FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

The end of the password era

Towards password-less authentication based on enhanced FIDO

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Mathias Bachl, BSc
Matrikelnummer 1125616

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Mitwirkung: Univ.Lektorin Dipl.-Ing. Dr.techn. Katharina Krombholz-Reindl

Wien, 22. Dezember 2016

Mathias Bachl	Edgar Weippl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

The end of the password era

Towards password-less authentication based on enhanced FIDO

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Mathias Bachl, BSc
Registration Number 1125616

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Assistance: Univ.Lektorin Dipl.-Ing. Dr.techn. Katharina Krombholz-Reindl

Vienna, 22nd December, 2016

Mathias Bachl

Edgar Weippl

Erklärung zur Verfassung der Arbeit

Mathias Bachl, BSc
Schopenhauerstraße 21/24, 1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. Dezember 2016

Mathias Bachl

Danksagung

An erster Stelle möchte ich mich bei meiner assistierenden Betreuerin Katharina Krombholz-Reindl bedanken. Ihre zahlreichen Tipps haben sich während der Entstehung dieser Arbeit immer wieder als sehr wichtig erwiesen. Natürlich möchte ich mich auch bei meinem Betreuer Edgar Weippl für die unkomplizierte und flexible Abwicklung dieser Diplomarbeit bedanken.

Zu guter Letzt möchte ich allen Menschen danken, die mich während meines Studiums in den unterschiedlichsten Lebenslagen unterstützt haben.

Acknowledgements

I would like to thank my assisting advisor Katharina Krombholz-Reindl for giving me important and helpful hints regarding numerous aspects of this work, and of course my advisor Edgar Weippl for his uncomplicated and flexible supervision throughout the entire time of creation of this thesis.

Last but not least, I would like to thank all people who supported me in all kinds of regards during the entire time of my studies.

Kurzfassung

Sicherheitsvorfälle, die auf kompromittierte Passwörter zurückzuführend sind, sind ein allgegenwärtiges Problem. Entsprechende Lösungen für sicherheitskritische Anwendungen existieren zwar, sind aber für viele Dienste aufgrund zahlreicher Einschränkungen im Hinblick auf deren Benutzerfreundlichkeit und Einsetzbarkeit ungeeignet.
Unter der Vielzahl an Ansätzen, die eine Lösung dieses Problems versprechen, scheint der *FIDO Standard* der vielversprechendste Kandidat zu sein. Einige Größen der IT-Industrie haben bereits begonnen, diesen Standard in ihren Produkten zu implementieren.
Dennoch gibt es nach wie vor einige zu lösende Probleme, bevor FIDO das *Ende des Passwortzeitalters* einläuten und der Standard-Authentifizierungsmechanismus für das Internet of Things werden kann.
FIDO bietet einen hohen Sicherheitsstandard, da sowohl ein Gerät im Besitz des Benutzers (mittels asymmetrischer Kryptographie) als auch der Benutzer gegenüber dem Gerät selbst (mittels Biometrie, PINs, etc.) authentifiziert wird. Unglücklicherweise birgt dieses Konzept Probleme, wenn ein Client-Gerät authentifiziert werden soll, welches sich nicht unter der exklusiven Kontrolle des Benutzers befindet (z.B. an dessen Arbeitsplatz). In solchen Szenarien ist ein zusätzliches Gerät (Authenticator) im Besitz des Benutzers notwendig, welches den Benutzer authentifiziert. Die dazu benötigte Schnittstelle zwischen den beiden Komponenten birgt allerdings potentielle Sicherheitsrisiken (bei drahtlosen Verbindungen), als auch mögliche Einschränkungen im Hinblick auf die Einsetzbarkeit (v.a. bei kabelgebundenen Schnittstellen). Diese Arbeit präsentiert sowohl eine benutzerfreundliche Lösung zur Sicherung dieses Kommunikationskanals, als auch eine Erweiterung des Authentifizierungsprotokolls, wo der Kommunikationskanal komplett entfällt.
Da der Authenticator einen Single point of failure (SPoF) im Hinblick auf die Sicherheit darstellt, besteht zudem die Notwendigkeit, kompromittierte Geräte auf globaler Ebene widerrufen zu können. Diese Arbeit schlägt mehrere Ansätze zur zentralen Widerrufbarkeit vor, welche die Privatsphäre des Benutzers nicht gefährden (Unlinkability).
Die letzten Kapitel dieser Arbeit widmen sich der kritischen Evaluierung der präsentierten Lösungen, und zeigen Themen für weitere Forschungsarbeit auf.

Abstract

Security incidents related to breached passwords are an omnipresent issue. Solutions for security-critical applications like two-factor authentication exist, but are no option for intensively used applications, especially due to usability- and deployability-issues.
Under the huge amount of scientific work and (commercial) products that address this issue, the *FIDO specification* seems to be the most promising candidate. Some of the major players in the IT industry already started adopting this standard in their products. However, there are still a number of problems that need to be solved before FIDO can herald the *end of the password era* and become the standard authentication mechanism for the Internet of Things (IoT).
FIDO brings strong security by authenticating a device in possession of the user (using asymmetric cryptography) as well as the user operating this device himself (using biometrics, PINs, etc.). Unfortunately, this concept involves issues if a client device should be authenticated that is not under the user's exclusive control, e.g. at the user's workplace. In such situations, an additional authenticator device owned by the user is needed. However, the required interface between the two components involves potential security risks (with wireless connections), as well as potential limitations regarding deployability (especially for wired interfaces). This thesis proposes both a usable solution for securing this communication channel, and an extension of the authentication protocol that allows to go without the direct interconnection at all.
Aside from that, as the authenticator device introduces a Single point of failure (SPoF) regarding security, a method that allows to globally invalidate a compromised authenticator is needed. This work proposes multiple approaches for central revocation, that do not threaten the user's privacy (unlinkability).
In the final chapters of this work, the proposed approaches are evaluated critically, and directions for future research on this topic are given.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Security incidents related to breached passwords are an everyday issue. This is not very surprising, as passwords can be seen as the de-facto authentication mechanism on the Web. If passwords are used properly, they are a very simple to implement and secure way of authenticating to a computer system. But unfortunately, there is always a trade-off between security and usability. To use passwords properly, they need to be complex, unique for every service and stored securely, ideally only in the user's memory. However, if these criteria are fulfilled, passwords are usually not usable. For convenience, people tend to use weak passwords, write them down on Post-its and use the same password for multiple services. As long as users are not forced to use an authentication mechanism in a secure way, there is a high risk that they won't do so.
Securely maintaining a list of dozens of good passwords and keeping this list within reach is not a trivial task. Unluckily, the current situation is not expected to get better: Due to the rising adaption of mobile computing, the existing problems will get even worse and new challenges will be introduced. People will need to authenticate to a single service from an ever-changing set of multiple devices, which implies that current „solutions“ like password-managers are unlikely to be used in a secure way. Also the issue of entering a password on a mobile device in public is a well-known problem which is intensively being researched on, but where still no proper or elegant solution has been found for.
If we go one step further, and think about authentication for the „Internet of Things“, it is obvious that the mentioned issues will become even more problematic, and as long as we are not able to solve them, the authentication- or security-aspects in general will probably prevent the ideas of ubiquitous computing from becoming reality at all.

1.2 Problem statement

Replacing the password as the de-facto authentication mechanism on the Web is not a straightforward task. Apart from the central question of this paper, finding an mechanism that allows usable authentication on multiple and mobile devices, there are further challenges that need to be coped with:
Various service providers that tried to improve their password-based authentication processes using 2-factor authentication, like Facebook [1] or Outlook.com [2], implemented the mechanism insufficiently. Facebook enforces enabled two-factor authentication only if accessed using the web-site, but not on the mobile app. Outlook.com has to cope with the fact that Hypertext Transfer Protocol (HTTP) isn't the only protocol that is used on the Internet, and developed a very unusable workaround: Users can configure their account to require an additional authentication step to protect it if it's accessed via the web portal, but the users, or an attacker, can still access the mailbox via Post Office Protocol version 3 (POP3) or Internet Message Access Protocol (IMAP) using password-based authentication only. Microsoft introduced special auto-generated passwords for non-HTTP access, however the number of users that actually understands this feature is probably very low.
Furthermore, the typical „cheap“ implementation of 2-factor authentication, where a smart-phone is used to deliver a one-time password, has a fundamental security issue per design: If a user utilizes the same smart-phone for accessing a service and for generating the one-time password, the second authentication factor is worthless and does not add additional security.
Last but not least, if the password is going to be replaced, various authentication related processes like handling users with forgotten passwords or lost devices need to be thought over.

Certainly there exist well-known alternatives to passwords, like biometric authentication methods or smart-cards. However, all of these technologies have serious disadvantages in respect of security, usability, costs or complexity.

Of course, there also exists a lot of current research that tries to solve the problem: A typical approach is to improve the security of the password without changing the core authentication mechanism, as proposed with *Tapas* [3] or *UniAuth* [4].
Also completely new authentication mechanisms are proposed, like *UbiKiMa* [5] or some commercial solutions around the Universal Authentication Framework (UAF) standard of the *FIDO Alliance* [6].
However, all of the stated attempts have serious security and usability issues, or do not cover enough real-world usage scenarios.

1.3 Aim of the work

The aim of this thesis is to show how public-key cryptography can be used to build a secure and usable authentication mechanism that is a real alternative to passwords. The basic idea of public-key authentication is nothing new and has been successfully used in high-security smart-card based authentication scenarios for years.

The UAF standard of the *FIDO Alliance* defines an authentication architecture where public-key cryptography is used to authenticate a device to an online service, and the private key on the user's device is protected by a local authentication mechanism like a PIN or fingerprint.
However, the FIDO specification does not cover a couple of security and usability challenges that arise in real-world usage scenarios. Therefore, simply adopting this standard is not enough.

FIDO's idea assumes that the user's client device is trustworthy. However, this assumption does not hold for some environments, especially where multiple persons use a single machine. This situation is obviously given in business environments, universities or public Internet terminals for example, but the problem basically occurs in all situations where a user wants to log-in on a machine other than his own.
As the FIDO specification even defines the interface for the communication between the user agent (application client) and the FIDO authenticator, it is basically possible to use a separate, trustworthy device as an authenticator to log-in on a foreign machine. However, the specification does not define how a secure communication channel between these devices can be established. This is one of the problems this thesis addresses.

Furthermore, the hardware that is required to interconnect the user's workstation with the authenticator might not be given. The specification as well as the few implementations that are available consider the *smartphone* as the typical authenticator-device for FIDO. Obviously, this is for a reason, as the smartphone has ideal hardware preconditions for protecting the keys (e.g. fingerprint reader, camera, microphone, secure element, ...) or communicating with other devices (e.g. Near Field Communication (NFC), Bluetooth, Universal Serial Bus (USB), ...).
However, password-based authentication ideally should be replaced everywhere. Hence, there exist potential deployments that don't offer the required hardware, like client machines without USB and Bluetooth for example. A way to allow FIDO authenticators to be used without a direct, bidirectional data interface is needed. This is a topic this thesis addresses as well.

Last but not least, the specification does not provide the ability to block stolen or lost devices centrally without having to revoke them individually for each registered account. In a future Internet of Things, authenticating devices might require dozens of registrations

with depending services. Without a standardized mechanism for central revocation, lost devices might pose an extreme privacy issue to the user.

To sum it up, the aim of this work is to design adaptations and enhancements of the FIDO specification, to be able to implement an authentication system that takes over the ideas of the FIDO standards, but represents a more comprehensive solution, that can replace the password for as many of the typical end-user scenarios as possible.

1.4 Methodological approach

The results of this work have been achieved by performing the following steps:

1.4.1 Literature research

Search for related and state-of-the-art work using the literature databases of field-related scientific societies like *ACM* or *IEEE*. Investigation of the work of known researchers in the field, like *Blase Ur*, *Joseph Bonneau* or *Alexander de Luca*. Review of current articles of related popular scientific conferences like *CCS* or *SOUPS*.

1.4.2 Evaluation of the FIDO architecture & design of improvements

This phase started with the analysis of the UAF architecture and the identification of potential issues. Afterwards, possible solution approaches have been designed and analyzed theoretically.
During the design phase of the adaptations, various aspects have been taken into account, including:

- Does the adaptation affect security/privacy goals of the original specification?
- Does the specification/a third party already address this problem?
- Does the developed concept have platform limitations, e.g. requirement of a Trusted Platform Module (TPM).
- Is the scheme secure (intensive examination using common attack models).
- Is the scheme feasible regarding usability, deployability or scalability?
- Is it beneficial to stick to the FIDO architecture?
- Is the targeted usage scenario relevant?

1.4.3 Prototype design & implementation

To show that the proposed concepts can be implemented in practice, a prototype has been developed.

The released FIDO UAF 1.0 specification already must be considered obsolete, as the structure of the *WebAuthn/FIDO 2.0* draft has changed completely. This also explains why there seems no well-documented FIDO library available in public.

For this reason, our prototype does not use a FIDO library nor is fully compatible to the specification. However, proposed concepts are designed to be compatible with the specification, e.g. be implementable as an extension (the UAF specification[1] as well as the *WebAuthn* draft[2] provide means for doing this).

Furthermore, the concepts of the FIDO specification were adopted whenever possible. Anyway, the aim of the prototype is to demonstrate the proposed concepts rather than covering details of the FIDO protocols.

FIDO-based authentication can be realized for numerous usage scenarios. As some of our proposed approaches specifically target distributed deployment of FIDO/application client and authenticator, a prototype that is feasible for this usage scenario was created, although it might be not the most common one. Hence, it consists of the following components:

FIDO authenticator An Android application, authenticating the user using the fingerprint scanner.

FIDO Authenticator-Specific Module (ASM) A Windows application, communicating with the authenticator over Bluetooth.

FIDO Client A browser plugin for *Google Chrome*, communicating with the ASM over HTTP.

Demo application A fictional web application, consisting of

a frontend part, communicating with the FIDO client using HTTP AJAX-requests,

and a backend part, the FIDO server (Relying Party (RP)).

Revocation authority (RA) A web service that manages a list of revoked authenticator devices.

