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 20, 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.

Abstract

Kurzfassung

Keywords— authentication, multi-factor authentication, mfa, two-factor authentication, 2fa, fido, fido2, 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. 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 alongside 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 of course for proofreading this thesis.

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

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

Finally, 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 and understanding during this time.

Last but not least, I thank my parents for always supporting me and my decisions, as well as enabling me such a course of studies.

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, even forcing it into shutdown such as the bitcoin marketplace Mt. Gox.[2]

To make things worse, very few people are using multi-factor authentication (MFA) and even fewer a password manager 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 solution 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 it.

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

Nevertheless, the majority of the users are not using MFA at all, even if weak MFA 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 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 into MFA and the different authentication factors such as »knowledge, possession and biometrics« including the technical functionality, usability in web projects and respectively web browsers and their security threats alongside an introduction to the Web Authentication API. Those methods of authentication need to be mapped to actual forms of authentication such as passwords, security keys, and fingerprint sensors, that need to be again evaluated security-wise.

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 and 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 be used in conjunction. Besides that, questions such as

- What are the risks of not using MFA?
- Why are weak passwords and password re-usage such a big issue?
- Is there a 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 and understandable for end-users?

are taken into account and answered.

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 towards 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 if the Web Authentication API suits their needs.

1.4 Delimitation of this Thesis

Existing 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 solution, nor the Web Authentication API, but to take 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 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 such as phishing or Man-in-the-Middle (MITM) attacks.

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

In Chapter 7, 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 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,[5] but other sources also introduce or take new methods into account such as the location- or time-based authentication.[6] Therefore, this thesis accounts for them, too, and describes the methods in the following sections briefly including a diagram of an example authentication flow. A detailed analysis of the security, especially potential threats and vulnerabilities, follows in Chapter 3.

2.1.1 Knowledge

The most common 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 a good example for usage, e.g., in banking (ATM's, credit cards) or telephony (subscriber identity module (SIM)). The security relies on the fact that the knowledge method 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 are that the user may has to replace the used knowledge everywhere in case of re-use.[7]

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

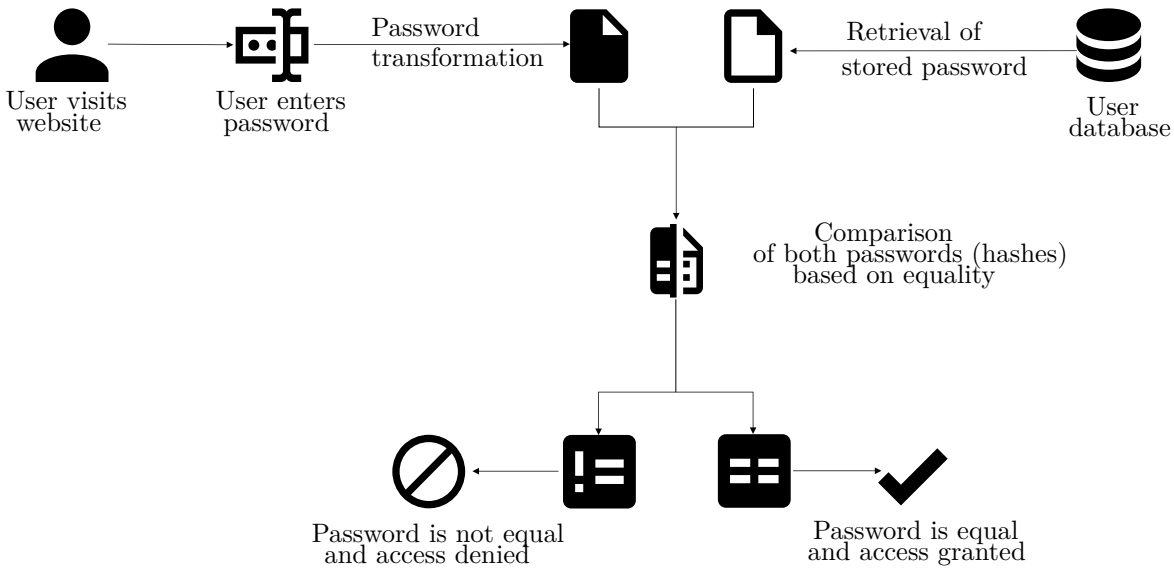


Figure 2.1: Exemplary, but simplified, authentication by knowledge flow[8]

Figure 2.1 shows a simplified authentication by knowledge flow. First, the user visits a website in this example 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 user is known in the database, then the stored (hash of) the password is retrieved and compared to the given 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 if the authentication is made by the 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 has« (physically). 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 a hardware (such as a YubiKey, a RSA SecurID or a smartcard) or software (e.g., a smartphone application) token. They can either be disconnected, connected (e.g., via USB or as a smartcard) or contactless (e.g., via near-field communication (NFC), Bluetooth Low Energy (BLE) or RFID). Sometimes these tokens contain a display itself that can show further

[8]Source: diagram by author.

information.[9]

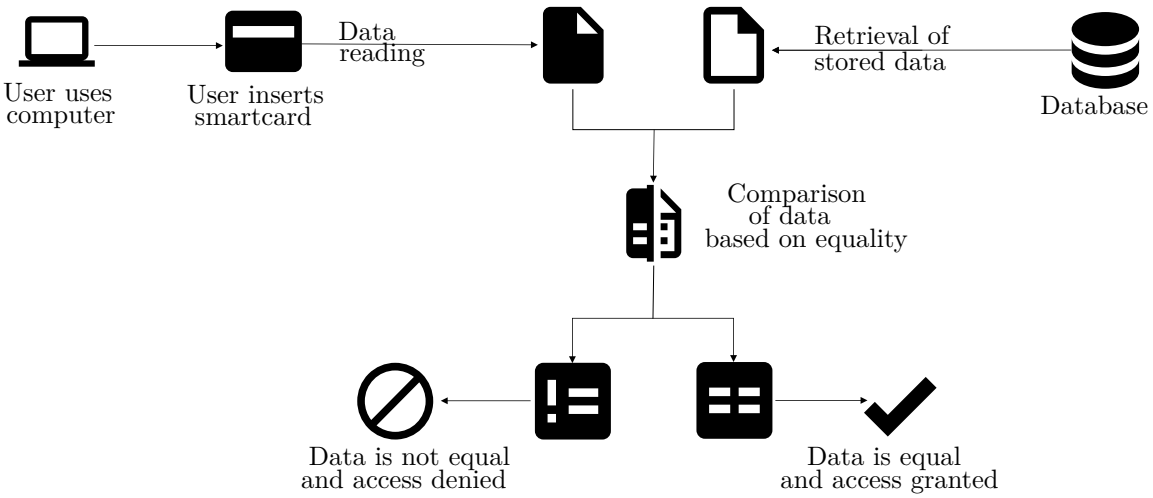


Figure 2.2: Exemplary, but simplified, authentication by possession flow[10]

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 are in possession of 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.[11]
These intrinsic factors are sometimes referred to as traits or inherence, too.[12]

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 false acceptance rate (FAR), i.e., an imposter is granted access, needs to be accounted for the usage.

[9] See Tod07, p. 24; DLE19; See MM17, pp. 8–11.
[10] Source: diagram by author
[11] See JRN11, pp. 30–34.
[12] 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.[13]

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 a compromised biometrics. The usability varies because of the quality of the used biometrics module, the chosen biometrics itself, and the availability of the biometrics.

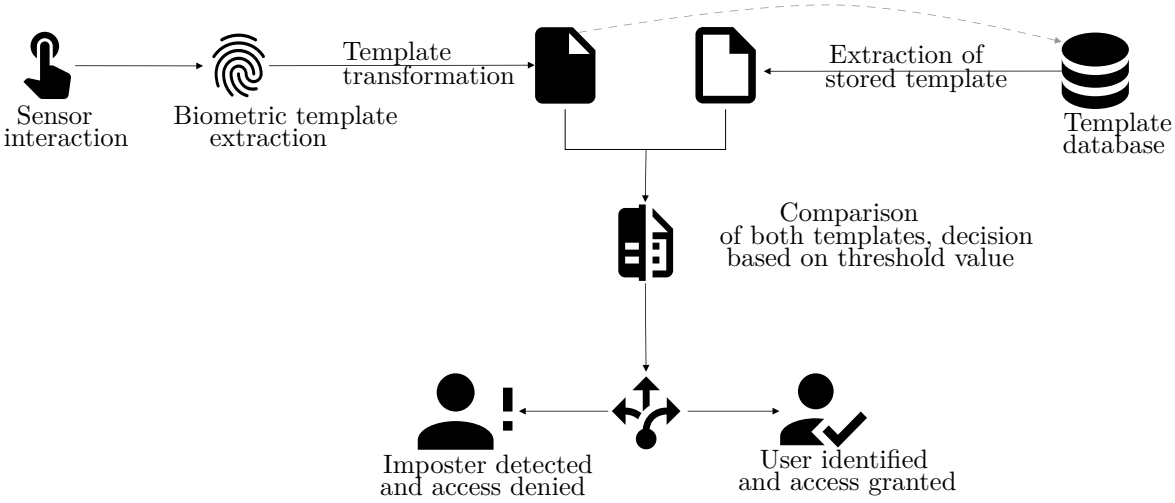


Figure 2.3: Exemplary, but simplified, authentication by biometrics flow[14]

Figure 2.3 shows an exemplary authentication flow using biometrics, in this case with a fingerprint. First, the user interacts with the sensor that 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 that determines how much of difference is tolerable finally decides if the authentication attempt can proceed or has to be aborted and access denied. If the authentication succeeds, the stored template can be updated in the database, as denoted by the dotted grey arrow.

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 location of the

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

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.[15]

Another form 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 instance, phishing attacks from abroad, because the access is granted or denied based on the time and usual time routines where, for instance, a user logs typically on.[16]

Further methods of authentication are, for example, 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, too.[17] Besides these methods, »something the user does« is another form of authentication. Examples range from keystrokes to online shopping behavior.[18]

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

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.[19]

[15] ZKM12; See Bis18, Chapter 13.9.
[16] See DRN17, p. 191.
[17] See Bra+06; See Sho14, pp. 278–279.
[18] See Shi+11; See Oud16.
[19] See Eck14, p. 545.

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.[20] The biometric passport authenticates against a reading device with an asymmetric challenge-response protocol. This security measure helps to identify cloned passports.[21]

2.2.2 Passive Authentication

In contrast to the active authentication process, in the passive authentication process the user is authenticated without 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 the keystroke or touch screen usage patterns. In comparison with active authentication, this process is more low-friction.[22] The biometric passport provides a way to calculate the integrity and authenticity from a reading device to improve the protection against forgery.[23]

2.2.3 Continuous Authentication

Further, the process of continuous authentication exists. 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. The authentication must happen in a non-intrusiveness way. Commonly used for continuous authentication are biometrics, such as the fingerprints, facial recognition, or keystroke patterns.[24]

Unfortunately, the term active authentication is often used to describe continuous authentication, too. To avoid confusion, solely term continuous authentication is used to refer to this process of authentication, while any mentions of active authentication refer to the process described in subsection 2.2.1.

[20] See DZZ14, pp. 185–186.
[21] See Eck14, p. 545.
[22] See DZZ14, p. 186; See XZL14.
[23] See Eck14, p. 545.
[24] See DRN17, pp. 236–238; See Fri+17.

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 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).[25]

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. That 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 for the client (e.g., a random number) and sends it to the client the challenge c
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 to the client for verification. A typical protocol is Fiat-Shamir.[26]

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 disclose the secret or parts of it to the other or third-parties. An example is the

[25] See Fen+17; See MM17, p. 501; See Cok+11, p. 4; See Cel+17, p. 100.
[26] See Was17, Chapter 13.6; See Eck14, pp. 489–491.

