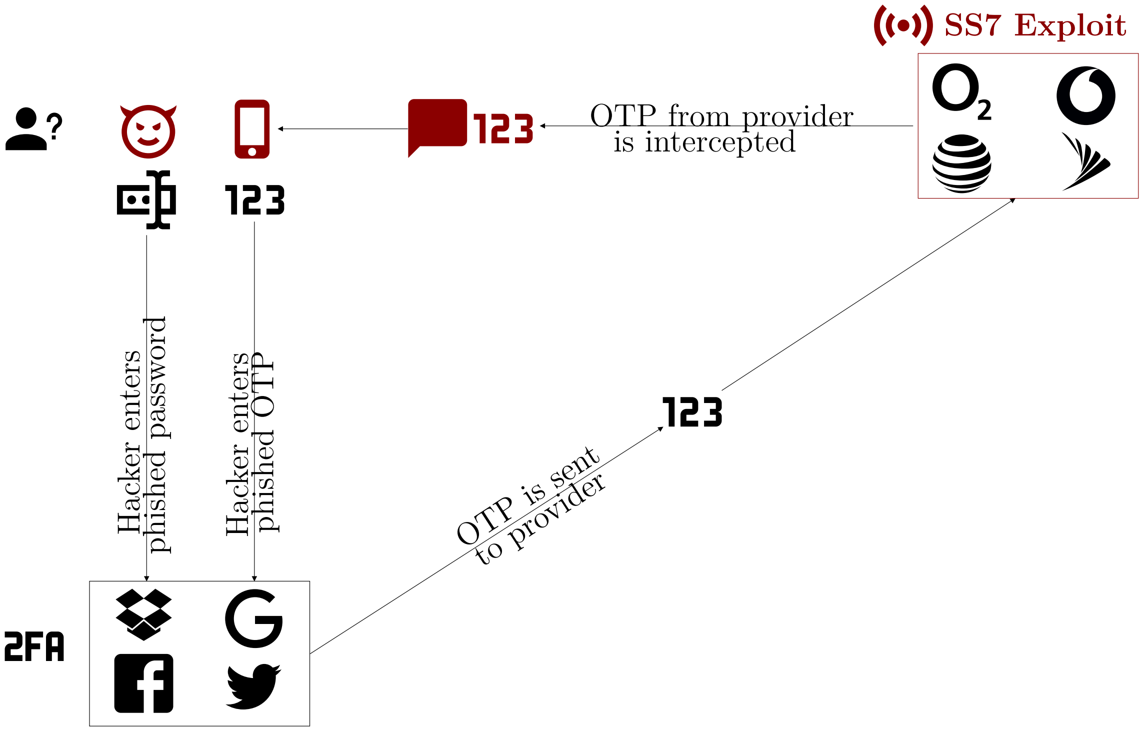

Figure 5.2: SS7 exploit to phish an OTP used in MFA[129]

Another negative aspect of SMS transportation is routing. Many companies rely on third-party providers to send SMS to the user. These providers are using countries where SMS are very cheap, or route them the cheapest way, but on the other hand, the SS7 security measures such as SMS home routing and not enforced. This results in a higher security risk that the SMS is compromised while delivered to the user. Also, third party providers are given access to the OTP, which enables the risk of a malicious insider.[130]

In contrast to the web and e-mail, the user is not aware of phishing attacks in the SMS context. However, studies show that a new technique called the Verification Code Forwarding Attack (VCFA) is already in use. In this scenario, the attacker sends the victim a (spoofed) SMS and impersonates the service provider. They tell the user that, e.g., fraudulent access was detected and in order to block this attempt, the user needs to reply with the OTP code for security measure.[131]

[129] Source: diagram by author
[130] See Mul+13, p. 153; See Cer18, pp. 4, 9, 12.
[131] See Jak18, pp. 6–7; See Sia+17, pp. 4–5.

Figure 5.3 shows an example of a VCFA. An attack logs in to the user's account with, e.g., hacked or phished credentials. Because MFA protects the account, the user receives verification code via SMS or smartphone. The attacker now sends a fake SMS to the victim, stating that the service has detected unusual activity. In order to block this attempt and prove that the user is the legitimate account owner, they should reply with the just received verification code. Of course, the attacker now has access to the OTP, too, since they convinced the user into forwarding their code.[132]

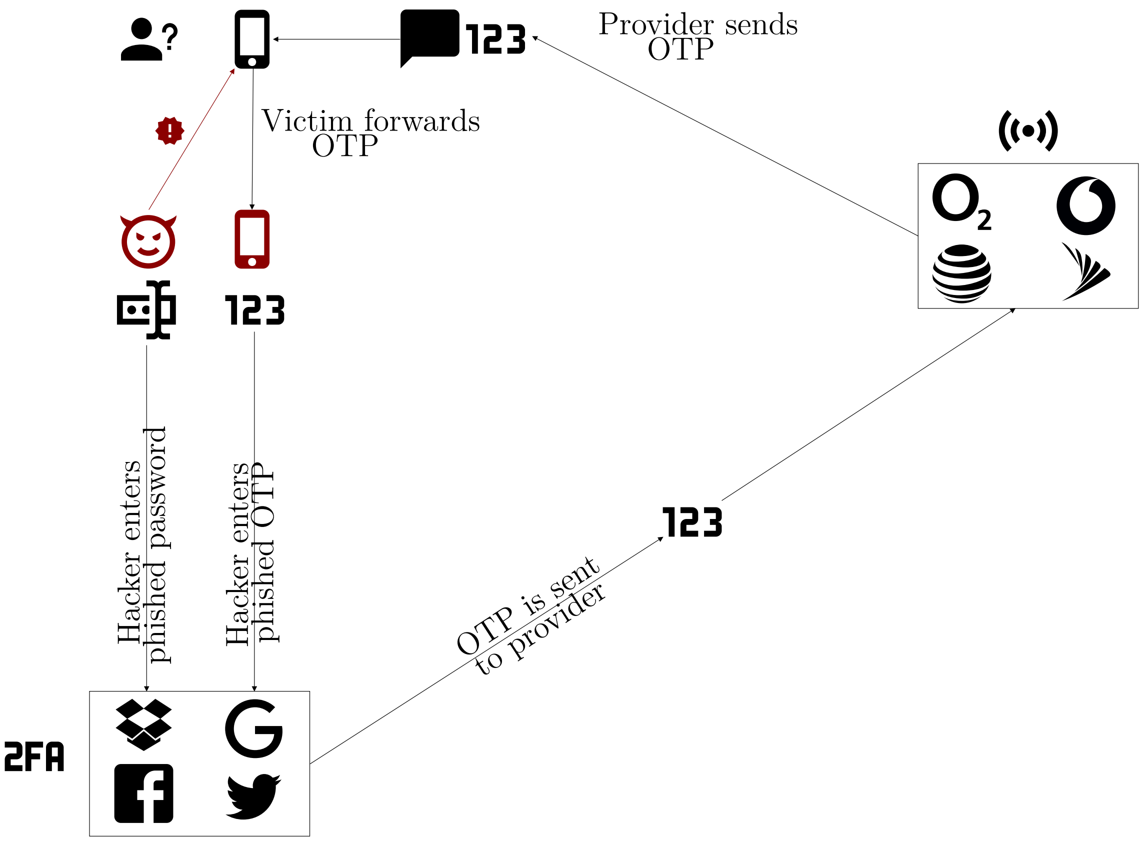

Figure 5.3: VCFA to phish an OTP used in MFA[133]

Another threat is specialized malware for mobile phones. Especially for Android, there exist multiple SMS trojans that are capable of intercepting the SMS, too. Some of them even target specific banking apps. Additionally, apps disguise as a useful application or are a repackaged legitimate app with a backdoor.[134]

[132] See SNM16, p. 66.
[133] Source: diagram by author
[134] See Dmi+14b, pp. 146–149; See Mul+13, pp. 152–154; See HDK17, p. 114; See Ste19.

Moreover, social engineering attacks that target the mobile service operator are an attack vector. Figure 5.4 shows a MFA flow using TOTP, but in this case, with another phishing scenario that targets the service provider. An attacker has again access to the user's password, e.g., from a previous, successful phishing attack. In order to obtain or phish, respectively, they target the human weakness in the cell phone provider of the user. They successfully convince them to activate another SIM card for the victim's phone number and receive the SMS with the TOTP, too, which enables the attacker to complete the MFA flow successfully. This type of attack is also called SIM swap scam or fraud. In the recent past, Twitter's CEO was a victim of such an attack that lead to an account takeover.[135]

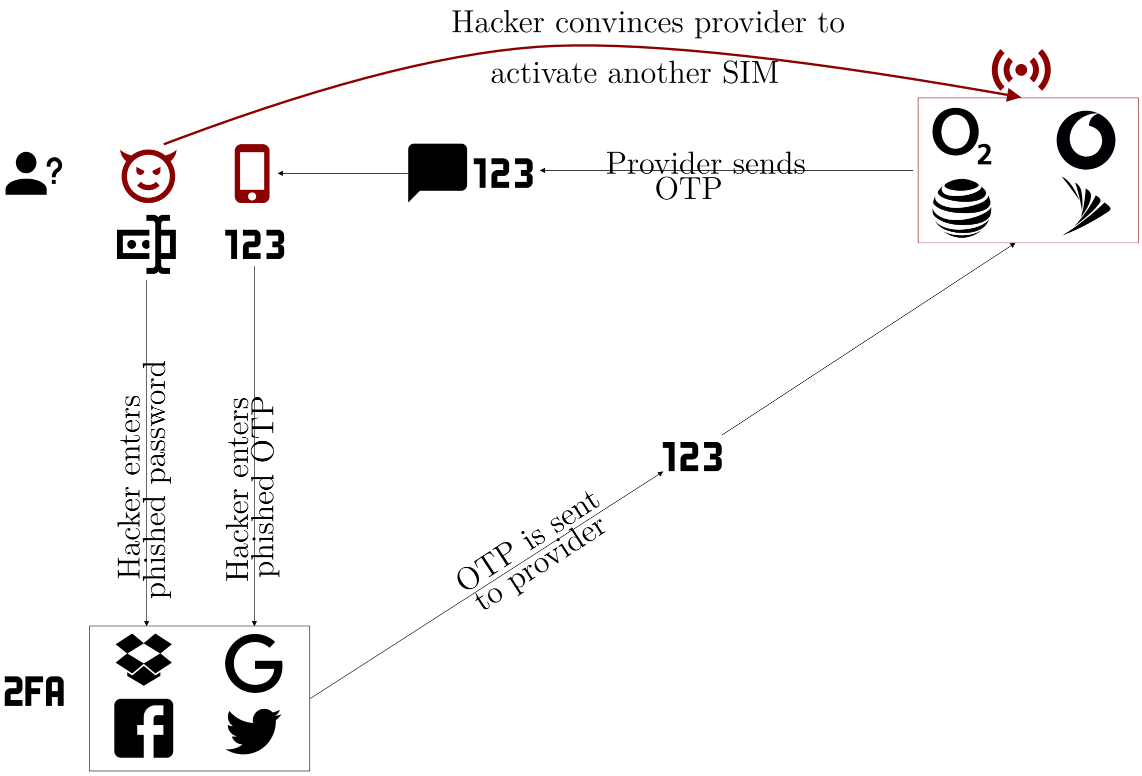

Figure 5.4: Social engineering used to phish an OTP in MFA[136]

Another variant that is technically more complex, but feasible, is the SIM card cloning. This allows the attacker to intercept the TOTP, too. Even if the phone number cannot be registered twice, it still enables the eavesdropping.[137]

Given all these facts, SMS transportation should be avoided for the usage with OTPs, since there are multiple flaws in the SS7 network itself and the process of

[135] See Bla17, p. 19; See Con19; See JS11, pp. 74–76.
[136] Source: diagram by author
[137] See Eck14, p. 873; See He07, pp. 11–12.

how the SMS reaches the user. It is also not resistant against phishing or mobile phone trojans. Both the BSI and the NIST advise service providers not to use SMS for the transmission of OTPs and TANs anymore.[138]

Further, it cannot be guaranteed that the user has a working mobile network, that the registered mobile phone number is still active, or that the user receives the SMS in time. These non-influenceable, external factors strengthen the fact that SMS is not a secure and reliable choice for the transportation medium of OTPs.[139]

App

In contrast to the transportation of the OTP via SMS, using a standalone app such as Google Authenticator, Authy, or even the mobile OTP app for Java-based phones offer some advantages. While the user has to be connected to the cellular network when receiving the OTP via SMS, the app solution works in offline or bad network connectivity use cases, too.[140]

Furthermore, the app solution is cheaper because no transaction fee has to be paid by the sender or receiver. It also solves the roaming problem. On the other hand, the app needs to be maintained and updated to protect against, e.g., vulnerabilities in third-party libraries and to ensure compatibility with future devices and OS versions.

The setup of the OTP is not phishing resistant either. A malware on, e.g., the desktop can intercept the shared secret, or the mobile phone can contain malware. This malware can intercept, for instance, the camera, or later access the generated OTP, or access the app's database where the secrets are stored.[141]

Figure 5.5 shows the scenario where an attacker successfully infected a smartphone of a user with mobile malware. The hacker now has the possibility to access either the OTP secret or to forward the secret when a user opens the app. This enables an attacker to bypass 2FA if, e.g., beforehand, the password was phished or obtained from another data breach.

[138] See Jak18, p. 8; See Inf19b, p. 27; See Gra+17, p. 19; See KVB17, p. 407.
[139] See EAK11, p. 327.
[140] See UY14, p. 228.
[141] See Dmi+14a, p. 371; See KVB17, p. 407.

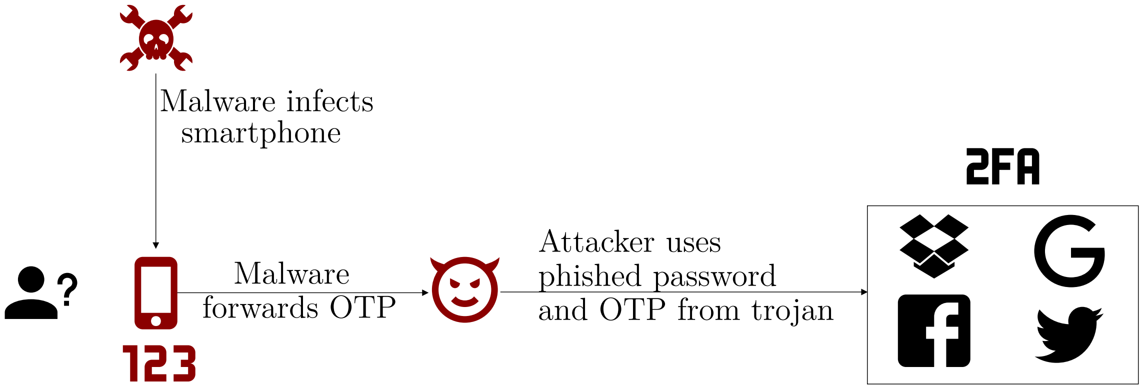

Figure 5.5: Mobile malware used to phish an OTP in MFA[142]

Besides that, the app itself can contain vulnerabilities. For instance, the 2FA app Authy suffered from a vulnerable backend that could be exploited to bypass the 2FA. Additionally, the user has to ask if the app is legitimate and from a trustworthy source and needs to have faith in the app that it keeps their data safe.[143]

E-Mail

Another widely used form to transmit the OTP from the server to the user is the distribution via e-mail. E-mails are by comparison with apps more accepted by the users and do not require a user to provide their cellphone number, but rather only the e-mail address that is mostly provided to the service nevertheless.

However, e-mail traffic comes with threats, too. In the first place, unencrypted e-mail traffic can be intercepted by a MITM, therefore exposing the OTP. For example, in August 2019, only 90% of Google Mail's outgoing traffic was encrypted. Malware on the desktop or smartphone can intercept incoming e-mail and even delete the message without the users' knowledge, the Figure 5.5 applies here, too, with the variation that the OTP is not generated on the device but instead only received.[144]

Besides that, e-mails re-introduce the problem of delayed reception of the OTP. Especially techniques such as grey-listing delay incoming messages to avoid spam. Examples such as network connectivity problems, dedicated attack (distributed denial of service (DDoS)) or exceeded e-mail storage quota further increase the potential of unreliable OTP transportation.[145]

[142] Source: diagram by author
[143] See Ste19; See Hom15.
[144] See Gooa.
[145] See KC12.

Unfortunately, e-mails are not phishing resistant, either. The same threats of SMS apply, too. A malware, both on the desktop or mobile, can intercept and forward a received OTP to an attacker.