[1] `https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-protocol-v1.0-ps-20141208.html#extension-dictionary` (accessed 2016-12-22)

[2] `https://www.w3.org/TR/2016/WD-webauthn-20160531/#extensions` (accessed 2016-12-22)

1.4.4 Verification & Critical reflection

Fortunately, there already exist methods to verify authentication schemes in a formal way, thus we also used these methods to show the relevance of our work.
As already mentioned, there is a close link between security and usability. It is important to push the user to utilize the system in a secure way. But of course, it is also important that the usability is sufficient enough to encourage users to accept the new authentication mechanism. Therefore, usability considerations have been part of our verification methodology as well.

Verification against FIDO security requirements The FIDO Security Reference [7] provides a detailed elaboration of Security goals the FIDO specification is intended to meet, including a detailed threat analysis. We showed that our adaptations of the specification do not weaken the compliance with these goals by performing a threat analysis analogous to the one done by the FIDO document [7].

Evaluation using Bonneau's framework Bonneau et al. proposed a framework[8] for comparative evaluation of password alternatives. Using this framework, we performed an evaluation of the FIDO standard, with and without our adaptations, which shows that our proposed ideas improve the results of the evaluation.

Relevance of the work We could show that our adaptations improve FIDO-like authentication in *security-*, *usability-* as well as *deployability*-aspects.

Design with usability in mind We showed that our adaptations can be implemented in a usable fashion and are not just theoretically feasible.

1.5 Structure of the work

Chapter 2 gives an overview of the state of the art regarding scientific work, open standards & specifications but also (commercial) services & products that provide alternatives to password-based authentication. Particular attention is given to the FIDO specification, which is elaborated in detail in this chapter.

Chapter 3 is the main part of this work, and proposes three major enhancements to the FIDO specification.

Chapter 4 describes the most important aspects of the prototype, including design decisions, APIs, etc.

To show the scientific relevance of this work, chapter 5 critically analyzes our proposed adaptations of the FIDO specification. This is done, among others, by comparing them

with the original specification using a formal framework for evaluating password alternatives. It also points out potential issues with our adaptations.

Chapter 6 finally summaries the results of this thesis, but also gives directions for future research in this field.

CHAPTER 2

State of the art / analysis of existing approaches

2.1 Scientific literature

The approaches of the related and state of the art scientific literature can be divided into two categories: Attempts to improve the security of password-based authentication, by helping or forcing the users to use passwords in a more secure way, and the attempts to establish new, conceptually different authentication mechanisms, usually based on public-key cryptography, and/or improving their usability.

Bonneau et al. [8] developed a framework for comparison of current and future approaches to replace the password as the de-facto authentication mechanism on the Web, including even exotic ones that are beyond the scope of this thesis. However, Bonneau et al.'s paper gives an excellent overview of how many aspects need to be taken into account, to be able to find a realistic candidate for replacing password-based authentication.

2.1.1 Improving password security

2.1.1.1 Tapas

McCarney et al. [3] propose *Tapas*, a distributed password manager to solve typical security issues of currently established password managers. The key idea is that two devices are needed to use the password manager, a manager-device, where the stored passwords can be used to login to services, and a wallet-device, where the passwords are stored encrypted. The concept *does not need a master password*, and therefore solves the related security issues. Security is guaranteed in the way that user-interaction on *both* devices in a narrow time-frame is needed to retrieve a password.

However, aside from the obvious usability issues, i.e. feasibility for mobile-only users, the concept requires a basic understanding of how to use passwords properly, regardless of whether a password manager is used or not. Therefore, improving the security of password managers can not be seen as the ideal solution for solving the security issues of password-based authentication.

2.1.1.2 UniAuth & Knock x Knock

Hayashi and Hong [4] propose *UniAuth* and *Knock x Knock*, a novel password manager architecture and the corresponding implementation for the iPhone and the Mac, an approach very similar to *Tapas*. However, it introduces a lot of advanced features that tackle many of the usability and security risks *Tapas* cannot solve.
They use Bluetooth Low Energy (BLE) for the communication channel between the password vault (the smartphone) and the user's desktop computer. This approach does not only allow the system to be used in offline scenarios, it further increases security, as an attacker would need to be physically co-located to be able to attack it.
Furthermore, the paper introduces the Universal Identity Management Protocol (UIMP), that allows the password manager (the smartphone) to communicate with web services that might support this protocol in the future. This not only enables the system to reliably and fully automate the registration or login processes, it furthermore enables the system to enforce password-quality, renewal or revocation rules.
Due to its design, the system can immediately be used for password-based services, but provides additional features which, if service providers adopt UIMP, further increase security and usability.
Unfortunately, as most of the other related work, *UniAuth* does not cover various real-world usage-scenarios sufficiently, like mobile-only usage (utilization of the system directly on the password vault device), or scenarios where the (master) vault is not available.

2.1.1.3 WebTicket

Hayashi et al. [9] propose *WebTicket*, a sort of paper-based password manager. The prototype encodes the username- and password combinations as Quick Response (QR) codes and prints them out on paper, one „ticket“ per account. To add additional security, the login information is encrypted using a key stored on a computer, which can be printed out as a QR code analogously. To be able to used the *tickets* on another machine, the user has to import the key first.
The design has many obvious usability issues, as devices with a camera and the paper-based tickets are required to authenticate to a service. Similar to *Tapas*, the system does not force the user to utilize it in a secure way. People might keep the tickets together with the decryption key and/or keep them insecurely otherwise. The users would need to understand the security concept, which cannot be expected from the average user.

2.1.2 New authentication mechanisms

2.1.2.1 UbiKiMa

Everts et al. [5] propose *UbiKiMa*, and idea very similar to *Tapas*. However, the proposed prototype offers two operating modes. The first one is more or less equivalent to the distributed password-manager architecture of *Tapas*. The second one solves the mentioned security issues by replacing the password completely and utilizing public-key cryptography based authentication. It adopts the abstract concept of authenticating a web browser to a web service using an additional device like a smart-phone, however uses asymmetric encryption to transparently authenticate the user.
Obviously, this idea also does not cover usage scenarios where only a single device is available, and also shares many of the other mentioned usability issues of related approaches.

2.2 Standards & specifications

2.2.1 FIDO Alliance

> *The FIDO (Fast IDentity Online) Alliance is a 501(c)6 non-profit organization nominally formed in July 2012 to address the lack of interoperability among strong authentication devices as well as the problems users face with creating and remembering multiple usernames and passwords. The FIDO Alliance plans to change the nature of authentication by developing specifications that define an open, scalable, interoperable set of mechanisms that supplant reliance on passwords to securely authenticate users of online services. This new standard for security devices and browser plugins will allow any website or cloud application to interface with a broad variety of existing and future FIDO-enabled devices that the user has for online security. [6]*

FIDO develops two standards: Universal Authentication Framework (UAF) and Universal Second Factor (U2F). The former describes a protocol for public-key cryptography-based authentication and key management, the latter describes a protocol for two-factor authentication and the related procedures. As we want to *replace* passwords, we give our attention especially to the former standard, UAF.
UAF combines public-key based authentication of a trustworthy device under the user's exclusive control with simple authentication methods like biometrics or PINs that are used to approve the trusted device's actions. However, for deployments where the user's client device cannot be trusted (because multiple users have access to it for example), a separate secure authenticator device is needed. The UAF architecture principally supports such deployments, as it also defines interfaces between the user agent and the authenticator component, however it doesn't specify how this interface can be implemented securely. This thesis will address this issue in detail.

2.2.2 W3C Web Authentication

The World Wide Web Consortium (W3C) develops the *WebAuthn*-specification („*Web Authentication*“): It describes an API for web developers that allows to utilize FIDO authenticators from within a web browser, independent of the underlying implementation. A first draft has already been released. [10]

2.3 Commercial services & products

2.3.1 LaunchKey

LaunchKey [11] offers commercial authentication services based on similar concepts as proposed by the FIDO Alliance. Unfortunately, their solution does not consider many potential real-world usage scenarios, such as authenticating without a smart-phone. Furthermore, a commercial product that doesn't rely on open standards can never be the answer to the global authentication problem, as it is unlikely that such a product will be adapted by a majority of service providers as well as users. It isn't even clear if the product can be used in a global environment, without having to rely on their commercial service, which acts as an authentication proxy.

2.3.2 Nok Nok

Nok Nok Labs [12] is one of the major stakeholders behind the FIDO Alliance, and offers a complete authentication solution based on the UAF specification, including server and client components. Unfortunately, there are many real-world use cases this specification does not propose solutions for, which their commercial solution does not do either. These issues and missing features is where this thesis ties on.

2.3.3 Windows Hello & Microsoft Edge

Microsoft announced that their browser *Edge* includes an experimental implementation of the WebAuthn Application Programming Interface (API) [13]. *Windows Hello*, the Windows 10 feature that allows biometric authentication to the operating system [14], can be used as a compatible FIDO authenticator.

2.4 The FIDO UAF specification

The aim of the following section is to give an overview of the UAF specification and the most important concepts. The latest version of the specifications can be downloaded here [15].

2.4.1 A note on FIDO versions

FIDO 1.0 is the only version that has been released as a *final* version [15]. Therefore, the UAF part of this specification, which is the part relevant for our work, is elaborated

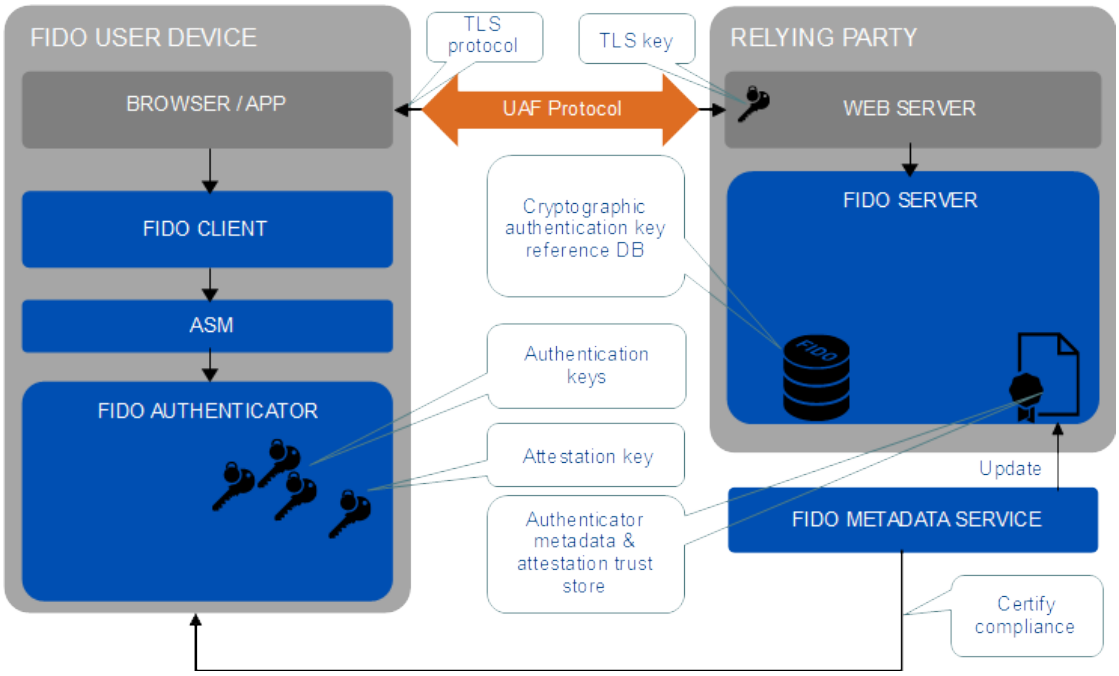

Figure 2.1: FIDO UAF High-Level Architecture [16]

here in detail.
FIDO 2.0 hasn't been published by the FIDO alliance at of this writing. However, parts of the specification have been submitted to the W3C and were published as draft of the *Web Authentication API[10]*. According to the W3C, FIDO 2.0 will unify the two standards of FIDO 1.0, UAF and U2F, to a single specification[1].
As the overall concepts stayed the same, the specification is principally simply referred to as *FIDO*. However, if an aspect only concerns a certain version of the standard, this is stated accordingly, e.g. *UAF* refers to FIDO 1.0 and *WebAuthn* refers to *WebAuthentication*/FIDO 2.0.

2.4.2 Overview

Figure 2.1 shows the overall architecture of the UAF specification.

2.4.2.1 Components of the UAF architecture

FIDO Authenticator The authenticator generates and stores the private keys that are used to sign the server's authentication challenges. To initiate a registration or authentication operation he authenticates the user by biometric or other methods. An important design aspect of this component is how the private data can be protected from unauthorized access.

[1] `https://www.w3.org/Submission/2015/02/Comment/` (accessed 2016-12-22)

FIDO Client The FIDO client negotiates authentications and related processes between the user agent (e.g. web browser) and the authenticator. It locates available authenticators using the ASM API.

ASM Authenticator-Specific Modules are the link between FIDO clients and authenticators, and provide an API that allows clients to detect available authenticators. On Windows platforms for example, the location of the ASM libraries can be detected by looking into predefined registry paths (e.g. *HKLM\Software\FIDO\UAF\ASM*).

FIDO server The FIDO server is the counterpart to the authenticator, and negotiates authentications, registrations or related processes with it. It stores the required information like the authorized public keys and related metadata.

FIDO metadata service The metadata service is operated by the FIDO Alliance and provides compliance information like attested authenticators to FIDO servers.

2.4.3 Privacy considerations

The following privacy goals stated in the FIDO specification [16] are important to understand the concepts proposed by the standard:

User accounts are not correlatable. There exists no global identifier that allows to link together two FIDO accounts that originate from the same authenticator device. Key-pairs are generated on a per-device, per-user account and per-relying party basis. If a user creates two accounts for the same relying party using the same authenticator, the relying party has no possibility to link the two accounts together, using the FIDO information alone. Furthermore, if two relying parties exchange user data, they cannot conclude from their accounts to a single user device respectively person.
To preserve anonymity despite authenticator attestation, attestation data is always shared among a large user base (at least 100000 installations).

Biometric data never leaves the authenticator. Biometric data is just used to locally authenticate the user against the FIDO authenticator. It never gets transmitted to the relying party or other entities.