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.[27]

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 different 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.[28]

The naming of the chosen authentication or verification methods by companies is often confusing or difficult to understand. The terms used by companies vary from two-factor authentication (2FA), often just calling it 2FA,[29] to two-step-verification, sometimes written as 2-Step Verification, too.[30]

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.[31]

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.

[27] See Eck14, p. 492; See BKW14, Chapter 28.3.7; See FB17, pp. 769–770; See FFS88.
[28] See DRN17, pp. 186–188.
[29] See Sup19a.
[30] See Sup19b; See Pla; See Goo; See Mic19.
[31] See Gri17, p. 117.

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 Universal Authentication Framework (UAF) and Universal Second Factor (U2F).[32]

Another goal of the FIDO alliance is the 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. In addition, 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.[33]

2.7.2 Universal Authentication Framework

The Universal Authentication Framework (UAF) is FIDO's solution for a passwordless experience. 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 a unified user experience between different operating systems and clients.[34]

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

[32] See Eck14, p. 583; See Sch19, p. 17.
[33] See All14, pp. 6–7.
[34] See SM18, p. 249; See DRN17, pp. 197–198.

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.[35]

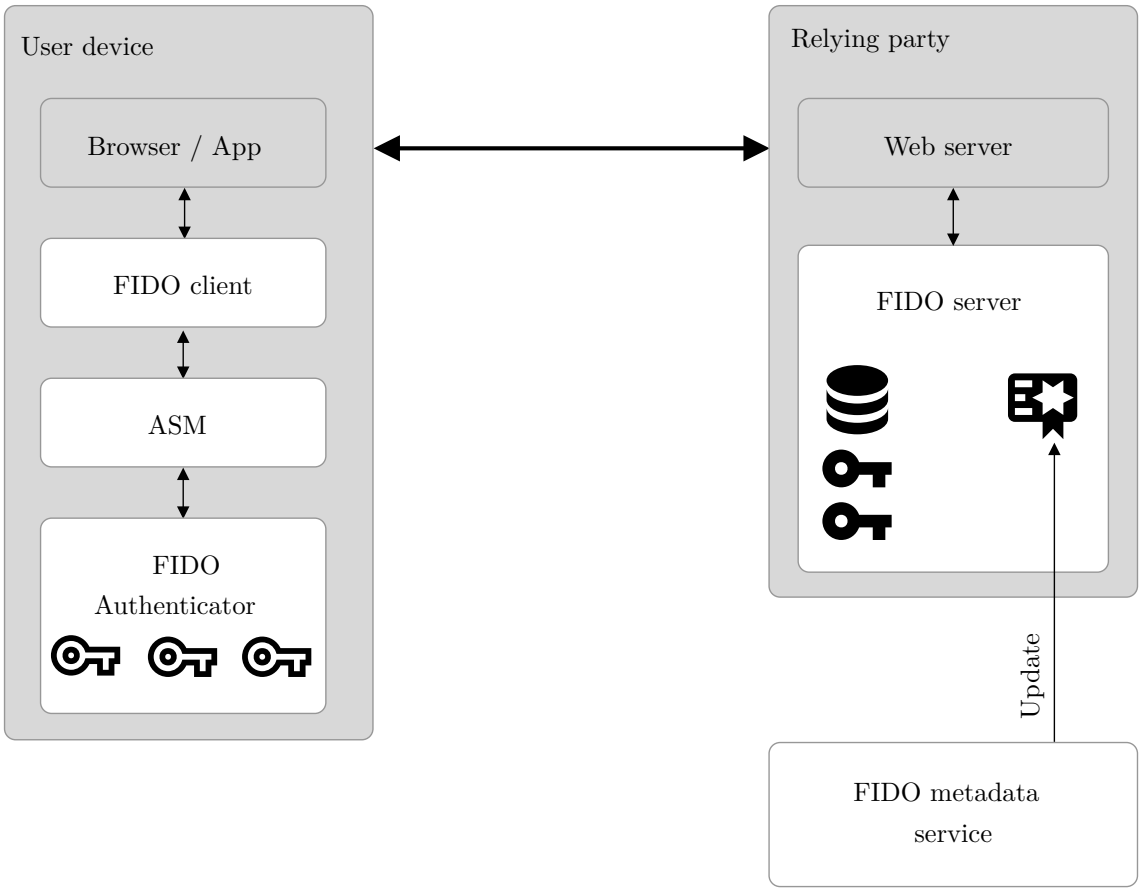


Figure 2.4: UAF architecture overview[36]

Figure 2.4 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

[35]See LJ15, p. 145; See LT17, p. 8.
[36]Source: diagram by author, based on Mac+17, p. 4.

service updates the database of approved, genuine, and certificated authenticators that the FIDO uses for authenticator validation. The protocol defines four different uses cases, which are explained in more detail below:[37]

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

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 is 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 for a valid FacetID is »https://admin.timbrust.de«, where »https://auth.wings.de« is an invalid FacetID, because the origin is different.[38]

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).[39]

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 a 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*

[37] See LT17, p. 4.
[38] See Pan+17, pp. 131–132; See LT17, pp. 17–19; See Lin17, pp. 3–4.
[39] See Pan+17, pp. 131–132; See LT17, pp. 17–19.

handle in its secure storage. The key handle consists of the *public key*, KHAccessToken hash, and username. In additio, might 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.[40]

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.[41]

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. In addition, 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.[42]

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. 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.[43]

When the authenticator receives the key handle, KHAccessToken and the hash of the FCP, this data can 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

[40] See LK17a, pp. 9, 16–17.
[41] See Ang18, pp. 12–13; See LT17, p. 22; See LK17b, p. 17.
[42] See HZ16, pp. 192–193; See LT17, p. 23.
[43] See Pan+17, pp. 132–133.

data to the RP. Finally, the RP can cryptographically verify the sent data by the signature and proceed with the authentication.[44]

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. The specifications list the authenticator display as optional, though. In case of the absence of a display, the ASM can offer the display functions as a software solution.[45]

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. It is sent by the FIDO client and ASM. To ensure the genuineness of the request, the client checks that the AppID matches the origin of the request.[46]

[44] See LK17b, pp. 20–21; See Ang18, p. 15.
[45] See Mac+17, p. 4; See SM18, p. 251.
[46] See LT17, p. 31; See Mac+17, p. 7.

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/Registration/Enrollment

A more general threat is the registration, initialization or enrollment of the authentication. 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 therefor compromised. A computer virus could also intercept a USB connection from a security key, both when registering the device and while using it.[47]

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. Mobile phones are subjects to trojans, too, enabling 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. With the recent rise of the Internet of Things (IoT) devices, e.g., the mentioned security cameras might be compromised by an attacker.[48]

3.1.2 Transmission

Further, the chosen transmission channel is an important fact to take into account. Entering a password on an unencrypted website (Hypertext Transfer Protocol (HTTP)) enables network sniffing because the password is transmitted in cleartext

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

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 which enable the attacker to perform an MITM attack and even steal the passwords that are sent via an encrypted channel.[49]

The risk also applies to other authentication methods, too. A manipulated USB or smartcard port could copy the data on a security key or smartcard, or a tampered sensor can capture the fingerprint of a user. SMS traffic is at high risk of being intercepted or eavesdropped (e.g., the transmission of PINs or transaction authentication numbers (TANs)) as well as unencrypted e-mail traffic containing, e.g., temporary passwords or TOTPs.[50]

3.2 Knowledge

»Passwords are both the bane and the foundation of [...] security«,[51] yet the most used authentication method remains knowledge, in IT, especially passwords.[52] 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 difficulty 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.[53]
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[54]

(b) using something easy to guess or knowledge that is tied to a personal object, such as birthdays or names[55]

[49] See She+19, p. 518.
[50] See GB17, p. 103; See Dot19, p. 58; See MM17, p. 6.
[51] Har05, p. 206.
[52] See Bid06, p. 424.
[53] See Las18, pp. 7, 9.
[54] See You18, p. 8; See Löw16, p. 14; See Las18, p. 7.
[55] See Fri19; See And08, p. 34.

(c) writing down the username and passwords, e.g., on a piece of paper that is accessible easily for others, storing PINs, e.g., in the briefcase or saving an unencrypted file on their computer or smartphone[56]

This enables an attacker to steal the login credentials of a user easily. Written down post-it notes enable any physical attacker to steal the credentials. It might be captured by a camera or a colleague looking over the shoulder, too. Leaving an unencrypted file on the computer enables computer viruses and trojans to send the file to an attacker. Mobile devices are affected, too, since, e.g., mobile trojans exist, too.
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.[57]

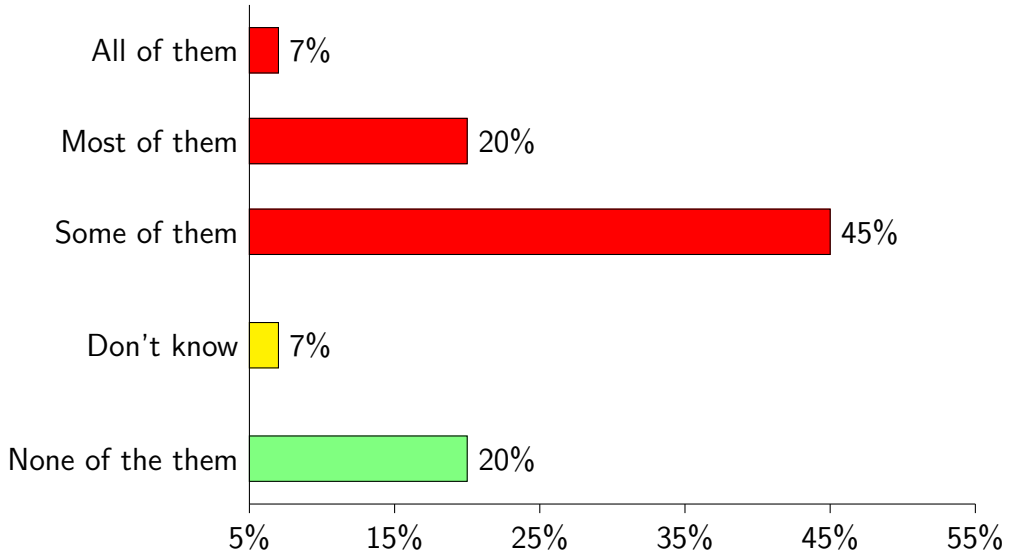


Figure 3.1: Percentage of online accounts sharing the same password in the United States in 2018[58]

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. While there is only a marginal difference between

[56]See Fri19; See You19, p. 6.
[57]See Kis19, Chapter 4.1.
[58]Source: You18, p. 8.

