STRENGTHENING PASSWORD-BASED WEB AUTHENTICATION THROUGH MULTIPLE SUPPLEMENTARY MECHANISMS

by

Furkan Alaca

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Carleton University
Ottawa, Ontario
May 2018

In memory of my grandparents.

Table of Contents

List of Tables

List of Figures

Abstract

User authentication is one of the primary mechanisms that protects online accounts from break-in by attackers. Password-based authentication is currently the most widespread form of user authentication, but has many well-documented usability and security drawbacks. As an increasing number of consumer, financial, governmental, and other organizations move towards offering services online, users are burdened with creating and managing increasingly large portfolios of online accounts; this increased user burden exacerbates the drawbacks of password-based authentication. This thesis contributes to the reinforcement of password-based authentication by pursuing parallel mechanisms that improve security without further burdening users—this is a prominent avenue of improvement, given the continued dominance of password authentication. To that end, our contributions achieve three broad goals. First, we identify, develop, and evaluate device fingerprinting mechanisms for use alongside passwords, and offer guidance on their use, to enhance the security of password-based web authentication. Second, we expand on the concept of mimicry resistance, a dimension that has thus far been overlooked in the design and study of web authentication schemes. We develop a comprehensive methodology for evaluating the mimicry resistance of web authentication schemes and provide guidance on how to combine multiple schemes alongside password authentication to maximize the benefits gained. Third, we perform a comprehensive analysis and evaluation of a broad range of single sign-on (SSO) schemes, which reduce password fatigue by allowing users to access a multitude of online services through a single master password. We identify design properties of SSO schemes and develop an evaluation framework that highlights their benefits and drawbacks, revealing trade-offs between different designs. These three contributions encompass complementary approaches that can be used together to improve online security with minimal impact on usability.

Acknowledgements

I would like to thank my supervisor, Dr. Paul van Oorschot, for his support and valuable guidance throughout the course of my PhD studies. I am grateful for his mentorship over the years, which has helped me to become a better academic. Thanks also to my thesis examination committee members Dr. Scott Knight, Dr. Carlisle Adams, Dr. Andy Adler, and Dr. Andrew Patrick for carefully reviewing my work and providing valuable suggestions towards the improvement of this thesis, and to Dr. Anil Somayaji for the many helpful discussions.

Thanks to all my friends and colleagues from the Carleton Computer Security Lab for their support, and especially to AbdelRahman Abdou for the countless hours of helpful research discussions, and to David Barrera, Daniel McCarney, and Mohamed Alsharnouby for always being willing to offer their technical insight and feedback.

I gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada–Canada Graduate Scholarship program, the Ontario Graduate Scholarship program, and Carleton University for funding this research.

Last, but not least, I am deeply grateful to my parents, Drs. Ayşe and Şaban Alaca, and to my sisters Zahide and Betül for their moral support throughout the course of my studies.

Chapter 1

Introduction

The Internet has changed many aspects of our daily lives, from the way we communicate to the way we shop, consume media, and pay our bills. These changes continue to reshape our lives even as of the time of writing, with new services changing the way we track our health, automate household tasks, and transit from one part of town to another; the number and the diversity of such online services will only continue to grow. A common requirement for accessing the vast majority of these online services is to create an account with the corresponding service provider (SP). Online accounts serve firstly to identify ourselves to SPs, and secondly to prevent other users from accessing our private information stored on those accounts or from performing unauthorized transactions on our behalf. When accessing online accounts, the overwhelming majority of SPs require users to identify themselves by a user name and prove that identity with a password selected at the time of account creation. Password-based authentication has many well-documented security and usability drawbacks, which we summarize as follows:

Security: Password capture and replay. Password authentication is a knowledge-based scheme, meaning that any attacker that learns a user's account password can break into the account. This is in contrast to, for example, possession-based schemes that require users to be in possession of a hardware device (e.g., a USB token) to access their accounts. Passwords are susceptible to being captured via a number of categories of attacks, e.g., by malicious software installed on a public PC that records all keystrokes, by fraudulent websites that masquerade as legitimate websites and ask users to type in their password, or by observing a user (either in person or remotely through a camera) as they type in their password. Attackers may also attempt to guess users' passwords; to improve the effectiveness of guessing attacks, attackers may leverage databases containing user passwords that have been stolen or leaked from different websites.

Usability: Password selection and re-use. It is difficult for users to select passwords that they can easily remember but which are also difficult for attackers to guess. This difficulty is exacerbated by the advice given by computer professionals to users that they should select unique passwords across different accounts, to ensure that a compromised password for one account does not lead to users' other accounts being compromised (cf. the above mention of attackers' use of stolen password databases). Due to the large (and growing) number of online accounts that users must manage, this advice is impractical for users to implement.

Although numerous alternatives have been proposed to replace password authentication, none have been successful in displacing it as the dominant scheme for web authentication. While some current password alternatives offer more security and usability benefits, none retain the *full* set of benefits that passwords offer. For example, possession-based authentication schemes may not require users to memorize a password, but they do require users to carry around a hardware device with them. Moreover, *all* password alternatives face obstacles to adoption, e.g., due to development costs for SPs or the need for preserving compatibility with existing end-user systems. Since passwords do not seem to be disappearing [86], a prominent avenue of improvement is to reinforce their security by parallel mechanisms [26] without further burdening users.

1.1 Motivation and Thesis Goals

We believe that there is a need for practical mechanisms that can be easily deployed to improve security with minimal impact on usability, without necessarily having to supplant current password-based authentication that is already familiar to users. The general motivation behind the research contributions in this thesis is to contribute towards identifying and developing mechanisms to fulfill the aforementioned need. More specifically, the principal goals of this thesis are as follows:

G1. Identify and develop mechanisms, and offer guidance on their use, for enhancing the security of password-based web authentication with minimal (or no) negative impact on usability. We focus especially on the use of device fingerprinting, a mechanism by which SPs can identify diverse attributes of users' devices to assemble device identifiers, analogous to how humans can be identified by their physical fingerprints.

G2. Extend the concept of mimicry resistance in user authentication (a concept that has been considered for user authentication to software applications running on the user's device) to user authentication to web applications; doing so requires determining the differences in threats relevant to user-to-device and user-to-web authentication, determining what classes of schemes possess mimicry resistance on the web, how to use and combine such schemes together, and how to evaluate the benefits that they provide. Mimicry here refers to user impersonation techniques employed by attackers (e.g., by attempting to appear as if authenticating from a location from which a user usually authenticates) to gain unauthorized access to users' online accounts. In line with our previously-stated objective of minimizing any negative impact on usability, we place special emphasis on mimicry-resistant schemes that can operate without any (or with minimal) explicit new user actions.

G3. Perform a comprehensive analysis and evaluation of a broad range of authentication schemes designed to allow users to securely leverage a single password to access multiple services. These authentication schemes are known as single sign-on (SSO) schemes, and they are designed to alleviate the discussed usability drawbacks (namely, password selection and re-use) of conventional password-based authentication by significantly reducing the number of passwords that users are required to remember. A typical SSO scheme requires the user to authenticate to an Identity Provider (IdP) to establish an authenticated session; throughout the session, the IdP provides proof-of-identity to authenticate the user to other SPs without requiring any additional authentication tasks from the user. While a wide variety of SSO schemes have been proposed over the last decade, there is a lack of clarity regarding what benefits are offered by different schemes, which benefits are more desirable to different stakeholders (namely users, SPs, and IdPs) and whether there are any competing interests between them, and which benefits are better suited in different environments or use-case scenarios (e.g., medium-value vs. high-value accounts). Therefore, there is a need to study the high-level design of SSO protocols to determine the design choices to be made, the benefits and drawbacks that result from different choices, and the inherent trade-offs that are involved in SSO protocol design.