5.3 Smartcards and Security Tokens

Smartcard and physical security tokens face common security threats. As both are authentication by possession, they are at risk of being stolen, damaged, lost, or rendered inoperable in any other way. Especially security tokens that are carried around on a keychain are exposed to the threat of being left on the desk and therefore being accessible for other people.

Further, physical tokens and smartcard are at risk of being cloned or disassembled in order to gain access to the underlying chip. Also, malicious applets for Java-enabled smartcards can try to exploit software vulnerabilities.[146]

The first generation of RSA SecurID contained a vulnerable algorithm that led to a successful adaptively chosen-plaintext attack. Even though RSA replaced the algorithm in their security tokens, in March 2011, an intruder successfully managed to gain access to RSA's internal seed and serial number database that lead to a replacement of over 40 million RSA SecurID tokens.[147]

Besides vulnerabilities in the companies server, both Google's Titan Key and YubiKeys suffered from vulnerable firmware. Due to the token design, it is not possible to update the firmware, and the only security mitigation remains the replacement of affected devices.[148]

In order to attack the chips inside smartcards and security tokens, a common attack is side-channel attacks such as a differential power analysis (DPA). YubiKeys were successfully attacked, and their private AES could be extracted. Therefore, an attacker was able to gain access to the Yubico OTP generation. Additionally, an attacker can target the EEPROM of a smartcard, e.g., by trying to freeze and copy the values.[149]
Moreover, given the fact that a smartcard operates as a USB-HID and some RSA SecurID tokens contain a USB port for smartcard compatibility, a malware might be able to intercept the USB communication.

[146] See Wit02, pp. 14–16.
[147] See Eck14, p. 480; See BLP05, p. 369; See HF15, p. 8.
[148] See Wes19b; See Bra19a.
[149] See ORP13, pp. 210, 212, 219; See And08, pp. 502–503, 509.

As software tokens mainly generate OTPs and the generation by an application already has been discussed, the reader is referred to page 54.

5.4 Universal Second Factor

The security of the U2F protocol extends the security threats of physical security tokens and smartcards that were introduced in the previous section, for example, loss or theft.[150]

In contrast to OTP, the U2F protocol is phishing resistant. This resistance is achieved by binding the token, registration, and authentication to a specific origin. Given the fact U2F token compares the origin, phishing sites that target, e.g., typos in the Uniform Resource Locator (URL) do not expose a threat.

However, the U2F protocol is not resistant to malware. A malware that controls the USB ports of a computer can communicate with the token, too.[151]

When using the TLS ChannelID, also called token binding, a possible extension of the U2F protocol, the user is protected against MITM attacks. Token binding enables mutual authentication in TLS by using cryptographic certificates on both sides. It has to be noted though, that using the TLS ChannelID is defined as optional in the specification.[152]

Security reachers were able to find a potential vulnerability in the optional key wrapping of the key material if the counters are not correctly increased and checked. This attack could lead to a successful cloning attack. The security token vendors and the FIDO alliance since have adjusted the specification and recommendations. Further, researches were able to control the token via the WebUSB API by emulating the NFC capabilities of a YubiKey to use it as a smartcard.[153]

Albeit it is not a vulnerability, the lack of a display on the authenticator can render a threat. An attacker might be able to change the transaction a user signs with the test of user presence. Further, the absence of a display might confuse a user if a registration or authentication was successful. Also, malware is able to continually request the touch of a button to complete a MFA authentication flow with, e.g., phished or keylogged credentials. Due to the lack of a display, the chances are high

[150] See Lin17b, pp. 12–13.
[151] See JK18, pp. 10–1; See Sri+17, p. 9.
[152] See Sri+17, pp. 6–7.
[153] See Kan19, p. 3.

that a user accidentally confirms a fraudulent authentication from the malware. Additionally, the user might not notice it and presses the button again due to the lack of feedback from the token.[154]

In theory, an attacker could manufacture a backdoor variant of the U2F token, which either leaks or sends all the private keys to the attacker. This attack has a low probability compared to the other attack vectors, though.[155]

Moreover, the centralized managed metadata service for device attestation can expose a threat. Given an attacker takes over this server, they can either enable counterfeit security tokens to become legitimate attested tokens. On the other hand, legitimate tokens can be flagged as banned or compromised. Besides that, the service is subject to a DDoS attack that prevents RPs from updating their metadata database. Since vendors offer their attestation metadata independently, mitigation is to implement an independent metadata service.[156]

5.5 Overall Comparison of Threats

The following section sums the introduce methods of authentication and analyzed MFA solutions up and shows their key threats.

Table 5.1 shows the introduced authentication methods grouped by the known authentication methods knowledge, possession, and biometrics. It shows that primary authentication by possession is used as an additional authentication factor, given the fact that passwords are the de-facto standard on the internet as the first factor. Biometrics can be used, for instance, with the UAF, but in practice there exist very few applications that use biometrics as an additional factor for internet-based login flows.

Further, it shows that no authentication method is free of vulnerabilities, even when combined as 2FA or MFA. The most present vulnerability is the missing phishing resistance alongside the threat of interception followed by physical theft. The table shows that every introduced method of OTPs is subject to phishing attacks and that the only phishing resistant solution is U2F. Unfortunately, hardware tokens itself are often not protected.

[154]See Lan+17, p. 434; See DDC18, p. 15; See Ree+19, pp. 1518–1519; See Rey+18, p. 884.
[155]See Dau+18, p. 3.
[156]See Pow18; See Cha+17, p. 17.
[157]Sources: table based on analysis from previous chapters and additionally from Gra+17, pp. 41–45.

	Authentication	MFA	Threats
Knowledge	Passwords	-	Phishing, sharing, guessing, brute-force, theft, replay attacks, interception, theft (e.g., written down passwords), social engineering
	PINs	-	
	Security/Recovery questions	-	
Possession	Hardware OTPs	✓	Theft of the device, phishing, interception, replay attacks, brute-force, damage, oblivion, loss
	App OTPs	✓	Theft of the device, phishing, interception, replay attacks, brute-force
	SMS OTPs	✓	Theft of the device, phishing, interception, replay attacks, brute-force, unavailability
	E-Mail OTPs	✓	Theft of the device (in case of mobile phones), interception, phishing, brute-force, unavailability
	Smartcards	✓	Cloning, theft, damage, oblivion, loss, side-channel attacks, phishing (in case of OTP generation
	Security Tokens	✓	Cloning, theft, damage, oblivion, loss, side-channel attacks, phishing (in case of OTP generation
	U2F	✓	Cloning, theft, damage, oblivion, loss, side-channel attacks
Biometrics	Fingerprints	(✓)	Replica, forgery, replay attacks, injuries, unavailability of the sensor
	Facial scan	(✓)	
	Iris scan	(✓)	

Table 5.1: Overall comparison of threats[157]

Independently of the used MFA solution, the service provider must require the usage of MFA for sensitive transactions, such as the change of the user's password, the de-activation of 2FA or the initiation of an account recovery process. Failing to do so enables an account takeover if an attacker can successfully perform, for example, a session hijacking because no additional confirmation is necessary to de-activate the MFA. Alternatively, if for instance, an attacker controls a victim's e-mail account, they can reset any password for every account that is registered with the e-mail if the service provider does not enforce 2FA for this operation.[158]

[158]See Dmi+14a, p. 370.

6 Introduction to the Web Authentication API

6.1 Goal of the Web Authentication API

The goal of the Web Authentication API is to enable »the creation and use of strong, attested, scoped, public key-based credentials by web applications, for the purpose of strongly authenticating users«.[159] Each public key credential is scoped to the relying party (RP), i.e., cannot be re-used for other websites (RPs). The authenticator has the duties and responsibilities to create, store, and access these credentials. These actions always require users' consent. The user agent, i.e., web browser performs the communication with the authenticators and RPs to preserve the users' privacy. Attestation ensures that each operation from an alleged authenticator is legitimate and cryptographically verifiable. The primary use cases for the Web Authentication API are passwordless registrations and logins, but also to provide a second-factor or to sign specific transactions.[160]

6.2 History and Evolution

The Web Authentication API is an outcome of joint efforts between the FIDO alliance and the World Wide Web Consortium (W3C). It is an outcome from preceding industry standards, namely Universal Authentication Framework (UAF) and Universal Second Factor (U2F). This chapter introduces the Web Authentication API with a focus on the technical implementations, protocols, and used techniques.[161]

The first specification version of the U2F was the starting point for the development of the Web Authentication API in joint efforts with the W3C. The CTAP is based on the U2F specification version 1.2, which complements the Web Authentication API. Both projects are part of the FIDO2 project.[162]

[159] See Bal+19, Abstract.
[160] See Bal+19, Abstract, Chapter 1.2.
[161] See Eik19, p. 24.
[162] See Gri17, pp. 169–170.

6.3 Technical Implementation and Details

6.3.1 FIDO2

As already briefly introduced in subsection 2.7.1, the FIDO2 project is a joint effort of the W3C and the FIDO alliance. It consists of the JS standard, the Web Authentication API, and the Client-to-Authenticator Protocol (CTAP). The Web Authentication API is standardized and managed by the W3C, while the CTAP is authored by the FIDO alliance and standardized in ITU recommendation ITU-T X.1278. However, the FIDO alliance also initially developed the Web Authentication API under the name FIDO 2.0 before officially handing it over to the W3C.[163]

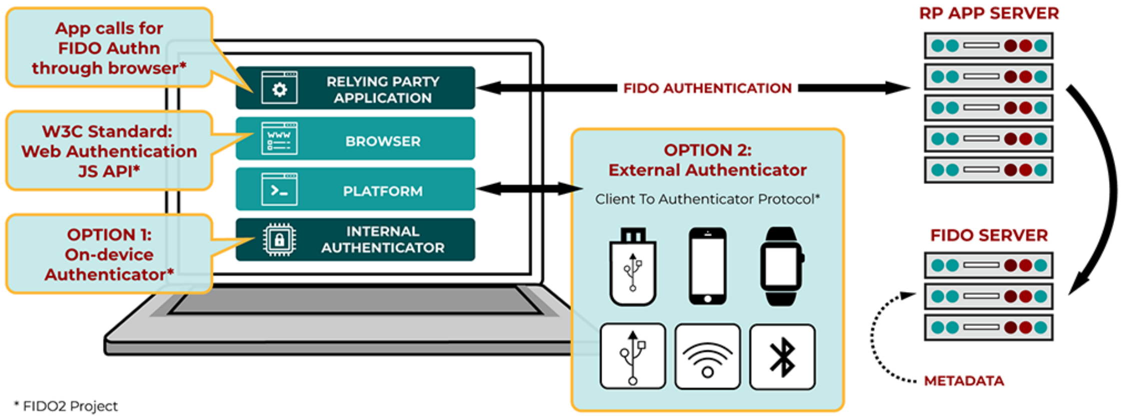

Figure 6.1: FIDO2 architecture overview[164]

Figure 6.1 shows the overview of the FIDO2 project. A noteworthy change in contrast to the U2F specification is the possibility to use either a *roaming*, i.e., an external authenticator or an authenticator that is built into the device or platform, respectively.

6.3.2 Client to Authenticator Protocol 2

The Client-to-Authenticator Protocol (CTAP) 2 is based on the U2F protocol version 1.2 and defines three parts:

1. the authenticator API
2. message encoding

[163]See SM18, p. 254; See GH18, p. 3; See ITU18b.
[164]Source: https://fidoalliance.org/specifications/, last accessed on 09/14/2019

3. transport-specific bindings

The key methods of the authenticator API are explained in more detail below. Message encoding describes the process of encoding the corresponding message in a binary form called Concise Binary Object Representation (CBOR). This binary form is suitable for, e.g., the transport over BLE, because plain text strings and JavaScript Object Notation (JSON) objects can be too big for a means of transport over low data rate protocols. The transport-specific bindings define the required transformation and bindings in order to comply with the transport protocol specifications.[165]

An essential difference between CTAP2 and the preceding standard U2F is the fact that CTAP2 describes only the communication between the client, i.e., web browser and the authenticator, as opposed to U2F where the standard also defines the JS API in order to communicate with the authenticator. Figure 6.2 shows this architectural difference.[166]

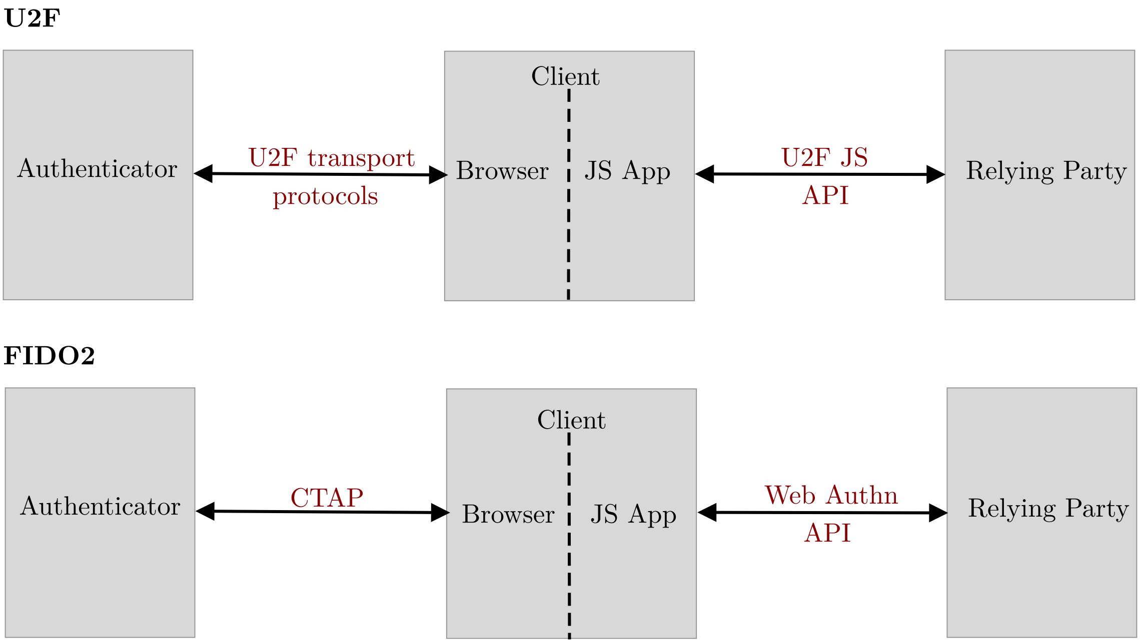

Figure 6.2: Architectural differences between U2F and CTAP2[167]

[165] See Bra+19, pp. 4–5.
[166] See Ngu15, p. 51; See SM18, p. 254.
[167] Source: diagram by author, based on Sri+17, p. 4; Bal+19, Chapter 6.

Registration

The registration procedure invokes the method *authenticatorMakeCredential*. The input parameters are identical to the ones defined in the higher-level Web Authentication API and are further explained in subsection 6.3.3. Upon reception of the required data, the authenticator first checks if the *excludeList* contains a credential ID that is already registered with the authenticator. This prevents a user from registering multiple accounts for the same RP. If the user verification or presence option is passed, the authenticator has to ensure a legitimate user is present. Upon successful user verification, the authenticator generates a new credential key-pair for the specified algorithm.[168]

After that, the authenticator generates the attestation object. It consists of the authentication data, which contains the hash of the RP ID, a counter, flags if the user has been verified, and the public key with its unique credential ID. Besides that, the authenticator also sends the attestation statement, if required. This statement can, for example, be issued by the TPM, Android Key attestation, or be generated with the private attestation key of the authenticator token.[169]

Authentication and Transaction Confirmation

Authentication is performed by using the method *authenticatorGetAssertion* of the CTAP, where the higher-level Web Authentication API defines the input parameters. The identifier of the RP is sent to the authenticator and optionally a list of public keys the authenticator is allowed to retrieve. After optional user verification and presence detection, the authenticator displays the data to the user if it has a display to do so. When these checks succeeded, the authenticator accesses the corresponding credential.[170]

The authenticator generates an assertion signature over the received hash of the client data and the authenticator data. The authenticator data consist of the hashed RP ID, the flags for user presence and verification, a counter, and the attested credential data. The attested credential data comprises the Authenticator Attestation Globally Unique ID (AAGUID), credential ID, and the credential public key.[171]

[168] See Bra+19, p. 9.
[169] See Bra+19, p. 9; See Bal+19, Chapter 8.
[170] See Bra+19, pp. 11–13.
[171] See Bal+19, Chapter 6.4.1.