2.4.4 Important security concepts

2.4.4.1 Application isolation

It is important to allow the user to limit generated keys to certain applications, FIDO clients and ASMs. Otherwise, malicious software could try to temp the user to grant it access to credentials created for legitimate applications. To know which Relying Partys (RPs) the user is registered with is not that important, as the attacker can simply try popular services. Of course, the user needs to approve the authentication operation to the FIDO client by sweeping his finger over the sensor. However, the probability that a

careless user does this without questioning the reason for the authentication request is very high. The fingerprint prompt could easily be taken for an unlock request for example.

A paranoid solution to this problem would be to further tighten the security concept of the FIDO specification, and generate key-pairs on a per-account, per-device **and per client application** basis. It is obvious, that this approach wouldn't be acceptable from a usability point of view, as a browser change for example would force the user to register the new browser for every service he utilized on the old browser. Therefore, we need something more flexible.

Furthermore, in some situations even more flexible access to the keys is necessary, e.g. when a user wants to use his smartphone as a FIDO authenticator to authenticate on a public desktop computer. For that reason, the UAF specification defines two use-cases for application isolation: Bound and roaming authenticators.

For these reasons, the UAF ASM API specification [17] proposes a complex application isolation concept. It introduces a number of identifiers and secret tokens, that are used to recognize trusted FIDO components:

AppID Identifies the relying party, e.g. the Uniform Resource Locator (URL) of a web application. Points to a list of FacetIDs.

FacetID A relying party specifies a list of trusted FacetIDs, which are allowed to access the application's credentials. What this FacetID is generated from, depends on the user agent of an application. In the case of a web application, FacetIDs may constitute subdomains or paths that are allowed to access the credentials stored for a specific AppID. This is necessary as complex web applications typically have more than one HTTP endpoint. Additionally, users may want to use the same credential for a completely different user agent, e.g. access their e-mails using a web browser *and* a native application, without having to register each application to the FIDO authenticator individually. In this case, a FacetID also may represent an identifier for a native application, e.g. the hash of the Android application package (APK) signing certificate on the Android platform. See the FIDO AppID and Facet specification for details [18].

CallerID Identifies the FIDO client. How the CallerID is determined depends on the environment. On Android platforms for example, this could be the hash of the APK signing certificate of the FIDO client application.

PersonaID In typical multi-user environments, like on desktop computers, it is desirable to limit the accessible credentials to the user account that created them. Therefore, the PersonaID typically represents the operating system user account.

ASMToken The ASM generates this random secret on first use.

KHAccessToken Key Handle Access Token. The specification distinguishes bound and roaming authenticators.

$$KHAccessToken_{bound} = Hash(AppID|ASMToken|PersonaID|CallerID)$$

$$KHAccesstoken_{roaming} = Hash(AppID)$$

(Note: „|“ denotes byte wise concatenation.)

2.4.4.2 Authenticator attestation

Authenticator attestation allows the FIDO server to verify, that the authenticator application is what it claims to be, e.g. is a *serious* product and no malware. This allows the relying party to allow only software products that comply with certain security requirements, for example.

The FIDO 1.0 UAF protocol specification[19] supports the following attestation flavours:

Full Basic Attestation All authenticator models of a specific type share a key-pair that is used to sign their compliance metadata. The private key is stored within the authenticator application and must not be extractable by the user. The authenticator vendor provides the public key to the relying parties to allow them to verify the authenticator's attestation information. The specification prescribes that at least 100000 installations must share the same key-pair, otherwise the unlinkability of user accounts cannot be ensured. This attestation flavour only makes sense if the target platform provides the possibility to store the attestation private key in a way that nobody except the manufacturer can extract it. An example for such platform could be dedicated hardware authenticators that are delivered with a pre-installed attestation key.

Surrogate Basic Attestation There is no dedicated attestation key-pair, instead, the account-specific authentication keys are used to sign the attestation information. This scheme cannot provide a real compliance proof.

Additionally, the current working draft of the WebAuthn/FIDO 2.0 specification [10], published by the W3C, lists the following additional attestation models:

Privacy Certificate Authority (CA) The *Privacy CA* is a service which is assumed to be operated by the vendor of the authenticator software in most cases. Each deployment of the authenticator gets equipped with an *endorsement key-pair*, with the public part being preserved by the *Privacy CA*. This key-pair is specific to a single installation of the authenticator software, therefore the integrity of the attestation architecture is not threatened if the user manages to extract the

corresponding private key. The public part of the endorsement key-pair is used to authenticate against the *Privacy CA*. The authenticator generates a new *attestation key-pair* for each account, and signs the public part with the private endorsement key.
For each transaction with a relying party, the *Privacy CA* checks the validity of the attestation key by verifying if the associated public endorsement key hasn't been revoked. On success, it issues a certificate for it using a private key. The key-pair that is used by the Privacy CA for this process is the same for a large number of installations, like for *Full Basic Attestation*, and therefore is an extremely threatened secret.

Direct Anonymous Attestation (DAA) DAA is a concept also known from the TPM specification and is based on the idea of *group signatures*. There exists a group private key, which is needed to generate new member private keys, and a single group public key. The idea is that group members can use their individual private key to sign messages, while their anonymity is preserved, as a single group public key can be used to verify the signatures. However, group signatures are still researched very actively, as there are still some unsolved issues, e.g. how member keys can be revoked.
Brickell and Li from Intel proposed Enhanced privacy ID (EPID) [20], a DAA scheme with enhanced revocation capabilities. However, the efficiency of the proposed algorithms practically is still very limited, as both the member and the verifier have to perform computations linear to the size of the list of revoked members. Furthermore, similar as for the *Privacy CA* model, some data that was used for member key-pair generation needs to be preserved by the issuer/revocation manager to be able to revoke members in the future. This limitation might be problematic for some deployments (see [20] for details about the revocation protocol).
The W3C announced that a DAA algorithm based on elliptic curve cryptography, called Elliptic Curve DAA (ECDAA) is going to be published [10].

Android Attestation This attestation strategy is conceptually completely different from the others. It is based on Google's *SafetyNet*[2] API and *Google Play Services*, hence it can only be utilized on Android based authenticators. With the SafetyNet API, Android applications can request a signed statement from Google's servers, that attests that the operating environment fulfils certain security requirements. These might include security relevant system settings, but also the absence of known dangerous software for example. Relying parties can verify the authenticity of the attestation information using a public key provided by Google. In contrast to the other attestation models, this model doesn't really provide information about the authenticator application itself, but rather about the system environment of the authenticator. However, this information might also be very valuable.

[2] `https://developer.android.com/training/safetynet/index.html` (accessed 2016-12-22)

2.4.4.3 Implementation challenges

How the *AppID*, *FacetID*, *CallerID* and *PersonaID* can be reliably determined by the FIDO components depends on the operating environment and may be problematic to implement on some platforms. While application isolation in Android[3] for example, was built in and enforced from the ground up, comparable concepts for Windows are very new[4] and are not enabled by default. This explains why the the prevalence of this feature still seems to be very low.

Similar challenges come along with the requirement to store secret information. Not only the keys on the authenticator, but also the *ASMToken* and *KHAccessToken* must be stored privately. While the *Android Keystore System*[21] supports storage of private data even without special hardware, Windows requires a TPM per design, as user applications can be run with unrestricted privileges as long as application isolation is not enforced consequently.

[3] `https://source.android.com/security/` (accessed 2016-12-22)

[4] `https://msdn.microsoft.com/en-us/library/dd548340(v=ws.10).aspx` (accessed 2016-12-22)

CHAPTER 3

Improvements to FIDO specification

3.1 Centralized authenticator revocation

The UAF specification does not propose a concept for *centralized authenticator revocation*. This means that if an authenticator device gets lost, stolen or compromised in another way (e.g. by malware) so that the private authentication keys cannot be longer considered safe, the device needs to be revoked for each relying party individually. There doesn't exist a centralized global list of stolen authenticator devices.

The associated usability and implicated security issues are obvious: The probability that the user does not even remember which services his authenticator device was linked to is very high, and despite of this issue the possibility to revoke keys in a user-friendly way might not be given for every relying party at all.

Implementing a centralized revocation scheme while maintaining anonymity is non-trivial. The subsequent section explains why.

3.1.1 Challenges

Traditional Public Key Infrastructure (PKI)-based environments use a well established concept named Certificate Revocation Lists (CRLs). The CRL is an autonomous service that provides a list of revoked certificates. It is queried online by the relying parties on every authentication request, or downloaded for offline use and refreshed from time to time, depending on the security requirements of the application.

However, the FIDO UAF specification prescribes a unique key-pair for every account, on every device and for every RP. This ensures that an RP cannot link two accounts of the same user together and two RPs cannot link requests to a single user, respectively (see Security goal (SG)-4 *Unlinkability* in the FIDO Security Reference [7]). This fact makes it hard to implement a CRL-based key-revocation scheme, as there is not a single public key respectively certificate that can simply be put on the revocation list.

Furthermore, a CRL-alike revocation scheme for FIDO authentication implicates further privacy challenges traditional PKIs do not face, imposed by the *Unlinkability* property of the specification. However, as a revocation scheme is not part of the specification, the question to which degree a revocation scheme must guarantee unlinkability isn't neither. Unfortunately, it is important to clearly define what privacy requirements a revocation scheme must meet before developing one. Therefore, this question is discussed in detail in the subsequent section.

3.1.1.1 Unlinkability considerations

Note: For further reference, we define the term Revocation authority (RA). A RA is the entity that manages and provides the CRL.

SG-4 of the FIDO Security Reference is defined as follows:

> *Unlinkablity: Protect the protocol conversation such that any two relying parties cannot link the conversation to one user (i.e. be unlinkable). [7]*

However, to be able to determine the necessary privacy measures for a revocation protocol, the term unlinkability needs to be defined even more precisely. Concretely, the following three important aspects need to be addressed:

Pre-revocation unlinkability/Revocation-query unlinkability A revocation list may be either queried online for a specific account respectively public key, or offline by querying a list that is refreshed from time to time. In the case of online access, it must be ensured that the RA is not able to link a querying relying party to a specific user device. For this reason, we define a new security goal, SG-4.1.

Post-revocation unlinkability The question if the unlinkability property still needs to be fulfilled even if the authenticator device has been revoked further constrains the possible approaches for a revocation scheme. We define a new security goal, SG-4.2. Some of our proposed approaches will meet even this requirement. *Note: Post-revocation unlinkability entails pre-revocation unlinkability.*

Local vs. foreign unlinkability Independent of any revocation aspects, the wording of the definition of the unlinkability security goal is unlcear. The definition of SG-4 describes the term unlinkability as a measure to ensure that two relying

parties cannot link actions to a single user. For further reference, this is what we call *foreign unlinkability*. Section 4, *Privacy Considerations* of the architectural overview-document of the UAF specification [16] however, states the following:

> *Similarly, if two users share a UAF device and each has registered their account with the same relying party with this device, the relying party will not be able to discern that the two accounts share a device, based on the UAF protocol alone. [16]*

As this privacy goal also can be classified as an unlinkability requirement, we need to define the security goal more fine-grained.
Therefore, we introduce the security goals SG-4.3 *Foreign unlinkability* and SG-4.4 *Local unlinkability*. Local unlinkability is the stricter variant and entails remote unlinkability, as when even a single relying party cannot link two user accounts to a single authenticator device, another foreign relying party won't be able to do so either.

Note: Section 5.1.1.2 summarizes all security goals defined in this work.

3.1.2 Online vs. offline revocation list querying

Table 3.1 compares the strengths and drawbacks of online- and offline revocation list querying.

As the comparison shows, each approach has benefits and drawbacks. The suitability for a specific use case highly depends on the application's and user's needs. Therefore, this work proposes an online- as well as an offline-revocation protocol. The two protocols are fully compatible, this means every involved actor (relying party, revocation authority and client) can operate both protocols simultaneously.

3.1.3 Related work

With its *authenticator attestation* concept (see section 2.4.4.2), the FIDO specification provides some schemes that might also be used for implementing central authenticator revocation. Only those attestation models that attest authenticator instances *individually* (not by model type like *Full Basic Attestation* for example), are eligible: These are *Privacy CA* and *DAA* (or the improved variant *EPID*). Unfortunately, both models have several weaknesses regarding deployability, security, availability, scalability or privacy (for details about the weaknesses, see section 2.4.4.2):

hardware dependence Both schemes require the authenticator key-pair to be tamper-proof. The *Privacy CA* scheme however, might be slightly adapted to get rid of this requirement: RPs could store a fingerprint of the authenticator public key anonymized with the account-specific public key. This would allow RPs to detect altered keys.

	online querying	**offline querying**
availability	For maximum security, online querying requires the RA to be queried for each authentication request, which makes the RA a SPoF. Without further considerations, an outage of the RA's infrastructure prevents users from authenticating to relying parties.	In the case of an outage of the RA, the revocation list cannot be refreshed. This prevents recent revocations from being enforced by the relying parties, but doesn't prevent eligible users from authenticating successfully.
performance	For maximum security, online querying requires the RA to be queried for each authentication request. This implies bundled load on the RA's infrastructure. However, a well-scalable revocation scheme might be able to handle the load.	Low refreshing-rates entails low load on the RA servers.
privacy	Without further considerations, online querying allows the RA to identify the querying relying parties by their IP addresses. As the RA needs to identify user devices per design, it would be able to determine which services are used by a certain user. This would pose an unacceptable privacy violation.	For offline-querying, relying parties fetch the list with all revoked devices at once. The RA can determine which relying party connects, but cannot link it to a specific user.
security	For maximum security, online querying requires the RA to be queried for each authentication request. This ensures that if a user puts his device on the revocation list, all relying parties refuse authentication immediately.	Obviously, offline revocation lists pose the risk that a revoked device can still authenticate successfully, because the list hasn't been refreshed yet. There is a trade-off between security and performance.

Table 3.1: Online- vs. offline revocation list querying

RA data loss Both schemes need to store data about all valid (unrevoked) authenticators. If the RA looses this data because of a system crash for example, *Privacy CAs* cannot perform revocation status attestations anymore. EPID-based deployments continue working as long as the RA's public key is available, as revocation status verifications do not involve the RA directly. However, no new devices can be revoked. To restore full functionality, both schemes require all authenticators to generate new key-pairs.