1.2 Contributions

The research contributions in this thesis are focused on achieving the goals outlined above. Below, we summarize the contributions made in achieving each of the three principal goals of this thesis.

G1. Device fingerprinting for augmenting password-based web authentication.

1. Identified and classified 29 available device fingerprinting mechanisms, primarily browser-based and known, but including several network-based methods and other methods not discussed in prior academic literature.

2. Identified and assessed properties distinguishing device fingerprinting mechanisms that are more suitable for application to augment user authentication.

3. Defined a series of adversarial models within the context of device-fingerprint-augmented user authentication.

4. Offered guidance on practical issues in the implementation, deployment, and use of device fingerprinting within the context of user authentication.

G2. First systematic study of mimicry resistance for web authentication.

1. Investigated the mimicry-resistance dimension in web authentication, including ranking schemes under three sub-classes of mimicry resistance.

2. Investigated schemes that possess mimicry-resistant characteristics, namely device fingerprinting, user geolocation, and physically unclonable functions (PUFs), and evaluated their degree of mimicry resistance when used for web authentication.

3. Constructed a comprehensive evaluation methodology, which includes:

   (a) A two-dimensional chart that visually reflects the ability of authentication schemes to resist two categories of attacks leading to account break-in: (i) exposure or theft of an authentication secret (e.g., a password), and (ii) mimicry of user (or user device) actions.

   (b) A set of benefits that ideal mimicry-resistant authentication schemes may provide.

4. Used the evaluation methodology, in combination with a set of benefits defined in a prior well-known evaluation framework for web authentication schemes, for the first detailed exploration of the benefits of combining mimicry-resistant web authentication techniques with password-based authentication.

G3. Single Sign-On.

1. Prior work has explored various aspects of SSO security. We perform the first comprehensive analysis and comparison of a broad range of SSO systems proposed and/or deployed within the last decade, including two hardware-based SSO schemes that are undergoing large-scale deployment, FIDO UAF [117] and Mobile Connect [79].

2. Identified different design properties of SSO schemes and developed a taxonomy based on a categorization scheme across the design properties.

3. Developed an evaluation framework that highlights benefits and drawbacks of SSO schemes based on their design properties as defined under our taxonomy, and analyzed a representative set of 14 SSO schemes under this framework.

4. Identified trade-offs between different design goals, and identified how various SSO schemes can be augmented with existing techniques to achieve specific benefits while forgoing others. Such trade-offs allow SSO schemes to be tailored to different needs and scenarios.

1.3 Publications

Work from Chapter 3 has been published at a peer-reviewed conference, and is available in the conference proceedings:

> F. Alaca, P. C. van Oorschot. Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods. ACSAC 2016, Dec. 5–9, Los Angeles.

The work from Chapters 4 and 5 has been submitted for publication; a preprint is available:

F. Alaca, A. Abdou, P. C. van Oorschot. Comparative Analysis and Framework Evaluating Mimicry-Resistant and Invisible Web Authentication Schemes. arXiv:1708.01706 [cs.CR] (arXiv.org e-print, Cornell University Library, 16 pages, 4 Aug. 2017).

The above work was done in collaboration with Dr. AbdelRahman Abdou when he was a postdoctoral fellow at Carleton. Abdou (whose own PhD research was on secure location verification) was involved in research discussions throughout the course of the work, and also contributed to the writing—he was particularly instrumental in identifying, classifying, and evaluating geolocation schemes suitable for web authentication.

The work from Chapter 6 has been submitted for publication; a preprint is available:

F. Alaca, P. C. van Oorschot. Comparative Analysis and Framework Evaluating Web Single Sign-On Systems. arXiv:1805.00094 [cs.CR] (arXiv.org e-print, Cornell University Library, 38 pages, 30 Apr. 2018).

All of the above papers were written under the supervision of Prof. Paul van Oorschot, who offered mentorship and valuable feedback throughout the research and writing process.

1.4 Scope of Thesis

This thesis is focused primarily on user authentication on the web. However, user authentication is also relevant to other contexts which may have different constraints or requirements. For example, many smartphones implement user authentication mechanisms (e.g., 4-digit PIN, fingerprint recognition, or face recognition) to deny unauthorized access to the phone's contents and functionality—this is an example of *local* authentication. We explain several differences in the security requirements and relevant attack models between local authentication and web authentication schemes in Sections 2.1.2 and 6.2.5. In the interest of completeness, we provide an overview of local authentication schemes from the literature, particularly those based on environmental or behavioural measurements (e.g., searching for nearby wireless networks to

determine whether the user is logging in from a familiar location), that offer mimicry-resistance benefits.

1.5 Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background and related work on web authentication, device fingerprinting, the concept of mimicry resistance, and single sign-on. Chapter 3 covers our work on augmenting password-based authentication with device fingerprinting. Chapter 4 covers our framework for evaluating mimicry resistance in web authentication schemes, and Chapter 5 performs an evaluation of a representative set of schemes, highlighting and discussing the benefits offered by different schemes and how different schemes can be combined to maximize the benefits offered. Chapter 6 presents our taxonomy and evaluation framework for SSO schemes, along with an evaluation of a representative set of schemes. Chapter 7 provides an integrated discussion of insights and recommendations derived from this thesis, and identifies open problems for further research that have been identified as a result of our work.

Chapter 2

Background and Related Work

This chapter explains concepts and defines terms used throughout the remainder of this thesis, and further defines our scope by positioning our work in relation to existing literature. While this chapter provides more general background and related work relevant to the overall context of the thesis, more specific background and related work is presented in subsequent chapters.

2.1 User Authentication

Authentication is the process that takes place between a user (i.e., the *claimant*) that is claiming a certain identity and a *verifier* that assesses the validity of the user's claim. In web authentication, verifiers are websites that control access to user accounts, i.e., by ensuring that only account holders should be able to access and perform transactions on their respective online accounts. Typically, users claim an identity by entering a user name, and websites verify user identity by asking users to enter a secret text password that is typically selected by users at the time of account creation.

2.1.1 Approaches to User Authentication

User authentication is conventionally based on one of four approaches, as follows:

1. **Personal knowledge (i.e., something the user knows).** Examples include text passwords, 4-digit PINs, personal knowledge questions (e.g., *"What is your mother's maiden name?"*), and graphical passwords (e.g., drawing a pattern on a touchscreen device).

2. **Physical possession (i.e., something the user has).** Examples include hardware USB tokens, or mobile phones. User possession of a USB token may be verified by the user plugging it into the device they are authenticating from;

possession of a mobile phone may be verified by the verifier website sending the user a randomly-generated text message, and asking the user to type it into the website.

3. **Physical biometrics (i.e., something the user is).** Examples include fingerprint, iris, or facial recognition.

4. **Behavioural biometrics (i.e., something the user does).** Examples include user recognition via typing rhythm, mouse or touchscreen usage dynamics, or gait analysis (e.g., by collecting smartphone accelerometer data while the user is walking).

Other approaches, as will be discussed later, include behavioural characteristics (other than biometrics) that may involve geographic location, habitual user actions, and capabilities (e.g., being able to respond to an e-mail sent to a specific address or domain).

Local vs. Remote Authentication

Local user authentication refers to authentication between a user and a system running on a device in their physical possession. Examples include logging into a laptop or desktop computer with a password, or unlocking a smartphone using a 4-digit PIN or physical biometric. In contrast, *remote* user authentication refers to authentication between a user and a service running on a remote server, typically over the Internet or other network link. Remote authentication occurs through a *user agent* that communicates to the remote server on the user's behalf; for example, in web authentication (our primary area of focus), remote servers consist of websites, and user agents consist of web browsers.