Factory Reset

The CTAP defines a method to completely factory reset the authenticator in order to de-register every user account and key material stored on it. This is similar to the UAF de-registration process, but contrary to the U2F specification which lacks a possibility to do so. To avoid the accidental deletion of all user accounts, the protocol specifies that the authenticator may ask for user confirmation. However, it is not possible to delete a specific user account and key-pair.[172]

6.3.3 Web Authentication API

The API defined in the Web Authentication API is actually an extension of the Credential Management API. This is another API in development by the W3C, but currently in a draft state and not a recommendation yet.[173] The Credential Management API defines the *navigator.credentials* property with the *create* and *get* methods. Its goal is to offer an API for programmatically accessing the user agent's password storage capabilities. The Web Authentication API is adding further method overloads to support public-key based credentials, too.[174]

Authentication and registration in the context of the Web Authentication API are a particular form of network protocols, called a *ceremony*. A ceremony describes the concept of extending a network protocol to include human nodes, too. This allows the specification to take the human factor into account, too.[175]

The Web Authentication API is backward compatible with the U2F protocol, thus making every security token that is usable for U2F compatible with the Web Authentication API. However, a severe restriction of the legacy U2F protocol in usage with FIDO2 is that it is only usable as a second factor and not for passwordless logins.[176]

Registration

A new key-pair for the registration with an RP is generated when the client invokes the asynchronous method *navigator.credentials.create* with an object that contains

[172] See Bra+19, p. 26.
[173] The W3C standardization process can be viewed in the Section A.1
[174] See Bal+19, Chapter 1; See Wes19a, Chapter 1.1.
[175] See Ell07, p. 2.
[176] See Bal+19, Chapter 2.2.1, 6.1.2.

the required *publicKey* property. The publicKey object contains the ID, name of the RP, and the user information consisting of a username, display name, and a unique ID. Given the fact that, e.g., the authenticator stores the ID value, it should not consist of any information that can be linked to the user. Further, the publicKey object contains a random challenge generated by the server to prevent replay attacks.[177]

Besides that, with the public key credential parameters array (*pubKeyCredParams*), it is possible for the RP to define the desired algorithm that should be used for the key-pair generation. The algorithm (*alg*) IDs are obtained from the Internet Assigned Numbers Authority (IANA) registry of CBOR Object Signing and Encryption (COSE) algorithms. The ID -7 expands to ECDSA with SHA-256, while -257 links to the RSA algorithm in conjunction with SHA-256. The order in the array describes the preferred algorithm, but also accepted fallback algorithms.[178]

Additionally, the RP can define a list of credentials (*excludeCredentials*) that need to be check if they exist, e.g., prevent the user from creating multiple key-pairs for the same RP. Furthermore, a *timeout* value can be specified in which the operation should succeed or fail.[179]

Moreover, the RP can set the *authenticatorSelection* property which defines the requirement if a user needs to be verified (possible values are *required*, *preferred*, *discouraged*), the option if the credentials need to be stored on the authenticator (*requireResidentKey*). Further, the RP can specify the authenticator attachment modality, i.e., if the authenticator should be platform-specific or a cross-platform (roaming) authenticator.[180]

Finally, the *attestation* property is of importance. The RP can specify either a *direct*, *indirect*, or *none* attestation. None means that the RP is not interested in the attestation of the authenticator at all. A direct attestation requires a signed attestation statement generated by the authenticator to verify its authenticity. In contrast, an indirect attestation leaves the authenticator in charge of how to generate the attestation certificate. The authenticator may use a per-origin certificate authority (CA) to protect the users' privacy or implement and use the ECDSA.[181]

[177] See Bal+19, Chapter 5.1.3.
[178] See Bal+19, Chapter 5.3, 11.3.
[179] See Bal+19, Chapter 5.4.
[180] See Bal+19, Chapter 6.2.1.
[181] See Bal+19, Chapter 5.4.6.

Listing 6.1: Exemplary Web Authentication API registration request

```
const publicKeyOptions = {
  challenge: 'Wings2019', // normally a random string from the
   ↪  server in binary form (Uint8Array)
  rp: {
    name: 'Web Authn Test',
    id: 'https://timbrust.de'
  },
  user: {
    id: 'C0E3F2BFCFA8179F', // usually in binary form
     ↪  (Uint8Array)
    name: 'me@timbrust.de',
    displayName: 'tim'
  },
  pubKeyCredParams: [{ alg: -7, type: 'public-key' }],
  authenticatorSelection: {
    authenticatorAttachment: 'cross-platform'
  },
  timeout: 600,
  attestation: 'none'
};

const credential = await navigator.credentials.create({
  publicKey: publicKeyOptions
});
```

Listing 6.1 shows an example payload for the registration of a new credential with the Web Authentication API consisting of the previously introduced parameters. The *publicKeyOptions* payload generated and sent by the RP can be passed to the authenticator by calling the CTAP *authenticatorMakeCredential* method. The client passes the user and RP to the authenticator. Further, it is evaluated if the authenticator should verify the user or if a check of user presence is sufficient. For this evaluation, the property *userVerification* is used. Besides that, the list of credentials to exclude, the public key credential parameters, and the hash of the client data is provided to the authenticator. The client data comprises the server provided challenge, its origin, and the string type *webauthn.create*.[182]

Upon reception of the *attestationObject* from the authenticator, the client can generate the credential to be returned to the RP, as shown in Listing 6.2.

Listing 6.2: Web Authentication API registration response

```
const credential = {
  id: 'BShOCQ2c32dv4aqyy3oWmcu_9s4tz0VIob81U5tg [...]',
  rawId: ArrayBuffer(59),
  response: {
    clientDataJSON: ArrayBuffer(121),
    attestationObject: ArrayBuffer(306)
  },
  type: 'public-key'
};
```

The created *credential* object consists of an *ID*, both as a string and binary representation, the *type* that is always set to »public-key« and the *response* object. The response object is constructed from the returned *attestationObject* and the client data, which is also shown in Listing 6.3.[183]

Listing 6.3: Web Authentication API registration client data

```
const clientDataJSON = {
  challenge: 'Wings2019',
  origin: 'https://timbrust.de',
  type: 'webauthn.create'
};
```

Listing 6.3 shows the decoded client data from the Web Authentication API registration response from Listing 6.1. The data contains the *challenge* sent by the RP and the *origin* of the RP. Each registration is flagged with the *type* of »webauthn.create«.[184]

[182] See Bal+19, Chapter 5.4, 6.4.2.
[183] See Bal+19, Chapter 5.1.
[184] See Bal+19, Chapter 5.10.1.

In contrast, the *attestationObject* sent from the authenticator contains more properties and is shown in Listing 6.4 on the next page. On the first hierarchy level, it contains the attestation statement format identifier (*fmt*), such as »packed, tpm, or fido-u2f«, the attestation statement (*attmStmt*), and the authentication data (*authData*). The authentication data includes the evaluated user flags, for instance, if the user was present or verified, a signature counter, if supported by the authenticator, and the hash of the RP ID. In detail, the attested credential data of the authentication data contains the public key ID, the public key itself, e.g., the point on an EC and the AAGUID if attestation is not set to none. Otherwise ,it is set to 16 zero bytes.[185]

[185] See Bal+19, Chapter 5.1.3, 6.1.

Listing 6.4: Web Authentication API registration attestation

```
const attestationObject = {
  fmt: 'fido-u2f',
  attStmt: {
    sig: '[...]',
    x5c: []
  },
  authData: {
    rpIdHash: '068a7ad7f858dadbf691af6f2f7ca86d4dee5a080b
    ↪  [...]',
    flags: {
      userPresent: true,
      reserved1: false,
      userVerified: false,
      reserved2: '0',
      attestedCredentialData: true,
      extensionDataIncluded: false
    },
    signCount: 0,
    attestedCredentialData: {
      aaguid: '0000000000000000',
      credentialIdLength: 96,
      credentialId: ArrayBuffer(59), // identical to
      ↪  publicKeyCredential.id
      credentialPublicKey: {
        kty: 'EC',
        alg: 'ECDSA_w_SHA256',
        crv: 'P-256',
        x: 'xHxgcBFgJolQ5lvukADki+cUzTPcmk50tfjOYGH3nYE=',
        y: 'W1OKIxfc6pIE/ANeTD7MqnNVjBXdOL7We9xZ3Hx6nD8='
      }
    }
  }
};
```

Authentication

An authentication ceremony is started by calling the client's asynchronous method *navigator.credentials.get*. As in the registration procedure, the RP needs to generate a *publicKeyOptions* object. For the authentication, a random challenge, a timeout value, and the allowed credentials array are required. The RP can send a list of associated credentials that are suitable for the user assertion. The RP may provide an *rpId* and flag for *user verification*, too. If omitted the origin of the RP and default of preferred user authentication is used instead. Each allowed credential is identified by its ID and an optional array of transports the client is allowed to perform to retrieve the credential. Listing 6.5 on the next page shows the example payload for the assertion.[186]

Subsequently, the user agents generate the *client data* consisting of the origin, challenge, and the type that is always set to *webauthn.get*. It passes the hash of the client data, the ID of RP, user presence and verification flags, and list of allowed credentials to the authenticator by calling its *authenticatorGetAssertion* method.[187]

[186]See Bal+19, Chapter 5.1.4., 5.5, 5.10.3.
[187]See Bal+19, Chapter 6.3.3.

Listing 6.5: Exemplary Web Authentication API authentication request

```
const publicKeyOptions = {
  challenge: 'Wings2019Auth', // normally a random string from
  ↪ the server in binary form (Uint8Array)
  allowCredentials: [
    {
      id: 'BShOCQ2c32dv4aqyy3oWmcu_9s4tzOVIob81U5tg [...]',
      type: 'public-key',
      transports: ['usb', 'ble', 'nfc']
    }
  ],
  timeout: 6000
};

const assertion = await navigator.credentials.get({
  publicKey: publicKeyOptions
});
```

Upon reception of a response from the authenticator, the client can generate the response, i.e., *assertion* object. Listing 6.6 on the next page shows the response that can be sent to the RP. The response is similar to a registration response and also contains the credential ID in binary and string representation. Further, the client data is returned to the RP, too. The authenticator data is equal to the registration procedure. The most important property is the returned *signature* value over the authenticator data and the client data hash. The RP can cryptographically verify the signature with the corresponding public key of the user.[188]

[188]See Bal+19, Chapter 5.1.4.1, 5.2.2, 6.3.3.

Listing 6.6: Web Authentication API authentication response

```
const assertion = {
  id: 'BShOCQ2c32dv4aqyy3oWmcu_9s4tz0VIob81U5tg [...]',
  rawId: ArrayBuffer(59),
  response: {
    authenticatorData: ArrayBuffer(191),
    signature: ArrayBuffer(59),
    clientDataJSON: {
      challenge: 'Wings2019 Auth',
      origin: 'https://timbrust.de',
      type: 'webauthn.get'
    }
  },
  type: 'public-key'
};
```

6.3.4 Web Browser Support

Table 6.1 shows the web browser support status of the Web Authentication API, both for desktop and mobile web browsers, and if they support the API. If so, the table shows the version which initially added support for the Web Authentication API alongside the release date. The following subsections will explain the web browser support more detailed.

The global web browser support as of August 2019 adds up around 70%, given the fact that web browser usage statistics vary between services.[189]

[189] The obtained data is available in the Appendix Section A.2, see Staa.

	Web browser	Supported	Version	Release Date
Desktop	Chrome	✓	67	May 2018
	Firefox	✓	60	May 2018
	Opera	✓	54	June 2018
	Internet Explorer	✗	-	-
	Edge	✓	18	November 2018
	Safari	✓	13	September 2019
Mobile	Opera Mobile	✗	-	-
	IE Mobile	✗	-	-
	Safari (iOS)	✗	-	-
	Google Chrome (iOS)	✗	-	-
	Firefox (iOS)	✗	-	-
	Brave (iOS)	✓	1.11.3	August 2019
Android	LineageOS Stock Browser	✗	-	-
	Chrome for Android	✓	70	October 2018
	Firefox for Android (Fennec)	✓	68	July 2019
	Firefox Preview (Fenix)	✗	-	
	Opera	✗	-	-
	Opera mini	✗	-	-
	Edge	✗	-	-
	Samsung Internet	✗	-	-
	UC Browser	✗	-	-
	Mint Browser	✗	-	-
	360 Secure Browser	✗	-	-
	QQ Browser	✗	-	-
	Yandex Browser	✗	-	-
	Brave Browser	✗	-	-

Table 6.1: Web browser support of the Web Authentication API[190]

Desktop Support

The Web Authentication API is supported from Chrome 67 onwards, which was released in May 2018. Firefox added support for the Web Authentication API in May 2018 with its version 60 as well.

Microsoft added support for the Web Authentication API in Edge 13 which was released in November 2015. However, the implementation is based on an earlier draft version of the Web Authentication API. Support for the FIDO 2.0 specification was added in Edge 14 (released in December 2016). The feature is hidden behind a

[190]Sources: BK18; JT18; Dav18; Ger18; Jon19, a detailed analysis of Android web browsers is available on the USB flash drive in the appendix.

configuration option though and was enabled for all users with the release of Edge 17 in November 2018.[191]

Web browsers such as Opera, Vivaldi, Brave, and upcoming Edge versions that are all based on Chromium, the web browser and source code behind Google's Chrome web browser, have support for the Web Authentication API, too.[192]

As the development for the Internet Explorer halted, and it is only receiving security updates, no support is available for new web APIs including the Web Authentication API. Even though it is still used by over 4% of all desktop web browser users and remains supported for the operating system Windows 7, 8.1 and 10. This is an important fact to take into account when evaluating the usability of the Web Authentication API since especially enterprise users often cannot upgrade or switch their web browser.[193]

Safari added support for the Web Authentication API feature in December 2018 but only for the preview variant of the web browser, called the Safari Technology Preview. On September 20 Safari 13 was released for the OS versions macOS High Sierra and Mojave. The support is limited to USB-HID enabled authenticators though.[194]

Besides that, Windows 10 also added support for MFA by incorporating the technology described in the FIDO2 standard. This allows biometric authentication with, e.g., fingerprints when a reader is available or to use the facial recognition technology or iris scans. The feature is called »Windows Hello«. Credentials are only stored locally and are protected by asymmetric encryption. Besides biometric authentication Windows Hello also supports PINs. The TPM stores this PIN. Windows Hello can be used in desktop web browsers, i.e., delegating the Web Authentication API functionality to the OS.[195]

Mobile Support

The support for the Web Authentication API in mobile web browsers is inferior to desktop support. While Chrome for Android supports the Web Authentication API since October 2018 and Firefox since July 2019, stock iOS completely lacks support

[191] See Jac19, p. 112.
[192] See Kis19, Chapter 7.1.
[193] See Sup19c; See Stab.
[194] See Dav18; See Sch19c.
[195] See Bio16; See JK, p. 6.

for the Web Authentication API. Even though in the iOS 13 settings the feature can be enabled in the »Experimental Features« section, the API remains unsupported or at least there is no way to add an authenticator in the web browser yet.[196]

The only ray of hope is that the Brave web browser for iOS added support for the security key »YubiKey 5Ci« which enables the Web Authentication API for iOS by using an Apple certified Lightning accessory. Figure 6.3 shows a successful authentication with the YubiKey 5Ci and the Brave web browser for iOS. iPad devices with a USB-C connector currently do not work yet. Further, the YubiKey 5Ci is not recognized in the Safari web browser, too.[197]

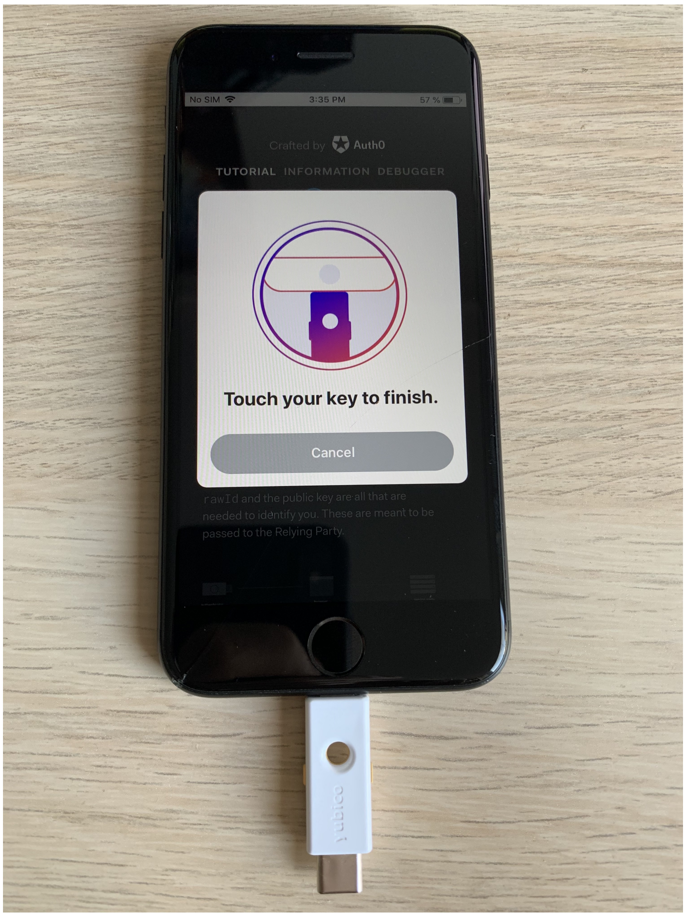

Figure 6.3: Successful use of the Web Authentication API with the Brave web browser on an iPhone 7 with the YubiKey 5Ci[198]

However, Figure 6.4 shows the try to use an existing Security Key by Yubico with a lightning dongle in the Brave web browser. While the token has power, Brave does

[196]See Ger18; Jon19, See.
[197]See Bra19b; See Bra19c; See Mah19.
[198]Source: author's own photograph

not recognize it. Neither is it usable. Safari does not show an overlay for the key usage either.