RA data leakage For both schemes, the RA uses a single key-pair that is used for all revocation status attestations. For *EPID*, if the private part of this key-pair is leaked and the key-pair needs to be re-generated, all authenticator key-pairs become invalid. As stated above, *Privacy CA* could be adapted to allow re-generation of the authenticator key-pairs. EPID requires that the authenticator key-pair isn't modifiable by the user, therefore key-regeneration is very problematic. It would require manual out-of-band re-attestation of all issued authenticator devices, e.g. the user would have to prove by other means that his authenticator device hasn't been revoked before the data crash. As *Privacy CA*-based revocation schemes could be realized without tamper-proof authenticator key-pairs (see above), key re-generation would be possible, at least theoretically, because this would invalidate *all* registered accounts.

Scalability EPID doesn't scale well for a large number of revoked devices: Both the authenticator and the RP have to perform computations linear to the size of the list of revoked authenticators (see [20] for details about the revocation protocol).

Post-revocation unlinkability EPID does not provide *Post-revocation unlinkability*: The revocation list contains data that can be used to uniquely identify an authenticator (see [20] for details about the revocation protocol).

Fortunately, in contrast to attestation schemes, a revocation scheme doesn't necessarily require an authenticator to be able to proof it's validity without any data being stored about it at registered RPs. This means that during registration of a new account with a RP, we can store information that can be used later to verify the authenticator's revocation status. Of course, it is important that this information does not threaten the stated privacy requirements, i.e. does not violate the unlinkability property.
In the following sections, we propose offline revocation schemes that meet this requirement *nearly completely*, and an online revocation scheme that meets this requirement *completely*. All proposed schemes can be implemented independently of the FIDO UAF code, and therefore might be enforced by relying parties optionally only, depending on the security requirements of the application or the user.

3.1.4 Revocation protocols with offline-verification

This section proposes two offline revocation protocols, which differ in their unlinkability properties.
The first scheme guarantees *foreign post-revocation unlinkability*. *Local unlinkability* isn't provided by this scheme at any time.
The second scheme provides *foreign post-revocation unlinkability*, as well as *local pre-revocation unlinkability*. This means that after an authenticator device has been revoked, if two accounts had been registered with this authenticator at a single RP, this RP can link the two accounts together.

As offline revocation offers some benefits over online revocation per design, and such strict unlinkability characteristics might not be required or maybe even not desired, this section describes both protocols in detail.

3.1.4.1 Offline revocation with foreign post-revocation unlinkability

The concept uses the following entities:

- A relying-party identifier Id_{rp}. It is important that the relying party is something the authentication client is able to verify. Otherwise, multiple relying parties could share an identifier intentionally to be able to link user accounts together. The *AppID* defined by the FIDO specification[18] meets this requirement and can be used as Id_{rp}, for example.
- A device identifier Id_d
- The revocation hash $H_r = Hash(Id_{rp} + Id_d)$
- A revocation list RL, consisting of multiple revoked Id_d, maintained by the RA and kept private.
- An anonymized revocation list RL_a, consisting of corresponding H_r entries.
- An URL U_r that locates the offline revocation list. This can be an HTTP or File Transfer Protocol (FTP) endpoint for example, depending on the implementation.

The following step needs to happen before an actual registration or authentication operation is performed:

- Id_d is generated on the first use of the authenticator application.

On registration of an authenticator with a RP, the following steps are performed additionally to the registration steps defined by the FIDO specification:

1. Id_{rp} is verified by the authenticator.
2. H_r is computed.
3. H_r and U_r are transmitted to the relying party and stored permanently.
4. The relying party starts fetching the anonymized revocation list RL_a periodically.

The anonymized revocation list RL_a is generated by the RA for each relying party individually. This happens in the following way:

1. The relying party requests its anonymized copy of RL by sending its Id_{rp} to the RA.
2. The RA calculates H_r for each entry of RL.
3. The RA randomizes the order of the entries. This is important because otherwise two relying parties could link two accounts together by comparing the positions of the corresponding H_r in the revocation list.
4. The RA sends RL_a to the requesting relying party.
5. The relying party stores RL_a permanently.

The actual revocation status verification, which needs to be performed for every authentication procedure, consists of the following steps:

1. RP looks up the stored H_r of the authenticating account in RL_a.
2. If the hash can be found, the corresponding device has been revoked, otherwise the authentication process was successful.

3.1.4.2 Offline revocation with local pre-revocation unlinkability and foreign post-revocation unlinkability

The concept uses the following entities:

- A RP identifier Id_{rp}. It is important that this identifier is something the authentication client is able to verify. Otherwise, multiple relying parties could share an identifier intentionally to be able to link user accounts together. The *AppID* defined by the FIDO specification[18] meets this requirement and can be used as Id_{rp}, for example.
- A device identifier Id_d
- An account salt Sa_a
- The revocation hash $H_r = Hash(Id_{rp} + Id_d)$
- The per-account revocation hash $H_{ra} = Hash(H_r + Sa_a)$
- A revocation list RL, consisting of multiple revoked Id_d, maintained by the RA and kept private.
- An anonymized revocation list RL_a, consisting of corresponding H_r entries.
- An URL U_r that locates the offline revocation list. This can be an HTTP or FTP endpoint for example, depending on the implementation.

The following step needs to happen before an actual registration- or authentication operation is performed:

- Id_d is generated on the first use of the authenticator application.

On registration of an authenticator with a RP, the following steps are performed additionally to the registration steps defined by the FIDO specification:

1. Id_{rp} is verified by the authenticator.
2. Sa_a is generated by the authenticator. This is important, because if Sa_a would be generated by the relying party, two relying parties could use a static, identical value for Sa_a intentionally, to be able to link user accounts to an authenticator device.
3. H_r and H_{ra} are computed.
4. H_{ra}, Sa_a and U_r are transmitted to the relying party and stored permanently.
5. The relying party starts fetching the anonymized revocation list RL_a periodically.

The anonymized revocation list RL_a is generated by the RA for each relying party individually. This happens in the following way:

1. The relying party requests its anonymized copy of RL by sending its Id_{rp} to the RA.
2. The RA calculates H_r for each entry of RL.
3. The RA randomizes the order of the entries. This is important because otherwise two relying parties could link two accounts together by comparing the positions of the corresponding H_r in the revocation list.
4. The RA sends RL_a to the requesting relying party.
5. The relying party stores RL_a permanently.

The actual revocation status verification, which needs to be performed for every authentication procedure, consists of the following steps:

1. For each H_r in RL_a, the RP computes H_{ra} with Sa_a of the authenticating account, and compares the resulting hash with the H_{ra} stored with the account information.
2. If a matching hash can be found, the corresponding device has been revoked, otherwise the authentication process was successful.

3.1.4.2.1 Performance considerations We can observe that a large revocation list will constitute a performance issue, as the relying party needs to compute n_{RL} hashes for every authentication operation, where n_{RL} is the number of revoked devices for the RA. (*Note: A RP only needs to check the revocation list of the RA the authenticating client utilizes.*)
To handle this issue, *precomputed hashes* or *caching* of already computed values for H_{ra} could be used. However, the size of the cache might become very large, especially if the relying party has a large number of accounts, as the following calculation shows:

Let s_H be the size of a hash, n_a the number of accounts at a relying party, and s_{cache} be the size of the cache. Let $s_H = 32\ bytes$, $n_a = 1000000$ and $n_{RL} = 100$. Then

$$s_{cache} = s_H * n_a * n_{RL} = 3,2\ GB$$

To reduce the size of the cache, a truncated version of H_{ra}, H'_{ra} could be used to generate the cache. Due to the low size, the probability that for multiple H_{ra} the resulting hashes are identical is very high. As these duplicate hash entries do not need to be stored, the size of the cache can be reduced significantly. The worst case size of the hash cache can be calculated as follows:

Let $s_{H'}$ be the size of the truncated hash, and $R(s)$ be the function that calculates the number of possible hash values for a given size s. Let $s_H = 32\ bytes$, $s_{H'} = 2\ bytes$ and $n_{rl} = 100$. Then

$$s_{cache} = (s_H + s_{H'}) * R(s_{H'}) * n_{RL} = 844,8\ KB$$

As we can see, the size of the cache is significantly smaller with this improvement.

For every authentication operation, the RP first performs the revocation lookup using H'_{ra}. If there is no match, it can be concluded that the device isn't revoked. If there is a match, the RP needs to calculate H_{ra} for every H_r stored in the cache with the matched H'_{ra}. In the worst case, n_{RL} computations have to be performed.

3.1.4.3 Issues with both offline revocation schemes

Privacy issues in small deployments For both proposed offline revocation protocols, it is important to point out that the grade of anonymity of revoked devices depends on the number of revoked devices. If a RA only revoked one or a few devices, it is easy for two relying parties to link accounts to a single authenticator device.

Hash collisions Due to the nature of hash functions, it is possible that two authenticator devices produce identical values for H_r or H_{ra}, even if all the values that are used to compute these hashes are distinct. This can lead to the situation where a valid authenticator device is blocked because another, revoked authenticator device

produces an identical value for H_r or H_{ra}.

As this issue cannot occur for the online revocation scheme proposed in the subsequent section, offline revocation verification could always be deployed in combination with online verification: An unintentionally blocked authenticator device could be given the possibility to proof its valid status by performing an online revocation verification request in the case the offline verification results in a rejection.

Fortunately, we were able to develop an revocation scheme that uses online-querying and isn't susceptible to the issues stated above. Furthermore, it can guarantee *local post-revocation unlinkability*. This approach is proposed below.

3.1.5 A revocation protocol with online-querying

Note: As the previously described revocation schemes that use offline-checking have serious drawbacks (see above) and require further research, the following protocol, which is based on online-querying*, is considered the final solution relating to this work.*

To recapitulate, the relevant involved parties are the

- FIDO authenticator
- Relying Party (RP)
- Revocation authority (RA)

The concept uses the following entities:

- A per-account key-pair, consisting of a private key k_a and a public key K_a
- An authentication challenge c_a
- An authentication signature $S_a = Sign(c_a, k_a)$
- A randomly generated account salt Sa_a
- A device identifier Id_d
- The revocation hash $H_r = Hash(size(Id_d) + Id_d + Sa_a)$
- A key-pair the revocation service uses to sign query responses, consisting of a private key k_r and a public key K_r
- The revocation signature $S_r = Sign(H_r + c_a, k_r)$
- An URL L_r that locates the revocation service. This can be a HTTP or Domain Name System (DNS) endpoint for example, depending on the implementation.

The following steps need to happen before an actual registration or authentication operation is performed:

1. Id_d is generated on the first use of the authenticator application.
2. k_r and K_r are generated by the revocation service and are only changed in the case of a leakage of k_r.
3. K_r is published on L_r.

To register an authenticator with a RP, the following steps are performed:

1. k_a and K_a are generated by the authenticator.
2. Sa_a is generated by the authenticator.
3. H_r is calculated.
4. K_a, H_r and L_r are transmitted to the relying party and stored permanently.

The actual authentication procedure consists of the following steps:

1. RP generates a challenge and sends it to the authenticator.
2. The authenticator computes the authentication signature S_a with the challenge c_a and k_a.
3. RP verifies the signature with K_a.
4. The authenticator sends c_a, Sa_a and Id_d to the RA.
5. RA looks up Id_d in its list of blocked devices. If the identifier is found, RA sends an error response and the authentication procedure is aborted. If not, the procedure continues.
6. RA calculates H_r, and computes the signature $S_r = Sign(H_r + c_a, k_r)$ and returns it.
7. The authenticator forwards the signature to the RP.
8. RP decrypts the signature with K_r and checks if H_r and c_a match the stored versions.
9. If both match, the authentication process was successful.

3.1.5.1 Performance considerations

It is obvious that the RA is a SPoF regarding availability as well as performance, as it is involved in every authentication operation. The following paragraphs discuss who unavailable RAs can be handled and how the load on their infrastructure can be limited.

3.1.5.1.1 Query intervals To reduce the associated risk of inability to perform authentications, the revocation verification mechanism at the RP might tolerate a certain number of failed revocation checks, or perform a check only every n authentication operation to reduce the load on the RA's infrastructure. RPs may enforce different check intervals, according to the maximum time a blocked device can be tolerated with regard to the level of security required by the application.
RAs might limit the number of possible revocation queries per timespan, and customers could be forced by RPs to purchase premium versions of authenticator software that utilizes RAs which allow more frequent revocation checks than others to be compatible with security-critical services.

3.1.5.1.2 Caching Aside from that, the load on the RA could be slightly lowered by caching the results of the computation of H_r. As the value doesn't change over time, it could be cached using efficient data structures like trees and looked up by the parameters of the hash function on demand.

3.1.5.1.3 Load balancing As the only permanent data the RA's servers depend on is the list of revoked device identifiers, which should be a relatively small dataset, multiple instances of revocation servers can be deployed easily. An issue could be the size of an eventually used revocation hash cache, which might grow to large for a single machine. However, instead of randomly balancing the load on the revocation servers, it might be balanced by device identifier, which will keep the size of the revocation hash cache small.

3.1.5.2 A security note on concatenated arguments of hash functions

The size parameter $size(Id_d)$ in the definition of the revocation hash function, $H_r = Hash(size(Id_d) + Id_d + Sa_a)$, is a very security critical component.

Example: Let $Id_d =$ "$imei : 239480234802123$" and $Sa_a = a3254bcb2356de65$.

If the definition of the revocation hash function would be $H_r = Hash(Id_d + Sa_a)$, where $+$ denotes byte wise concatenation, an attacker that possesses a revoked authenticator device could send the following tampered parameters to the revocation service: $Id_d =$ "$imei : 23948023480212$" and $Sa_a = 3a3254bcb2356de65$.
Note: The only thing that has changed is that one character of the device identifier was

shifted to the account salt.

Due to the changed device identifier, the revocation service would not identify the requesting authenticator device as a revoked one, and as the concatenated value of both parameters results in the same value as for the original ones, it would generate the valid values for the hash and the associated signature!

With the inclusion of the calculated size of the device identifier into the calculation of the hash, the described attack becomes impossible. It is enough to include the size of one parameter only.