Start-of-Session vs. Continuous Authentication

Conventionally, websites authenticate users once at the beginning of each browser session; we refer to this as *start-of-session* authentication. Subsequent to verifying user identity (e.g., by validating a user-entered password), websites provide the browser with a *session cookie* that remains valid for a pre-determined period of time (e.g., several hours) or until the user explicitly logs out of their account. For as long as

the session cookie remains valid, users typically do not need to re-authenticate. In addition to start-of-session authentication, some websites may also require users to complete additional authentication tasks before carrying out highly sensitive transactions; this practice is referred to as *progressive* authentication [166].

Continuous authentication refers to mechanisms that continuously authenticate users *throughout* a session, typically by performing unobtrusive measurements without requiring any deliberate user action. A benefit of continuous authentication schemes is that they can also *de-authenticate* users when the obtained measurements deviate beyond a certain threshold of what is expected. Two categories of schemes proposed thus far in the literature for continuous authentication are:

1. **Implicit authentication.** These schemes are based on behavioural biometrics that can be measured without any *deliberate* action by users, i.e., by obtaining measurements from actions that users perform throughout the regular use of an application, such as swiping on a touchscreen.

2. **Zero-effort authentication.** These schemes allow verifiers to automatically authenticate users by detecting physical user possession of one or more devices (e.g., a miniature hardware device specifically designed to act as an *authenticator device*).

Jakobsson et al. [94] propose an implicit authentication framework that continuously obtains measurements using smartphone sensors (e.g., GPS, accelerometer, WiFi and Bluetooth connections) throughout the day to maintain an authentication score that increases when habitual user actions are observed, and decreases when unusual actions are observed. They propose that the authentication score can be used either to replace passwords or as a second authentication factor used in combination with passwords. Khan et al. [101] evaluate six different smartphone-based implicit authentication schemes (that measure, e.g., touchscreen input, touchscreen keyboard usage, gait pattern, phone call/text usage), and offer relevant insights. For example, schemes vary in the amount of data collection needed to make a decision—some require only milliseconds, whereas others may require on the order of 10 seconds.

ZIA (Zero-Interaction Authentication) [44, 144] is an example of zero-effort authentication; users wear a small authenticator device that communicates over a short-range wireless link with their computer. When the authenticator device is within range, it automatically provides the user access to the computer's encrypted file system. Conversely, the encrypted file system is made inaccessible after the authenticator device leaves the wireless range of the computer.

Implicit schemes and zero-effort schemes have generally been proposed for local authentication. With many of these techniques, remote authentication is more difficult to secure, since they require verifiers to rely on measurements collected from user-controlled devices. Therefore, attacker-controlled devices may impersonate users by sending falsified measurements to remote verifiers. Such threats may be mitigated with hardware-based trusted computing[1] technologies designed to allow verifiers to validate that the measurements collected by user devices have not been tampered with. For example, GPS coordinates can be obtained from user devices through a hardware Trusted Platform Module (TPM) [154, 121].

Theoretically, implicit and zero-effort authentication schemes may be "converted" to start-of-session authentication schemes, but they may not necessarily be practical or efficient to use. For example, a scheme that measures touchscreen keyboard usage could ask users to type a paragraph on their phone, or a scheme that measures gait may ask users to stand up and walk around to collect accelerometer measurements.

Bilateral Authentication

Bilateral authentication refers to schemes that cross-check or compare information observed by two different devices—generally between an authenticator device in the user's possession, and the device that the user is authenticating to. For example, for in-person credit card transactions, the verifier may obtain the GPS co-ordinates from the mobile phone of the credit card owner and compare it with the GPS co-ordinates of the payment terminal [197, 121, 201]; this ensures that the credit card owner is physically present when the transaction is being made.

[1] Trusted computing refers generally to hardware technologies that allow verifiers to cryptographically verify that a device (e.g., a smartphone or personal computer) has executed a computer program without altering the underlying code or tampering with any program output [185].

Authentication schemes may be both bilateral and zero-effort: Zero-Effort Bilateral Recurrent Authentication (ZEBRA) [120] is a local authentication scheme that requires users to wear a bracelet equipped with an accelerometer. While users type on their computer keyboard, the computer wirelessly receives accelerometer readings from the bracelet, and compares the readings with the keystrokes made on the keyboard. Comparing the accelerometer data with the keystrokes allows the computer to verify that the user typing on the keyboard is wearing the bracelet. Unlike ZIA (discussed previously), which relies only on users' physical proximity to their computers (so, e.g., a physically-present attacker may be able to use a victim's computer while the user is nearby but has turned their back), ZEBRA verifies that the bracelet owner is actively using the computer.

Sound-Proof [99] is a bilateral remote authentication scheme that determines whether the user's smartphone and the computer the user is authenticating from are in close physical proximity. The verifier website records the ambient sound from the microphone on the computer (through a browser API) and transmits it (over the Internet) to the user's smartphone. The Sound-Proof application on the user's smartphone also records the ambient sound from its own microphone, and compares it with the sound file received from the user's browser. If the two sound files match, this indicates that the smartphone and computer are in close physical proximity, and the smartphone application sends a cryptographically-signed assertion to the website to approve the user authentication. To protect against attackers in close physical proximity to users (e.g., an attacker seated at a coffee shop nearby the user; also cf. the example above for ZIA), Sound-Proof requires explicit user consent at the time of authentication by tapping a button on the smartphone application to approve the authentication attempt.

2.1.2 Attacks on User Authentication

In this thesis, we categorize attacks on user authentication into two broad categories: (1) Credential capture and reuse, and (2) Mimicry attacks.

Credential Capture and Reuse

The vast majority of user authentication schemes rely on the protection of a secret, known as a *credential*. Knowledge-based schemes, such as passwords, require users to protect the secrecy of their password. Possession-based schemes, such as hardware USB tokens, require the token to protect the secrecy of cryptographic keys. Many physical biometrics also require secrecy and are arguably unsuitable for remote authentication (e.g., [25]). For example, attackers have used high-resolution public photographs to "clone" fingerprints of high-profile individuals [103]. Moreover, in remote authentication schemes where a digital representation of the fingerprint is sent from the user's device to the remote server, an attacker that captures the digital representation could later reuse it to impersonate the user.

Bonneau et al. [25] evaluate a wide range of web authentication schemes proposed as password replacements. In their evaluation framework, they develop evaluation criteria that reflect each authentication scheme's susceptibility to different types of attacks, all of which relate to credential capture and reuse. We summarize these attacks as follows, and highlight how each attack relates to password authentication:

1. **Online guessing attacks** involve guessing the user credential. Passwords are vulnerable to these attacks, since users tend to choose predictable passwords that are easy for attackers to guess. A helpful precaution against such attacks is to limit the rate at which users can attempt to authenticate (e.g., lock the account for several hours after 5 unsuccessful attempts) [8].

2. **Physical observation** involves physically observing the user while they are authenticating, e.g., watching them type in their password.

3. **Internal observation** involves capturing credentials via malware installed on the user's device or by intercepting communication between the user's device and the verifier.

4. **Leaks from other verifiers** involve attackers impersonating users to verifiers using information stolen from a different verifier. For example, users often reuse passwords across multiple websites; this allows attackers to steal a password

database from one website and attempt to use the same passwords to impersonate users on another website. Websites can reduce the scalability of such attacks by not directly storing user passwords; instead, the recommended practice is to use secure password-hashing functions, which generate cryptographic transformations of text passwords that cannot be easily reversed.

5. **Offline guessing attacks** involve stealing a password database containing secure hashes of passwords, and finding password strings whose transformations match with the stored hashes. More computationally-intensive (i.e., "slow") hash functions increase the time cost of each individual guess; however, while slow hash functions are helpful for reducing the "guessability" of medium- to high-strength passwords, they are ineffective for protecting the simplest passwords (e.g., "123456") that would only take several attempts to guess (the optimal attack strategy is to guess in descending order of the most commonly-used passwords).