Figure 6.4: Failed try to use the Web Authentication API with the Brave web browser on an iPhone 6[199]

It has to be noted though, that other Android web browser vendors need to implement the functionality themselves. Other geographic regions use a variety of different web browsers, e.g., the UC Browser, 360 Security Browser, Mint Browser from Xiaomi, or the QQ Browser from Tencent. Neither they, nor web browsers such as Samsung Internet, Opera (mini) for Android, Edge, or the Android Stock web browser are currently supporting the Web Authentication API. The current Firefox for Android (codenamed »Fennec«) web browser is based on Chromium, too, in contrast to the desktop web browser which is powered by Mozilla's own web browser engine Gecko. A new Firefox for Android web browser, currently called Firefox Preview (codename »Fenix«), which uses a mobile compatible version of Gecko, lacks support for the Web Authentication API, too. However, Android offers support for FIDO2 as an API, and the OS itself is FIDO certified.[200]

Other mobile OSs, for example, Windows Phone 8, BlackBerry OS, BlackBerry 10 or KaiOS do not support the Web Authentication API. Further live demonstration

[199]Source: author's own photograph
[200]See Eik19, p. 24.

captures are available on the attached USB flash drive in the appendix.

6.3.5 Usability

One of the main goals of the Web Authentication API is the »it just works« feeling, by providing a secure but abstract solution for the end user. The chosen web browser and OS are responsible for the design of the login and registration windows, often being native overlays, while in contrast, the website designs the traditional login masks and forms. In order to maintain high usability, the user should be able to use a variety of tokens, e.g., built-in key stores protected by biometrics or an external token that uses BLE, NFC, or a USB-A or USB-C interface. Unfortunately, the »it just works« cannot be fulfilled on macOS or iOS yet.

While the desktop variant of Safari at least contains support for the Web Authentication API, the CTAP is only implemented for USB-HID based tokens. Unfortunately no indication, for instance, an overlay or popup, that indicates the user needs to interact with their authenticator is shown in Safari. Additionally, Firefox only supports USB-HID based authenticators on other operating systems than Windows 10.[201]

Besides that, an external token that contains a vulnerability in its firmware often needs to be replaced, making this both a massive usability loss, as well as increasing security risk when not replacing the affected token. Both Google's Titan Key and YubiKey's were affected in the past and needed to be replaced.[202]

Further, external tokens are exposed to the environment and suffer from the same problems, such as the regular tokens do.[203]

An additional usage implication is the recommendation to have at least two registered tokens for each RP. In case one token is lost, stolen, damaged or in any other inaccessible the user still possesses a backup token to gain access to their accounts. Moreover, the Web Authentication API does not specify a way to backup registered credentials. Unfortunately, the different USB interfaces and wireless transportations are not supported on all devices and OS, which further decreases the usability and interoperability.[204]

[201]See Moz.
[202]See Wes19b; See Bra19a.
[203]See Section 5.3
[204]See Bal+19, Chapter 13.6; See DDC18, p. 15.

Also, the fact that different certification levels of security tokens by the FIDO alliance exist does not make it easier for the end user to pick the right security token. Technically inexperienced users may not know the difference between a U2F and FIDO2 certified product, even though only the latter is capable of a complete passwordless process. Additionally, end users may not be technically experienced enough to understand the differences in the certification levels of the FIDO alliance.[205]

Finally, different studies showed that users struggle to enable MFA with roaming tokens due to a lack of feedback and guidance from both the RP and the web browsers. The built-in platform authenticator might be able to change this challenge.[206]

6.4 Security Aspects

The following sections build upon the security of MFA, especially security tokens and U2F by further concentrating on the specifics of the Web Authentication API and the architectural changes opposite to the U2F protocol.

6.4.1 Problems

The problems that are transferred to the Web Authentication API are the ones of authentication by possession already described in Section 3.3 and further specified for security keys in Section 5.3 and U2F in Section 5.4. If the Web Authentication API is used with a physical security key, then the same threats of damage, loss, or theft exist. Besides that, if the security key itself is not protected (by, e.g., fingerprints), an attacker can gain access to an account if he steals or copies the authenticator. Built-in key stores in devices, such as smartphones or laptops, do not protect against theft, either. Furthermore, the roaming authenticators are subject to physical attacks, in particular, side-channel attacks such as a DPA.

Security-wise the Web Authentication API has received little attention yet. A first security analysis showed some weaknesses. These are the following ones, described more detailed below. It has to be noted though, that these security considerations are only from one source and outlined only theoretical attack vectors. In contrast, the security of the Web Authentication API was formally verified, and findings only resulted in privacy concerns, and no security issues were found.[207]

205 See FIa.
206 See Rey+18; See Ley19, p. 884; See DDC18, p. 15.
207 See Sta18; See GH18, p. 9.

The first problem of the Web Authentication API is the registered COSE algorithms defined in section 11.3 of the specification. Support for RSASSA-Public Key Cryptography Standards (PKCS)#1 v1.5 is explicitly required, making it vulnerable for the over twenty years known »Bleichenbacher attack«.[208]

Further, the Elliptic Curve Digital Signature Algorithm (ECDAA) does not specifically point compression. This can lead to invalid curve attacks, where an attacker can send a chosen point that is assumed to be on the elliptic curve. However, if the point is not on the curve, it can lead to the leakage of the private key. Random values for the secret key in ECDSA expose a threat, too. It is recommended to use determinist nonces instead.

A further critique is the choice of the EC. The specification of the Web Authentication API defines two curves. Both are Barreto-Naehrig (BN) curves. Recent attacks showed that they do not provide enough protection against the elliptic curve discrete logarithm problem (ECDLP) and suffer from a reduced amount of bits for security.[209]

Additionally, the usage of the random number generator (RNG) is not further specified. Weak implementations might use the *standard* RNG and not a suitable cryptographically secure pseudo-random number generator (CSPRNG) for the ECDAA. Moreover, the attestation can be criticized because the specification does not require an implementer to use ECDAA but also allow to use of the private attestation key of the security token. In combination with a centralized attestation CA, this can de-crease the user's anonymity and break the principle of the unlinkability between users and RPs.

6.4.2 Mitigations

As mitigations against theoretical security threats, the following changes should be taken into account for future revisions of the Web Authentication API. For example, even when the standard does not specify which RNG has to be used, an implementer of such a cryptographic API should always know the danger of potential lack of randomness when not using a CSPRNG.

The Bleichenbacher attack and padding oracle can be avoided by not implementing the RSA PKCS#1 v1.5 padding requirement, although this is a violation of the

[208] See Ble98; See Bal+19, Chapter 11.3.
[209] See KB16, p. 562.

specification. Additionally, when, for instance, the RP does not implement the API themselves, they have control over the public key credential parameters array (*pubKeyCredParams*) specified in the registration ceremony. Omitting the identifier for the RSA PKCS#1 v1.5 padding ensures that the key material is generated with a different algorithm.

Regarding the reduced security against the ECDLP, the specification needs to be updated. A definition of different ECs can mitigate against an increased risk of a successful attack against the curve.

It has to be highlighted that all these mitigations are based on theoretical vulnerabilities and that all of the security threats can be solved on a protocol level, i.e., by updating the specification. As further versions of the Web Authentication API are in development, these recommendations can be taken into account for updated versions of the specification.

6.5 Comparison with Other Multi-Factor Authentications

Because the Web Authentication API is an evaluation of the U2F protocol, the comparison of the Web Authentication API extends the comparison of U2F to other MFA solutions. The key advantage of the Web Authentication API in contrast to OTPs is the built-in phishing resistance. The specifications of the FIDO2 project set the requirements for the protection against phishing attacks by verifying the RP. This solves one of the biggest threats in other MFA solutions.

In contrast to the U2F protocol, the Web Authentication API enables the possibility to use a platform, i.e., built-in authenticators. This greatly enhances the usability and simplicity for the end user by enabling already learned authentication flows, such as unlocking a device with a fingerprint for the registration and login on the web. Built-in authenticators can, additionally, enable protection by biometrics. This is possible due to prior works of, e.g., the UAF.

However, the current hurdles of the Web Authentication API are the missing interoperability of different transport protocols and the lack of web browser support for Internet Explorer, iOS, and many Android web browsers. In contrast, OTPs work in these web browsers, too, and do not require an update of the web browser. Also, the user is advised to keep a backup authenticator in case of theft or loss of their primary authenticator, but this essentials aspect is not well highlighted and advertised. Nonetheless, this behavior is identical to OTP, where the user has the possibility to save backup codes to prevent an account lockout.[210]

Additionally, the Web Authentication API cannot protect against an account takeover if the system itself is, for example, infected by malware. Malware is still able to steal a user's session, which can lead to further attacks. Out of scope for the Web Authentication API is transportation, which is defined by the CTAP. Malware can access the Bluetooth or USB interfaces and intercept the transmissions. However, the private key material never leaves the authenticator. As long as the secure storage of the authenticator is tamper-resistant, the private keys are not at risk. In contrast, malware for smartphones might be able to steal the secret used to generate OTPs if it is not stored in a secure manner.

An unweighted fact is the current rate of adaption. Since the Web Authentication API was certified in March 2019, very few services, have implemented the API - in comparison with MFA solutions, such as OTPs and RSA SecurID, that are in use for five to ten years.[211]

Regardless, the Web Authentication API has the potential to obsolete the necessity of a second-factor at all by replacing traditional passwords with public-key cryptography that are secure, unique, and phishing resistant for every RP the user registers with. Moreover, the user does not need to worry about remembering passwords anymore because they can use their biometrics to unlock, for instance, a platform authenticator. No other introduced MFA solution has the potential to achieve the same.

[210] See SMJ16, p. 36.
[211] See Yub.

7 Conclusion and Outlook

This thesis starts with a citation that highlighted that passwords are not suitable for secure authentication on the web anymore. To prove this statement, the reader has been introduced into the fundamental methods and concepts of authentication. The knowledge-based method of authentication in the form of passwords is the most common method of authentication on the internet. Further, but less used methods of authentication are, for example, possession and biometrics, but the time or location can also be used to authenticate a user. Additionally, the building blocks for the UAF, the attestation and challenge-response authentication have been explained, followed by the first example of passwordless authentication.

Based thereupon, each of the authentication methods has been analyzed in regard to their security. Chapter 3 showed that knowledge-based authentication exposes significant threats. Since the human brain has difficulties remembering the different passwords, the user re-uses them, chooses simple passwords and variants, or stores them somewhere in plain text. Moreover, service providers hat only hash passwords without salting or peppering. It increases the chances of a successful brute-force or dictionary attack when their database is breached. However, even physical possession is not safe against theft, loss, copies, or damages and therefore cannot be used as a replacement for knowledge-based authentication. Additionally, biometrics is more expensive to use and do not protect against an impersonation, either. Further, the initialization and transmission of authentication are subject to MITM attacks, albeit a secure communications channel is used. This reinforces these that additional security measures need to be employed.

As a countermeasure to protect a user's account even in case of leaked, hacked, or reversed credentials from a hash, different MFA solutions have been introduced. Instead of relying on one factor, a user has to provide two or more distinct factors authentication. An example of an authentication method besides passwords is the OTP. It is an (alpha)numerical password that is only valid once. In order to generate the next password, either based on time or an event, both parties need to possess a shared secret, or the RP calculates the OTP and sends it to the client. However, both the transmission of the OTP and the generation contain potential weaknesses. The transmission via SMS or unencrypted e-mails can be eavesdropped or phished. The SS7 network is not secure enough for the transmission of confidential information. Further, relying on third party providers for the transmission increases the attack surface for malicious insiders, social engineering, or VCFAs. Nonetheless, the generation of the OTP on the client device can be intercepted by malware, too.

Another concept to achieve MFA is the use of the U2F protocol. In contrary to the OTP, this authentication relies on the public-key cryptography that is handled by dedicated security tokens. They provide resistance against phishing since the origin, the client (and token) is communicating with, is verified by the web browser, therefore taking this task off the user's duty. While the security in comparison with OTPs is increased, the usability is worse. Requiring the user to purchase a dedicated piece of hardware, as well as the difference of ports a computer and mobile phone has, and transport protocols hinder the interoperability and ease of use. Further, only a few web browsers support U2F. The outlined security threats the Web Authentication API faces are all of a theoretical nature, as no known and successful exploit of vulnerabilities exists. Future revisions of the specification can further clarify vague sections or introduce new EC for the key material.

As an evolution of the U2F protocol, a central part of this thesis was the introduction of the Web Authentication API. By keeping the same secure concept of public-key cryptography that is abstracted from the user, the phishing resistance is ensured, too. A key difference and advancement are the division into two protocols, the low-level CTAP which defines the communication between the web browser (client) and the token and the high-level JS API, the Web Authentication API. Furthermore, the Web Authentication API does not require that the token an external, i.e., roaming authenticator, but instead, it can be a built-in platform authenticator. This enabled the user to utilize already know and learned techniques, such as using their fingerprint sensor. Likewise, the Web Authentication API allows passwordless registration and authentication, which in contrast, the U2F protocol did not allow. Given the fact that both protocols are standardized by either the W3C and the ITU, the adoption rate of the API is improved because the web browser vendors usually follow the recommendations and implement the functionality in a manageable time frame.

Additionally, the Web Authentication API was still in development and only a W3C draft[212] while finding the topic of this thesis, and the API is only standardized since March 2019. This highlights the relevance of research on this topic. At the beginning of writing this thesis, Safari had only limited, opt-in support for the Web Authentication API, and iOS had no working solution at all. During the development of this work, Yubico in partnership with Brave announced and released a new variant of the YubiKey series to support iOS. Moreover, Apple released the stable release of Safari 13 with support for the Web Authentication API and Firefox for Android added support for the API. This rapid development further strengthens the interest in this topic and the need to replace passwords.

The questions that were defined at the beginning of this thesis in Section 1.2, such as the threats of not using MFA, problems that result from weak and re-used passwords, but also the question of whether MFA can be made secure, have been answered.

Since the thesis focused on the use of MFA solutions on the web, but at the same time delimitated concepts as OpenID Connect, OAuth 2.0 and SSO, further research can be done on these topics. For instance, SSO is an essential factor in enterprises, often used in conjunction with services and protocols such as Lightweight Directory Access Protocol (LDAP) or Active Directory (AD) from Microsoft. It has yet to be researched, whether the Web Authentication API is combinable with SSO providers.

In the consumer section, OAuth plays a relevant aspect as, e.g., Google, Facebook, or Microsoft allow the registration on a website with the account a user has on their site registered. Most recently, Apple announced support for OAuth with the Apple ID, too. A matter to clarify and study in this regard is the competition between the Web Authentication API and OAuth. In particular, the privacy aspect is interesting, because the providing party has the knowledge on which websites the user registered with their account. This allows extensive user tracking.[213]

One the one hand, emerging technologies such as IoT and Industry 4.0 are becoming more and more connected, but on the other hand, they increase the attack surface drastically. An open question for continued research is if the Web Authentication API is a suitable use case for secure and robust authentication which is done without user interaction, but instead by machines and computers autonomously. Further, it has to be evaluated if these, especially low-power computing, devices are capable of implementing both secure channels, such as TLS and calculations on ECs.

[212]The W3C standardization process can be viewed in the Section A.1
[213]See Sch19a, p. 18; See Wal16, p. 4.