Other possibilities to prevent this attack would be to combine the two parameters by XOR operations, or to pass hashed values as parameters to the revocation hash function, e.g. $H_r = Hash(Hash(Id_d) + Hash(Sa_a))$ or $H_r = Hash(Hash(Id_d) + Sa_a)$. However, the proposed method above is more efficient regarding performance, which can be an important factor in large deployments.

3.1.5.3 Implementation considerations

A further interesting aspect is that the proposed protocol has some similarities with DNS Security extensions (DNSSEC). DNSSEC protects DNS responses with digital signatures, and DNS implementations itself should be very efficient and matured handling extreme loads of small queries and load balancing. These opportunities could allow the proposed revocation concept to be quickly implemented using the DNSSEC protocol, without having to reinvent the wheel, as the signing mechanism is already provided by the DNSSEC implementation.

A DNS query to the RA might look like this:
<c_a>.<Sa_a>.<Id_r>.ra.authenticator-vendor.com

The response might be a signed TXT resource record and look like this:
ns1.authenticator-vendor.com. <TTL> IN TXT "<S_r>"

3.2 Securing wireless ASM-Authenticator connections

If FIDO authenticator and ASM respectively client are located on different devices, the physical connection between these devices might not be able to guarantee integrity and confidentiality of the transmitted data.

Bluetooth for example, must be considered as a potentially insecure communication channel. First, the PIN-based pairing mechanism should enforce very complex PINs to be secure. Furthermore, the built-in security mechanisms require the user to be aware of

how they actually work, to be really safe against Man-in-the-middle (MITM)-attacks (e.g. be aware that repeated PIN-prompts after an already successful pairing procedure are likely to be caused by an attacker [22]). With the introduction of Bluetooth 2.1, many security-related issues have been fixed, however the user still plays an active role in establishing a secure connection, e.g. when *numeric comparison* is used, he needs to assure himself that the connection is secure [23].

Due to the design of the FIDO UAF protocol, there is no need for a communication channel between ASM and authenticator that guarantees *confidentiality* of the transmitted data, as the protocol messages never contain sensitive unencrypted data. However, it is important that the underlying communication channel guarantees *integrity and authenticity*. Otherwise, there would be no protection mechanism that guarantees the user that an authentication request actually comes from the client device he is working on, rather than from an attacker's machine.

This issue can be solved by attaching Hash Message Authentication Codes (HMACs) to every message exchange between ASM and authenticator, exchanging the HMAC secret using asymmetric encryption, and verifying the integrity of the key exchange using an out-of-band mechanism.

Our proposed approach uses QR-codes displayed on the client machines to provide the fingerprint of the public key that is used to encrypt the HMAC secret to the authenticator device. The authenticator device needs to be equipped with a camera to be able to scan the QR-code. The use of QR-codes additionally brings the possibility to transmit protocol-dependent pairing information (e.g. Bluetooth device names) at one go, hence reducing the number of required user interactions. Our proposed approach assumes that *Bluetooth* is used as the underlying communication channel.

3.2.1 Protocol description

The concept uses the following entities:

- An ASM key-pair consisting of a private key k_{asm} and a public key K_{asm}.
- The fingerprint of the ASM key-pair $fp_{asm} = Hash(K_{asm})$
- The random HMAC secret key k_{hmac}.
- The pairing information PI (e.g. Bluetooth device name).

Before secure messages can be exchanged, the ASM needs to generate the ASM key-pair.

Pairing an authenticator to an unkown ASM is done by the following steps:

1. The ASM application encodes $PI + fp_{asm}$ as QR-code and displays it.
2. The user scans the QR-code with his authenticator device/application.
3. The authenticator uses PI to connect to the desired ASM.
4. The ASM sends K_{asm} to the authenticator.
5. The authenticator validates the received K_{asm} by computing fp_{asm} and comparing it to the value received via the QR-code. If the values don't match, the connection is aborted.
6. The authenticator generates a random k_{hmac}, encrypts it with K_{asm} and sends the resulting ciphertext back to the ASM.
7. The ASM decrypts the ciphertext to gain k_{hmac}.

For every subsequent UAF message, whether it goes from the ASM to the authenticator or the other way round, a HMAC hash of the message payload is computed, attached to the message and verified by the other end.

To be able to communicate with each other in the future without having to pair again, both the key-pair at the ASM and $PI + fp_{asm}$ at the authenticator might be stored permanently for a certain period of time.

Note: The data encoded in the QR-code is not private. Attackers spying the QR-code displayed on the users monitor cannot extract any secret information from it.

To be able to attack the connection, e.g. making the authenticator use a malicious public ASM-key, the attacker would need to have the ability to modify the QR-code displayed on the user's client machine.

3.3 Indirect authentication

There exist a number of potential use cases where FIDO authenticator and ASM respectively FIDO client do not reside on the same device, *and* no physical connection between the stated components is possible. A possible scenario might be a user that wants to use his smartphone as an authenticator to login in on a public desktop in an Internet cafe, or on an office computer that isn't equipped with a Bluetooth interface for cost- or security reasons, for example.

The UAF specification does not cover such use cases. For this reason, we propose a mechanism called *Indirect authentication*. The proposed protocol allows camera-equipped devices (e.g. smart phones) to be used as authenticators without the need for a physical connection between the authenticator and the ASM.

3.3.1 Protocol description

The concept uses the following entities:

- An indirect authentication token t_{ia}
- The URL that is used as an authentication endpoint by the authenticator, U_{ia}.

No steps need to be performed before indirect authentication operations can be performed. The relying party allows indirect authentication by specifying an a U_{ia} in its published list of valid *FacetID*s [18].

The indirect authentication procedure consists of the following steps:

1. The user requests indirect authentication in the client application.
2. The relying party generates a random t_{ia} and stores it temporarily until the authentication procedure is completed or a timeout is reached.
3. The relying party encodes $U_{ia} + t_{ia}$ (e.g. *http://www.myapp.com/fido/ia.php?tia=ab33fcd3e3557acf8bdb8954678686bc*) as QR code and displays it to the user.
4. The user opens his FIDO authenticator application on his authenticator device and scans the QR code.
5. The authenticator performs a regular FIDO authentication procedure using $U_{ia}+t_{ia}$ as endpoint.
6. The relying party authenticates the session associated with t_{ia} and invalidates the token.

3.3.2 Security aspects of the indirect-authentication token

t_{ia} is no secret. The person that knows t_{ia} can only authenticate the session that is associated with it, if she possesses a registered authenticator, of course. She cannot steal associated sessions or gain any other private information.

Theoretically, there is no need to encode t_{ia} as a QR code. A human-readable string that is displayed on the client device and entered manually using a (virtual) keyboard on the

authenticator device would also be a possible solution for transferring t_{ia}, that would even obviate the need for a camera on the authenticator device.

However, the length of t_{ia} is still a relevant security factor. If the token is very short, and the user has to enter it manually, the probability that a single erroneously entered character makes the user authenticating a foreign session unintentionally because he accidentally entered a valid token of another session, is very high. Theoretically, undetected errors also might occur with QR codes, however due to the integrated error correction[1], the probability is very low.

[1] `http://www.qrcode.com/en/about/error_correction.html` (accessed 2016-12-22)

CHAPTER 4

Prototype

4.1 Prototype components

There are dozens of scenarios FIDO-based authentication can be implemented for. The UAF architecture consists of numerous components, including authenticator, ASM, FIDO client, client application, Relying Party and other optional components like the Revocation authority introduced in this work. Some of them may reside on the same physical devices or be distributed across multiple ones, on different hard- and software platforms and interconnected using various communication technologies.

The architecture of the prototype's components was designed in a way, that the three key improvements to the FIDO specification proposed in this work, *Centralized authentication*, *Integrity protection of ASM/authenticator-communication* and *Indirect authentication*, can be implemented and that their usefulness can be shown using appropriate usage scenarios.

The following listing describes the key aspects of the prototype's components:

- A native Android Java application serving as FIDO authenticator, authenticating the user using the fingerprint scanner, and communicating with the ASM over the Bluetooth interface. Some use-cases use the built-in camera to scan QR-codes displayed on the user's client device.
- A Windows application serving as the ASM, communicating with the authenticator over Bluetooth, and providing the ASM API to the FIDO client via a HTTP web service. The application is based on the Microsoft .NET-framework and written in C#.

- A browser plugin for Google Chrome, written in JavaScript, that provides a FIDO-alike JavaScript API to the demo website and communicates with the ASM's web service over HTTP.
- A fictional web application, consisting of

  a frontend part, utilizing the FIDO JavaScript-API provided by the browser plugin,

  and a backend part, the FIDO server (relying party), written in PHP.
- The revocation service, a PHP-based web service, providing an online revocation verification service via HTTP.

4.2 Design decisions & specification coverage

4.2.1 No component identification

As we chose a distributed scenario with Windows as the platform for the authenticating client and ASM, we didn't have the possibility to implement secure component identification, as it would have been possible for a single-device scenario using Android, for example (see section 2.4.4.1 for theoretical background information).

4.2.2 No utilization of hardware-assisted security

Unfortunately, our client- as well as our authenticator hardware we developed on also wasn't equipped with a TPM. Therefore we couldn't make use of the security features it would have provided. Fortunately, Android provides near TPM-grade secure generation and storage of private keys implemented in software, thus we made use of this feature called *Android Keystore System*[21].

4.2.3 Simplified but secure application isolation

The UAF specification defines complex mechanisms for application isolation, that cannot be implemented on our target platforms in a reasonable way (see section 2.4.4.1 for details). Therefore, for our prototype, we had to choose between security and versatility. We decided to focus on security: A credential can only be used to authenticate to websites using the *hostname* or *IP address* it was registered with.

4.3 Authenticator application

4.3.1 Authenticator commands

The ASM and the authenticator use JavaScript Object Notation (JSON)-encoded messages to communicate over the Bluetooth Radio Frequency Communication (RFCOMM)-channel. The JSON messages are encoded by the respective libraries in a single line, and

terminated by a Line feed (LF) symbol. This allows the receiver to detect the end of a message.

Only the ASM can trigger operations on the authenticator, but not vice versa. Therefore, messages sent from the ASM to the authenticator are referred as *Commands*, whereas messages sent from the authenticator to the ASM are referred as *Responses*.

Nested data structures are always stored in encoded form in their respective parent structure. This allows a party to pass on the data contained in child fields without having to know their structure.

The message structures of the prototype are loosely based on the structures described in the *Authenticator Commands* document[24] of the FIDO 1.0 specification. As the message formats described in this document aren't even mentioned in the draft of the newer *Web Authentication* standard published by the W3C, we expect them not to be standardized in the future anymore or at least to change fundamentally. Therefore, strict protocol conformance was not an objective of the protocol design for the prototype. Anyway, the standards conformance of the protocol for ASM/authenticator communication would only be relevant if it should be possible to communicate between ASMs and authenticators of different vendors.

As already stated, the prototype's protocol uses JSON-encoded messages, whereas the UAF standard proposes a Tag Length Value (TLV)-based protocol. This decision was made in favor of better understandability in consequence of the decision to neglect standards compliance for this protocol.

Listing 4.1 describes the JSON data structures of the various messages using Web Interface Definition Language (IDL) (Second Edition) [25].

Listing 4.1: Authenticator commands - JSON data structures

```
typedef DOMstring Base64; //maps a Base64 encoded byte array
typedef DOMstring JSON; //escaped JSON data

dictionary HmacMessage {
        required DOMString jsonPayload;
        required DOMString hash;
};

enum Command {
        GetInfo,
```

```
        Register,
        Sign
};

dictionary AuthenticatorCommand {
        required Command commandType;
        required JSON args;
};

dictionary AuthenticatorResponse {
        required boolean statusCode;
        required JSON responseData;
        required DOMString deviceID;
};

dictionary RegisterCommand {
        required DOMString appID;
        required DOMString username;
};

dictionary RegisterResponse {
        required Base64 assertion;
        required Base64 fingerprint;
        required DOMString revocationPublicKeyUrl;
};

dictionary SignCommand {
        required Base64 finalChallenge;
        required DOMString appID;
};

dictionary SignResponse {
        required Base64 assertion;
        required Base64 fingerprint;
        required DOMString revocationCheckUrl;
        required DOMString username;
};
```

4.4 ASM

4.4.1 ASM API

The FIDO client communicates with the ASM over a HTTP API. However, to stay compliant with the ASM API specification[17] of the FIDO alliance as much as possible, the interface was not designed as a Representational State Transfer (REST)-like API.

Instead, there is only one HTTP-endpoint, `/ASM`, accepting a single POST-parameter `message` that must contain a JSON-encoded `ASMRequest` data structure. If the query is correct, the web service always responds with the status code `200 OK` and a JSON-encoded `ASMResponse` structure. Detailed status information is encoded in the data structure.

Listing 4.2 describes the mentioned data structures and there respective child structures using Web IDL (Second Edition) [25].

Listing 4.2: ASM HTTP API - JSON data structures

```
typedef DOMstring Base64; //maps a Base64 encoded byte array
typedef DOMstring JSON; //escaped JSON data

enum Request {
        GetInfo,
        Register,
        Authenticate
};

dictionary ASMRequest {
        required Request requestType;
        required JSON args;
};

dictionary ASMResponse {
        required boolean statusCode;
        required JSON responseData;
};

dictionary RegisterIn {
        required DOMString appID;
        required DOMString username;
};
```

```
dictionary RegisterOut {
        required Base64 assertion;
        required Base64 revocationHash;
        required DOMString revocationPublicKeyUrl;
};

dictionary AuthenticateIn {
        required Base64 finalChallenge;
        required DOMString appID;
};

dictionary AuthenticateOut {
        required Base64 assertion;
        required Base64 revocationSignature;
        required DOMString username;
};
```

4.5 Client browser plugin

4.5.1 JavaScript API

Listing 4.3 describes the JavaScript API provided to web sites by the *Chrome* browser plugin to allow them to utilize the connected FIDO authenticators on the local system. The API is based on the draft of the *Web Authentication API* published by the W3C [10].