6. **Phishing** involves deceiving users by directing them to a fraudulent website that pretends and appears to be a website (e.g., a bank) that users normally use. A common technique is to e-mail users a message asking them to log in to their account (e.g., to view a new notification) and to include a link to a phishing website. When the phishing website asks users to log in to access their account, users may provide their credentials (e.g., passwords) if they believe the website to be legitimate. This technique allows attackers to harvest large numbers of passwords.

7. **Targeted impersonation** involves attacker impersonation of users they are personally acquainted with, by exploiting knowledge of personal details such as birth date or names of relatives. While there is no evidence to suggest that passwords can be guessed in this manner, personal knowledge questions commonly used for password reset (e.g., "What is your mother's maiden name?") may be guessed in this manner [27].

8. **Physical theft** involves stealing a user's device, such as a USB authenticator token, that stores secret credentials.

Mimicry Attacks

Mimicry is the impersonation of users not through re-use of a captured secret, but through imitation of user behaviour or user device (i.e., hardware and software) behaviour. Mimicry attacks have been studied within the context of intrusion detection systems [202]; for example, an attacker may hijack control of an application running on a device and attempt to evade detection by simulating the normal behaviour of the program while inserting a malicious sequence at some point during execution [203]. Mimicry attacks are easier to carry out if the expected behaviour of legitimate applications is known to attackers, which is typically the case for publicly-available software.

Implicit authentication schemes for local authentication do not necessarily rely on an explicit secret, but instead their security typically relies on the difficulty of mimicking user behaviour that is readily observable. However, mimicry attacks have been found to be feasible against some implicit schemes [101, 16]; for example, an attacker may observe and imitate a user's keyboard usage dynamics. However, such attacks are highly unscalable, since they are highly targeted and require attacker possession of the user's device. Mimicry attacks can also be made more difficult by combining multiple implicit authentication schemes.

In Chapters 4 and 5, we extend the concept of mimicry resistance to the domain of web authentication.

2.1.3 Account Recovery Schemes

It is common practice for verifiers to provide fallback authentication schemes for account recovery purposes, in case users lose their credentials for their primary means of authentication. Some authentication schemes that may be too unusable or inefficient to use as a primary means of authentication may be acceptable to use as a fallback mechanism, since it will be used relatively infrequently. However, choosing a fallback scheme that is less secure than the primary authentication scheme weakens the overall security of the system, since attackers can bypass the primary authentication mechanism and attack the fallback mechanism instead. For example, personal knowledge questions are sometimes used for account recovery, but have been demonstrated to be even weaker against guessing attacks relative to passwords [27]. We revisit this

challenge throughout our discussions on mimicry-resistant authentication schemes in Chapters 4 and 5, and single sign-on systems in Chapter 6.

2.1.4 Multi-Factor Authentication

Multi-factor authentication combines two or more types of authentication mechanisms (i.e., factors) to provide stronger security. The overall security of the multi-factor authentication scheme depends on the security of each factor, and the "independence" of each factor from the other(s). For example, carrying two USB hardware tokens on the same keychain would provide little benefit compared to a single hardware token. In Chapter 5, we offer further discussion and insight on combining multiple authentication schemes and pitfalls to avoid.

2.2 Machine Authentication

Authentication mechanisms can also be classified based on the type of entity that is being authenticated. While user authentication encompasses all schemes that authenticate users to machines (whether local or remote), there is also a need to authenticate machines (either to users or to other machines).

2.2.1 Device Authentication

Device authentication involves authenticating a device to a verifier such as a remote server. For example, a mobile phone needs to authenticate itself, typically via a cryptographic secret stored on a SIM card, to connect to a mobile network. Some systems may implement *two-stage* authentication, which combines a *local user authentication* stage and a *device authentication* stage; for example, when making a credit card payment with chip-and-pin technology, the payment terminal first communicates to an authentication server on the payment network to validate the authenticity of the credit card itself (i.e., device authentication), and users are then authenticated by entering a numerical PIN that is verified locally by the payment device (i.e., local user authentication). Two-stage authentication can also be applied to web authentication, as discussed in subsequent chapters of this thesis.

Device authentication can also occur between devices (i.e., machine-to-machine

authentication). For example, self-driving cars may authenticate each other when communicating information about traffic flow or obstacles on the road—in this particular example, the purpose of the authentication may be to ensure that the traffic information being received is from a car (and not, e.g., a rogue transmitter device) that is manufactured by a legitimate car maker and has therefore undergone any necessary regulatory certification.

Device authentication is often easier to do securely than user authentication, since it is not constrained by the requirement to be usable by human users. A typical means of device authentication is through cryptographic challenge-response protocols (using either symmetric-key or public-key cryptography); for example, the verifier may send a cryptographic challenge to the device, which in turn generates a response that proves (to the verifier) the device's possession of a cryptographic key.

2.2.2 Server Authentication

Server authentication involves authenticating remote servers to users. On the web, this is achieved through HTTPS, which is built upon the TLS public-key infrastructure. For example, when a user visits the website www.carleton.ca over HTTPS, the web browser authenticates the web server to ensure that the communication between the web browser and www.carleton.ca is not being intercepted by an attacker. Upon successful authentication, web browsers display a visual cue in the border area of the browser interface (which cannot be manipulated by the web content being displayed), such as a green lock icon, to indicate to users that the web server has been authenticated. Server authentication is out of the scope of this thesis. However, techniques discussed to augment user authentication (e.g., device fingerprinting and geolocation) can also be used to strengthen the security of server authentication. For example, Abdou [3] proposes a scheme for server location verification to augment TLS authentication in web browsers.

2.3 Device Fingerprinting

Device fingerprinting is a technique by which a server collects information about a device's software and/or hardware configuration for the purpose of identification. Nmap [116] is a classic tool that can fingerprint devices over the network by sending

specially crafted packets to hosts and analyzing the response packets to find features that are attributable to specific hardware, operating systems, or software stacks. More recently, web analytics and advertising services have been performing device fingerprinting through web browsers to track and analyze users' browsing habits [56, 4].

2.3.1 Browser-based Device Fingerprinting

Web browsers reveal information to websites about the host system both explicitly by exposing information such as screen resolution, local time, or OS version, and implicitly by leaking information about the device's software or hardware configuration through observable differences in browser behaviour. Eckersley [56] published the first research paper discussing in detail the concept of browser-based device fingerprinting and showed that websites could identify users and track their browsing habits by collecting basic information such as device IP address, time zone, screen resolution, and a list of supported fonts and plugins. Web browsers are intentionally designed to provide extensive device information to allow websites to tailor and optimize content for a wide variety of target devices; for example, when a website detects that the user is on a smartphone, it may provide a user interface more suitable for a small touchscreen display.

Mowery et al. [131, 132] later proposed two more advanced fingerprinting methods; the first measures performance characteristics of the browser's JavaScript engine, and the second renders text in an HTML5 canvas element[2] to distinguish font-rendering techniques across different software and hardware platforms. Bojinov et al. [23] demonstrate how smartphones can be fingerprinted via a multitude of onboard sensors; in particular, they use the device's accelerometer calibration error and the frequency response of the speakerphone-microphone system.

2.3.2 Web Tracking

Empirical studies by Acar et al. [5, 4] and Nikiforakis et al. [146] reveal extensive use of device fingerprinting by advertisers as a fallback mechanism to track users, should

[2] An HTML5 canvas element creates a rectangular area on a web page in which 2D and 3D graphics can be rendered using JavaScript.

they clear their browser cookies[3]. Advertisers also save identifying information via less conventional storage mechanisms, e.g., Flash cookies, which are more difficult for users to delete [124]; this information may be used to reconstruct browser cookies, should the user clear them [182, 13, 126, 4]. Vastel et al. [200] use machine learning to correlate browser fingerprints that change over time, e.g., due to software updates or changes to device configuration.