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 more different passwords. Other surveys strengthen the observation that millennials are re-using passwords more often.[59]

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«, enabling 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 the security questions wrong than honest and correct or to provide custom ones, when allowed by the service. Additionally, it is not uncommon to be able to guess the partner's password.[60]

Unfortunately, in the history it was thought that a forced change of password increases the security and a lot of enterprises, policies, and standards still contain sections regarding the enforced password rotation.[61] However, studies show that the 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.[62]

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 (best case)

(d) encrypted

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

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, or in case of an e-mail provider breach, re-issue a new password with the »forgotten password« mechanism.[63]

Even if the password is hashed, but not salted, it renders the credentials at risk. Weaker hashing algorithms such as Message Digest (MD) version 5 or Secure Hash Algorithm (SHA)-1 might be broken in the future, but besides that, if it is a weak password, too, the hash might already be reversed. Having 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 vector.[64]

Obtaining a password hash is often enough for an attacker to gain access to further user accounts, by reversing easy hashes and then automatically trying to gain access with these credentials on other websites. This form of attack is called »credential stuffing« and a subform of the brute-force attack.[65]

A better 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 decrease the and harden the brute-force, dictionary, and rainbow attacks.[66]

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 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 comparing with the stored hash when authenticating a user. An example of the effects the salt and pepper have is shown in Table 3.1. The beginning of identical hashes are marked

[63] See Sho14, p. 277.
[64] See Tho+17, p. 1425; See Bid06, pp. 427–430; See And08, pp. 56–57.
[65] See Hun17; See Tho+19, p. 1565; See Zab19, Chapter 5.5.
[66] See LM16, pp. 32–34; See BB17, pp. 130–131; See Gra+17, p. 15.

bold. The table shows that just relying on a pepper is not sufficient to hide users that share the same password.[67]

Both techniques can be combined and each of them independently strengthen the generated hash of a potential weak password by increasing its length and complexity, therefore reducing the risk of a collision.

user	salt	pepper	password	resulting SHA-1 hash
tim			Wings	**daaf17**ba041ff1a2184a2b02 ↩ 4a9f83442a7ca3ee
caro			Wings	**daaf17**ba041ff1a2184a2b02 ↩ 4a9f83442a7ca3ee
tim	c261012e		Wings	2e35d46e345fd77317e54735 ↩ 86f15d681e89b9a3
caro	5f40720d		Wings	ee229d4f4c8f3a9137f98e7f ↩ 8b5d46f26d9c9b8d
tim		18e6c63a	Wings	**2e6536**c7a16feaaca34b6b83 ↩ a311a0880ad0f80e
caro		18e6c63a	Wings	**2e6536**c7a16feaaca34b6b83 ↩ a311a0880ad0f80e
tim	c261012e	18e6c63a	Wings	7d707f1b6dd8f811fabd17e3 ↩ 11e01d35015ce9cd
caro	5f40720d	18e6c63a	Wings	33c1b9d955d6d0b7f4208719 ↩ 07e822ccbe708249

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

3.3 Possession

The primary risks 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 so, e.g., a keycard to open a door can be used by the attacker, too.

Another usage implication is that it must be carried with the user and can be forgotten, which makes the authentication impossible if no access to the possession is possible and no backup or different authentication methods are available. A different risk 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.[68]

[67] See LM16, pp. 33–35; See Gra+17, p. 15; See Man96, p. 173.
[68] See Sho14, pp. 263–264.

Especially possessions that use wireless transmissions such as BLE, NFC, or RFID can be copied even over some distances. For instance, an attacker could copy credit cards in crowded places such as trains or buses.[69]

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 happened to be vulnerable, it can cause severe problems depending on the number of keys that need to be replaced.[70]

3.4 Biometrics

In contrast to possession and knowledge, the biometric trait cannot easily be stolen, but it can be copied, e.g., the fingerprint from high-resolution photographs or face models 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.[71]

Further implications are that the biometric characteristics can change over time or be temporarily unavailable because of injuries. While some 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 will never be 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 high value decreases the usability and increases the FRR.[72]
Traits such as facial recognition must also be usable with, e.g., different amounts of facial hair, hairstyles, or with and without glasses.[73]

Another high risk is data privacy and security. Over 50% of the users fear about data usage, both legitimate and abusive, and collection of their biometrics, yet

[69] See KSM14.
[70] See DRN17, p. 18; See BLP05; See Wes19b.
[71] See FKH14; See FSS18; See Mar13, e199; See Kre14.
[72] See JRN11, pp. 13–17, 52.
[73] See JRN11, p. 98.

the majority of the user states that biometrics is 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.[74]

However, the primary threat remains the difficulty of replacing a compromised biometric template. A password or a security key can be changed or replaced, but 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 instance, only a hash of the fingerprint and not store the *image* of the fingerprint itself.[75]

Further, 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.[76]

3.5 Further Methods

A high risk of location-based authentication is the spoofing of the actual location by an attacker. An attacker can choose different attack vectors, 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.[77]

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.[78]

[74] See Kes18, p. 8.
[75] See Sho14, p. 266.
[76] See Tod07, p. 37.
[77] See Har05, pp. 138–145; See Yua05, Chapter 4.5.3; See Eck14, pp. 115–116, 133.
[78] See Mal+15.

4 Multi-Factor Authentication

In this chapter, a variety of different MFA solutions are described in detail. MFA 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, the first factor is always assumed as knowledge, i.e., in the majority of the use cases, passwords. Therefore, further knowledge-based authentication methods are not taken into account in this chapter.

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 is not officially and often different defined.[79]

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 various security threats exist, which are independent from the specific authentication method. These make user accounts, for instance, vulnerable to theft, impersonation, or phishing.
In order to decrease or even eliminate these threats, the user needs to deploy additional security measures that are explained in this chapter.

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.[80]

[79] See Nat18, p. 47; See Sic19, p. 11.
[80] 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 works if the different factor is not transmitted over the first channel, which might be intercepted. Besides that, it does not work if the different factor is, for instance, entered on a phishing website, too.[81]

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 the generated OTP.

To fully understand how the OTP works, first, the basics and origins, especially the underlying message authentication code (MAC), have to be introduced. 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 TOTP, are introduced. Both extend the keyed-hash message authentication code (HMAC).

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 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 alongside with

[81] See Gra+17, p. 17; See Bid06, p. 441; See BB17, p. 140; See GB17, p. 106.

the message. Upon retrieval of the message, the recipient calculates the checksum (MAC) themself. If it differs, then the message has been manipulated, or there might have been a faulty transmission. 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).[82]

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.[83]

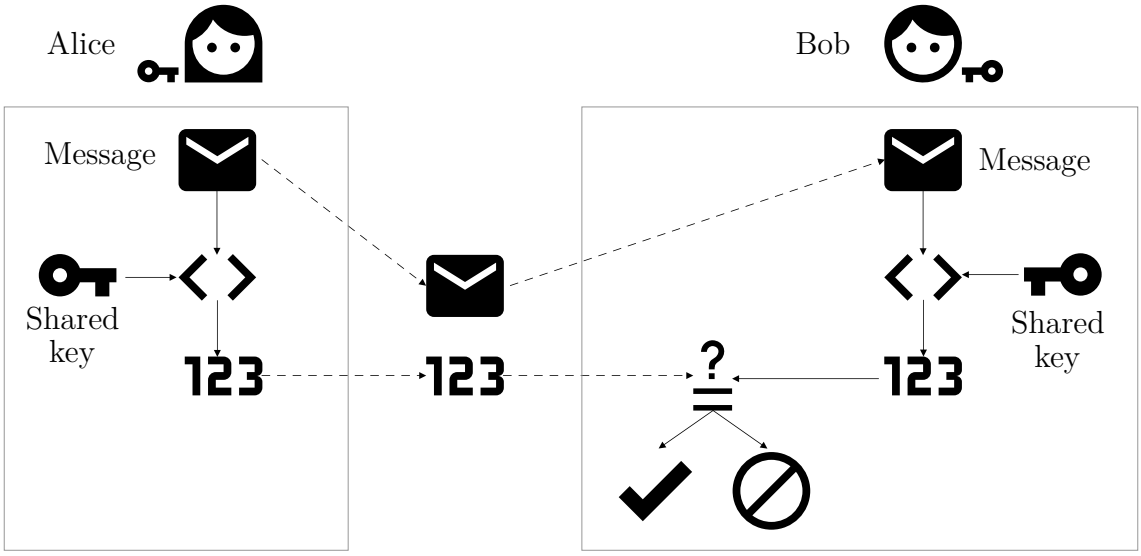


Figure 4.1: Message authentication code used to protect a sent message[84]

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 wants to send 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 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.

[82] See Bid06, p. 565; See And08, pp. 163–168; See Eck14, pp. 391–393.
[83] See ST08; See Inf19; See ISO11a; See ISO11b.
[84] Source: diagram by author

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.[85]

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 used in network protocols. Additionally, the MIC does not prove authenticity since an attacker can modify the message and re-generate the MIC of the modified message.[86]

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, given that both the encrypted traffic, but also the encrypted malware is increasing.[87]

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.[88] Besides authentication the HMAC is, e.g., used in TLS and JSON Web Token (JWT) to ensure data authenticity and integrity.[89]

Mathematically, the HMAC is defined as follows:

[85] See PP11, p. 320.
[86] See Tod07, pp. 60–62.
[87] See Wel15, p. 100.
[88] See KBC97; See ST08.
[89] See DR08, p. 14; See JBS15, p. 8; See TC11, pp. 3–4.

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

Where K is the shared secret key, 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.

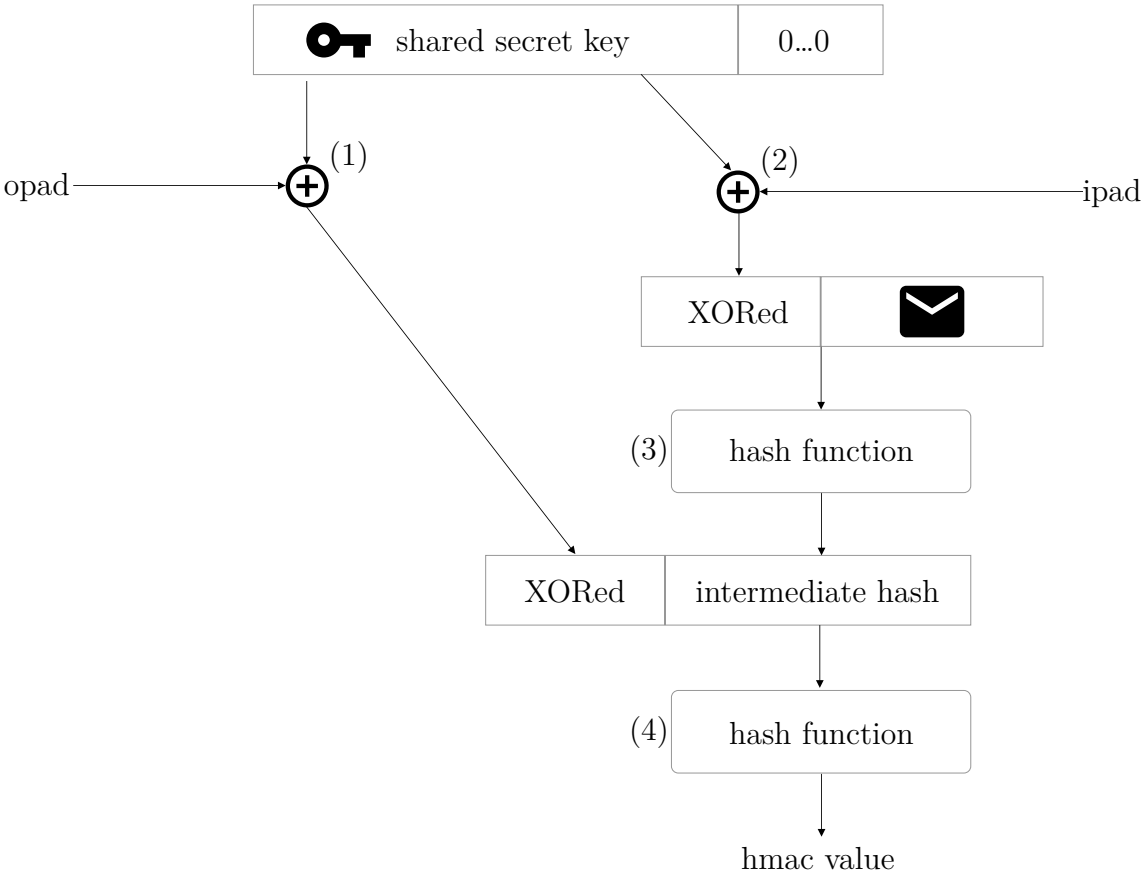


Figure 4.2: Visualization of the HMAC algorithm[90]

[90]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, for example, length and chosen alphabet, of the secret key. The best-known attacks against HMAC remain the brute force and birthday attack. Further security anlysis is performed in section 5.2.[91]

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 and joint efforts between the Internet Engineering Task Force (IETF) and the Initiative for Open Authentication (OATH). It is an algorithm for the generation of OTPs, in contrast to the HMAC not an algorithm for message authentication and integrity. 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 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, but it is also possible to replace the cryptographic hash function, although the implementation will not comply with the RFC anymore.[92]

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

[91] See Bis18, Chapter 10.4.1; See Sta17, p. 398; See BCK96, pp. 3, 10–13; See PO95.
[92] 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. Extracting 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 that the server only increases the counter on successful authentication, the out of sync scenario can occur. 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.[93] 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 counterfeit brute-force attacks. Further analysis is done in section 5.2.