All in all, this thesis showed that internet users and their accounts are at risk because passwords are not suited for secure authentication in the way they are currently used. Weak passwords, password re-usage, and data breaches expose a constant threat. It is crucial to sensitize the user that these threats affect each and every one. Moreover, existing MFA solutions are prone to phishing attacks since the way of transmission via SMS and e-mail is insecure and can be eavesdropped. Additionally, they are not resistant to social engineering attacks or malware on a user's device. U2F and in particular the Web Authentication API are the only solutions to protect against phishing. While the Web Authentication API has the potential to (finally) replace passwords, it is not yet usable enough in all web browsers across different OSs. If the browser vendors implement the missing transportation protocols (BLE and NFC) and if Apple adds support on iOS, this matter is solved though. A topic that is not well advertised, nor explained is the advice to use at least two authenticators to possess a backup method. With the help of the web browser vendors and a growing rate of adoption, the Web Authentication API is on the right way towards an internet with strong, secure, and simple authentication, even without passwords.

Bibliography

[Alp12] Jed Alpert. *The Mobile Marketing Revolution: How Your Brand Can Have a One-to-One Conversation with Everyone*. New York, NY, USA: McGraw-Hill Education, Apr. 2012. ISBN: 978-0-07-178818-2.

[And08] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2nd ed. Indianapolis, IN, USA: Wiley, Apr. 2008. ISBN: 978-0-470-06852-6.

[Ang18] Anna Angelogianni. "Analysis and implementation of the FIDO protocol in a trusted environment." Masters's Thesis. Piraeus, Greece: University of Piraeus, July 2018.

[BB17] Lee Brotherston and Amanda Berlin. *Defensive Security Handbook: Best Practices for Securing Infrastructure*. Sebastopol, CA, USA: O'Reilly Media, Apr. 2017. ISBN: 978-1-491-96038-7.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Keying Hash Functions for Message Authentication." In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Aug. 1996, pp. 1–15. ISBN: 978-3-540-68697-2. DOI: 10.1007/3-540-68697-5_1.

[Bio16] "Microsoft Windows Hello biometric login now works with websites." In: *Biometric Technology Today* 2016.4 (Apr. 2016), p. 12. ISSN: 0969-4765. DOI: 10.1016/S0969-4765(16)30071-6.

[Bis18] Matthew Bishop. *Computer Security: Art and Science*. 2nd ed. Boston, MA, USA: Addison-Wesley Professional, Dec. 2018. ISBN: 978-0-13-409714-5.

[BKW14] Seymour Bosworth, Michel E. Kabay, and Eric Whyne. *Computer Security Handbook*. 6th ed. Hoboken, NJ, USA: Wiley, Mar. 2014. ISBN: 978-1-118-12706-3.

[Bla17] Andrew Blaich. "Recognising the real danger of SIM fraud." In: *Computer Fraud & Security* 2017.11 (Nov. 2017), pp. 19–20. ISSN: 1361-3723. DOI: 10.1016/S1361-3723(17)30084-2.

[Ble98] Daniel Bleichenbacher. "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1." In: *Advances in Cryptology — CRYPTO '98*. Ed. by Hugo Krawczyk. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Aug. 1998, pp. 1–12. ISBN: 978-3-540-68462-6. DOI: 10.1007/BFb0055716.

[BLP05] Alex Biryukov, Joseph Lano, and Bart Preneel. “Recent attacks on alleged SecurID and their practical implications.” In: *Computers & Security* 24.5 (Aug. 2005), pp. 364–370. ISSN: 0167-4048. DOI: 10.1016/j.cose.2005.04.006.

[Bon+15] Joseph Bonneau et al. “Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google.” In: *Proceedings of the 24th International Conference on World Wide Web*. WWW ’15. Florence, Italy: International World Wide Web Conferences Steering Committee, May 2015, pp. 141–150. ISBN: 978-1-4503-3469-3. DOI: 10.1145/2736277.2741691.

[Bra+06] John Brainard et al. “Fourth-factor Authentication: Somebody You Know.” In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. Alexandria, VA, USA: ACM, Oct. 2006, pp. 168–178. ISBN: 978-1-59593-518-2. DOI: 10.1145/1180405.1180427.

[Cel+17] Antonio Celesti et al. “Secure Registration and Remote Attestation of IoT Devices Joining the Cloud: The Stack4Things Case of Study.” In: *Security and Privacy in Cyber-Physical Systems*. Ed. by Houbing Song, Glenn A. Fink, and Sabina Jeschke. Hoboken, NJ, USA: Wiley, Oct. 2017. Chap. 7, pp. 137–156. ISBN: 978-1-119-22607-9. DOI: 10.1002/9781119226079.ch7.

[Cer18] Stefan Certic. “Two-Factor Authentication Vulnerabilities.” In: *SSRN Electronic Journal* (Apr. 2018). DOI: 10.2139/ssrn.3158601.

[Cha+17] Donghoon Chang et al. “On Making U2F Protocol Leakage-Resilient via Re-keying.” In: *IACR Cryptology ePrint Archive* (2017).

[Cok+11] George Coker et al. “Principles of Remote Attestation.” In: *International Journal of Information Security* 10.2 (June 2011), pp. 63–81. ISSN: 1615-5262. DOI: 10.1007/s10207-011-0124-7.

[CPS10] Liqun Chen, Dan Page, and Nigel P. Smart. “On the Design and Implementation of an Efficient DAA Scheme.” In: *Smart Card Research and Advanced Application*. Ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny. CARDIS ’10. Passau, Germany: Springer, Berlin, Heidelberg, Apr. 2010, pp. 223–237. ISBN: 978-3-642-12510-2. DOI: 10.1007/978-3-642-12510-2_16.

[Dau+18] Emma Dauterman et al. “True2F: Backdoor-resistant authentication tokens.” In: *CoRR* abs/1810.04660 (Oct. 2018). arXiv: 1810.04660.

[DDC18] Sanchari Das, Andrew Dingman, and L Jean Camp. “Why Johnny Doesn’t Use Two Factor A Two-Phase Usability Study of the FIDO U2F Security Key.” In: *2018 International Conference on Financial Cryptography and Data Security*. FC ’18. Curaçao, 2018.

[DLE19] Thomas Dressel, Eik List, and Florian Echtler. “SecuriCast: Zero-touch Two-factor Authentication Using WebBluetooth.” In: *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS ’19. Valencia, Spain: ACM, June 2019, 6:1–6:6. ISBN: 978-1-4503-6745-5. DOI: 10.1145/3319499.3328225.

[Dmi+14a] Alexandra Dmitrienko et al. “On the (In)Security of Mobile Two-Factor Authentication.” In: *Financial Cryptography and Data Security*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Christ Church, Barbados: Springer, Berlin, Heidelberg, Mar. 2014, pp. 365–383. ISBN: 978-3-662-45472-5. DOI: 10.1007/978-3-662-45472-5_24.

[Dmi+14b] Alexandra Dmitrienko et al. “Security Analysis of Mobile Two-Factor Authentication Schemes.” In: *Intel Technology Journal* 18.4 (June 2014). ISSN: 1535-864X.

[Doe+19] Periwinkle Doerfler et al. “Evaluating Login Challenges as a Defense Against Account Takeover.” In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: ACM, May 2019, pp. 372–382. ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313481.

[Dot19] Chris Dotson. *Practical Cloud Security: A Guide for Secure Design and Deployment*. Sebastopol, CA, USA: O’Reilly Media, Mar. 2019. ISBN: 978-1-4920-3751-4.

[DR08] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. RFC Editor, Aug. 2008. DOI: 10.17487/RFC5246.

[DRN17] Dipankar Dasgupta, Arunava Roy, and Abhijit Nag. *Advances in User Authentication*. Cham, Switzerland: Springer International Publishing, Sept. 2017. ISBN: 978-3-319-58808-7. DOI: 10.1007/978-3-319-58808-7_5.

[DZZ14] Benjamin Draffin, Jiang Zhu, and Joy Zhang. “KeySens: Passive User Authentication through Micro-behavior Modeling of Soft Keyboard Interaction.” In: *Mobile Computing, Applications, and Services*. Ed. by Gérard Memmi and Ulf Blanke. Paris, France: Springer International Publishing, Nov. 2014, pp. 184–201. ISBN: 978-3-319-05452-0. DOI: 10.1007/978-3-319-05452-0_14.

[EAK11] Mohamed Hamdy Eldefrawy, Khaled Alghathbar, and Muhammad Khurram Khan. “OTP-Based Two-Factor Authentication Using Mobile Phones.” In: *2011 Eighth International Conference on Information Technology: New Generations*. Las Vegas, NV, USA, Apr. 2011, pp. 327–331. ISBN: 978-1-61284-427-5. DOI: 10.1109/ITNG.2011.64.

[Eck14] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. 9th ed. Munich, Germany: De Gruyter, Oct. 2014. ISBN: 978-3-486-77848-9.

[Eik19] Ronald Eikenberg. “Schlüssel zum Glück.” In: *c’t* 18 (Aug. 2019), pp. 20–24. ISSN: 0724-8679.

[Ell07] Carl M. Ellison. “Ceremony Design and Analysis.” In: *IACR Cryptology ePrint Archive* (2007).

[FB17] Simone Fischer-Hübner and Stefan Berthold. “Privacy-Enhancing Technologies.” In: *Computer and Information Security Handbook*. Ed. by John R. Vacca. 3rd ed. Cambridge, MA, USA: Morgan Kaufmann, June 2017. Chap. 53, pp. 759–778. ISBN: 978-0-12-803843-7. DOI: 10.1016/B978-0-12-803843-7.00053-3.

[Fen+17] Dengguo Feng et al. *Trusted Computing*. Munich, Germany: De Gruyter, Dec. 2017. ISBN: 978-3-11-047759-7.

[Fer15] Nuno Alvarez Fernandes. “Reliable electronic certification on mobile devices.” Masters’s Thesis. Lisbon, Portugal: Instituto Superior Técnico, Nov. 2015. DOI: 10.13140/RG.2.1.2397.8323.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. “Zero-knowledge proofs of identity.” In: *Journal of Cryptology* 1.2 (June 1988), pp. 77–94. ISSN: 1432-1378. DOI: 10.1007/BF02351717.

[FKH14] Tobias Fiebig, Jan Krissler, and Ronny Hänsch. “Security Impact of High Resolution Smartphone Cameras.” In: *8th USENIX Workshop on Offensive Technologies*. WOOT ’14. San Diego, CA, USA: USENIX Association, Aug. 2014.

[Fri+17] Lex Fridman et al. “Active Authentication on Mobile Devices via Stylometry, Application Usage, Web Browsing, and GPS Location.” In: *IEEE Systems Journal* 11.2 (June 2017), pp. 513–521. ISSN: 1932-8184. DOI: 10.1109/JSYST.2015.2472579.

[FSS18] Julian Fietkau, Starbug, and Jean-Pierre Seifert. “Swipe Your Fingerprints! How Biometric Authentication Simplifies Payment, Access and Identity Fraud.” In: *12th USENIX Workshop on Offensive Technologies*. WOOT ’18. Baltimore, MD, USA: USENIX Association, Aug. 2018.

[GB17] Evan Gilman and Doug Barth. *Zero Trust Networks: Building Secure Systems in Untrusted Networks*. Sebastopol, CA, USA: O’Reilly Media, July 2017. ISBN: 978-1-4919-6216-9.

[GH18] Iness Ben Guirat and Harry Halpin. “Formal Verification of the W3C Web Authentication Protocol.” In: *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. HoTSoS ’18. Raleigh, NC, USA: ACM, Apr. 2018, pp. 1–10. ISBN: 978-1-4503-6455-3. DOI: 10.1145/3190619.3190640.

[Gra+17] Paul A. Grassi et al. *Digital Identity Guidelines: Authentication and Lifecycle*. Special Publication 800-63b. National Institute of Standards and Technology, June 2017. DOI: 10.6028/NIST.SP.800-63b.

[Gri17] Roger A. Grimes. *Hacking the Hacker: Learn From the Experts Who Take Down Hackers*. Indianapolis, IN, USA: Wiley, May 2017. ISBN: 978-1-119-39621-5.

[Han+07] Weili Han et al. “Anti-Phishing by Smart Mobile Device.” In: *2007 IFIP International Conference on Network and Parallel Computing Workshops*. NPC 2007. Liaoning, China, Sept. 2007, pp. 295–302. ISBN: 978-0-7695-2943-1. DOI: 10.1109/NPC.2007.68.

[Har05] Jan L. Harrington. *Network Security: A Practical Approach (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann, Apr. 2005. ISBN: 978-0-12-311633-8.

[HDK17] Tarfa Hamed, Rozita Dara, and Stefan C. Kremer. “Intrusion Detection in Contemporary Environments.” In: *Computer and Information Security Handbook*. Ed. by John R. Vacca. 3rd ed. Cambridge, MA, USA: Morgan Kaufmann, June 2017. Chap. 6, pp. 109–130. ISBN: 978-0-12-803843-7. DOI: 10.1016/B978-0-12-803843-7.00006-5.

[He07] Sheng He. “SIM Card Security.” Seminar Work. Bochum, Germany: Ruhr-University of Bochum, July 2007.

[HF15] Christopher Hadnagy and Michele Fincher. *Phishing Dark Waters: The Offensive and Defensive Sides of Malicious Emails*. Indianapolis, IN, USA: Wiley, Apr. 2015. ISBN: 978-1-118-95847-6.

[HJT07] Steffen Hallsteinsen, Ivar Jørstad, and Do Van Thanh. “Using the mobile phone as a security token for unified authentication.” In: *2007 Second International Conference on Systems and Networks Communications*. ICSNC ’07. Cap Esterel, France, Aug. 2007. ISBN: 978-0-7695-2938-7. DOI: 10.1109/ICSNC.2007.82.

[HO17] Silke Holtmanns and Ian Oliver. “SMS and One-Time-Password Interception in LTE Networks.” In: *2017 IEEE International Conference on Communications*. ICC ’17. Paris, France, May 2017, pp. 1–6. ISBN: 978-1-4673-8999-0. DOI: 10.1109/ICC.2017.7997246.

[HS17] Emin Huseynov and Jean-Marc Seigneur. “Context-Aware Multifactor Authentication Survey.” In: *Computer and Information Security Handbook*. Ed. by John R. Vacca. 3rd ed. Cambridge, MA, USA: Morgan Kaufmann, June 2017. Chap. 50, pp. 715–726. ISBN: 978-0-12-803843-7. DOI: 10.1016/B978-0-12-803843-7.00050-8.

[HZ16] Kexin Hu and Zhenfeng Zhang. “Security analysis of an attractive online authentication standard: FIDO UAF protocol.” In: *China Communications* 13.12 (Dec. 2016), pp. 189–198. ISSN: 1673-5447. DOI: 10.1109/CC.2016.7897543.

[Inf19a] Federal Office for Information Security. “Cryptographic Mechanisms: Recommendations and Key Lengths.” In: *Technical Guideline TR-02102-1* 1 (Feb. 2019).

[Inf19b] Federal Office for Information Security. “Electronic Identities and Trust Services in E-Government.” In: *Technical Guideline TR-03107-1* 1.1.1 (May 2019).

[Isl+18] Burhan Ul Islam Khan et al. "Offline OTP Based Solution for Secure Internet Banking Access." In: *2018 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*. Langkawi Island, Malaysia, Nov. 2018, pp. 167–172. ISBN: 978-1-5386-7263-1. DOI: 10.1109/IC3e.2018.8632643.

[ISO11a] ISO/IEC 9797-1:2011. *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*. Standard 9797-1. Geneva, Switzerland: International Organization for Standardization, Mar. 2011.

[ISO11b] ISO/IEC 9797-2:2011. *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a dedicated hash-function*. Standard 9797-2. Geneva, Switzerland: International Organization for Standardization, May 2011.

[ISO11c] ISO/IEC 7816-1:2011. *Identification cards – Integrated circuit cards – Part 1: Cards with contacts – Physical characteristics*. Standard 7816-1. Geneva, Switzerland: International Organization for Standardization, Feb. 2011.

[ITU18a] ITU Telecommunication Standardization Sector. *Universal authentication framework*. Recommendation X.1277. Geneva, Switzerland: International Telecommunication Union, Nov. 2018. DOI: 11.1002/1000/13727.

[ITU18b] ITU Telecommunication Standardization Sector. *Client to authenticator protocol/Universal 2-factor framework*. Recommendation X.1278. Geneva, Switzerland: International Telecommunication Union, Nov. 2018. DOI: 11.1002/1000/13728.

[Jac16] Todd A. Jacobs. "Secure Token-based Authentication with YubiKey 4." In: *Linux Journal* 2016.265 (May 2016). ISSN: 1075-3583.

[Jac19] Todd A. Jacobs. "WebAuthn Web Authentication with YubiKey 5." In: *Linux Journal* 2019.295 (Feb. 2019). ISSN: 1075-3583.

[Jak18] Markus Jakobsson. "Two-factor inauthentication – the rise in SMS phishing attacks." In: *Computer Fraud & Security* 2018.6 (June 2018), pp. 6–8. ISSN: 1361-3723. DOI: 10.1016/S1361-3723(18)30052-6.