The original specification uses quite complex data types as it also includes mechanisms for negotiating various FIDO features and is intended to provide maximum flexibility. As these mechanisms are not the focus of this thesis, our prototype's interface only implements a subset of the interfaces described in the W3C specification, and sometimes uses simplified data types. However, it should give a reasonable insight how our adaptations could be implemented within the *Web Authentication API* in a non-interfering way.
Anyway, we unfortunately couldn't find any publicly available website which fully supports this specification that would have enabled us to evaluate the compatibility of our implementation in practice.

As the W3C draft, the interfaces of the prototype's API are described using Web IDL (Second Edition) [25].

Note: Parameters with the data type `any` *and the keyword* `optional` *are not handled by the prototype in any way. They just exist in the method declaration to allow the methods to be potentially called by web sites implementing a larger subset of the specification.*

Listing 4.3: Client browser plugin - JavaScript API

```
typedef DOMstring Base64; //maps a Base64 encoded byte array

partial interface Window {
        readonly attribute WebAuthentication webauthn;
};

interface WebAuthentication {
        Promise < ScopedCredentialInfo > makeCredential (
                Account accountInformation,
                optional any cryptoParameters,
                optional any attestationChallenge,
                optional any credentialTimeoutSeconds,
                optional any blacklist,
                optional any credentialExtensions
        );

        Promise < WebAuthnAssertion > getAssertion (
                Base64 assertionChallenge,
                optional any assertionTimeoutSeconds,
                optional any whitelist,
                optional any assertionExtensions
        );
};

dictionary Account {
        required DOMString name;
};

interface ScopedCredentialInfo {
        readonly attribute Base64 publicKey;
        readonly attribute Base64 revocationHash;
        readonly attribute DOMString revocationPublicKeyUrl;
};

interface WebAuthnAssertion {
        readonly attribute DOMString credential;
        readonly attribute Base64 signature;
        readonly attribute Base64 revocationSignature;
};
```

4.6 Demo web application

4.6.1 HTTP API

All server-side actions of the demo web application are handled over a REST-like HTTP-API.

Endpoint	Method	Parameters	Response	Description

/authenticate.php	GET	-	200 OK: {
			challenge: Base64
			}
				Generates an authentication challenge, temporarily stores it as session variable and returns it as JSON-field

/authenticate.php	POST	assertion: { username: String, assertion: Base64, revocationSignature: Base64 }, indirectToken: String	200 OK: <Info message> 403 Forbidden: Revocation verification failed. 403 Forbidden: Authentication failed. 404 Not Found: User not found. 424 Failed Dependency: Need to generate challenge first.	Validates the revocation signature. Validates the authentication signature (assertion). On Success, it authenticates the session identified by the given session ID, or the session associated with an optionally given indirect authentication token.

/register.php	POST	username: String,
		assertion: {
		publicKey: Base64,
		revocationHash: Base64,
		revocationPublicKeyUrl: String
		}
			200 OK: <Info message>
			409 Conflict: User exists.
				Persists all the given parameters into a user-specific, JSON-formatted file.

/createIndirectToken.php	POST	-	200 OK: <Token as hexadecimal number>	Creates and temporarily persists a randomly generated token.
/loginStatus.php	GET	-	200 OK: <Info message>	Returns the login status for the given session ID.
/logout.php	POST	-	200 OK: -	Deauthenticates the given session.

Table 4.1: Demo web application - HTTP API

Note: All endpoints respond with a `400 Bad Request` status code if required parameters are missing, and with a `500 Internal Server Error` on unexpected errors like failed file operations.

4.7 Revocation service

4.7.1 HTTP API

The revocation service provides a REST-like HTTP-API.

Endpoint	Method	Parameters	Response	Description
/public.pem	GET	-	200 OK: —BEGIN PUBLIC KEY— ... —END PUBLIC KEY—	The PEM-encoded public revocation key
/checkRevocation Status.php	GET	deviceId: String, fingerprint: Base64, challenge: Base64	200 OK: { revocationHash: Base64, signature: Base64 } 403 Forbidden: Device has been revoked.	The signature of the revocation hash, and the computed revocation hash (for debugging purposes)

Table 4.2: Revocation service - HTTP API

CHAPTER 5

Critical reflection

5.1 Verification against FIDO security requirements

The FIDO Security Reference [7] already provides a systematic security analysis of the UAF specification. It defines Security goals, and shows how the specification meets this goals by means of a detailed threat analysis.

The aim of this section is to show that our extensions of the FIDO specification proposed in chapter 3 do not violate these security goals and even meet additional goals defined in this work.

Note: As our proposed revocation scheme with online-querying provides significant benefits over the scheme based on offline-verification, the comparative evaluations in this chapter primarily consider the online variant.

5.1.1 Security goals & definitions

5.1.1.1 UAF Security Goals

For better understandability of the subsequent relevance analysis, this section cites the UAF Security goals defined by the FIDO Security Reference [7].

SG-1 Strong User Authentication *Authenticate (i.e. recognize) a user and/or a device to a relying party with high (cryptographic) strength.*

SG-2 Credential Guessing Resilience *Provide robust protection against eavesdroppers, e.g. be resilient to physical observation, resilient to targeted impersonation, resilient to throttled and unthrottled guessing.*

SG-3 Credential Disclosure Resilience *Be resilient to phishing attacks and real-time phishing attack, including resilience to online attacks by adversaries able to actively manipulate network traffic.*

SG-4 Unlinkability *Protect the protocol conversation such that any two relying parties cannot link the conversation to one user (i.e. be unlinkable).*

SG-5 Verifier Leak Resilience *Be resilient to leaks from other relying parties. I.e., nothing that a verifier could possibly leak can help an attacker impersonate the user to another relying party.*

SG-6 Authenticator Leak Resilience *Be resilient to leaks from other FIDO Authenticators. I.e., nothing that a particular FIDO Authenticator could possibly leak can help an attacker to impersonate any other user to any relying party.*

SG-7 User Consent *Notify the user before a relationship to a new relying party is being established (requiring explicit consent).*

SG-8 Limited Personal identifiable information (PII) *Limit the amount of personal identifiable information (PII) exposed to the relying party to the absolute minimum.*

SG-9 Attestable Properties *Relying Party must be able to verify FIDO Authenticator model/type (in order to calculate the associated risk).*

SG-10 Denial of Service (DoS) Resistance *Be resilient to Denial of Service Attacks. I.e. prevent attackers from inserting invalid registration information for a legitimate user for the next login phase. Afterward, the legitimate user will not be able to login successfully anymore.*

SG-11 Forgery Resistance *Be resilient to Forgery Attacks (Impersonation Attacks). I.e. prevent attackers from attempting to modify intercepted communications in order to masquerade as the legitimate user and login to the system.*

SG-12 Parallel Session Resistance *Be resilient to Parallel Session Attacks. Without knowing a user's authentication credential, an attacker can masquerade as the legitimate user by creating a valid authentication message out of some eavesdropped communication between the user and the server.*

SG-13 Forwarding Resistance *Be resilient to Forwarding and Replay Attacks. Having intercepted previous communications, an attacker can impersonate the legal user to authenticate to the system. The attacker can replay or forward the intercepted messages.*

SG-14 Transaction Non-Repudiation *Provide strong cryptographic non-repudiation for secure transactions.*

SG-15 Respect for Operating Environment Security Boundaries *Ensure that registrations and key material as a shared system resource is appropriately protected according to the operating environment privilege boundaries in place on the FIDO user device.*

5.1.1.2 Refined and new security goals

To be able assess our introduced security features that are not covered by the UAF security goals, we refine some of them and introduce new ones.

SG-4.1 Pre-revocation unlinkability/Revocation-query unlinkability See section 3.1.1.1 for details.

SG-4.2 Post-revocation unlinkability See section 3.1.1.1 for details.

SG-4.3 Remote unlinkability See section 3.1.1.1 for details.

SG-4.4 Local unlinkability See section 3.1.1.1 for details.

SG-A1 RA Leak Resilience Be resilient to leaks from the RA, i.e. data leaked by the RA cannot be used to evade the revocation check.

5.1.2 Relevant security goals

The objective of this security analysis is to show that our improvements do not violate the security goals specified by the FIDO security reference. However, only a small subset of them are relevant in the context of our improvements. Therefore, table 5.1 first identifies relevant and irrelevant FIDO security goals.

Note: To keep the size of the table small, the proposed improvements are abbreviated as follows: Centralized authenticator revocation (CAR), Secure ASM-authenticator channel (SAAC) and Indirect authentication (IA).

Security goal	**Relevant features**	**Justification**
SG-1 Strong User Authentication		None of the proposed improvements affect the core principles of the authentication mechanism.
SG-2 Credential Guessing Resilience	SAAC, IA	CAR: The user doesn't have to enter any information nor gets shown any relevant information.
SG-3 Credential Disclosure Resilience	IA	CAR & SAAC are not affected by the user's behavior regarding usage of the web browser.

Security goal	**Relevant features**	**Justification**
SG-4 Unlinkability	CAR	SAAC does not affect the RP. For IA, no additional information is sent *to* the RP by the user.
SG-5 Verifier Leak Resilience	CAR	SAAC does not affect the verifier (RP). IA does not store any additional non-public information.
SG-6 Authenticator Leak Resilience		None of the proposed improvements share information among authenticators.
SG-7 User Consent	IA	SAAC does not affect RPs. CAR-related communication with RPs is only initiated after successful authentication.
SG-8 Limited PII	CAR, IA	SAAC does not affect RPs.
SG-9 Attestable Properties		None of the proposed improvements introduce new independent user-sided software or hardware components.
SG-10 DoS Resistance	CAR	SAAC doesn't affect publicly accessible components. IA doesn't process user-created data/input (aside from the data also processed by ordinary authentication).
SG-11 Forgery Resistance	CAR, SAAC	IA doesn't process user-created data/input (aside from the data also processed by ordinary authentication).
SG-12 Parallel Session Resistance	SAAC, IA	CAR: A revoked device is blocked by the RPs. Therefore, there are no sessions that could be spied on in parallel.
SG-13 Forwarding Resistance	CAR, SAAC, IA	
SG-14 Transaction Non-Repudiation		None of the proposed improvements affect the transaction feature.
SG-15 Respect for Operating Environment Security Boundaries		None of the proposed improvements intend an operating environment differing from those covered by the FIDO specification.
SG-A1 RA Leak Resilience	CAR	RAs are only relevant for CAR.

Table 5.1: FIDO UAF security goals relevance

5.1.3 Threat analysis

5.1.3.1 Attacks against the revocation mechanism

Table 5.2 shows a detailed threat analysis of attacks against the proposed (online) revocation mechanism.

Affected security goal	**Attack action**	**Consequences of a successful attack**	**Countermeasures**	**Consequences of a prevented attack**
SG-4	Attacker tries to identify user accounts on one or multiple RPs that registered the same authenticator.	The privacy of the user is affected.	The revocation hash stored by the RP is based, among other information, on a random salt generated by the authenticator for each account.	The revocation hash is different for every account, correlation is impossible.
SG-5	The RP leaks revocation related data.	The leaked data could be used by attackers to evade revocation checks.	The revocation mechanism only stores the public key of the RA and the revocation hash at the RP.	This information cannot be used to evade the revocation check.
SG-8	The RP leaks revocation related data.	The leaked data could be used by attackers to identify the person associated with the account.	The revocation mechanism only stores the public key of the RA and the revocation hash at the RP.	This information cannot be used to identify the person associated with an account.
SG-10	An attacker performs a DoS-attack on the RA's infrastructure.	Users might not be able to authenticate as mandatory revocation checks cannot be performed due to unavailability of the RA.	RPs might allow the revocation check to be skipped if the RA is down.	Authentication operations can be completed without a revocation check. However, this lowers the level of security, of course.
SG-11	Attacker sends forged device ID.	The authenticator device is not recognized as a revoked one.	The RP stores a hash that is based, among other information, on the device ID.	The revocation authority generates an invalid revocation hash. The relying party refuses authentication because of the invalid revocation hash.
SG-13	The attacker sends a previously generated valid revocation signature to the relying party.	The RP approves the authenticator as a valid, non-revoked device, even if it's on the list of revoked devices.	The RP generates a random challenge for each revocation check, which is used as input to the revocation signature function.	The signature verification fails, as the used challenge is not the current one.

Affected security goal	Attack action	Consequences of a successful attack	Countermeasures	Consequences of a prevented attack
SG-A1	The attacker gains access to the private key of the revocation authority.	The attacker can generate valid revocation signatures for arbitrary device IDs, account salts and authentication challenges.	A compromised revocation key-pair can be replaced easily, as the RPs do not store the corresponding public key permanently. Instead, they need to fetch it dynamically using an URL transmitted by the authenticator during registration of an account. The public key may be cached of course to limit the load on the RA's infrastructure, but it should be refreshed frequently to limit the imposed security risk.	If the new public key of the replaced key pair has been fetched by all RPs, the compromised old private key is worthless.

Table 5.2: Threat analysis - Attacks on the revocation mechanism

5.1.3.2 Attacks on the communication channel between ASM and authenticator

Table 5.3 shows a detailed threat analysis of attacks on the communication channel between ASM and authenticator.

Affected security goal	Attack action	Consequences of a successful attack	Countermeasures	Consequences of a prevented attack
SG-2	Attacker scans and decodes the QR-code displayed on the user's client machine during pairing.	The attacker could use the information to attack the integrity of the communication between ASM and authenticator.	The QR-code only contains the Bluetooth MAC-address of the client machine and the fingerprint of the ASM's public key.	The information cannot be used to gain sensitive information like the ASM's private key or the HMAC-secret.