Additionally, the HOTP allows bidirectional 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.[94]

4.3.4 Time-based

The time-based one-time password (TOTP) is again an extension of the HOTP that is time-based instead of counter-based. It is a joint efforts of the IETF and the OATH, too, resulting in standardization in RFC 6238.[95]

Mathematically the TOTP is defined equally to the HOTP:

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

[93] See SM18, p. 236; See Bis18, Chapter 13.5.1.
[94] See HS17, p. 716; See MRa+05, p. 14.
[95] See MRa+11.

The only difference is that the counter is substituted by T, where T is a computed time value derived from 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.

More formally 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 introduced 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 an increased 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.

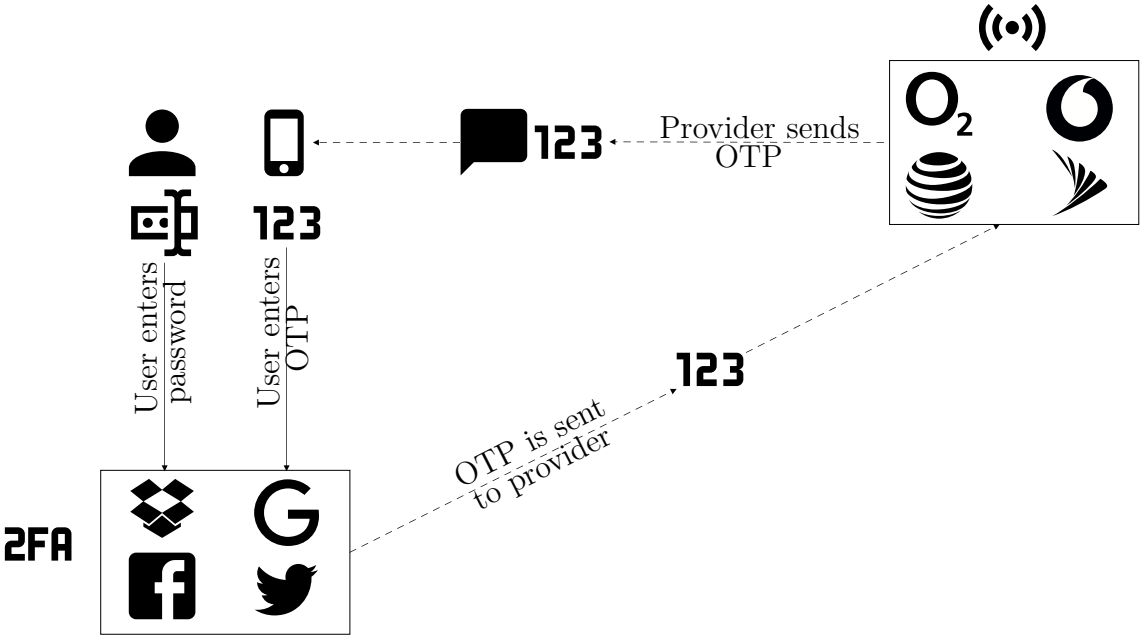


Figure 4.3: Exemplary MFA flow[96]

[96]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; 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; 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 TOTP and HOTP, the Swedish company Yubico developed a proprietary OTP protocol, too. It is available for all their produced and sold YubiKeys. The produced OTP is a 44-characters long string which is constructed by using Advanced Encryption Standard (AES) with 128-bit 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.[97]

The constructed OTP can be structured into the following groups:

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

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 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.[98]

[97] KS12; See Jac16, pp. 84–86.
[98] See Yub15, pp. 8–9, 33–34; See ORP13, pp. 209–210.

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 key.[99]

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 and 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 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 defines in FIPS 201-2 the usage of smartcards for Personal Identity Verification (PIV) of federal employees.[100]

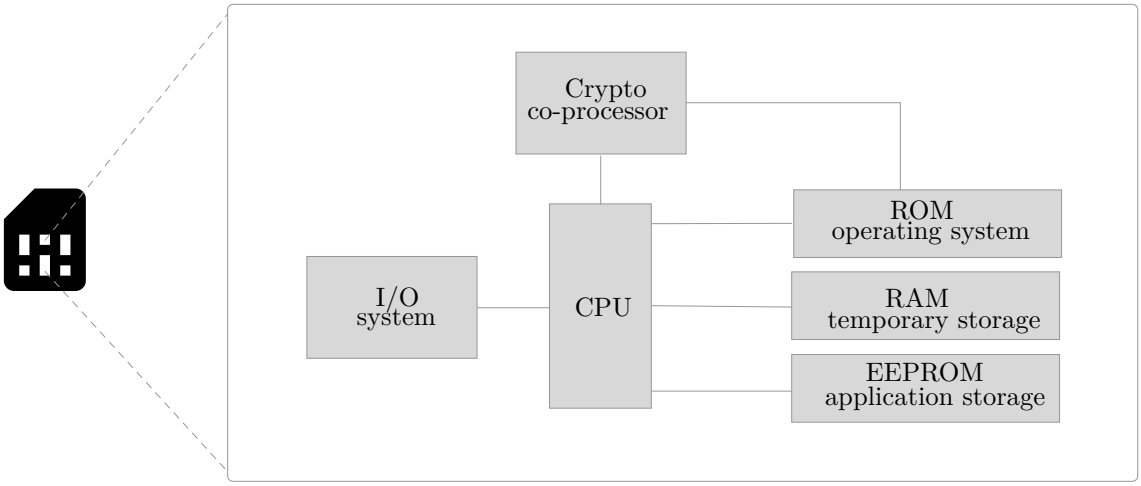


Figure 4.4: Typical smartcard architecture[101]

[99] See Yub12, pp. 8–9.
[100] See Eck14, pp. 525–527; See ISO11c; See MM17, pp. 6–9; See ST13.

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

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.[102]

In the aspect of usability, the smartcard always requires dedicated hardware, either external or built-in, a smartcard reader in order to use the smartcard as an authentication method. While especially enterprise notebooks contain a smartcard slot, e.g., mobile phones do not. With the chip card interface device (CCID) protocol which defines a USB protocol 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, opening the development possibilities for smartcard application further.[103]

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 define in, e.g., the NIST FIPS 140-3 standard.[104] This 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.[105]

As many security tokens support the U2F, these security tokens are not part of this section. Instead, the underlying concepts of the U2F API are explained and

[101]Source: diagram by author, based on Fer15, p. 33; MM17, p. 228.
[102]See Tod07, p. 34; See MM17, p. 228.
[103]See MM17, p. 65; See Eck14, p. 539.
[104]See ST19.
[105]See BKW14, Chapter 28.4.3.

analyzed in Chapter 6, since the Web Authentication API originated from the U2F specification.

4.5.1 RSA SecurID

The RSA SecurID exists in several variants, both hardware and software token. 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. Additional form-factors, such as credit card sized variants, exist, too. Each token contains a burned in seed, 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.[106]

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.[107]

4.5.2 YubiKey

Besides a proprietary OTP algorithm, the company Yubico is best known for their physical security tokens, the YubiKey. A variety of tokens exist, ranging from different USB-A and USB-C variants or 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« are FIPS 140-2 certified, i.e., their cryptographic modules are approved by the U.S. government, and are usable for PIV.[108]

[106] See Eck14, pp. 479–480; See Han+07, p. 296.
[107] See WPR16, pp. 3–6; See LB10, p. 49.
[108] 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 the method of »authentication by push notifications«, where a user just 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).[109]

4.6 Universal Second Factor

The Universal Second Factor (U2F) is the second open standard developed by the FIDO alliance prior to the Web Authentication API. It explicitly defines a second factor for the password-based login flow. It is like the UAF 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 orl for instance, an 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:[110]

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)

[109] See HS17, p. 717; See MS14, p. 111; See ULC19, p. 60; See DRN17, pp. 222–223; See HJT07, p. 3.
[110] 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).[111]

Because U2F relies on the web in contrast to the UAF protocol, browser support has to be taken into account. Due to the fact, that U2F is superseded by FIDO2, the browser support of U2F is not of interest for this thesis.

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.[112]

Registration

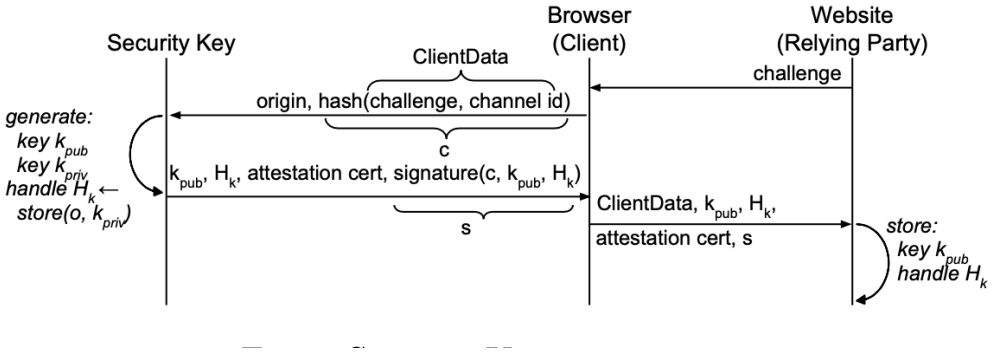


Figure 4.5: U2F registration process[113]

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

[111] See RKM16, p. 3; See BEL17, p. 3.
[112] See Bra+19, p. 4.
[113] Source: diagram by author, based on PRW18, p. 69; Lan+17, p. 428.

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.[114]

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 wrapped, i.e., encrypted, back to the client. In addition, 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.[115]

Finally, the client forwards the registration and client data to the RP, which can cryptographically verify the data with the signature and attestation certificate.[116]

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

Listing 4.1: Example U2F registration code

```
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)
});
```

[114] See BBL17, pp. 4–5; See Lan+17, p. 431.
[115] See BEL17, pp. 4–5; See PRW18, p. 70.
[116] See RKM16, p. 3.