[JBS15] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. RFC Editor, May 2015. DOI: 10.17487/RFC7519.

[JK18] Charlie Jacomme and Steve Kremer. "An Extensive Formal Analysis of Multi-factor Authentication Protocols." In: *2018 IEEE 31st Computer Security Foundations Symposium*. CSF '07. Oxford, United Kingdom, July 2018, pp. 1–15. ISBN: 978-1-5386-6680-7. DOI: 10.1109/CSF.2018.00008.

[JRN11] Anil K. Jain, Arun A. Ross, and Karthik Nandakumar. *Introduction to Biometrics*. New York, NY, USA: Springer, Nov. 2011. ISBN: 978-0-387-77325-4. DOI: 10.1007/978-0-387-77326-1.

[JS11] Louis Jordaan and Basie von Solms. “A Biometrics-Based Solution to Combat SIM Swap Fraud.” In: *Open Research Problems in Network Security*. Ed. by Jan Camenisch, Valentin Kisimov, and Maria Dubovitskaya. Sofia, Bulgaria: Springer, Berlin, Heidelberg, Feb. 2011, pp. 70–87. ISBN: 978-3-642-19228-9. DOI: 10.1007/978-3-642-19228-9_7.

[Kan19] Wang Kang. “U2Fi: A Provisioning Scheme of IoT Devices with Universal Cryptographic Tokens.” In: *CoRR* abs/1906.06009 (2019). arXiv: 1906.06009.

[KB16] Taechan Kim and Razvan Barbulescu. “Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case.” In: *Advances in Cryptology – CRYPTO 2016*. Ed. by Matthew Robshaw and Jonathan Katz. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, July 2016, pp. 543–571. ISBN: 978-3-662-53018-4. DOI: 10.1007/978-3-662-53018-4_20.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-hashing for message authentication*. RFC 2104. RFC Editor, Feb. 1997. DOI: 10.17487/RFC2104.

[KC12] Murray S. Kucherawy and Dave Crocker. *Email Greylisting: An Applicability Statement for SMTP*. RFC 6647. RFC Editor, June 2012. DOI: 10.17487/RFC6647.

[Kis19] Joe Kissell. *Take Control of Your Passwords*. 3rd ed. San Diego, CA, USA: alt concepts inc., Apr. 2019. ISBN: 978-1-4920-6638-5.

[KS12] Robert Künnemann and Graham Steel. “YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM.” In: *Security and Trust Management*. Ed. by Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi. Pisa, Italy: Springer, Berlin, Heidelberg, Sept. 2012, pp. 257–272. ISBN: 978-3-642-38004-4. DOI: 10.1007/978-3-642-38004-4_17.

[KSM14] Gurudatt Kulkarni, Ramesh Sutar, and Sangita Mohite. ““RFID security issues challenges”.” In: *2014 International Conference on Electronics and Communication Systems*. ICECS ’14. Coimbatore, India, Feb. 2014, pp. 1–4. ISBN: 978-1-4799-2320-5. DOI: 10.1109/ECS.2014.6892730.

[KVB17] Radhesh Krishnan Konoth, Victor van der Veen, and Herbert Bos. “How Anywhere Computing Just Killed Your Phone-Based Two-Factor Authentication.” In: *Financial Cryptography and Data Security*. Ed. by Jens Grossklags and Bart Preneel. Christ Church, Barbados: Springer, Berlin, Heidelberg, May 2017, pp. 405–421. ISBN: 978-3-662-54970-4. DOI: 10.1007/978-3-662-54970-4_24.

[Lan+17] Juan Lang et al. "Security Keys: Practical Cryptographic Second Factors for the Modern Web." In: *Financial Cryptography and Data Security*. Ed. by Jens Grossklags and Bart Preneel. Christ Church, Barbados: Springer, Berlin, Heidelberg, May 2017, pp. 422–440. ISBN: 978-3-662-54970-4. DOI: 10.1007/978-3-662-54970-4_25.

[LB10] Jing-Chiou Liou and Sujith Bhashyam. "A Feasible and Cost Effective Two-Factor Authentication for Online Transactions." In: *The 2nd International Conference on Software Engineering and Data Mining*. Chengdu, China, June 2010, pp. 47–51.

[LJ15] Ijlal Loutfi and Audun Jøsang. "FIDO Trust Requirements." In: *Secure IT Systems*. Ed. by Sonja Buchegger and Mads Dam. Stockholm, Sweden: Springer International Publishing, Oct. 2015, pp. 139–155. ISBN: 978-3-319-26502-5. DOI: 10.1007/978-3-319-26502-5_10.

[LM16] Jonathan LeBlanc and Tim Messerschmidt. *Identity and Data Security for Web Development: Best Practices*. Sebastopol, CA, USA: O'Reilly Media, June 2016. ISBN: 978-1-4919-3701-3.

[MA17] Konstantinos Markantonakis and Raja Naeem Akram. "Multi- Application Smart Card Platforms and Operating Systems." In: *Smart Cards, Tokens, Security and Applications*. Ed. by Keith Mayes and Konstantinos Markantonakis. 2nd ed. Cham, Switzerland: Springer International Publishing, May 2017. Chap. 3, pp. 59–92. ISBN: 978-3-319-50500-8. DOI: 10.1007/978-3-319-50500-8_3.

[Mah19] Jan Mahn. "FIDO2 für iOS." In: *c't* 20/2019 (Sept. 2019), p. 84. ISSN: 0724-8679.

[Mal+15] Aanchal Malhotra et al. *Attacking the Network Time Protocol*. Boston, MA, USA, Oct. 2015.

[Man96] Udi Manber. "A simple scheme to make passwords based on one-way functions much harder to crack." In: *Computers & Security* 15.2 (1996), pp. 171–176. ISSN: 0167-4048. DOI: 10.1016/0167-4048(96)00003-X.

[Mar13] Luther Martin. "Biometrics." In: *Computer and Information Security Handbook*. Ed. by John R. Vacca. 3rd ed. Cambridge, MA, USA: Morgan Kaufmann, June 2013. Chap. e70, e189–e204. ISBN: 978-0-12-803843-7. DOI: 10.1016/B978-0-12-803843-7.00070-3.

[May17] Keith Mayes. "An Introduction to Smart Cards." In: *Smart Cards, Tokens, Security and Applications*. Ed. by Keith Mayes and Konstantinos Markantonakis. 2nd ed. Cham, Switzerland: Springer International Publishing, May 2017. Chap. 1, pp. 1–29. ISBN: 978-3-319-50500-8. DOI: 10.1007/978-3-319-50500-8_1.

[MRa+05] David M'Raihi et al. *HOTP: An HMAC-based one-time password algorithm*. RFC 4226. RFC Editor, Dec. 2005. DOI: 10.17487/RFC4226.

[MRa+11] David M'Raihi et al. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. RFC Editor, June 2011. DOI: 10.17487/RFC6238.

[MS14] Tewfiq El Maliki and Jean-Marc Seigneur. "Online Identity and User Management Services." In: *Managing Information Security*. Ed. by John R. Vacca. 2nd ed. Waltham, MA, USA: Syngress, May 2014. Chap. 4, pp. 75–118. ISBN: 978-0-12-416688-2. DOI: 10.1016/B978-0-12-416688-2.00004-0.

[Mul+13] Collin Mulliner et al. "SMS-Based One-Time Passwords: Attacks and Defense." In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Konrad Rieck, Patrick Stewin, and Jean-Pierre Seifert. Berlin, Germany: Springer, Berlin, Heidelberg, July 2013, pp. 150–159. ISBN: 978-3-642-39235-1. DOI: 10.1007/978-3-642-39235-1_9.

[Nat18] Committee on National Security Systems. *National Information Assurance (IA) Glossary*. Tech. rep. 4009. Committee on National Security Systems, Apr. 2018.

[Ngu15] Kim Nguyen. "Neue Wege der hardware-basierten Authentisierung und Identifikation: FIDO, PKI und mehr." In: *Sicherheit im Wandel von Technologien und Märkten*. Ed. by Udo Bub, Viktor Deleski, and Klaus-Dieter Wolfenstetter. Wiesbaden, Germany: Springer Fachmedien Wiesbaden, Sept. 2015, pp. 49–54. ISBN: 978-3-658-11274-5. DOI: 10.1007/978-3-658-11274-5_8.

[Noc18] Mark Noctor. "PSD2: Is the banking industry prepared?" In: *Computer Fraud & Security* 2018.6 (June 2018), pp. 9–11. ISSN: 1361-3723. DOI: 10.1016/S1361-3723(18)30053-8.

[ORP13] David Oswald, Bastian Richter, and Christof Paar. "Side-Channel Attacks on the Yubikey 2 One-Time Password Generator." In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright. Rodney Bay, St. Lucia: Springer, Berlin, Heidelberg, Oct. 2013, pp. 204–222. ISBN: 978-3-642-41284-4. DOI: 10.1007/978-3-642-41284-4_11.

[Oud16] Abdelkader Ouda. "A framework for next generation user authentication." In: *2016 3rd MEC International Conference on Big Data and Smart City*. ICBDSC '16. Muscat, Oman, Mar. 2016, pp. 1–4. ISBN: 978-1-4673-9584-7. DOI: 10.1109/ICBDSC.2016.7460349.

[Pan+17] Christoforos Panos et al. "A Security Evaluation of FIDO's UAF Protocol in Mobile and Embedded Devices." In: *Digital Communication. Towards a Smart and Secure Future Internet*. Ed. by Alessandro Piva, Ilenia Tinnirello, and Simone Morosi. Palermo, Italy: Springer International Publishing, Sept. 2017, pp. 127–142. ISBN: 978-3-319-67639-5. DOI: 10.1007/978-3-319-67639-5_11.

[Pan12] Raymond R. Panko. "Digital Signatures and Electronic Signatures." In: *Handbook of Information Security: Threats, Vulnerabilities, Prevention, Detection, and Management*. Ed. by Hossein Bidgoli. Vol. 3. Hoboken, NJ, USA: Wiley, Jan. 2012, pp. 562–570. ISBN: 978-0-471-64832-1.

[PO95] Bart Preneel and Paul C. van Oorschot. "MDx-MAC and Building Fast MACs from Hash Functions." In: *Advances in Cryptology — CRYPT0' 95.* Ed. by Don Coppersmith. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Aug. 1995, pp. 1–14. ISBN: 978-3-540-44750-4. DOI: 10.1007/3-540-44750-4_1.

[PP11] Christof Paar and Jan Pelzl. *Understanding Cryptography.* Berlin, Germany and Heidelberg, Germany: Springer, Berlin, Heidelberg, Oct. 2011. ISBN: 978-3-642-04100-6. DOI: 10.1007/978-3-642-04101-3.

[PRW18] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. "Formal Analysis of the FIDO 1.x Protocol." In: *Foundations and Practice of Security.* Ed. by Abdessamad Imine et al. Nancy, France: Springer International Publishing, Feb. 2018, pp. 68–82. ISBN: 978-3-319-75650-9. DOI: 10.1007/978-3-319-75650-9_5.

[Puz17] Sergey Puzankov. "Stealthy SS7 Attacks." In: *Journal of ICT Standardization* 5.1 (Jan. 2017), pp. 39–52. ISSN: 2246-0853.

[Rab08] Ariel Rabkin. "Personal Knowledge Questions for Fallback Authentication: Security Questions in the Era of Facebook." In: *Proceedings of the 4th Symposium on Usable Privacy and Security.* SOUPS '08. Pittsburgh, PA, USA: ACM, July 2008, pp. 13–23. ISBN: 978-1-60558-276-4. DOI: 10.1145/1408664.1408667.

[Ras12] Jeremy L. Rasmussen. "Password Authentication." In: *Handbook of Information Security: Threats, Vulnerabilities, Prevention, Detection, and Management.* Ed. by Hossein Bidgoli. Vol. 3. Hoboken, NJ, USA: Wiley, Jan. 2012, pp. 424–438. ISBN: 978-0-471-64832-1.

[Ree+19] Ken Reese et al. "A Usability Study of Five Two-Factor Authentication Methods." In: *Fifteenth Symposium on Usable Privacy and Security.* SOUPS '19. Santa Clara, CA, USA: USENIX Association, Aug. 2019.

[Rey+18] J. Reynolds et al. "A Tale of Two Studies: The Best and Worst of YubiKey Usability." In: *2018 IEEE Symposium on Security and Privacy.* SP '18. San Francisco, CA, USA, May 2018, pp. 872–888. DOI: 10.1109/SP.2018.00067.

[RKM16] Florian Reimair, Christian Kollmann, and Alexander Marsalek. "Emulating U2F authenticator devices." In: *2016 IEEE Conference on Communications and Network Security.* CNS '16. Philadelphia, PA, USA, Oct. 2016, pp. 543–551. DOI: 10.1109/CNS.2016.7860546.

[Ros18] Patrick Rosenberger. *Bitcoin und Blockchain: Vom Scheitern einer Ideologie und dem Erfolg einer revolutionären Technik.* Berlin, Germany and Heidelberg, Germany: Springer, Berlin, Heidelberg, June 2018. ISBN: 978-3-662-56088-4. DOI: 10.1007/978-3-662-56088-4.

[SBE09] Stuart Schechter, A. J. Bernheim Brush, and Serge Egelman. "It's No Secret. Measuring the Security and Reliability of Authentication via "Secret" Questions." In: *2009 30th IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE, May 2009, pp. 375–390. DOI: 10.1109/SP.2009.11.

[Sch16] Bruce Schneier. "Stop Trying to Fix the User." In: *IEEE Security Privacy* 14.5 (Sept. 2016), pp. 96–96. ISSN: 1540-7993. DOI: 10.1109/MSP.2016.101.

[Sch19a] Jürgen Schmidt. "Abschied vom Passwort." In: *c't* 18 (Aug. 2019), pp. 16–18. ISSN: 0724-8679.

[Sch19b] Jürgen Schmidt. "Verschlossen, nicht verrammelt." In: *c't* 18 (Aug. 2019), pp. 30–32. ISSN: 0724-8679.

[She+19] Jaryn Shen et al. "AMOGAP: Defending Against Man-in-the-Middle and Offline Guessing Attacks on Passwords." In: *Information Security and Privacy*. Ed. by Julian Jang-Jaccard and Fuchun Guo. Christchurch, New Zealand: Springer International Publishing, July 2019, pp. 514–532. ISBN: 978-3-030-21548-4. DOI: 10.1007/978-3-030-21548-4_28.

[Shi+11] Elaine Shi et al. "Implicit Authentication through Learning User Behavior." In: *Information Security*. Ed. by Mike Burmester et al. Boca Raton, FL, USA: Springer, Berlin, Heidelberg, Oct. 2011, pp. 99–113. ISBN: 978-3-642-18178-8. DOI: 10.1007/978-3-642-18178-8_9.

[Sho14] Adam Shostack. *Threat Modeling: Designing for Security*. Indianapolis, IN, USA: Wiley, Feb. 2014. ISBN: 978-1-118-81005-7.

[Sia+17] Hossein Siadati et al. "Mind your SMSes: Mitigating social engineering in second factor authentication." In: *Computers & Security* 65 (Mar. 2017), pp. 14–28. ISSN: 0167-4048. DOI: 10.1016/j.cose.2016.09.009.

[Sic16] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz-Kataloge*. 15th ed. Cologne, Germany: Bundesanzeiger Verlag, Apr. 2016. ISBN: 978-3-88784-915-3.

[Sic19] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz-Kompendium*. 2nd ed. Cologne, Germany: Bundesanzeiger Verlag, Feb. 2019. ISBN: 978-3-8462-0906-6.

[SM18] Michael Schwartz and Maciej Machulak. *Securing the Perimeter*. New York, NY, USA: Apress, Dec. 2018. ISBN: 978-1-4842-2601-8. DOI: 10.1007/978-1-4842-2601-8.

[SMJ16] Vlasta Stavova, Vashek Matyas, and Mike Just. "Codes v. People: A Comparative Usability Study of Two Password Recovery Mechanisms." In: *Information Security Theory and Practice*. Ed. by Sara Foresti and Javier Lopez. Heraklion, Crete, Greece: Springer International Publishing, Sept. 2016, pp. 35–50. ISBN: 978-3-319-45931-8. DOI: 10.1007/978-3-319-45931-8_3.

[SNM16] Hossein Siadati, Toan Nguyen, and Nasir Memon. "Verification Code Forwarding Attack (Short Paper)." In: *Technology and Practice of Passwords*. Ed. by Frank Stajano et al. Cambridge, United Kingdom: Springer International Publishing, Dec. 2016, pp. 65–71. ISBN: 978-3-319-29938-9. DOI: 10.1007/978-3-319-29938-9_5.