2.3.3 Fingerprint Diversity

Spooren et al. [183] argue that the relative lack of diversity in mobile device fingerprints, compared to desktop computers, makes them less reliable for risk-based authentication. For example, they found that over 90% of Android devices use an identical set of fonts, and that all iOS devices and Windows Phone devices respectively share an identical set of fonts. However, the study did not include many of the more recent and advanced fingerprinting mechanisms discussed herein, in Chapter 3. Laperdrix et al. [113] collected and analyzed fingerprints from 119,000 devices, and found that while some mobile device attributes are less diverse (e.g., limited browser plugin support), other attributes (e.g., user-agent string and canvas fingerprinting) are much more diverse than on desktops.

2.3.4 Fraud Detection Based on Device Fingerprinting

Eisen [57, 58] is the inventor on two patents relevant to device fingerprinting: the first describes how a server may detect fraudulent transactions by recording the difference between the server's local time and that of each client (to determine the client's time zone, observance of daylight saving time, and drift from UTC); the second describes a generalized framework wherein a server can obtain a fingerprint on each page that the client requests from the server and signal a warning for a session-tampering attempt when there is sufficient change in the fingerprint. A patent by Varghese et al. [198] describes how a client's device fingerprint can be used as an index to retrieve its associated risk of fraud from a central database of fingerprints for devices suspected of participating in fraud, and accordingly grant the client an appropriate level of

[3]A browser cookie consists of data which a website can save on the user's local machine upon being visited by the user, and can be updated or retrieved by the website on subsequent visits. Cookies are often used to save users' website preferences or to track their browsing behaviour.

access to the account. A number of commercial fraud detection services [123, 192] employ device fingerprinting to identify and block transactions initiated from devices known to be associated with a high risk of fraud.

2.3.5 Privacy Concerns

Most of the existing research on browser-based device fingerprinting focuses on privacy concerns, and aims to develop techniques that limit its usefulness for tracking users; e.g., by restricting the subset of browser functionality made available to websites [181], by adding random noise to output generated by various browser functions [145, 112], or by reducing the level of precision in browser-provided information [151]. Nikiforakis et al. [146] discuss the difficulties of thwarting fingerprinting techniques with client-side privacy software; e.g., differences in JavaScript implementations across browser vendors can reveal which browser is being used even if the user-agent string[4] is obscured; web proxies can often be circumvented to reveal the user's true IP address through external plugins such as Flash or Java (cf. [137]); and user manipulation of the device fingerprint may result in a less common fingerprint that is even more useful for identifying the user.

2.3.6 Augmenting Web Authentication with Device Fingerprinting

Chapter 3 of this thesis is focused on the little-explored use of device fingerprinting for the purpose of augmenting web authentication. Device fingerprinting overlaps both implicit and zero-effort authentication, since the information collected reflects both device-specific information (e.g., system performance) and user behaviour (e.g., user-selected preferences, installed browser extensions)—however, the distinction between the two categories may sometimes be ambiguous, since device-specific information reflects user behaviour through their choice of device. Since device fingerprinting leverages personal devices that users already use, it does not require users to carry an additional dedicated hardware token.

While implicit authentication schemes such as recognition of typing rhythms or touchscreen usage patterns could also be performed within web browsers, using them

[4]The user agent string is an identifying string sent by the web browser to a server, which can include information such as the browser and operating system versions.

for start-of-session authentication requires asking users to perform additional tasks such as typing a paragraph or swiping around objects on a touchscreen. Our interest in device fingerprinting stems primarily from the usability benefit that it does not require user interaction.

Aside from the following work that we summarize from the academic literature, to the best of our knowledge there is no other substantial exploration of techniques for augmenting web authentication with device fingerprinting. Unger et al. [194] propose enhancing session security through server-side monitoring of certain web browser attributes (e.g., user-agent string, supported CSS features) to help detect session hijacking.[5] Preuveneers and Joosen [162] propose a protocol that monitors various parameters throughout an authenticated session, such as user IP address and time of access; applies a similarity-preserving hash function to enable privacy-preserving fingerprint storage that allows similarity checks between stored fingerprints and subsequently-collected fingerprints; and uses a comparison algorithm that assigns a weight to each attribute (based on its usefulness for identifying that particular client) to determine if the fingerprint has changed significantly enough to ask the user to re-authenticate. Van Goethem et al. [196] propose an accelerometer-based device fingerprinting mechanism for multi-factor mobile authentication. Freeman et al. [66] propose a statistical framework to detect suspicious login attempts using various browser attributes and other parameters such as time of access.

2.4 Single Sign-On

Single Sign-On (SSO) is an umbrella term for authentication schemes and architectures that allow users to rely on a single master credential to access a multitude of online accounts. We categorize SSO schemes across two broad categories: federated identity systems (FIS) and credential managers (CM). FIS establishes a means of communicating user identity across administrative domains; this allows users to

[5] Session hijacking is an attack that bypasses user authentication by taking control of an existing authenticated session. On the web, this is typically achieved by session cookie theft, as discussed further in Section 2.4.1

authenticate to an Identity Provider (IdP) that can communicate proofs of user identity to Service Providers[6] (SPs), thereby granting users access to SP services without needing to re-authenticate. CM-based SSO methods store SP-specific credentials (e.g., passwords or cryptographic keys) and automatically send them to SPs on behalf of users; CMs are typically protected by a single master credential such as a password or a hardware token containing a cryptographic key (e.g., USB key or smart card). In Chapter 6, we classify both FIS- and CM-based SSO schemes into more granular subcategories, while identifying benefits and drawbacks associated with different approaches. While SSO has long been used in enterprise networks to enable users to access network services and applications with a single set of credentials (e.g., see Kerberos [143]), we focus specifically on SSO designed for web authentication.

2.4.1 Related Work on SSO

Bonneau et al. [25] evaluate 35 authentication schemes, including a number of SSO systems, based on a usability, deployability, and security (UDS) framework. Among all categories of schemes considered, FIS (e.g., OpenID) and CM (e.g., password managers) schemes retained the most usability and deployability benefits of passwords, while improving security. The usability benefits of SSO stem from the reduced user memory burden from having to remember fewer passwords—however, while password managers offer the benefit of working with all password-based websites, FIS have the drawback of having much more limited SP support. The security benefit of SSO in general hinges on the assumption that users will be able to choose and remember a stronger master password (since they will have fewer passwords to remember); additionally, the security benefits offered by CM-based schemes rely on users picking (or randomly generating) unique passwords across SP accounts, to provide resilience against online guessing attacks and to limit the damage caused by any single compromised password to a single corresponding SP. Our analysis of SSO schemes in Chapter 6 is largely applicable with or without these assumptions, and our taxonomy and evaluation framework are complementary to the UDS framework in that we identify and analyze additional usability, deployability, security, and privacy benefits

[6]Often also referred to as Relying Parties (RPs), i.e., when they rely on on other parties to authenticate users.

specifically relevant to SSO systems, independent of the user authentication mechanism used.

NIST's Digital Identity Guidelines [74, 75], as revised in 2017, provide technical and procedural guidelines for US government agencies implementing several FIS models, which are part of our analysis in Chapter 6. Pashalidis and Mitchell [155] categorize web SSO broadly into *pseudo-SSOs*, *true SSOs*, *proxy-based SSOs*, and *local SSOs*; the first two categories correspond loosely to what we call FIS and CM schemes, and the latter two correspond loosely to the **T1** and **T2** categories we define in Section 6.2.5. Our taxonomy and evaluation framework, developed as a result of an analysis of newer schemes that have been proposed and deployed since the analysis of Pashalidis and Mitchell, provides a more granular means by which modern schemes can be evaluated and compared.

Below we summarize existing research related to protocols evaluated in our analysis, leaving our protocol overviews for individual schemes to Chapter 6.