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.[117]

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 a 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.[118]

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'
};
```

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

Authentication

The authentication process involves signing a challenge from RP with the corresponding private key. This enables the RP to cryptographically verify the the response with the saved public key. 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 verify the received AppID to the origin it communicates with.[119]

Afterwards, 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.[120]

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 by each usage, and the ECDSA signature over the values user presence, counter, challenge parameters and the hashed AppID back to the client.[121]

The client forwards registration data, client data, and key handle to the RP, which in return can verify the signature data with the stored public key.[122]

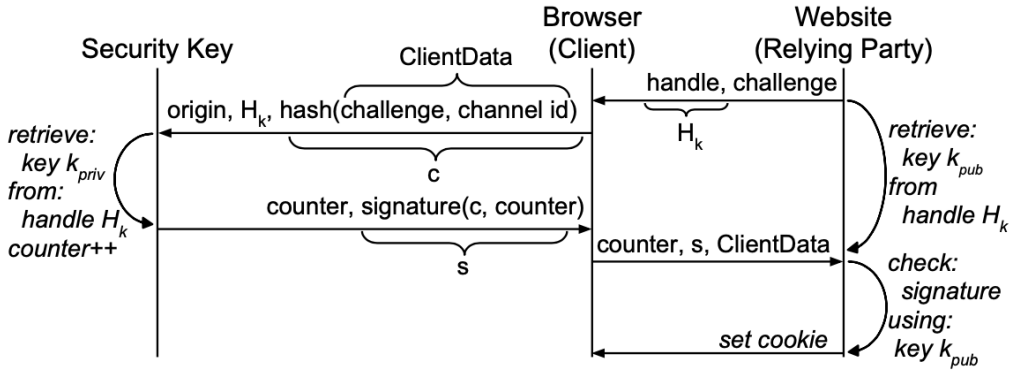


Figure 4.6: U2F authentication process[123]

[119]See RKM16, p. 3; See BBL17, p. 6.
[120]See BEL17, p. 6.
[121]Lan+17, p. 431; See BEL17, p. 7.
[122]See LM16, p. 118.
[123]Source: diagram by author, based on PRW18, p. 70; Lan+17, p. 428.

The Listing 4.3 shows an example, high-level JS API signing process. The RP sends the associated key handle, the protocol version, AppID and authentication challenge to the client. Listing 4.4 shows the generated response by the U2F token. The response object and decoded client data shows the Listing 4.4, where the clientData can also be generated beforehand by the client.[124]

Listing 4.3: Example U2F authentication code

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

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

[124] See BBL17, p. 3.

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'
};
```

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.

5 Security of Multi-Factor Authentication

5.1 Introduction

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

Figure 5.1 shows the already in Figure 4.3 explained MFA flow using a TOTP. In this figure the scenario is expanded with a phishing attack. The user visits a phishing copy of the website they want to use and do not recognize this. Because he 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; first factor) and TOTP (possession; second factor) to successfully login to the victims account and effectively bypassing the MFA solution.

5.2 One-Time Passwords

In this section the security of both HOTP and TOTP are being analyzed.

5.2.1 Algorithm

pros

1. Collisions in MD5 or SHA1 are no problem, already stated/analyzed in the RFC

[125]Source: diagram by author

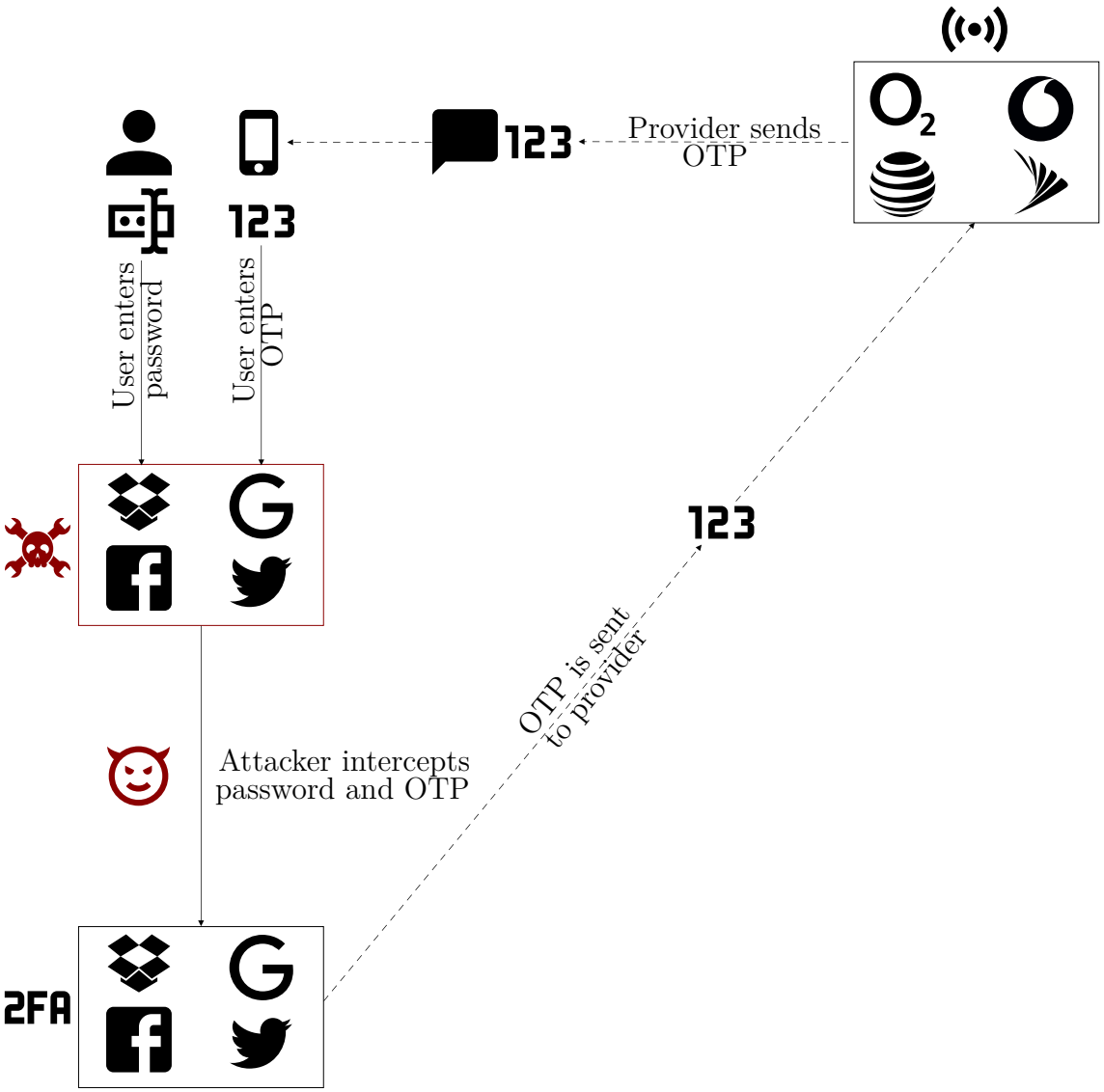


Figure 5.1: Exemplary 2FA phishing of an OTP[125]

cons

"Just an algorithm"

1. synchronization
2. invalidation
3. nobody knows how the algorithm is implemented
4. Differences (e.g. Steam - only 5 digits, limited Alphabet)
5. Brute Force if server does not limit
6. Not phishing resistant

As both the HOTP and the TOTP are based on the HMAC algorithm by building the OTP over the HMAC function of the secret key and the counter with a truncation, the underlying HMAC algorithm needs to be evaluated.
The important part here is the chosen cryptographic hash algorithm. Mostly SHA-1 is used, since it's the default of the RFC. 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 the collision resistance of the chosen cryptographic hash algorithm is not important for the security of the OTP generation those algorithms do not expose a threat.
The BSI lists these algorithms as secure for HMAC[126]

Citations:[127]

It is more important that the algorithm is implemented correctly, in the past e.g. Google did not issue OTP values with a leading zero. Besides that, the minimum length of the OTP values are six digits, meanwhile the RFC supports up to 10.
For example Steam, decided to use a different alphabet and character length.

A theoretical vulnerability is to use the time sync offset feature because it enables an attacker to use a token that's much longer valid than it should be. (as discussed in section xx - time sync/drift)

5.2.2 Transportation and Generation

Given that the generation of the OTP is considered secure the more important region to analyze is the transportation of these OTP. In this section the transportation mediums SMS, e-mail, and apps for the OTP generation are considered.

SMS

The biggest advantage of SMS as a transportation medium is every mobile, ranging from an old Nokia to a new iPhone XS, is capable of receiving SMS. All major mobile phone operation systems come with an SMS application pre-installed, so no external apps are required.
SMS are around 1999 and highly accepted and easy to use.

[126] Inf19.
[127] Ste+17.

While there are some key advantages with SMS transportation it also comes with a lot of downsides. Besides the cost aspect of SMS traffic, both for the sender and potentially for the receiver due to roaming fees, too, the current state of SMS traffic is considered insecure.
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]

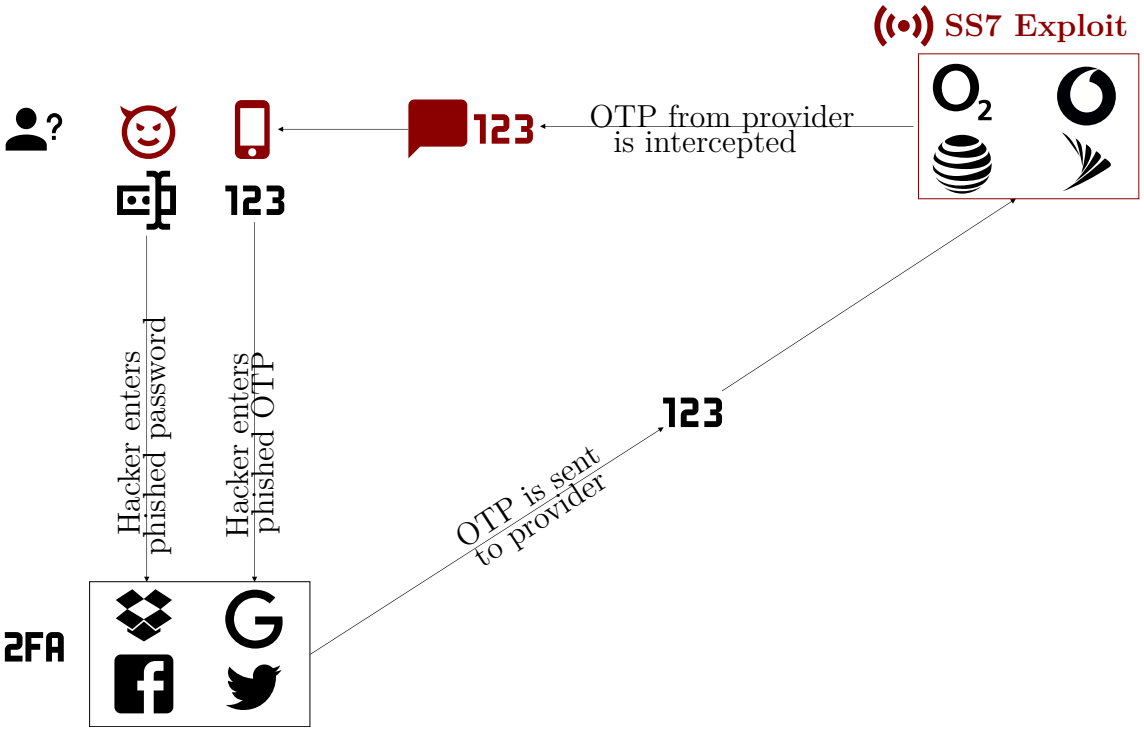


Figure 5.2: SS7 exploit to phish an OTP used in MFA[129]

Figure 5.2 again shows the described MFA flow using TOTP. In this scenario the attacker is still able to phish the TOTP designated for the user. The figure shows that the attacker uses an exploit in the SS7 network. This allows them to intercept all incoming SMSs.

[128] Wel17; HO17; Puz17.
[129] Source: diagram by author

Another negative aspect of SMS transportation is the routing. Many companies rely on third-party providers in order to send the SMS to the user. Often these providers like name some are using countries where SMS are very cheap, but on the other hand the SS7 security measures like SMS home routing and not enforced. This results in a higher security risk of the SMS being compromised while reaching the user. Also, the third party providers are given access to the OTP which enables the risk of a malicious insider because the security measures might be weaker than the original company.