Affected security goal	**Attack action**	**Consequences of a successful attack**	**Countermeasures**	**Consequences of a prevented attack**
SG-11	Attacker manipulates the key exchange of the HMAC secret.	If the attacker manages to gain knowledge of the HMAC secret or even manages to slip the ASM a faked one, he might be able to send forged FIDO commands to the ASM or authenticator.	The authenticator uses the public key received from the ASM to encrypt the randomly generated HMAC secret. Furthermore, the authenticator receives the fingerprint of the public key from the ASM over an independent communication channel (QR-code). Before sending the HMAC secret to the ASM, the authenticator verifies the received public key by computing the fingerprint of the public key himself and comparing it to the fingerprint retrieved from the QR code.	If the fingerprint check fails, the authenticator aborts the pairing process and does not send the HMAC secret.
SG-11	Attacker modifies a FIDO message.	If the attacker can modify FIDO messages, he can create faked ASMs or authenticators.	Modification of FIDO command messages is prevented by protecting every message sent from the ASM to the authenticator or vice versa with a HMAC.	If a tampered message is detected, the command is ignored.
SG-13	Attacker impersonates FIDO messages.	If the attacker can send valid FIDO messages to a proper ASM or authenticator, while pretending to be the proper counterpart, he can create faked ASMs or authenticators.	Impersonated FIDO command messages are prevented by protecting every message sent from the ASM to the authenticator or vice versa with a HMAC.	If a tampered message is detected, the command is ignored.
SG-12, SG-13	Attacker creates a fake ASM.	If an attacker manages to persuade an authenticator to communicate with a faked ASM under his control, while keeping the user believing that he is communicating with the genuine ASM, the attacker can persuade the user to authenticate a session controlled by the attacker.	To prevent faked ASMs, authenticators only accept HMAC protected messages.	If the HMAC check fails, the authenticator ignores the command.

Affected security goal	Attack action	Consequences of a successful attack	Countermeasures	Consequences of a prevented attack
SG-12, SG-13	Attacker creates a fake authenticator.	If an attacker manages to persuade an ASM to communicate with a faked authenticator under his control, while keeping the user believing that he is communicating with his genuine authenticator device, the attacker can persuade the user to register a credential generated by the attacker's authenticator with his account. From now on, the attacker can authenticate with his faked authenticator on the users behalf, while the legitimate user cannot. If the hijacked account is an existing one, sensitive data could be stolen.	To prevent faked authenticators, ASMs only accept HMAC protected messages.	If the HMAC check fails, the ASM ignores the command response.

Table 5.3: Threat analysis - Attacks on the ASM-authenticator interconnection

5.1.3.3 Attacks on the indirect authentication mechanism

Table 5.4 shows a detailed threat analysis of attacks on the proposed indirect authentication mechanism.

Affected security goal	Attack action	Consequences of a successful attack	Countermeasures	Consequences of a prevented attack
SG-2	Attacker scans and decodes the QR-code displayed on the user's client machine when initiating indirect authentication.	The attacker could use the information to impersonate the user.	The QR-code only contains a randomly generated token that allows the RP to associate the authentication request performed by the authenticator with the user's session on the client device.	The token does not represent sensitive information.

Affected security goal	**Attack action**	**Consequences of a successful attack**	**Countermeasures**	**Consequences of a prevented attack**
SG-3	Attacker triggers authentication to a trusted site the user's authenticator is registered with, by encoding the respective URL in a QR code and tempting the user to scan it with his authenticator.	If the user completes the authentication request with the respective authenticator, the attacker gains access to an authorized session of the service he persuaded the user to authenticate to.	The service identifier (e.g. domain name) indirected authentication is requested for is displayed on the authenticator's display. The user is responsible for deciding if authenticating the website that initiated the process with the credentials he is asked for is his intended action. By the design of indirect authentication, the web browser has no possibility to prevent this kind of attack.	If the user does not complete the indirect authentication request on the authenticator, the attacker's session isn't authenticated.
SG-7	Attacker triggers an authentication operation without the user's consent.	The attacker could impersonate the user.	The proposed indirect authentication mechanism requires the user to scan a QR-code on the client device with his authenticator, i.e. requires physical action.	-

Table 5.4: Threat analysis - Attacks on the indirect authentication mechanism

5.2 Evaluation using Bonneau's framework

Bonneau et al. [8] proposed a framework for comparative evaluation of alternative authentication mechanisms. They defined a list of *benefits* an arbitrary authentication mechanism can either provide or not, and analyzed a handful of more or less well-known password alternatives using this framework, but also the classical method of authenticating using a simple password. The benefits are composed of various characteristics of the three categories *Usability (U)*, *Deployability (D)* and *Security (S)*.

This section analyzes the authentication mechanism as defined in the FIDO specification (*without* our proposed improvements), as well as *with* our adaptations and enhancements taken into account.

Finally, a tabular breakdown shows that with our proposed enhancements, FIDO-like authentication can provide even more benefits than without them.

Note: The analysis only includes a short description of the benefits *defined by the framework. For a detailed explanation see the paper that introduces the framework [8].*

5.2.1 Usability

U1 Memorywise-Effortless *Do the users have to remember any secrets to use the scheme?* As opposed to passwords, users of FIDO-based authentication principally do not have to remember any secrets, and our adaptations do not impact this property. An exception is if it is used with PIN-based authenticator protection, of course.

U2 Scalable-for-Users *Can the user manage dozens of accounts, without having to remember separate information for each account for example?* To use classical passwords in a secure way, a user has to remember an individual password for each account, which is obviously not scalable. FIDO-based authentication doesn't have this issue, as the individual secrets are stored by the authenticator. However, as the FIDO specification doesn't provide a mechanism for central revocation, revoking all credentials of a compromised authenticator device for a large number of accounts individually poses an non-scalable task from a user's perspective. Therefore, the original FIDO-specification can only partially provide this benefit, whereas our adapted version can do fully, as we proposed a mechanism for central revocation.

U3 Nothing-to-Carry *Does the user need anything else other than himself to authenticate?* Unfortunately, as opposed to passwords, neither the original specification nor our adapted version can fully satisfy this property, as an authenticator device is needed. However, the framework defines that if the scheme only depends on a device that usually is carried anyway, like a smart phone, partial fulfillment of this benefit can be awarded to FIDO-like authentication schemes.

U4 Physically-Effortless *Does the user need to be able to perform particular physical actions to authenticate?* As opposed to passwords, which the user has to type in using a keyboard, FIDO-like schemes do not demand potentially challenging actions from the user. Of course, some authenticator protection mechanisms change this situation, but there are enough alternatives available (e.g. fingerprint). Our adaptations do not impact this property.

U5 Easy-to-Learn *Are the steps required for an authentication operation hard to learn?* For both original and adapted FIDO authentication, all associated use cases consist of just one or a few steps.

U6 Efficient-to-Use *Can authentication and registration operations be performed in a short time?* For both original and adapted FIDO authentication, all associated actions (pressing some buttons, swiping over the fingerprint sensor, scanning a QR-code) should be completable in a short period of time.

U7 Infrequent-Errors *Are the physical actions that are required for authentication prone to errors (e.g. typos)?* As opposed to passwords, the only scheme-specific physical action of both FIDO variants are scanning QR-codes and approving the operation on the authenticator. The former hardly can be influenced by the user, and also for the latter several relatively failsafe methods exist (e.g. fingerprints, short Personal Identification Numbers (PINs)).

U8 Easy-Recovery-from-Loss *If the authentication token is lost, can the access be restored easily?* FIDO-like authentication potentially offers similar backup strategies as passwords (e.g. backup of private keys). However, note that such measures might nullify some security advantages of FIDO-like authentication over passwords (e.g. storage of private keys in secure elements).

5.2.2 Deployability

D1 Accessible *Are there any physical requirements that might not be given in certain usage scenarios?* If client and authenticator reside on different physical devices, the deployability of original FIDO authentication might be limited, e.g. when no USB port is available. Our adaptations can help in these situations, with the introduction of indirect authentication and the possibility to establish wireless connections between client and authenticator in a secure and accessible way.

D2 Negligible-Cost-per-User *Does the user or verifier need special equipment?* FIDO authenticators can be implemented on common devices like smart phones or even on the client devices themselves.

D3 Server-Compatible *Do service providers need to adapt their password-based authentication systems to support the scheme?* Trivially true for password-based authentication. Not satisfiable for FIDO-like mechanisms, as they are conceptually different.

D4 Browser-Compatible *Do users have to install additional software on the client device?* Original FIDO requires browser support, either natively or through a plugin (like the one provided as part of our prototype). With the proposal of indirect authentication, our adapted FIDO-variant can provide this benefit!

D5 Mature *Is the authentication scheme widespread?* As many of the big players in the software industry are involved in the specification [6] and adoption [13] of FIDO authentication, and even the W3C is adopting the standard [10], FIDO authentication can definitely be classified as being on the way becoming mature.

Obviously, this cannot be true for our adaptations at the moment.

D6 Non-Proprietary *Can anyone implement the scheme?* Neither original FIDO nor our enhancements depend on proprietary software and the specifications are published openly [15].

5.2.3 Security

S1 Resilient-to-Physical-Observation *Can an attacker watching a user while authenticating gather enough sensitive information to impersonate him?* At least for the authentication process itself, no sensitive information needs to be entered nor is displayed during a FIDO authentication operation. The data encoded in the QR-codes used in our adaptations is not private. User-verification mechanisms on the authenticator that are susceptible to such kinds of attacks (e.g. PIN entry) are not considered here, of course.

S2 Resilient-to-Targeted-Impersonation *Can a user be tempted to reveal secret information accidentally?* The core authentication mechanisms of the original FIDO specification as well as our enhanced version solely rely on randomly generated secrets that the user never gets to know. User-verification mechanisms on the authenticator that are susceptible to such kinds of attacks (e.g. PINs, especially ones that can be chosen by the user) are not considered here, of course.

S3 Resilient-to-Throttled-Guessing *Can the authentication service be brute-forced, even if the service limits the number of authentication requests per timespan?* All security relevant mechanisms of both original FIDO and our added/adapted concepts rely on secret keys whose sizes are only limited by computing power. Therefore, brute-force attacks are not realistic, especially if the verifier can limit the number of requests per timespan. User-verification mechanisms on the authenticator that are susceptible to such kinds of attacks (e.g. PIN entry) are not considered here, of course.

S4 Resilient-to-Unthrottled-Guessing *Can the authentication service be brute-forced at all?* See S3. Note that if an attacker manages to retrieve public keys from the verifier for example, brute-forcing the associated private key might be eased, but still practically impossible if the key size is large enough.

S5 Resilient-to-Internal-Observation *Can malware perform authentication operations autonomously?* Generally true for both original FIDO and our adaptations, but if client and authenticator are different physical devices, both devices need

to be infected. For this case, the evaluation framework intends to award partial satisfiability of the benefit.

S6 Resilient-to-Leaks-from-Other-Verifiers *Can a compromised verifier be a threat to another verifier and its users?* Neither the original specification nor our adaptations use concepts where secret information is shared among verifiers.

S7 Resilient-to-Phishing *Is the authentication scheme phishing-safe?* Original FIDO is safe as the client reports the real hostname of the verifier to the authenticator, which verifies it. Our proposed revocation scheme and security improvements of the communication channel do not weaken this immunity. Our proposed mechanism for indirect authentication however, cannot be phishing-safe per design, as the browser is not involved in the authentication process.

S8 Resilient-to-Theft *If the user's account still safe if the authenticator device is stolen?* Both original FIDO and our adaptations intend to require user verification on the authenticator for all security relevant operations. As today's smartphones, which are suitable to be used as authenticator device, usually provide secure possibilities for user verification like fingerprint scanners, this benefit can be fully awarded here.

S9 No-Trusted-Third-Party *Does the authentication scheme rely on a trusted third party?* FIDO-like authentication does not rely on a third party. Indeed, our proposed revocation mechanism relies on a third party, however, this third party (the RA) does not process confidential information that can be abused by a public attacker. Note that potentially leaked revocation query data might be abused by a verifier to detect multiple accounts of a single authenticator device, but this risk is rated so minimal here that it does not justify to award this benefit partially only.

S10 Requiring-Explicit-Consent *Can authentication operations be performed without the user's awareness?* Both original FIDO and our adaptations require verification of the user on the authenticator for all security-critical operations.

S11 Unlinkable *Can colluding verifiers determine if the same authenticator is used for respective accounts?* The core authentication mechanisms of both original FIDO and our adaptations do not send data to the verifier which can be used to unambiguously identify an authenticator. Also our proposed revocation scheme was designed with unlinkability in mind.

5.2.4 Overview of results

Table 5.5 shows the results of the evaluation [8]. A filled circle (•) means that the authentication scheme provides the benefit. An empty circle (◦) means the scheme offers the scheme in most aspects but not in all. No circle means the scheme doesn't offer the benefit. *Note: The evaluation of classical password-based authentication (*Web passwords*) has been borrowed from Bonneau's paper [8].*

Note: To keep the size of the table heading small, the proposed improvements are abbreviated as follows: Centralized authenticator revocation (CAR), Secure ASM-authenticator channel (SAAC) and Indirect authentication (IA).

5.3 Relevance of the work

Our proposed work could enhance FIDO-based authentication in security- and usability aspects but also regarding potential deployment scenarios. This section summarizes the improvements that have been achieved.

5.3.1 Security aspects

Centralized revocation Without centralized revocability users would have to block stolen authenticator devices at each relying party separately. As some service providers must be expected to neglect to offer such a possibility at all, and users tend to neglect security if it requires cumbersome work, the possibility to revoke an authenticator device with a single step definitely is a security benefit. See section 3.1

Secure communication between ASM and authenticator If the deployment scenario requires a wireless communication channel between these FIDO components, our proposed secure key exchange scheme is a must to protect the user from certain attacks. See section 3.2 but also 5.1.3.2 for a detailed discussion of the potential threats prevented by this improvement.

5.3.2 Usability aspects

Centralized revocation From a usability perspective, centralized revocability reduces the number of revocation steps from n steps for n accounts to a single step.