[ST08] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication Series 198-1. National Institute of Standards and Technology, July 2008. DOI: 10.6028/NIST.FIPS.198-1.

[ST13] National Institute of Standards and Technology. *Personal Identity Verification (PIV) of Federal Employees and Contractors*. Federal Information Processing Standards Publication Series 201-2. National Institute of Standards and Technology, Aug. 2013. DOI: 10.6028/NIST.FIPS.201-2.

[ST19] National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules*. Federal Information Processing Standards Publication Series 140-3. National Institute of Standards and Technology, Mar. 2019. DOI: 10.6028/NIST.FIPS.140-3.

[Sta15] Mark Stanislav. *Two-Factor Authentication*. Ely, United Kingdom: IT Governance Publishing, Apr. 2015. ISBN: 978-1-84928-733-3.

[Sta17] William Stallings. *Cryptography and Network Security: Principles and Practice, Global Edition*. 7th ed. London, United Kingdom: Pearson, Oct. 2017. ISBN: 978-1-292-15858-7.

[Ste+17] Marc Stevens et al. "The First Collision for Full SHA-1." In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Santa Barbara, CA, USA: Springer International Publishing, Aug. 2017, pp. 570–596. ISBN: 978-3-319-63688-7. DOI: 10.1007/978-3-319-63688-7_19.

[Swe08] Christopher Swenson. *Modern Cryptanalysis: Techniques for Advanced Code Breaking*. Hoboken, NJ, USA: Wiley, Mar. 2008. ISBN: 978-0-470-13593-8.

[TC11] Sean Turner and Lily Chen. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. RFC 6151. RFC Editor, Mar. 2011. DOI: 10.17487/RFC6151.

[Tho+17] Kurt Thomas et al. "Data Breaches, Phishing, or Malware?: Understanding the Risks of Stolen Credentials." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, TX, USA: ACM, 2017, pp. 1421–1434. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134067.

[Tho+19] Kurt Thomas et al. "Protecting accounts from credential stuffing with password breach alerting." In: *28th USENIX Security Symposium*. USENIX Security '19. Santa Clara, CA, USA: USENIX Association, Aug. 2019, pp. 1556–1571. ISBN: 978-1-939133-06-9.

[Tod07] Dobromir Todorov. *Mechanics of User Identification and Authentication: Fundamentals of Identity Management*. Boca Raton, FL, USA: Auerbach Publications, June 2007. ISBN: 978-1-4200-5219-0.

[Tun17] Michael Tunstall. "Smart Card Security." In: *Smart Cards, Tokens, Security and Applications*. Ed. by Keith Mayes and Konstantinos Markantonakis. 2nd ed. Cham, Switzerland: Springer International Publishing, May 2017. Chap. 9, pp. 217–251. ISBN: 978-3-319-50500-8. DOI: 10.1007/978-3-319-50500-8_9.

[TW75] Rein Turn and W. H. Ware. "Privacy and Security in Computer Systems: The vulnerability of computerized information has prompted measures to protect both the rights of individual subjects and the confidentiality of research data bases." In: *American Scientist* 63.2 (1975), pp. 196–203. ISSN: 0003-0996.

[ULC19] Enis Ulqinaku, Daniele Lain, and Srdjan Capkun. "2FA-PP: 2nd Factor Phishing Prevention." In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '19. Miami, FL, USA: ACM, May 2019, pp. 60–70. ISBN: 978-1-4503-6726-4. DOI: 10.1145/3317549.3323404.

[UY14] Mariano Luis T. Uymatiao and William Emmanuel S. Yu. "Time-based OTP authentication via secure tunnel (TOAST): A mobile TOTP scheme using TLS seed exchange and encrypted offline keystore." In: *2014 4th IEEE International Conference on Information Science and Technology*. Shenzhen, China, Apr. 2014, pp. 225–229. ISBN: 978-1-4799-4808-6. DOI: 10.1109/ICIST.2014.6920371.

[Wal16] Elizabeth Walkup. *The Password Problem*. Tech. rep. SAND2016-5208T. Albuquerque, NM, USA: Office of Scientific and Technical Information, June 2016.

[Was17] Marvin Waschke. *Personal Cybersecurity: How to Avoid and Recover from Cybercrime*. New York, NY, USA: Apress, Jan. 2017. ISBN: 978-1-4842-2430-4.

[Wel15] Marcus K. Weldon. *The Future X Network: A Bell Labs Perspective*. Boca Raton, FL, USA: CRC Press, May 2015. ISBN: 978-1-4987-5927-4.

[Wel17] Bill Welch. "Exploiting the weaknesses of SS7." In: *Network Security* 2017.1 (Jan. 2017), pp. 17–19. ISSN: 1353-4858. DOI: 10.1016/S1353-4858(17)30008-9.

[Wit02] Marc Witteman. "Advances in Smartcard Security." In: *Information Security Bulletin* 07 (2002).

[WPR16] Keith Winnard, John Petreshock, and Philippe Richard. *IBM MFA V1R1: TouchToken, PassTicket, and Application Bypass Support*. Armonk, NY, USA: International Technical Support Organization, Dec. 2016. ISBN: 978-0-7384-5573-0.

[Xia18] Liang Xian Liang. "Two-Factor Human Authentication." Bachelor's Thesis. Perak, Malaysia: Universiti Tunku Abdul Rahman, May 2018.

[XZL14] Hui Xu, Yangfan Zhou, and Michael R. Lyu. “Towards Continuous and Passive Authentication via Touch Biometrics: An Experimental Study on Smartphones.” In: *10th Symposium On Usable Privacy and Security*. SOUPS ’14. Menlo Park, CA, USA: USENIX Association, July 2014, pp. 187–198. ISBN: 978-1-931971-13-3.

[Yua05] Michael Juntao Yuan. *Nokia Smartphone Hacks*. Sebastopol, CA, USA: O’Reilly Media, July 2005. ISBN: 978-0-596-00961-8.

[Zab19] Roman Zabicki. *Practical Security: Simple Practices for Defending Your Systems*. Raleigh, NC, USA: Pragmatic Programmers, LLC, Feb. 2019. ISBN: 978-1-68050-634-1.

[ZKM12] Feng Zhang, Aron Kondoro, and Sead Muftic. “Location-Based Authentication and Authorization Using Smart Phones.” In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. TrustCom ’12. Liverpool, United Kingdom, June 2012, pp. 1285–1292. DOI: 10.1109/TrustCom.2012.198.

Internet Sources

[Bal+19] Dirk Balfanz et al. *Web Authentication: An API for accessing Public Key Credentials Level 1*. Mar. 4, 2019. URL: https://www.w3.org/TR/2019/REC-webauthn-1-20190304/ (last accessed on 09/06/2019).

[BBL17] Dirk Balfanz, Arnar Birgisson, and Juan Lang. *FIDO U2F JavaScript API*. Apr. 11, 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-javascript-api-v1.2-ps-20170411.pdf (last accessed on 09/09/2019).

[BEL17] Dirk Balfanz, Jakob Ehrensvärd, and Juan Lang. *FIDO U2F Raw Message Formats*. Apr. 11, 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-raw-message-formats-v1.2-ps-20170411.pdf (last accessed on 09/19/2019).

[BK18] Christiaan Brand and Eiji Kitamura. *Enabling Strong Authentication with WebAuthn*. May 29, 2018. URL: https://developers.google.com/web/updates/2018/05/webauthn (last accessed on 08/15/2019).

[Bra+19] Christiaan Brand et al. *Client to Authenticator Protocol (CTAP)*. Jan. 30, 2019. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.pdf (last accessed on 09/09/2019).

[Bra19a] Christiaan Brand. *Advisory: Security Issue with Bluetooth Low Energy (BLE) Titan Security Keys*. May 19, 2019. URL: https://security.googleblog.com/2019/05/titan-keys-update.html (last accessed on 09/06/2019).

[Bra19b] Brave. *Adding YubiKey Support to Brave for iOS*. June 24, 2019. URL: https://brave.com/ios-yubikey-support/ (last accessed on 08/17/2019).

[Bra19c] Brave. *With Yubico partnership and support for the new YubiKey 5Ci, Brave is the first web browser to offer secure phishing-resistant authentication via robust security keys on iPhones & iPads*. Aug. 20, 2019. URL: https://brave.com/partnership-with-yubico/ (last accessed on 08/20/2019).

[Bun18] Bundeskriminalamt. *Cybercrime, Bundeslagebild 2017*. Sept. 27, 2018. URL: https://www.bka.de/SharedDocs/Downloads/DE/Publikationen/JahresberichteUndLagebilder/Cybercrime/cybercrimeBundeslagebild2017.html (last accessed on 08/20/2019).

[Bun19] Bundesnetzagentur. *Jahresbericht 2018*. May 16, 2019. URL: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Allgemeines/Bundesnetzagentur/Publikationen/Berichte/2019/JB2018.pdf (last accessed on 09/22/2019).

[Col16] Katie Collins. *Facebook buys black market passwords to keep your account safe*. Nov. 9, 2016. URL: https://www.cnet.com/news/facebook-chief-security-officer-alex-stamos-web-summit-lisbon-hackers/ (last accessed on 08/18/2019).

[Con19] Kate Conger. *Twitter C.E.O. Jack Dorsey's Account Hacked*. Aug. 30, 2019. URL: https://www.nytimes.com/2019/08/30/technology/jack-dorsey-twitter-account-hacked.html (last accessed on 08/31/2019).

[Dav18] Jon Davis. *Release Notes for Safari Technology Preview 71*. Dec. 5, 2018. URL: https://webkit.org/blog/8517/release-notes-for-safari-technology-preview-71/ (last accessed on 08/15/2019).

[dim19] infratest dimap. *ARD – DeutschlandTREND Januar 2019*. Jan. 2019. URL: https://www.infratest-dimap.de/fileadmin/user_upload/dt1901_bericht.pdf (last accessed on 08/20/2019).

[FIa] FIDO Alliance. *Certified Authenticator Levels*. URL: https://fidoalliance.org/certification/authenticator-certification-levels/ (last accessed on 09/24/2019).

[FIb] FIDO Alliance. *History of FIDO Alliance*. URL: https://fidoalliance.org/overview/history/ (last accessed on 09/20/2019).

[FI14] FIDO Alliance. *FIDO Privacy Principles*. Feb. 1, 2014. URL: https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf (last accessed on 09/18/2019).

[Fri19] Christian Friemel. *Trotz „Collection #1-5": Beim Passwortschutz lernen deutsche Internet-Nutzer nur langsam dazu*. Mar. 27, 2019. URL: https://newsroom.web.de/2019/03/27/trotz-collection-1-5-beim-passwortschutz-lernen-deutsche-internet-nutzer-nur-langsam-dazu/ (last accessed on 08/20/2019).

[Ger18] Chromium Gerrit. *Enable WebAuthN on Android by default (If95f7508) · Gerrit Code Review*. Aug. 16, 2018. URL: https://chromium-review.googlesource.com/c/chromium/src/+/1176736/ (last accessed on 08/15/2019).

[Gooa] Google. *Email encryption in transit*. URL: https://transparencyreport.google.com/safer-email/overview?encrypt_out=start:1564617600000;end:1567295999999;series:outbound&lu=encrypt_in&encrypt_in=start:1564617600000;end:1567295999999;series:inbound (last accessed on 09/22/2019).

[Goob] Google. *Google 2-Step Verification*. URL: https://www.google.com/landing/2step/ (last accessed on 08/01/2019).

[Hom15] Egor Homakov. *How "../sms" could bypass Authy 2 Factor Authentication*. Mar. 15, 2015. URL: https://sakurity.com/blog/2015/03/15/authy_bypass.html (last accessed on 08/09/2019).

[Hun17] Troy Hunt. *Password reuse, credential stuffing and another billion records in Have I been pwned*. May 5, 2017. URL: https://www.troyhunt.com/password-reuse-credential-stuffing-and-another-1-billion-records-in-have-i-been-pwned/ (last accessed on 09/15/2019).

[JK] Maneesh Jhinger and Pooja Kuntal. *FAST IDENTITYONLINE(FIDO): PASSWORD-LESSAUTHENTICATION*. URL: https://developer.amd.com/wp-content/resources/FIDO_Whitepaper.pdf (last accessed on 09/13/2019).

[Jon19] J.C. Jones. *Web Authentication in Firefox for Android*. Aug. 5, 2019. URL: https://blog.mozilla.org/security/2019/08/05/web-authentication-in-firefox-for-android/ (last accessed on 08/15/2019).

[JT18] J.C. Jones and Tim Taubert. *Using Hardware Token-based 2FA with the WebAuthn API*. Jan. 16, 2018. URL: https://hacks.mozilla.org/2018/01/using-hardware-token-based-2fa-with-the-webauthn-api/ (last accessed on 08/15/2019).

[Kes18] Limor Kessem. *IBM Security: Future of Identity Study*. Jan. 2018. URL: https://www.ibm.com/downloads/cas/QRBY08NO (last accessed on 08/20/2019).

[Kre14] Stefan Krempl. *31C3: CCC-Tüftler hackt Merkels Iris und von der Leyens Fingerabdruck*. Dec. 28, 2014. URL: https://heise.de/-2506929 (last accessed on 08/09/2019).

[Las18] LastPass. *Psychology of Passwords: Neglect is Helping Hackers Win*. May 1, 2018. URL: https://lp-cdn.lastpass.com/lporcamedia/document-library/lastpass/pdf/en/logmein-lastpass-survey-ebook-v8.pdf (last accessed on 08/30/2019).

[Ley19] John Leyden. *U2F nowhere near ready for prime time*. Feb. 6, 2019. URL: https://portswigger.net/daily-swig/u2f-nowhere-near-ready-for-prime-time (last accessed on 09/06/2019).

[Lin17a] Rolf Lindemann. *FIDO AppID and Facet Specification*. Feb. 2, 2017. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-appid-and-facets-v1.1-ps-20170202.pdf (last accessed on 09/10/2019).

[Lin17b] Rolf Lindemann. *FIDO Security Reference*. Apr. 11, 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-security-ref-v1.2-ps-20170411.pdf (last accessed on 09/09/2019).

[LK17a] Rolf Lindemann and John Kemp. *FIDO UAF Authenticator Commands*. Feb. 2, 2017. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-authnr-cmds-v1.1-ps-20170202.pdf (last accessed on 09/10/2019).

[LK17b] Rolf Lindemann and John Kemp. *FIDO UAF Authenticator-Specific Module API*. Feb. 2, 2017. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-asm-api-v1.1-ps-20170202.pdf (last accessed on 09/09/2019).

[Löw16] Anne-Marie Eklund Löwinder. *Lösenord för alla*. Sept. 1, 2016. URL: https://internetstiftelsen.se/docs/Rapport_Losenord_for_alla.pdf (last accessed on 08/30/2019).

[LT17] Rolf Lindemann and Eric Tiffany. *FIDO UAF Protocol Specification*. Feb. 2, 2017. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-protocol-v1.1-ps-20170202.pdf (last accessed on 09/09/2019).

[Mac+17] Salah Machani et al. *FIDO UAF Architectural Overview*. Feb. 2, 2017. URL: https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-overview-v1.1-ps-20170202.pdf (last accessed on 09/09/2019).

[Mic19] Microsoft. *How to use two-step verification with your Microsoft account*. July 25, 2019. URL: https://support.microsoft.com/en-us/help/12408/microsoft-account-how-to-use-two-step-verification (last accessed on 08/01/2019).

[Moz] Mozilla. *Rust library to interact with Security Keys, used by Firefox*. URL: https://github.com/mozilla/authenticator-rs/tree/master (last accessed on 09/21/2019).

[Pla] PlayStation. *2-Step Verification*. URL: https://www.playstation.com/en-us/account-security/2-step-verification/ (last accessed on 08/01/2019).

[Pow18] Adam Powers. *FIDO TechNotes: The Truth about Attestation*. July 19, 2018. URL: https://fidoalliance.org/fido-technotes-the-truth-about-attestation/ (last accessed on 09/24/2019).

[RR19] Natasha Rooney and Florian Rivoal. *World Wide Web Consortium Process Document*. Mar. 1, 2019. URL: https://www.w3.org/2019/Process-20190301/ (last accessed on 09/17/2019).

[Sch19c] Ben Schwan. *Safari 13 ist verfügbar*. Sept. 20, 2019. URL: https://heise.de/-4534466 (last accessed on 09/20/2019).

[Sri+17] Sampath Srinivas et al. *Universal 2nd Factor (U2F) Overview*. Apr. 11, 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.pdf (last accessed on 09/09/2019).