SSO Vulnerabilities

Whereas our taxonomy and evaluation framework for SSO schemes is based on high-level design properties, in practice SSO security also depends on the secure design and implementation of the underlying protocols. Many SSO vulnerabilities arise due to widespread implementation errors (an analogy can be drawn to the widespread practice of websites using password authentication that insecurely store passwords). Sun et al. [188] analyze 96 popular OAuth 2.0 based SPs using Facebook SSO, and find that the majority are vulnerable to at least one serious implementation-related vulnerability such as access token theft via a cross-site scripting (XSS) attack. Chen et al. [39] analyze over 600 mobile applications using OAuth 2.0 for SSO authentication, and find that about 60% were vulnerable to exploit due to incorrect implementation. Fett et al. [61] develop a formal analysis framework for web SSO implementations, and apply it to BrowserID (the protocol underlying Mozilla Persona) to find a number of server-side IdP implementation-related vulnerabilities; they also [62] conduct a formal analysis of OpenID Connect to discover new classes of attacks and outline implementation guidelines for corresponding defenses.

Some SP vulnerabilities can be traced back to inadequate documentation in protocol specifications or SDKs. Sun et al. [190] conducted a formal analysis of OpenID 2.0 and found that protection against session tampering required the implementation of safeguards not discussed in the specifications; an empirical study revealed that many OpenID 2.0 RPs were vulnerable to cross-site request forgery (CSRF) attacks since they did not have the safeguards in place. Wang et al. [204] formally analyzed OAuth 2.0 SDKs provided by Microsoft and Facebook (two major IdPs) to determine whether SP developers can securely build SSO into their applications by following only explicitly-stated assumptions in the SDKs; it was found that many apps built with these SDKs were vulnerable to major exploits due to violations of unstated assumptions by IdPs (e.g., the expected sequence of API calls made by an SP).

Implementation-related vulnerabilities can also impact password managers (but potentially to a lesser degree than FIS—IdP-side vulnerabilities may be more likely to be promptly fixed than SP-side vulnerabilities [212]). For example, flawed password auto-fill policies may result in password compromise [177] by, e.g., filling in the user's password on a page retrieved via a non-HTTPS connection. Web-based password managers that operate by locally injecting JavaScript into websites via a browser extension or bookmarklet can be vulnerable to various XSS or CSRF attacks [115].

Automated SP Vulnerability Scanners

Although implementation-related vulnerabilities are widespread, automated vulnerability scanners may help SP developers secure their applications by remotely scanning for common implementation errors. Zhou et al. [212] develop an automatic vulnerability checker to test for five different vulnerabilities; they scanned over 1660 sites that use Facebook SSO, finding that about 20% were susceptible to at least one vulnerability. Mainka et al. [118] analyze common implementation-based vulnerabilities, categorize them (e.g., one category is replay attacks allowing attackers to impersonate users by sending expired access tokens to SPs that fail to check the nonce or expiry parameters), and develop an automated vulnerability assessment tool that scans OpenID Connect SPs for common vulnerabilities; they also discovered protocol-level vulnerabilities in OpenID Connect.

Hardening SSO Protocols

A common threat to virtually all current web authentication schemes, including SSO, is that of session hijacking. Typically, successful user authentication to a website is followed by the provision of an HTTP session cookie allowing users to browse across different pages of the website without re-authenticating. Session cookie theft (e.g., via XSS attacks [152]) allows attackers to bypass user authentication by hijacking an existing authenticated user's session. Token Binding [160] (formerly TLS ChannelID [15] and Origin-Bound Certificates [52]) allows cryptographically binding tokens (e.g., session cookies, access tokens sent from IdPs to SPs via the user's browser) to browsers using client-side dynamically generated TLS certificates [76]. Token binding can be applied across different SSO protocols to defend against various token theft attacks (such as session cookie theft or identity assertion reuse), e.g., Dietz et al. [53] implement token binding for Mozilla Persona, and FIDO UAF [14] supports it as an optional feature (for backwards-compatibility with platforms that do not yet support token binding).

SSO Adoption

Sun et al. investigate SSO adoption barriers, both from users' [191] and SPs' [189] perspectives. Their user study reveals various reasons behind users' hesitance to adopt SSO, such as inaccurate user mental models (e.g., incorrectly believing that they were sharing their password with the RP) and concerns relating to privacy and potential phishing attacks. In contrast, a user study by Stobert [186] indicates that users are much more willing to adopt password managers built into the OS or web browser—this motivates our analysis of CM-based schemes such as Firefox Sync and FIDO UAF[7]. McCarney [125] discusses other classes of password managers, e.g., managers that use a master password to generate SP-specific passwords—such schemes have not received much interest, due to practical issues such as the need to reset all SP passwords when the master password is changed.

[7]FIDO UAF is currently built into a number of major platforms that advertise it under their own branding, e.g., "Windows Hello" on Windows 10 and "Samsung Pass" on Samsung smartphones.

Chapter 3

Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods

We explore the state-of-the-art of device fingerprinting, albeit with a special focus: to augment web authentication mechanisms, and especially password-based methods. Despite usability and security drawbacks, the latter remains the dominant form of authentication on the web—in part because more secure alternatives have their own usability and deployability issues [25].

Large web services can mitigate insecure passwords in part by using server-side intelligence in the authentication process with "multidimensional" authentication [26], wherein conventional passwords are augmented by multiple implicit signals collected from the user's device, ideally without requiring explicit new user actions. We explore the applicability of various fingerprinting techniques to strengthening web authentication and evaluate them based on the security benefits they provide. We aim to identify techniques that can improve security, while being ideally invisible to users and compatible with existing web browsers, thereby imposing low cost on usability and deployability.

We identify and classify 29 available device fingerprinting mechanisms, primarily browser-based and known, but including several network-based methods and others not in the literature; identify and assess their properties suitable for application to augment user authentication; define a series of adversarial models within the context of fingerprint-augmented authentication; and consider practical issues in the implementation, deployment, and use of device fingerprinting within the context of user authentication.

While prior work has mostly presented specific device fingerprinting mechanisms or their use by web trackers, we analyze a broad range of such mechanisms within the context of augmenting password-based web authentication.

3.1 Framework and Threat Model

Base definitions, replay and spoofing. In device fingerprinting, servers use one or more mechanisms (*fingerprinting vectors* or *vectors*) to extract and verify properties related to the software and/or hardware configuration of a device. "Pure" device fingerprinting is *stateless*, depositing no new client-side information; alternatives are *stateful*. By *device fingerprint* we mean the overall set of vectors a server employs, or depending on context, the output corresponding to such vectors. A base assumption is that attackers eventually learn the set of device fingerprinting vectors employed by websites. A server may use a collection of vectors and choose some (random or other) subset in particular instances; if so, we assume the attacker does not know the subset beforehand.

Many vectors require browsers to perform certain operations, e.g., via JavaScript, and return the output to the server. If a vector response is *static*, i.e., constant regardless of circumstances, an attacker observing or intercepting this can later *replay* it. If all vectors composing a device fingerprint are static, so is the device fingerprint (i.e., of a particular device), and simple replay suffices. By *spoofing* we mean an attacker trying to mimic the fingerprint of a target device; this is a form of *mimicry attack* (Chapters 4 and 5 discuss mimicry attacks and mimicry resistance in further detail). If a server varies the vectors, or uses individual vectors for which the response is dependent on conditions or a variable challenge of some form, then spoofing requires a more complex attack than simple capture and replay—e.g., successfully spoofing some forms of geolocation (as discussed later) requires attacker access to a proxy machine located near the victim (which we call a co-location attack). In the models below and elsewhere, we typically use the more general term spoofing, but often this requires only replay.

To facilitate analysis of device-fingerprinting-augmented authentication, we define a continuum of five attack models, in order of increasing attacker power and skill. Table 3.1 summarizes and compares attacker capabilities across these models. This modelling also serves to motivate and focus later discussion of desirable defensive properties of fingerprinting vectors (see Section 3.3 and Table 3.2).