In contrast to the web and e-mail the user is not very aware of phishing attacks in the SMS context. Studies however show that a new technique called forward phishing is already in use. In this scenario the attacker sends the victim a (spoofed) SMS from the fakes service provider to reply with the OTP code for security measures.[130]

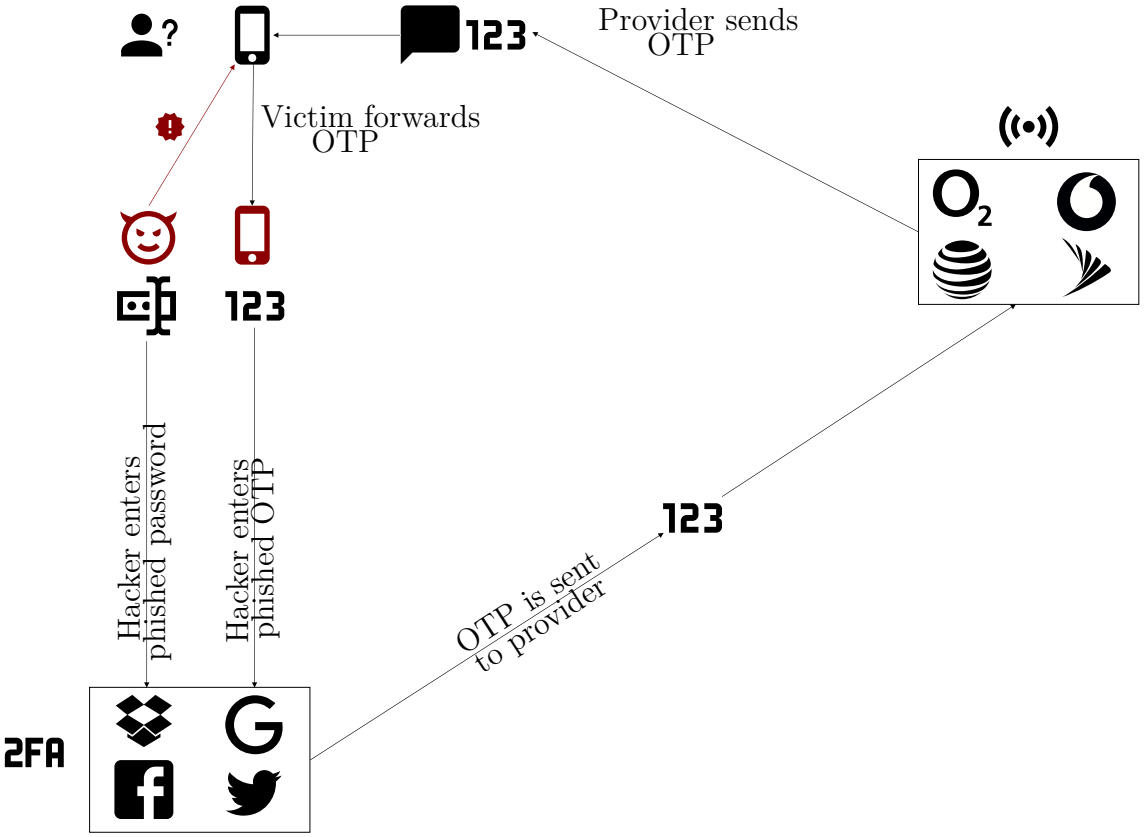


Figure 5.3: Verification Code Forwarding Attack (VCFA) to phish an OTP used in MFA[131]

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 the account is protected by MFA,

[130] Jak18; Sia+17.
[131] Source: diagram by author

the user receives a verification code via SMS or smartphone. The attacker send now a fake SMS to the victim, stating that the service has detected unusual activity and that the user should reply with the just received verification code in order to stop this activity and proof they are the account owner. Of course the attack now has access to the OTP, too, since they tricked the user into forwarding their code.

Especially for Android there exists multiple SMS trojans which are capable of intercepting the SMS, too.

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 on time. These non-influencable, external factors strengthen the fact that SMS are not a wise choice as the transportation medium.

Additional weaknesses of SIM are the attack risks of

(a) SIM cloning

(b) SIM swapping scam

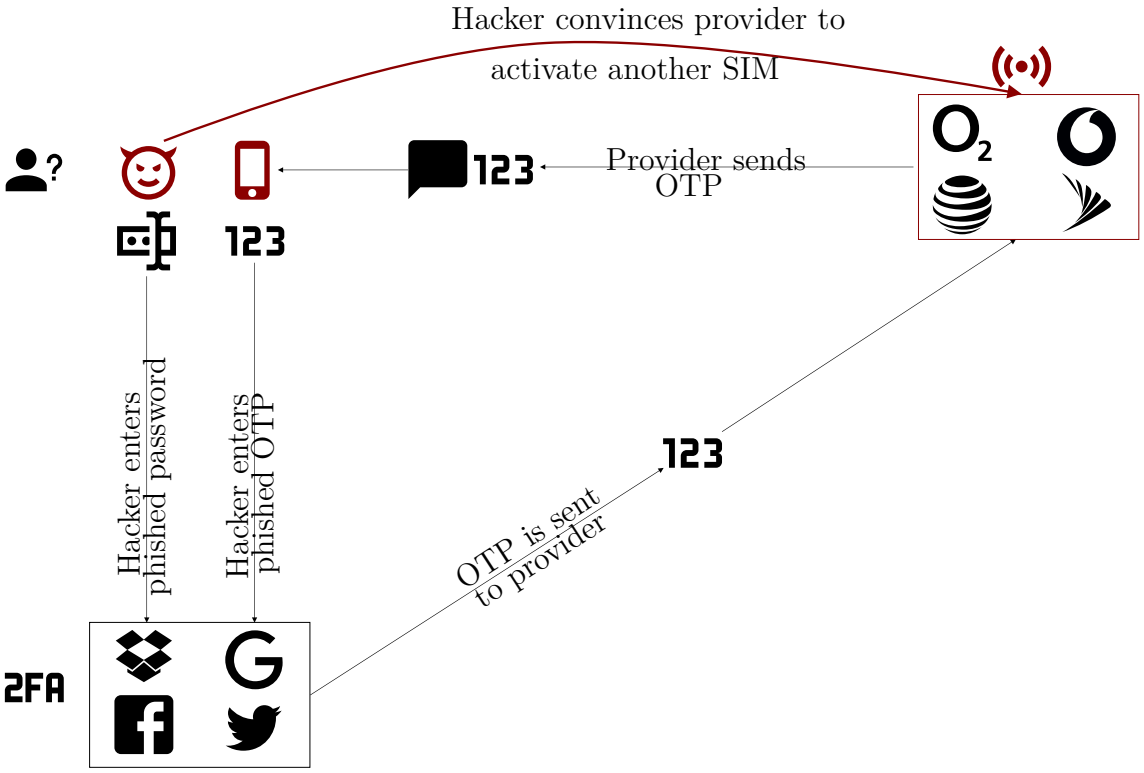


Figure 5.4: Social engineering used to phish an OTP in MFA[132]

[132]Source: diagram by author

Figure 5.4 shows presented MFA flow using TOTP again, but in this case another phishing scenario. 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 victims phone number and receive the SMS with the TOTP, too, which enables the attacker to successfully complete the MFA flow. Yet another variant which is technically more complex, but feasible, is the SIM card cloning. This allows the attacker to intercept the TOTP, too, by registering the phone number twice.

[ST08] [Gra+17] (Twitter CEO hack) [Con19]

Given all these facts SMS transportation should be avoided at all costs,[133] since there are multiple flaws in the SS7 network itself and the process how the SMS reaches the user. It's also not resistant against phishing or mobile phone trojans.

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 offers 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.

Furthermore, the app solution is cheaper because no transaction fee is due for the sender or receiver. It also eliminates 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, later access the generated OTP, or access the app's database where the secrets are stored.

Besides that, the app itself can contain vulnerabilities such as Authy suffered from a vulnerable backend that could be exploited to bypass the 2FA. Additionally, the

[133] Jak18.

user has to ask if the app is legtimite and from a trustworthy source and have faith in the app that it keeps their data safe.[134]

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 acceptable and do not require a user to give aways their cellphone number. While it eliminates the need to rely on external services and given the fact that the user already registered an account with the corresponding site, it is assumed the user owns an e-mail account.

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 secret. In 2019 ?? % of all e-mail traffic is still unencrypted. A malware on the desktop or smartphone can intercept incoming e-mail and even delete the message without the users knowledge.

Besides that, e-mails re-introduce the problem of delayed reception of the OTP since 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

5.3 Security Tokens

[Wes19b] [Bra19a] [ORP13] [Kan19]

5.4 Overall Comparison of Threats

The following sections 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

[134] See Ste19; See Hom15.

[135] 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	✓	Interception, phishing, brute-force, unavailability
	Smartcards	✓	Cloning, theft, damage, oblivion, loss, side-channel attacks
	Security Keys	✓	Cloning, theft, damage, oblivion, loss, side-channel attacks
Biometrics	Fingerprints	(✓)	Replica, forgery, replay attacks, damage, unavailability of the sensor
	Facial scan	(✓)	
	Iris scan	(✓)	

Table 5.1: All threats compared[135]

the possession methods of authentication are used as an additional authentication factor, given the fact that passwords are the de-facto standard in the internet as the first factor. Biometrics can be used, too, but in practice there exist very few applications that use biometrics as an additional factor.

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 the threat of interception followed by physical theft.

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«.[136] Each public key credential is scoped to the relying party (RP), i.e. can not be re-used for other websites (RPs). The authenticator has the duties and responsibilities to create, store, and access these credentials. These action always require the users consent. The user agent, i.e., browser performs the communication with the authenticators and RPs to persevere 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.[137]

6.2 History and Evolution

The Web Authentication API is an outcome of joint efforts between the FIDO alliance and 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.[138]

The first specification version of the U2F was the starting point for the development of the Web Authentication API in a 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.[139]

[136] See Bal+19, Abstract.
[137] See Bal+19, Abstract, Chapter 1.2.
[138] See Eik19, p. 24.
[139] 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 efforts 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. 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.[140]

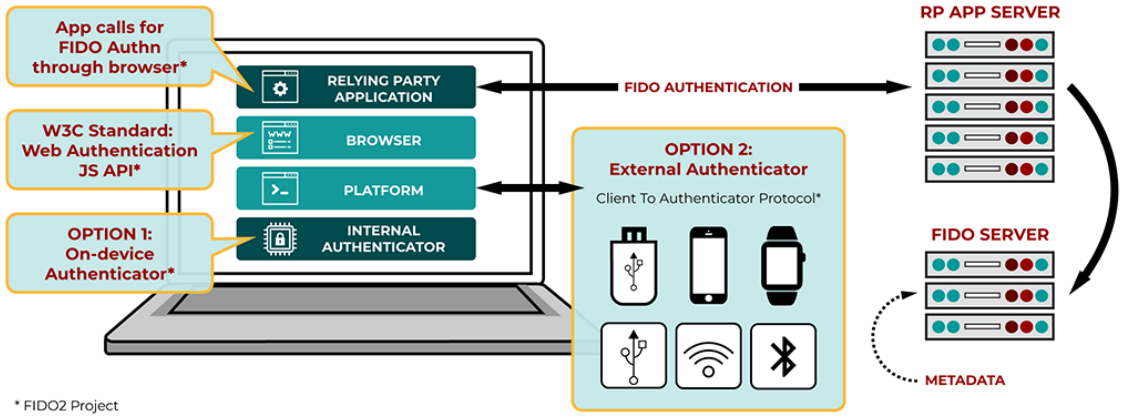


Figure 6.1: FIDO2 architecture overview[141]

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

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
3. transport-specifc binding

[140] See SM18, p. 254; See GH18, p. 3.
[141] Source: https://fidoalliance.org/specifications/, last access on 09/14/2019