5.3.3 New deployment options

Wireless communication channels between ASM and authenticator If the security threats stated above and in the referenced sections are not acceptable for a

		Web passwords	FIDO	FIDO+CAR/SAAC	FIDO+CAR/IA
U1	Memorywise-Effortless		●	●	●
U2	Scalable-for-Users		○	●	●
U3	Nothing-to-Carry	●	○	○	○
U4	Physically-Effortless		●	●	●
U5	Easy-to-Learn	●	●	●	●
U6	Efficient-to-Use	●	●	●	●
U7	Infrequent-Errors	○	●	●	●
U8	Easy-Recovery-from-Loss	●	●	●	●
D1	Accessible	●	○	●	●
D2	Negligible-Cost-per-User	●	●	●	●
D3	Server-Compatible	●			
D4	Browser-Compatible	●			●
D5	Mature	●	○		
D6	Non-Proprietary	●	●	●	●
S1	Resilient-to-Physical-Observation		●	●	●
S2	Resilient-to-Targeted-Impersonation	○	●	●	●
S3	Resilient-to-Throttled-Guessing		●	●	●
S4	Resilient-to-Unthrottled-Guessing		●	●	●
S5	Resilient-to-Internal-Observation		○	○	○
S6	Resilient-to-Leaks-from-Other-Verifiers		●	●	●
S7	Resilient-to-Phishing		●	●	
S8	Resilient-to-Theft	●	●	●	●
S9	No-Trusted-Third-Party	●	●	●	●
S10	Requiring-Explicit-Consent	●	●	●	●
S11	Unlinkable	●	●	●	●

Table 5.5: Bonneau-framework evaluation - Condensed results

specific deployment scenario, the stated security improvements bring the possibility to use wireless communication channels like Bluetooth for connecting FIDO authenticators with clients. This could be the case especially for public client devices (like Internet terminals at airports), where users can't be expected to carry an USB cable with them.

No connection between ASM and authenticator The proposed indirect authentication scheme allows FIDO-like authentication to be used even in deployments where no connection between the stated components is available at all. Again, this

could be relevant especially for public usage scenarios.

No FIDO support on the client platform The proposed indirect authentication scheme even allows FIDO-like authentication if the client platform doesn't specifically support the protocol. This allows the usage of FIDO-like authentication not only for devices where the system maintainer doesn't intend to ensure compatibility, but also for platforms that cannot be made compatible due to technical reasons.

5.4 Usability considerations

This section intends to show, that the proposed adaptations and the use-cases introduced with them were designed with usability in mind. We analyze our improvements of the FIDO specification by discussing the associated use cases individually.

Revoking an authenticator device Revoking a stolen device can be done by simply entering the device ID of the stolen authenticator device within a revocation user interface provided by the RA. The device identifier can be the smartphone's International Mobile Equipment Identity (IMEI) for example (like in our prototype).

Pairing an authenticator device with an ASM The proposed pairing scheme uses a single QR-code to encode the information which client device to connect to as well as the information that guarantees that a secure communication channel is established. Thanks to this design decision, aside from starting the respective applications on both devices, only a single step is required to complete the pairing process (scanning the QR-code).

Authenticating indirectly Also the proposed scheme for indirect authentication encodes all required information in a single QR-code. Aside from choosing indirect authentication in the client- as well as the authenticator-application, the only required step to initiate the authentication process is to scan the QR-code.

5.5 Known issues

5.5.1 Missing application isolation for indirect authentication

Due to the architecture of classical *direct* FIDO authentication, phishing attacks can be prevented easily: As the browser is responsible for forwarding FIDO commands to the authenticator, he can ensure that the website triggering an authentication operation is authorized to do that by verifying the URL.

Unfortunately, as one of the design goals of indirect authentication has been to be independent of browser support, the browser is not involved in forwarding FIDO commands (the authenticator directly communicates with the relying party over the Internet). Therefore,

phishing protection, as it can be implemented for direct authentication, cannot be realized for indirect authentication, at least not the same way.

To minimize the associated risks, it is important to display a warning message on the authenticator's display, that requests the user to check the hostname displayed in the browser's address bar, and asks him to confirm that it is his intention to authenticate to that website with the respective credentials.

CHAPTER 6

Summary and future work

6.1 Unsolved problems

6.1.1 Implementation challenges

To be able to unveil its full security potential, FIDO-like authentication schemes, including some of our adaptations, require certain features to be supported by the underlying hardware platforms and operating systems. Unfortunately, there are lot of potential target platforms that probably do not support some or all of these requirements at this time. See section 2.4.4.3 for a more detailed discussion which and why some platforms are affected.

Component identification Some of the security concepts described in the FIDO specification can only be implemented in a meaningful way if the underlying operating system provides FIDO software components a meaningful way to verify the identity and integrity of calling software components.

Storage of private data Some potential target platforms do not provide methods to prevent secret data from being accessed by unauthorized processes. However, this capability is required to store private keys securely.

6.2 Future work

6.2.1 Client & ASM Attestation

To further improve security, the attestation concept like it is proposed in the FIDO specification for verifying authenticators could be inherited for FIDO clients and ASMs.

6.2.2 Offline revocation

Aside from the problem that the proposed schemes for offline revocation checking cannot provide local (post-revocation) unlinkability, both schemes have two serious flaws.
The first is that anonymity may not be guaranteed in small deployments when the revocation list is too small. Maybe this issue can be coped with by filling the revocation list with dummy entries, but this idea needs further research.
The other disadvantage compared to online revocation checking is that hash collisions can occur, which lead to false positives. A solution to this problem could be perfect hash functions, but as perfect hash functions require a specific finite input set, a revocation scheme using them could be unfeasible in global deployments and therefore requires further research.

6.2.3 Application isolation

The application isolation concept of the UAF ASM API specification[17] ensures that unauthorized and potentially malicious service providers or client applications cannot access the credentials registered with a relying party, by allowing the RP to specify a list of trusted *FacetID*s (see section 2.4.4.1 for details). This however, prevents trustworthy service providers or applications to allow their users authenticating with the credentials of another relying party they are already registered with. On the other side, without this feature, users would be completely unprotected from phishing threats. Further research might bring out an application isolation strategy that represents a reasonable tradeoff between security and versatility.

6.2.4 Prototype

The primary design goal of the prototype was to show that the theoretical concepts proposed in this paper actually work rather than developing a perfectly usable software product. Therefore, several features that would be required by a complete software suite for FIDO authentication are missing. This section lists some of them to show that these issues were considered during the design of the theoretical concepts and the prototype, but also to give hints for further development.

6.2.4.1 Silent authentication

There exist numerous use cases where authentication needs to happen without any user interaction. Let's take an E-Mail client application for example. In password-based authentication scenarios, this problem is simply solved by permanently storing the credentials in a more or less secure way, and the E-Mail client can authenticate the user unattended every time the server asks for it. Having to provide your fingerprint every time you perform an action in your E-Mail client however would depict an unacceptable usability issue.

A simple solution would be to allow the user to approve subsequent authentication requests for a limited period of time for certain credentials. However, far more complex approaches are imaginable, like allowing to enable silent authentication for specific FacetIDs only, or developing elaborate algorithms that decide if user verification is required based on security biasing factors, like authenticator location or preceding successful authentications with other credentials.

6.2.5 Publish/subscribe-based central revocation

Theoretically, *pushing* the information about revoked authenticators to relying parties should be the best strategy regarding security as well as infrastructure load. New revocation information is propagated to the relying parties nearly in real time, and the load on the revocation authority's infrastructure is minimal, as it only transfers data if a authenticator is revoked, or if a relying party subscribes to the revocation authority and requests the revocation data about all already revoked devices.
However, as the issues with offline revocation showed, developing such a scheme isn't trivial and requires further research.

6.3 Conclusion

We could develop a feasible method for centrally managing revocations of compromised FIDO authenticators. This achievement makes FIDO-based authentication a real alternative to passwords even for users with a large number of accounts.
Furthermore we could improve the security and deployability of dedicated authenticator devices by developing our methods for secure pairing of wireless ASM-authenticator interconnections and indirect authentication.
Together, the proposed techniques brought FIDO-based authentication one step further in becoming the new standard authentication mechanism on the Internet.

List of Figures

List of Tables

Listings

Acronyms

AJAX Asynchronous JavaScript and XML. 5

API Application Programming Interface. 6, 12–15, 17, 37, 38, 41–45, 66, 69

APK Android application package. 15

ASM Authenticator-Specific Module. 5, 14–16, 31–34, 37–39, 41, 49, 52–54, 60–62, 65–67, 69, 72

BLE Bluetooth Low Energy. 10

CA Certificate Authority. 16, 17, 21–23

CAR Centralized authenticator revocation. 49, 50, 60

CRL Certificate Revocation List. 19, 20

DAA Direct Anonymous Attestation. 17, 21, 71

DNS Domain Name System. 28, 31, 71

DNSSEC DNS Security extensions. 31

DoS Denial of Service. 48, 51

ECDAA Elliptic Curve DAA. 17

EPID Enhanced privacy ID. 17, 21–23

FIDO Fast IDentity Online. xi, xiii, xv, 2–6, 11–16, 18, 20, 21, 23–26, 28, 31–34, 37–39, 41, 42, 47, 49, 50, 53, 55–62, 65–67, 69

FTP File Transfer Protocol. 24, 25

HMAC Hash Message Authentication Code. 32, 33, 52–54

HTTP Hypertext Transfer Protocol. 2, 5, 15, 24, 25, 28, 37, 38, 41, 44, 45, 69

IA Indirect authentication. 49, 50, 60

ID Identifier. 17, 44, 51, 52, 62, 71

IDL Interface Definition Language. 39, 41, 42

IMAP Internet Message Access Protocol. 2

IMEI International Mobile Equipment Identity. 62

IoT Internet of Things. xi, xiii, 3

IP Internet Protocol. 38

IT Information technology. xi, xiii

JSON JavaScript Object Notation. 38, 39, 41, 44, 69

LF Line feed. 39

MAC Media Access Control. 52

MITM Man-in-the-middle. 32

NFC Near Field Communication. 3

PEM Privacy Enhanced Mail. 45

PHP PHP Hypertext Preprocessor. 38, 72

PII Personal identifiable information. 48

PIN Personal Identification Number. xi, xiii, 11, 31, 32, 56–58

PKI Public Key Infrastructure. 19, 20

POP3 Post Office Protocol version 3. 2

QR Quick Response. 10, 32–35, 37, 52–58, 62

RA Revocation authority. 5, 20, 22–31, 37, 49–52, 59, 62

REST Representational State Transfer. 41, 44

RFCOMM Radio Frequency Communication. 38

RP Relying Party. 5, 14, 20, 21, 23–30, 37, 50–52, 54, 66

SAAC Secure ASM-authenticator channel. 49, 50, 60

SG Security goal. 6, 20, 21, 47–49

SPoF Single point of failure. xi, xiii, 22, 30

TLV Tag Length Value. 39

TPM Trusted Platform Module. 4, 17, 18, 38

U2F Universal Second Factor. 11, 13

UAF Universal Authentication Framework. 2–5, 11–13, 15, 16, 19–21, 23, 32–34, 37–39, 47, 49, 50, 66, 69

UIMP Universal Identity Management Protocol. 10

URL Uniform Resource Locator. 15, 24, 25, 28, 52, 55, 62

USB Universal Serial Bus. 3, 57, 61

W3C World Wide Web Consortium. 12, 13, 16, 17, 39, 42, 57

XML Extensible Markup Language. 71

Bibliography

[1] Introducing Login Approvals. Accessed 2016-04-17. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/introducing-login-approvals/10150172618258920/

[2] About two-step verification. Accessed 2016-04-17. [Online]. Available: http://windows.microsoft.com/en-GB/windows/two-step-verification-faq

[3] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot, "Tapas: Design, implementation, and usability evaluation of a password manager," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 89–98. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420964

[4] E. Hayashi and J. I. Hong, "Knock x knock: The design and evaluation of a unified authentication management system," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '15. New York, NY, USA: ACM, 2015, pp. 379–389. [Online]. Available: http://doi.acm.org/10.1145/2750858.2804279

[5] M. Everts, J.-H. Hoepman, and J. Siljee, "Ubikima: Ubiquitous authentication using a smartphone, migrating from passwords to strong cryptography," in *Proceedings of the 2013 ACM Workshop on Digital Identity Management*, ser. DIM '13. New York, NY, USA: ACM, 2013, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2517881.2517885

[6] FIDO Alliance. Accessed 2016-04-17. [Online]. Available: https://fidoalliance.org/

[7] FIDO Security Reference. Accessed 2016-07-11. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html

[8] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 553–567.

[9] E. Hayashi, B. Pendleton, F. Ozenc, and J. Hong, "Webticket: Account management using printable tokens," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 997–1006. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208545

[10] Web Authentication: A Web API for accessing scoped credentials. [Online]. Available: https://www.w3.org/TR/2016/WD-webauthn-20160531/

[11] LaunchKey. Accessed 2016-04-17. [Online]. Available: https://launchkey.com/

[12] Nok Nok Labs, Inc. Accessed 2016-04-17. [Online]. Available: https://www.noknok.com/

[13] A world without passwords: Windows Hello in Microsoft Edge. Accessed 2016-06-20. [Online]. Available: https://blogs.windows.com/msedgedev/2016/04/12/a-world-without-passwords-windows-hello-in-microsoft-edge/

[14] Windows Hello. Accessed 2016-06-20. [Online]. Available: http://windows.microsoft.com/de-at/windows-10/getstarted-what-is-hello

[15] FIDO Alliance - Download Specifications. Accessed 2016-07-02. [Online]. Available: https://fidoalliance.org/specifications/download/

[16] FIDO UAF Architectural Overview. Accessed 2016-07-05. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-overview-v1.0-ps-20141208.html

[17] FIDO UAF Authenticator-Specific Module API. Accessed 2016-07-05. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-asm-api-v1.0-ps-20141208.html

[18] FIDO AppID and Facet Specification v1.0. Accessed 2016-07-05. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-appid-and-facets-v1.0-ps-20141208.html

[19] FIDO UAF Protocol Specification v1.0. Accessed 2016-07-05. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-protocol-v1.0-ps-20141208.html

[20] E. Brickell and J. Li, "Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities," in *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, ser. WPES '07. New York, NY, USA: ACM, 2007, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/1314333.1314337

[21] Android Keystore System. Accessed 2016-12-01. [Online]. Available: https://developer.android.com/training/articles/keystore.html

[22] Y. Shaked and A. Wool, "Cracking the bluetooth pin," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '05. New York, NY, USA: ACM, 2005, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/1067170.1067176

[23] Bluetooth SIG. Simple Pairing Whitepaper. Accessed 2016-11-30. [Online]. Available: https://web.archive.org/web/20061018032605/http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf

[24] FIDO UAF Authenticator Commands v1.0. Accessed 2016-09-03. [Online]. Available: https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-authnr-cmds-v1.0-ps-20141208.html

[25] WebIDL (Second Edition). Accessed 2016-09-02. [Online]. Available: https://heycam.github.io/webidl/