Model M1: Naive attack. We consider a naive attack to be a conventional

Attack models	Optimized password-guessing	Optimized device fingerprint guessing	Obtain partial or full target device details	Phish victim password and device fingerprint	Spoof device fingerprint	Steal session cookie
M1: Naive attack	●					
M2: Optimized password & fingerprint guessing	●	●			●	
M3: Targeted password & fingerprint guessing	●	●	●		●	
M4: Fingerprint phishing & spoofing			●	●	●	
M5: Session hijack with fingerprint spoofing			●		●	●

Table 3.1: Attacker models (rows) and capabilities (columns).

online password-guessing attack that does not attempt to thwart any secondary authentication mechanisms. Such attacks will simply fail, since they involve guessing an account password while making no effort to spoof the device fingerprint of the account owner. The verifying server can thus determine, based on previously-collected device fingerprints, that the authentication attempt comes from a device that the account owner has never used. Advanced attackers will use the more sophisticated models below.

Model M2: Optimized password and fingerprint guessing attack. This attack follows an optimal guessing strategy, i.e., guessing passwords in order of popularity across many accounts [24] while iterating over device fingerprint "guesses" expected to maximize the probability of account break-in. The attacker may spoof or mimic device fingerprints that match a set of popular devices, potentially tailored to the target website audience; e.g., websites targeting users in particular countries are likely to have visitors within certain time zones, or vendor-specific technical support sites will likely have visitors using that vendor's devices. The attacker can then, based on the popularity of each password and device fingerprint combination, mount an attack by iterating through the list of password-fingerprint pairs in order of decreasing popularity across all accounts. In special cases, the attacker may reduce the search space by obtaining a list of passwords (e.g., from a leaked password database of another website, if users can be mapped between the two websites through information

such as e-mail addresses) or a list of device fingerprints (e.g., by exploiting a XSS vulnerability to obtain the necessary data from a large subset of users to spoof their device fingerprints).

Model M3: Targeted password and fingerprint guessing attack. The attacker aims to break into a specific account, and mounts a password guessing attack while also attempting to spoof a fingerprint to match a target device. This attack iterates over passwords and fingerprints in some optimized order, as in M2. The attacker may significantly reduce the search space by using known information (e.g., if the user is known to use a particular type of device) or in special cases may possess (e.g., due to *a priori* capture) the user's password or a fingerprint of the user's device, thereby reducing the attack to either fingerprint guessing or password guessing.

Model M4: Fingerprint phishing and spoofing. Whereas classic phishing attacks steal passwords, this related attack targets users of websites employing device-fingerprinting-augmented authentication. This attack involves luring users (e.g., by e-mailing users a phishing link) to an impostor website that captures both the victims' passwords and device fingerprints. The attacker then gains access to victims' accounts by spoofing the device fingerprints and replaying the captured passwords. This attack is harder if the device fingerprint is not static.

Model M5: Session hijack with fingerprint spoofing. The attacker aims to hijack an existing authenticated session, vs. attackers M1-M4 which attempt account break-in via password and fingerprint guessing or stealing. The M5 attacker is granted (i.e., assumed to have) the power to steal session cookies and execute client-side JavaScript by exploiting, e.g., improperly-configured HTTPS [178, 106] or XSS vulnerabilities. This enables the attacker to (1) capture device fingerprints sent from the browser to the server, and (2) perform any additional fingerprinting that would facilitate spoofing the user's device and thereby resuming the session from the attacker's device. A device fingerprint useful under M5 must thus be hard to spoof (beyond simply replaying a static string). Server-side fingerprint checking has been proposed as a defence against attacker reuse of stolen session cookies [194, 162].[1]

[1]The motivation behind this use case is similar to that of Channel ID (originally known as Origin-Bound Certificates [52]), which uses self-signed, domain-specific SSL client certificates dynamically generated by the client. Session cookies are cryptographically bound to the SSL session, thereby requiring an attacker to possess both the client's cookie and domain-specific private key to hijack a session.

We now make some simple observations on these models:

- Attacks under M1: Device fingerprinting completely stops naive attacks, even if fingerprints are replayable.
- Attacks under M2: Device fingerprinting significantly reduces the success probability of M2 attacks, even with replayable fingerprints, if the guessing space of the fingerprints is sufficiently large relative to the number of guesses the attacker is capable or allowed to submit.
- Attacks under M3: These attacks are more difficult to defend against, since the attacker targets a specific user and is assumed to have device-specific information. These attacks are also more difficult to carry out and are less scalable (i.e., harder to mount on a massive scale).
- Attacks under M4: These are the most difficult to prevent among M1-M4, since the M4 attacker lures the user by phishing to capture a password and device fingerprint. If the fingerprint is static, simple replay suffices for spoofing.
- M5 is specific to session hijacking, with fingerprinting used not simply to augment authentication at the start of a session, but throughout one; this is pursued in Section 3.4.2.

The above models and discussion help delineate properties important in our later analysis and classification.

3.2 Device Fingerprinting Vectors: Classification

We now identify, summarize, and classify fingerprinting vectors based on a review of research literature, informal sources (e.g., online sources, open-source fingerprinting libraries, online advertising and anti-fraud services), and other fairly obvious vectors including some not formally documented elsewhere to our knowledge. In most cases, we classify these vectors based on the method used to obtain the fingerprintable information from the client. Alternative classifications are possible, such as based on the type of information collected: hardware features, network information, user-defined preferences, and software installed (e.g., OS, applications, device drivers, shared libraries).

Category 1: Browser-provided information. Web browsers explicitly provide (e.g., via JavaScript) a wide range of system information that is of use to constructing a device fingerprint. Known vectors are listed below.

(a) **Major software and hardware details.** The navigator and window Browser Object Model (BOM) objects expose attributes such as browser/OS vendor and version, system language, platform, user-agent string (which includes the prior three and sometimes others, e.g., device model number), installed plugins, supported browser storage mechanisms (e.g., localStorage, indexedDB, sessionStorage, WebSQL via openDatabase), screen resolution, colour depth, and pixel density.

(b) **WebGL information.** WebGL [127], a JavaScript API for rendering graphics within web browsers, exposes various attributes (e.g., GL version, max texture size or render buffer size, supported WebGL extensions, vendor/renderer strings) of the underlying browser and hardware.

(c) **System time and clock drift.** The device's system time can be accessed via JavaScript and used to infer the device's time zone, observance of daylight saving time, and clock drift from Coordinated Universal Time (UTC) [57].

(d) **Battery information.** The HTML5 battery status API is suitable for fingerprinting when sufficiently precise readouts are provided [151]. The battery's charge level can be used for short-term tracking of clients across different websites; its capacity (which slowly degrades with battery age, but changes very little over relatively short periods such as days) can be estimated by monitoring the discharge rate for about 30 seconds, and used to aid identification.

(e) **Evercookies.**[2] An *evercookie* is a mechanism whereby a client identifier (and/or other tracking information commonly stored in a browser cookie) is stored on the device using a variety of techniques such as HTML5 local storage, HTTP ETags, or Flash cookies, thereby allowing websites to reconstruct user-deleted cookies [4].

(f) **WebRTC.** WebRTC is a set of W3C standards that support native (plugin-free) browser-to-browser applications such as voice and video chat [18]. Devices can be fingerprinted by enumerating supported WebRTC features and potentially connected

[2]Evercookies violate the stateless property of our "pure" device fingerprinting definition, but are nonetheless considered a form of fingerprinting by, e.g., the W3C: https://w3c.github.io/fingerprinting-guidance

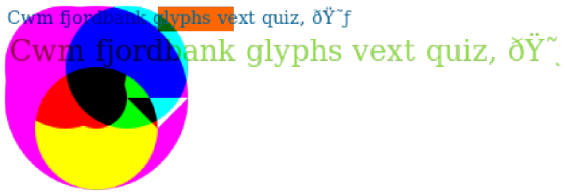

Figure 3.1: Illustration of text and graphics rendered in an HTML5 canvas for fingerprinting a device.

media devices such as microphones and webcams; browsers differ as to what types of devices can be enumerated without user permission. WebRTC also exposes the IP addresses assigned to all network interfaces present on the device [167], including private IP addresses assigned by a NAT router or VPN.