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 encoding called Concise Binary Object Representation (CBOR) that is suitable for, e.g., the transport over BLE, because plain text string and JavaScript Object Notation (JSON) objects can be too big for a transport over such protocols. The transport-specific bindings define the required transformation and bindings in order to comply with the transport protocol specifications.[142]

An important 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.[143]

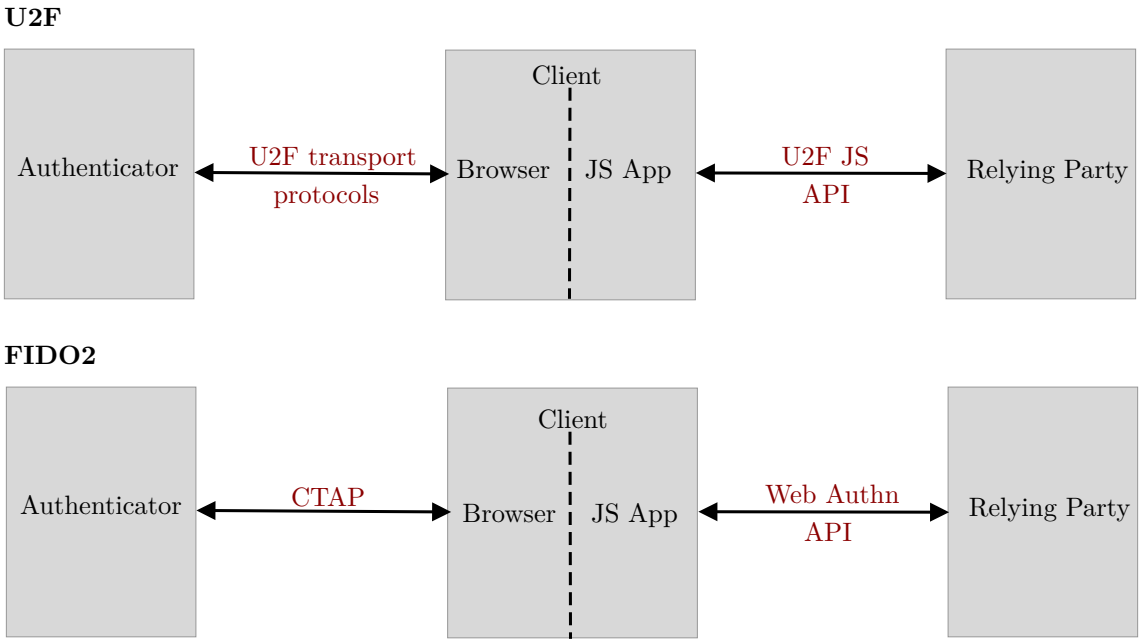


Figure 6.2: Architectural differences between U2F and CTAP2[144]

Registration

The registration procedure calls the method *authenticatorMakeCredential*. The input parameters are identical to the ones defined in the higher level Web Authen-

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

tication API and are further explained in subsection 6.3.3. Upon reception of the required data, the authenticator first checks if the *excludeList* if a credential ID is listed that is already registered on the authenticator. This prevents that a user registers multiple accounts. 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.[145]

After that, the authenticator generates the attestation object consisting of the authentication data, which contain 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 the private attestation key of the authenticator token.[146]

Authentication and transaction confirmation

Authentication is performed by using the method *authenticatorGetAssertion* of the CTAP protocol. Again 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 allowed 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 credentials.[147]

The authenticator generates an assertion signature over the received hash of the client data and the authenticator data which 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 of the Authenticator Attestation Globally Unique ID (AAGUID), credential ID and the credential public key.[148]

Factory reset

As the UAF protocol, but in contrast to the U2F specification, the CTAP does define a method to completely factory reset the authenticator in order to de-register every

[145] See Bra+19, p. 9.
[146] See Bra+19, p. 9; See Bal+19, Chapter 8.
[147] See Bra+19, pp. 11–13.
[148] See Bal+19, Chapter 6.4.1.

user account on it. To avoid 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.[149]

6.3.3 Web Authentication API

The API defined in the Web Authentication API is actually an extension of the Credential Management API, which is another API in development, but currently in a draft state.[150] The Credential Management API defines *navigator.credentials* property with the *create* and *get* methods and has the goal 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.[151]

Authentication and registration in the context of the Web Authentication API are a special form of network protocols, called a *ceremony*. A ceremony is the concept of extending a network protocol to include human nodes, too. This allows the specification to take the human factor into account, too.[152]

The Web Authentication API is backwards compatible to the U2F compatible, thus making every security token that is usable for U2F compatible with the Web Authentication API, too. However, a crucial 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.

Registration

A new key-pair for the registration with a RP is generated by calling the asynchronous method *navigator.credentials.create* with an object that contains the required *publicKey* property. The publicKey object contains the ID and name of the RP and the user information consisting of a username, optional display name and 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

[149] See Bra+19, p. 26.
[150] The W3C standardization process can be viewed in the section A.1
[151] See Bal+19, Chapter 1; See Wes19a, Chapter 1.1.
[152] See Ell07, p. 2.

publicKey object contains a random challenge generated by server to prevent replay attacks.[153]

Besides that, with the public key credential parameters (*pubKeyCredParams*) array it is possible 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 denotes that the RSA algorithm should be used in conjunction with SHA-256. The order in the array denotes the preferred but also accepted fallback algorithms.[154]

Additionally, the RP can define a list of credentials to exclude (*excludeCredentials*) to, e.g., prevent the user from creating multiple key-pairs that are registered with the RP. Furthermore a *timeout* value can be specified in which the operation should succeed or fail.[155]

Moreover, the RP can set the *authenticatorSelection* which defines the requirement if a user needs to be verified (»required«, »preferred«, »discouraged«), the option if the credentials need to be stored on the authenticator (»requireResidentKey«), and the authenticator attachment modality, i.e., if the authenticator should be platform specific or a cross-platform (roaming) authenticator.[156]

Finally, the *attestation* 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. 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 how to generate the attestation certificate. The authenticator may use a per-origin certificate authority (CA) to protect the users privacy.[157]

[153] See Bal+19, Chapter 5.1.3.
[154] See Bal+19, Chapter 5.3, 11.3.
[155] See Bal+19, Chapter 5.4.
[156] See Bal+19, Chapter 6.2.1.
[157] See Bal+19, Chapter 5.4.6.

Listing 6.1: Exemplary Web Authentication API registration

```
const publicKeyOptions = {
  challenge: 'Wings2019', // normally 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 with the previously introduced parameters. The publicKeyOptions payload can now be passed to the authenticator by calling the 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 of the server provided challenge, it's origin and the string type *webauthn.create*.

Upon reception of the *attestationObject* and included credential ID in it, 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 consists of an ID, both as a string and binary reprenstation, the type that is always *public-key* and the *response* object. The response object is constructed from the returned *attestationObject* and the client data.

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*.

In contrast, the attestation sent from the authenticator is much bigger. On the first hierarchy 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

public point on an EC and the AAGUID if attestation is. not set to none, otherwise 16 zero bytes.

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+cUzTPcmk50tfj0YGH3nYE=',
        y: 'W1OKIxfc6pIE/ANeTD7MqnNVjBXd0L7We9xZ3Hx6nD8='
      }
    }
  }
};
```

Authentication

An authentication ceremony is started by calling the method *navigator.credentials.get*.