[Staa] StatCounter. *Browser Market Share Worldwide*. URL: https://gs.statcounter.com/browser-market-share#monthly-201908-201908-bar (last accessed on 09/20/2019).

[Stab] StatCounter. *Desktop Browser Market Share Worldwide*. URL: https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201908-201908-bar (last accessed on 09/20/2019).

[Sta18] P.I.E. Staff. *Security Concerns Surrounding WebAuthn: Don't Implement ECDAA (Yet)*. Aug. 23, 2018. URL: https://paragonie.com/b/ya9unbDYhvmp2EUy (last accessed on 08/09/2019).

[Ste19] Lukas Stefanko. *Malware sidesteps Google permissions policy with new 2FA bypass technique*. June 17, 2019. URL: https://welivesecurity.com/2019/06/17/malware-google-permissions-2fa-bypass/ (last accessed on 08/09/2019).

[Sup19a] Apple Support. *Two-factor authentication for Apple ID*. July 30, 2019. URL: https://support.apple.com/en-us/HT204915 (last accessed on 08/01/2019).

[Sup19b] Apple Support. *Two-step verification for Apple ID*. May 29, 2019. URL: https://support.apple.com/en-us/HT204152 (last accessed on 08/01/2019).

[Sup19c] Microsoft Support. *Lifecycle FAQ—Internet Explorer and Edge*. June 12, 2019. URL: https://support.microsoft.com/en-us/help/17454/lifecycle-faq-internet-explorer (last accessed on 08/15/2019).

[Wes19a] Mike West. *Credential Management Level 1*. Jan. 17, 2019. URL: https://www.w3.org/TR/2019/WD-credential-management-1-20190117/ (last accessed on 09/17/2019).

[Wes19b] Olivia von Westernhagen. *YubiKey: Yubico ruft Security-Keys der FIPS-Serie zurück*. June 19, 2019. URL: https://heise.de/-4448794 (last accessed on 08/30/2019).

[You18] YouGov. *YouGov Online Passwords*. Oct. 2018. URL: https://d25d2506sfb94s.cloudfront.net/cumulus_uploads/document/81iu1qrr2x/Passwords%20results,%20Sept.%202017%20%E2%80%93%20Oct.%202018.pdf (last accessed on 08/30/2019).

[You19] YouGov. *YouGov Computer Protection 2019*. Jan. 2019. URL: https://d25d2506sfb94s.cloudfront.net/cumulus_uploads/document/ubt2ymvq7p/Computer%20protection,%20Jan.%2015%202019.pdf (last accessed on 08/30/2019).

[Yub] Yubico. *Works with YubiKey Catalog*. URL: https://www.yubico.com/works-with-yubikey/catalog/#protocol=webauthn&usecase=all&key=all (last accessed on 09/24/2019).

[Yub12] Yubico. *YubiKey Authentication Module DesignGuideline*. May 7, 2012. URL: https://www.yubico.com/wp-content/uploads/2012/10/YubiKey-Authentication-Module-Design-Guideline-v1.0.pdf (last accessed on 09/01/2019).

[Yub15] Yubico. *The YubiKey Manual*. Mar. 27, 2015. URL: https://www.yubico.com/wp-content/uploads/2015/03/YubiKeyManual_v3.4.pdf (last accessed on 09/01/2019).

List of Figures

List of Tables

List of Listings

Acronyms

Symbols

2FA Two-Factor Authentication

A

AAGUID Authenticator Attestation Globally Unique ID
AAID Authenticator Attestation ID
AD Active Directory
AES Advanced Encryption Standard
APDU Application Protocol Data Unit
API Application Programming Interface
ASM Authenticator-Specific Module

B

BLE Bluetooth Low Energy
BN Barreto-Naehrig
BSI Federal Office For Information Security

C

CA Certificate Authority
CAC Common Access Card
CBC-MAC Cipher Block Chaining Message Authentication Code
CBOR Concise Binary Object Representation
CCID Chip Card Interface Device
COSE CBOR Object Signing And Encryption
CPU Central Processing Unit
CRC Cyclic Redundancy Check
CSO Chief Security Officer
CSPRNG Cryptographically Secure Pseudo-Random Number Generator
CTAP Client-To-Authenticator Protocol

D

DAA Direct Anonymous Attestation
DDoS Distributed Denial Of Service
DES Data Encryption Standard
DNS Domain Name System
DPA Differential Power Analysis

E

EC Elliptic-Curve
ECC Elliptic-Curve Cryptography
ECDAA Elliptic Curve Digital Signature Algorithm
ECDLP Elliptic Curve Discrete Logarithm Problem
ECDSA Elliptic Curve Direct Anonymous Attestation
EEPROM Electrically Erasable Programmable Read-Only Memory
EPROM Erasable Programmable Read-Only Memory
EU European Union

F

FAR False Acceptance Rate
FCP Final Challenge Parameter
FIDO Fast IDentity Online
FIPS Federal Information Processing Standard Publication
FRR False Rejection Rate

G

GPS Global Positioning System

H

HID Human Interface Device
HMAC Keyed-Hash Message Authentication Code
HOTP HMAC-Based One-Time Password
HTTP Hypertext Transfer Protocol

I

IANA Internet Assigned Numbers Authority
ICC Integrated Circuit Card
IEEE Institute Of Electrical And Electronics Engineers
IETF Internet Engineering Task Force
IoT Internet Of Things
ISO International Organization For Standardization
IT Information Technology
ITU International Telecommunication Union

J

JCVM Java Card Virtual Machine
JS JavaScript
JSON JavaScript Object Notation
JWT JSON Web Token

K

KRD Key Registration Data

L

LDAP Lightweight Directory Access Protocol

M

MAC Media Access Control
MAC Message Authentication Code
MD Message Digest
MFA Multi-Factor Authentication
MIC Message Integrity Code
MITM Man In The Middle

N

NFC Near-Field Communication
NIST National Institute Of Standards And Technology
nonce Number Used Once
NTP Network Time Protocol

O

OATH Initiative For Open Authentication
OOB Out-Of-Band
OS Operating System
OTP One-Time Password

P

PIN Personal Identification Number
PIV Personal Identity Verification
PKCS Public Key Cryptography Standards
PoC Proof Of Concept
PSD Payment Services Directive

Q

QR Quick Response

R

RAM Random-Access Memory
RFC Request For Comments
RFID Radio-Frequency Identification
RNG Random Number Generator
ROM Read-Only Memory
RP Relying Party
RSA Rivest–Shamir–Adleman

S

SHA Secure Hash Algorithm
SIM Subscriber Identity Module
SMS Short Message Service
SS7 Signalling System No. 7
SSL Secure Sockets Layer
SSO Single Sign-On

T

TAN Transaction Authentication Number
TLS Transport Layer Security
TOTP Time-Based One-Time Password

TPM Trusted Platform Module

U

U2F Universal Second Factor
UAF Universal Authentication Framework
URL Uniform Resource Locator
US United States
USB Universal Serial Bus

V

VCFA Verification Code Forwarding Attack
VoIP Voice Over Internet Protocol
VPN Virtual Private Network

W

W3C World Wide Web Consortium
WAP Wireless Application Protocol

Z

ZKPP Zero-Knowledge Password Proof

A Appendix

A.1 W3C Standardization Process

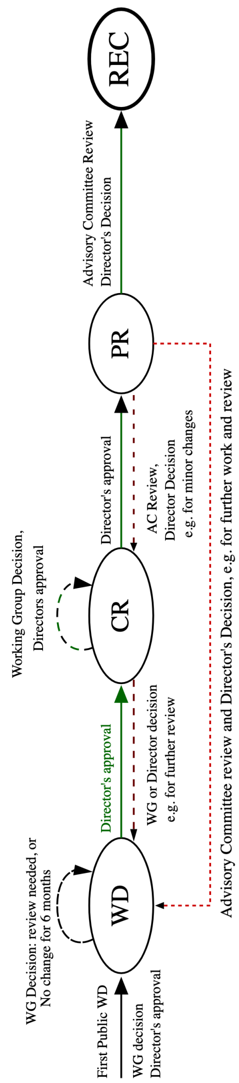

Figure A.1: W3C standardization process[214]

[214]Source: RR19, Chapter 6.7.

A.2 Web Browser Usage Statistics for August 2019

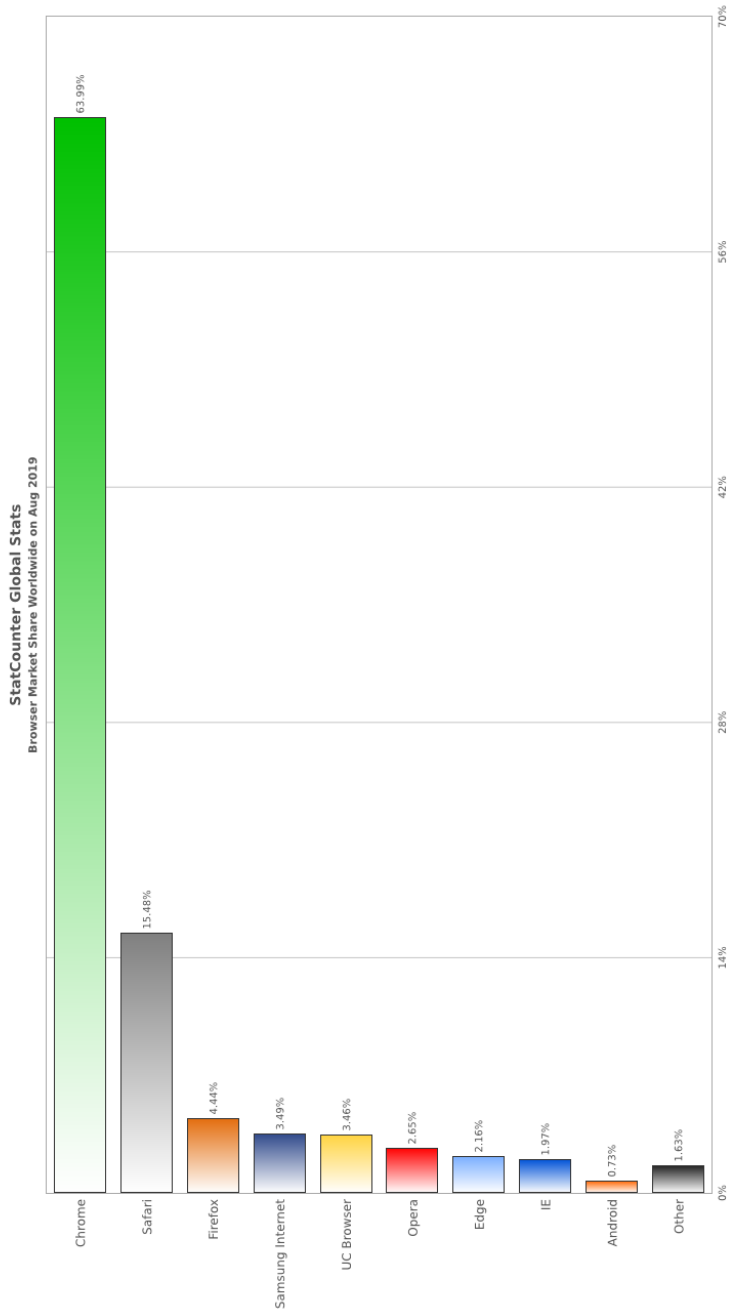

Figure A.2: Web Browser Usage Statistics for August 2019[215]

[215]Source Staa.

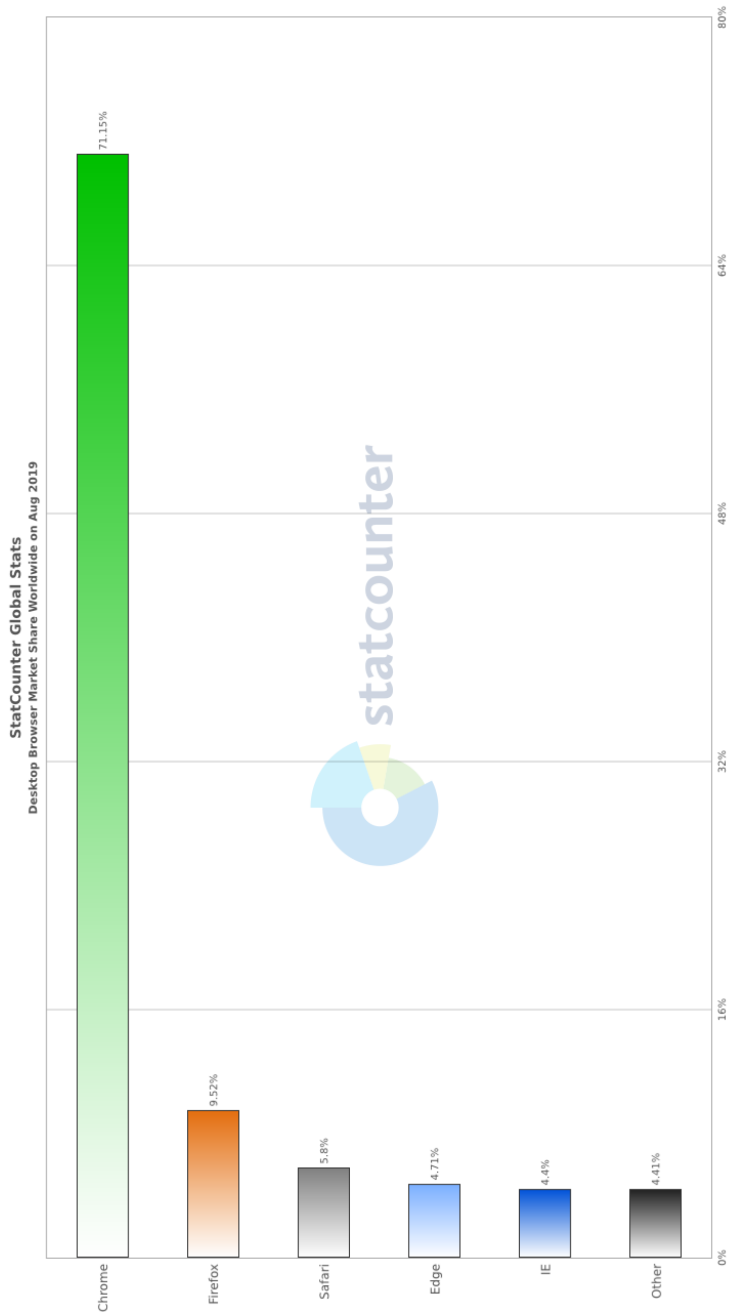

Figure A.3: Desktop Web Browser Usage Statistics for August 2019[216]

[216]Source Stab.

B Annex

B.1 Table of Contents of the USB Flash Drive

```
/
├── Live Demo
│   ├── Brave_Dongle_iPhone_6.MOV
│   ├── Telekom_Puls_OTG_1.jpg
│   ├── Telekom_Puls_OTG_2.jpg
│   ├── iPhone_6_Dongle.jpg
│   ├── iPhone_7_YubiKey_5Ci.jpg
│   ├── Chrome_OTG_Android_9.MOV
│   └── Brave_YubiKey_5Ci_iPhone_7.MOV
├── Web Authentication API Support Test
│   ├── Android
│   │   ├── Firefox
│   │   ├── Lightning
│   │   ├── Puffin
│   │   ├── Mint
│   │   ├── Firefox Klar
│   │   ├── Edge
│   │   ├── Brave
│   │   ├── UC
│   │   ├── Samsung Internet
│   │   ├── 460
│   │   ├── Firefox Preview
│   │   ├── Chrome
│   │   ├── Yandex
│   │   ├── Opera
│   │   ├── CM
│   │   ├── Stock Browser (Jelly)
│   │   ├── Opera Touch
│   │   ├── Pure
│   │   ├── Opera mini
│   │   └── QQ
│   ├── iOS
│   │   └── Brave
│   └── KaiOS
└── Internet Sources
```

Declaration of Academic Integrity

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are adequatley denoted as such.

Hamburg, September 27, 2019

Tim Brust

Theses

1. The status quo of password usage is terrible; often chosen passwords are re-used and weak.

2. Humans are the weakest link in a phishing scenario. Social engineering attacks can even target the service provider.

3. Multi-factor authentication is not phishing resistant, both the secret when setting it up and the second factor can be phished or stolen. Software solutions are more probable to be phished.

4. The biggest threat to multi-factor authentication is transportation, especially when using SMS or unencrypted e-mail traffic.

5. Multi-factor authentication can be made phishing resistant, but it requires more effort to do so by choosing different transportation algorithms.

6. The Web Authentication API is not yet usable enough nor widely adopted. This is in particular true for iOS, because it lacks support for API and the Internet Explorer is still widely used.

7. The user needs to be educated about passwords, the risk of password re-use, phishing, and how to protect themselves against typical internet threats.