Listing 6.5: Exemplary Web Authentication API authentication

```
const publicKeyOptions = {
  challenge: 'Wings2019Auth', // normally a random string from
  ↪  the server in binary form (Uint8Array)
  allowCredentials: [
    {
      id: 'BShOCQ2c32dv4aqyy3oWmcu_9s4tz0VIob81U5tg [...]',
      type: 'public-key',
      transports: ['usb', 'ble', 'nfc']
    }
  ],
  timeout: 6000
};

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

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 browser, and if they support the API. If so, the table shows the version which initially added support for the Web Authentication API alongside with the release date. The following subsections will explain the web browser support more detailed.

The global browser support as of August 2019 is 68%.

[158] Sources: BK18; JT18; Dav18; Ger18; Jon19, a detailed analysis of Android browsers is available on the CD in the appendix.

	Web browser	Supported	Version	Release Date
Dekstop	Chrome	✓	67	May 2018
	Firefox	✓	60	May 2018
	Opera	✓	54	June 2018
	Internet Explorer	✕	-	-
	Edge	✓	18	November 2018
	Safari	(✓)	(13)	-
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[158]

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 configuration option though and was enabled for all users with the release of Edge 17 in November 2018.[159]

[159]See Jac19, p. 112.

Browsers such as Opera, Vivaldi, or Brave, and upcoming Edge versions that are all based on Chromium, the browser and source code behind Google's Chrome browser, have support for the Web Authentication API, too.[160]

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 5% of all desktop browser users and remains supported for the operating system Windows 7, 8.1 and 10.[161] 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 browser.

Safari added support for the Web Authentication API feature in December 2018 but only for the preview variant of the browser, called the Safari Technology Preview. It is expected to be available for all users with the release of Safari 13 in mid to end September. The support is limited to USB-HID enabled authenticators though and only available for macOS Mojave and Catalina and yet unknown if older macOS version will receive an update to Safari 13.[162]

Besides that, Windows 10 also added support for MFA by incorporating the technology described in the FIDO standard. This allows biometric authentication with, e.g., fingerprints when a reader is available or to use the facial recognition technology or iris scans. This feature is called »Windows Hello«. The credentials are only stored locally and are protected by asymmetric encryption. Besides biometric authentication Windows Hello also supports PINs. The TPM stores this PIN. The Windows Hello technology can be used in desktop browsers, i.e., delegating the Web Authentication API functionality to the OS.[163][See][6]fido-whitepaper-amd

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, iOS completely lacks support for the Web Authentication API. Even though in the iOS 13 beta versions 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 browser yet.

[160] See Kis19, Chapter 7.1.
[161] See Sup19c.
[162] See Dav18.
[163] See Bio16.

The only ray of hope is that the Brave browser for iOS incorporated support for security key »YubiKey 5Ci« which enables U2F and the Web Authentication API for iOS by using an Apple certified Lightning accessory. A successful authentication with the YubiKey 5Ci and the Brave browser for iOS shows Figure 6.3. iPad devices with an USB-C connector currently do not work yet.[164]

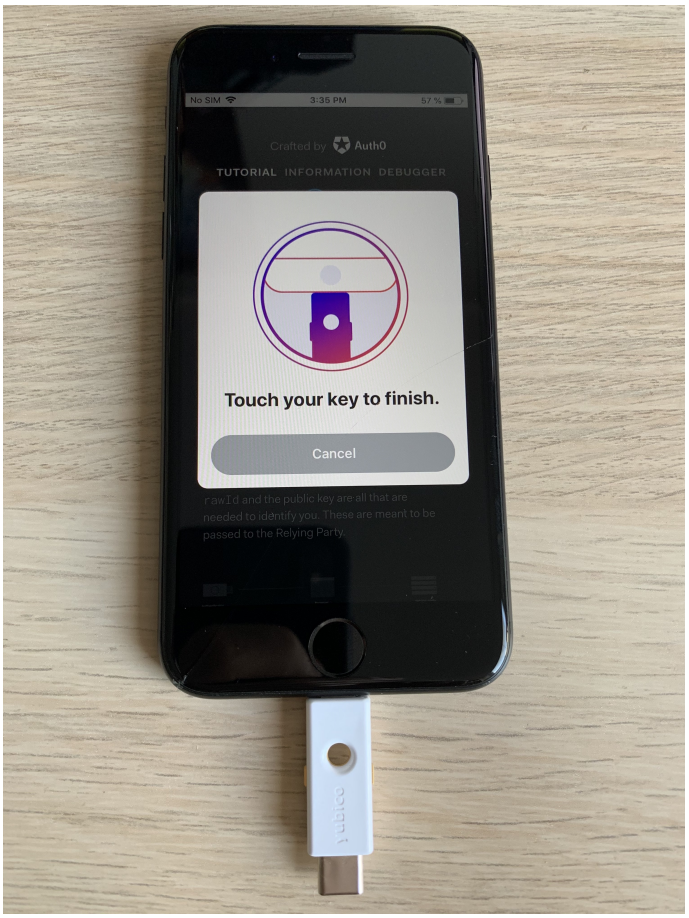


Figure 6.3: Successful use of the Web Authentication API with the Brave browser on an iPhone 7 with the YubiKey 5Ci[165]

However, Figure 6.4 shows the try to use an existing U2F YubiKey with a lightning dongle in the Brave browser on a website that offers support for the Web Authentication API. While the U2F YubiKey has power, Brave does not recognize it. Neither is it usable. Safari does not show an overlay either. Further, the YubiKey 5Ci is not recognized in the Safari browser, too.

It has to be noted though, that other Android browser vendors need to implement the functionality themselves. Other geographic regions use a variety of different

[164]See Bra19b; See Bra19c; See Mah19.
[165]Source: author's own photograph
[166]Source: author's own photograph

Figure 6.4: Failed try to use the Web Authentication API with the Brave browser on an iPhone 6[166]

browsers, e.g., the UC Browser, 360 Security Browser, Mint Browser from Xiaomi or the QQ Browser from Tencent. Neither they nor browsers such as Samsung Internet, Opera (mini) for Android, Edge or the Android Stock browser are currently supporting the Web Authentication API. The current Firefox for Android (codename »Fennec«) browser is based on Chromium, too, while in contrast the desktop browser is powered by Mozilla's own browser engine called »Gecko«. A new Firefox for Android browser, currently called Firefox Preview, which uses a mobile compatible version of Gecko is in development (codename »Fenix«), too, which yet lacks support for the Web Authentication API. However, Android offers support for FIDO2 as an API

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

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, while

in contrast the website designs the traditional login masks and forms. In order to maintain a 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« can not be reached on macOS or iOS yet. While the desktop varian of Safari at least contains an opt-in support, the CTAP is only implemented for USB-HID based tokens. Additionally, Firefox only supports USB-HID based authenticators on other operating systems than Windows 10, too.

While analyzing the market situation it was observed that the availability of FIDO U2F for end consumers is quite limited. Only a handful vendors offer security keys, quantity-wise BLE are the rarest keys, it seems they are succeeded by NFC.

Often token that contain a vulnerability in their firmware need to be replaced, making this both a heavy usability loss, as well as a security issue.

1. multi keys?!
2. durability of the keys
3. Implementation is hard (e.g. no SpringBoot, but many OSS. Often not certified)

6.4 Security Aspects

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. If the Web Authentication API is used with a security key, then the same risk of damage, loss or theft exist. Besides that, if the security key itself is not protected (by e.g. fingerprints) an attacker can easily gain access to an account if he steals or copies the authenticator. Built-in key stores in devices such as smartphone or laptops, do not protect against theft, either. In the past pyhsical security keys suffered from lack of randomness in their random number generator (RNG).

Security-wise the Web Authentication API had little attention yet. A first security analysis showed some weaknesses. These are the following ones, described more detailed below:

1. Support for RSA Public Key Cryptography Standards (PKCS)#1 v1.5 padding
2. The usage of Elliptic Curve Digital Signature Algorithm (ECDAA) without point compression
3. Randomness of ECDAA
4. Chosen ECDSA curves

It has to be noted though, that these security considerations are only from one source.[167]

The first problem of the Web Authentication API are the registered COSE algorithms in section 11.3. A support for RSASSA-PKCS#1 v1.5 is explicitly required, making it vulnerable for the over twenty years known »Bleichenbacher attack«.[168]. Further the ECDAA does not specify 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. Additionally the usage of the 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. Random values for the secret key in ECDSA expose a threat, too. It is recommended to use determinist nonces.

6.4.2 Mitigations

As mitigations against potential 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 keep the potential lack of randomness when not using a CSPRNG.

[167] See Sta18.
[168] See Ble98.

7 Comparison

Things to consider:

1. Do I need two or more (backup?) keys
2. pyhsical theft - relevant? how is the key protected, since it's the master one
3. cross device sync - possible?!
4. TODO password safe?!

8 Conclusion and Outlook

OAuth 2.0, KERERBOS, radius based, LDAP, AD, OpenID Connect, SAML, IoT, Login with Apple

1. not yet ready for the world (iOS, IE, only USB)
2. two keys recommended - expensive
3. user needs to be educated!
4. mfa is safe most of the time - sms and email and http not!

Bibliography

[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.

[Bid06] Hossein Bidgoli. *Handbook of Information Security: Threats, Vulnerabilities, Prevention, Detection, and Management*. Vol. 3. Handbook of Information Security. Hoboken, NJ, USA: Wiley, Jan. 2006. ISBN: 978-0-471-64832-1.

[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.

[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, Virginia, 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.

[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.

[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+14] 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.

[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.

[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-Hbner 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 '17. 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.

[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.

[Inf19] Federal Office for Information Security. "Cryptographic Mechanisms: Recommendations and Key Lengths." In: *Technical Guideline TR-02102-1* 1 (Feb. 2019).

[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.

[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.

[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.

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

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

[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.

[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, 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.

[Mah19] Jan Mahn. "FIDO2 für iOS." In: *c't* 20/2019 (Sept. 2019), pp. 84–84. 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.

[MM17] Keith Mayes and Konstantinos Markantonakis. *Smart Cards, Tokens, Security and Applications*. 2nd ed. Cham, Switzerland: Springer International Publishing, May 2017. ISBN: 978-3-319-50498-8. DOI: 10.1007/978-3-319-50500-8.

[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.

[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, 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.

[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.

[Sch19] Jürgen Schmidt. "Abschied vom Passwort." In: *c't* 18 (Aug. 2019), pp. 16–18. 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.

[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.

[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.

[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.

[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: 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.

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

[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

[All14] 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).

[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).

[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).

[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).

[Goo] 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).

[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).

[Lin17] 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).

[LK17a] Dr. 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).

[LK17b] 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).

[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] Dr. 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).

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

[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).

[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).

[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).

[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 Listings

List of Tables

Acronyms

Symbols

2FA Two-Factor Authentication

A

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

B

BLE Bluetooth Low Energy
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

E

EC Elliptic-Curve
ECC Elliptic-Curve Cryptography
ECDAA Elliptic Curve Digital Signature Algorithm
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

J

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

K

KRD Key Registration Data

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
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

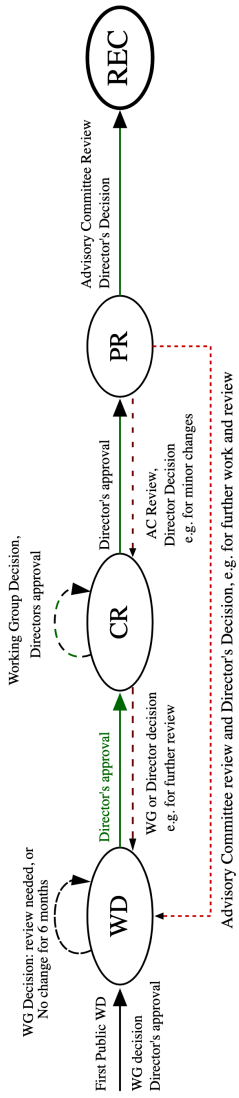


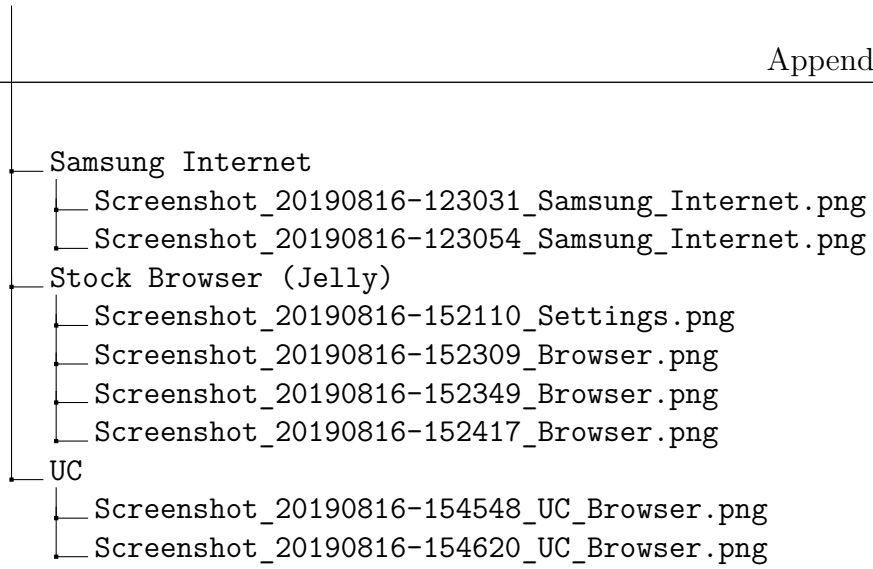
Figure A.1: W3C standardization process[169]

[169]Source: RR19, Chapter 6.7.

B Annex

B.1 Table of Content of the CD-Rom

```
/Volumes/CWniwQ7mThQP_0/Masterthesis/LaTeX/cd_rom
└── Web Authentication API Support Test Android
    ├── 360
    │   ├── Screenshot_20190816-154301_360.png
    │   ├── Screenshot_20190816-154306_360.png
    │   └── Screenshot_20190816-154438_Settings.png
    ├── Brave
    │   ├── Screenshot_20190816-153730_Brave.png
    │   ├── Screenshot_20190816-153901_Brave.png
    │   ├── Screenshot_20190816-153910_Brave.png
    │   └── Screenshot_20190816-153936_Brave.png
    ├── Chrome
    │   ├── Screenshot_20190816-121238_Google_Play_services.png
    │   ├── Screenshot_20190816-121245_Google_Play_services.png
    │   └── Screenshot_20190816-121304_Chrome.png
    ├── Edge
    │   ├── Screenshot_20190816-135319_Edge.png
    │   ├── Screenshot_20190816-135334_Edge.png
    │   └── Screenshot_20190816-135347_Edge.png
    ├── Firefox
    │   ├── Screenshot_20190816-121429_Firefox.png
    │   └── Screenshot_20190816-122820_Firefox.png
    ├── Mint
    ├── Opera
    │   ├── Screenshot_20190816-155037_Opera.png
    │   ├── Screenshot_20190816-155044_Opera.png
    │   └── Screenshot_20190816-155108_Opera.png
    ├── Opera mini
    │   ├── Screenshot_20190816-155738_Opera_Mini.png
    │   ├── Screenshot_20190816-155750_Opera_Mini.png
    │   └── Screenshot_20190816-155815_Opera_Mini.png
    └── QQ
        ├── Screenshot_20190816-121712_QQ.png
        ├── Screenshot_20190816-121734_QQ.png
        └── Screenshot_20190816-122522_Settings.png
```


Samsung Internet
Screenshot_20190816-123031_Samsung_Internet.png
Screenshot_20190816-123054_Samsung_Internet.png
Stock Browser (Jelly)
Screenshot_20190816-152110_Settings.png
Screenshot_20190816-152309_Browser.png
Screenshot_20190816-152349_Browser.png
Screenshot_20190816-152417_Browser.png
UC
Screenshot_20190816-154548_UC_Browser.png
Screenshot_20190816-154620_UC_Browser.png

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 20, 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.

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.

6. The Web Authentication API is not yet usable enough nor widely adopted; this is especially true because iOS lacks support for it 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.