(g) **Password autofill.** JavaScript can be used to detect if a password has been user-typed, or auto-filled by a browser or password manager. Our tests show that this is easily achieved using event listeners[3] to detect whether the user has typed any characters into the password field, e.g., by assigning an event listener for the keydown event (triggered when a key is pressed down) and keypress event (triggered after a key has been pressed and subsequently released). Since these events are triggered by physical key presses, their absence indicates password entry via autofill (or other automated means, e.g., an automated password-guessing attack).

Category 2: Inference based on device behaviour. Information about the device can be gleaned not only by querying the browser (as in Category 1), but also by executing specially-crafted JavaScript code on the browser and observing the effect, e.g., measuring execution time or analyzing generated output. The following fall in this category.

(a) **HTML5 canvas fingerprinting.** JavaScript can be used client-side to render a variety of text and graphics in an HTML5 canvas (e.g., as illustrated in Figure 3.1), and send the server a hash of the resulting bitmap image [132]. Subtly different images are generated by devices with different software/hardware configurations, e.g., fonts and font rendering (e.g., anti-aliasing) vary with OS and video driver, and emojis

[3]JavaScript event listeners wait for some event to be generated (typically as a result of user action, e.g., by clicking on a button or typing a key) and execute a function to process the event.

vary with OS and smartphone vendor [113]. Rendering text using a list of predefined fonts allows font detection [59]. Rendering complex graphics using WebGL provides further fingerprint diversity by a wide range of rendered hardware-dependent graphics output.

(b) **System performance.** Client-side JavaScript can be used to run JavaScript engine benchmarks over a range of computationally-intensive operations. The elapsed times are measured to infer device performance characteristics [131].

(c) **Hardware sensors.** Mobile device sensors can be fingerprinted based on variations in manufacturing and factory calibration, e.g., measuring calibration error of smartphone accelerometers (accessible via JavaScript) or the frequency response of speakerphone-microphone systems [23, 47].

(d) **Scroll wheel fingerprinting.** Various aspects of user pointing devices can be inferred via JavaScript by listening for the WheelEvent event, triggered whenever the user scrolls using a mouse wheel or touchpad [148]. For example, a mouse wheel scrolls the page by fixed increments when triggered; a touchpad can scroll by varying increments. Measuring the rate at which a document is scrolled reveals information both about user scrolling behaviour and the value of the OS user-configurable scrolling speed.

(e) **CSS feature detection.** The browser vendor and version can be inferred by testing CSS features that are not uniformly supported across browsers [194], e.g., by setting the desired CSS property on a target element and subsequently querying the element to determine if the change was applied. This vector, and vectors 2(f) and 2(g) below, yields a subset of the information obtained from the user-agent string via 1(a) or 4(e). If a device fingerprint already extracts user-agent string information by another vector, then 2(e) here can also be used to test if that information has been manipulated.

(f) **JavaScript standards conformance.** Browsers differ in conformance to the JavaScript standard, allowing fingerprinting based on behaviour in corner cases. Various JavaScript conformance tests run thousands of test cases that can collectively take over 30 minutes. Mulazzani et al. [138] developed a technique that leverages the fact that modern browsers fail very few of these tests. This technique can be used

to verify the browser vendor and version reported in the user-agent string by using a decision tree to select a very small subset of tests that run in negligible time.

(g) **URL scheme handlers.** Some browsers implement non-standard schemes to access local resources; e.g., res:// in Microsoft IE exposes images stored in DLL files in the Windows system directory, and moz-icon://, jar:resource:// and resource:// in Mozilla Firefox expose various built-in browser and OS resources. Although recent browser versions disallow websites from accessing local files via file:// (to prevent leakage of user private data), this restriction does not extend to the aforementioned schemes. Websites can thus create HTML image tags with the source address set to a local resource and use the onerror event handler to detect if the image was loaded. By iterating through a list of resources preloaded with various browser or OS versions, this vector can enumerate the non-standard schemes supported and the resources successfully loaded [92]. This offers an alternative to 2(e), 2(f), and 1(a) to infer browser vendor and version.

(h) **Video RAM detection.** The amount of Video RAM (VRAM) available to the GPU, while not explicitly available via the WebGL API, can be inferred by repeatedly allocating textures until VRAM is full, after which textures start being swapped into system main memory. By observing the elapsed time for each texture allocation and recording the step at which a large spike in elapsed time is observed,[4] it can be inferred that GPU VRAM is at full utilization. After this point, the browser can continue allocating textures until an OUT_OF_MEMORY[5] error is returned.

(i) **Font detection.** While installed fonts cannot be enumerated via JavaScript, text can be formatted with fonts from a predefined list; the dimensions of the resulting text can distinguish different font rendering settings and allow the presence of each font (and even versions of the same font) to be inferred [146, 63].

(j) **Audio processing.** The HTML5 AudioContext API allows the creation of audio visualizations by providing an interface for real-time frequency- and time-domain

[4]For example, on a system we tested, about 1ms elapsed while allocating each 16MB texture. This spiked to 10-40ms after the GPU VRAM reached full capacity.

[5]This error does not necessarily indicate all system memory is depleted, but that WebGL can no longer allocate memory. It is undesirable to continue allocating until this error triggers, as the user may be presented an error message (in our testing, desktop Google Chrome does not produce an error; Chrome for Android does).

analysis of audio playback. As with 2(a) HTML5 canvas fingerprinting, audio processing varies by browser and software/hardware configuration [59].

Category 3: Browser extensions and plugins. Browser plugins and extensions can be leveraged for fingerprinting in a manner similar to Categories 1, 2, and 4 (i.e., by directly querying for information, by inference, and/or by protocol-level analysis). The following are such methods.

(a) **Browser plugin fingerprinting.** Browser plugins such as Java, Flash, and Silverlight can be queried (via plugin objects embedded in a webpage) to reveal system information, often in more detail than available via JavaScript. For example, Flash provides the full OS kernel version; both Flash and Java plugins allow all system fonts to be enumerated; even the order in which system fonts are enumerated can vary across systems, increasing the distinguishability of fingerprints [146, 56].

(b) **Browser extension fingerprinting.** If a JavaScript-blocking extension[6] is installed (disabling JavaScript by default for all websites, except those whitelisted by the user), a website can attempt to load scripts from a large set of websites (e.g., Alexa Top 1000) to detect which are on the user whitelist [131]. Similarly, ad blockers can be detected by embedding a "fake" ad such as a hidden image or iframe with a source URL containing words commonly blacklisted by ad blockers (e.g., "ads"); JavaScript can then detect if the fake ad has been loaded, and return the result to the server. Other extensions may be fingerprinted by other methods, e.g., some browser extensions add custom HTTP headers [113].

(c) **System-fingerprinting plugins.** Websites may install specialized plugins (e.g., by bundling them with other software available for download) to provide more powerful fingerprinting information such as hardware identifiers, OS installation date, and installed driver versions [146]. Such plugins are generally considered *spyware*.

Category 4: Network- and protocol-level techniques. The prior categories involve accessing application-level APIs on client devices. Network- and protocol-level techniques can also be used to fingerprint devices as follows.

[6]NoScript is a popular JavaScript-blocking extension for Firefox; similar extensions are available for Chrome.

(a) **IP address.** A client IP address can be used as an identifier, or to query regional Internet registries (via the WHOIS protocol [45]) to obtain further information such as the Autonomous System (AS) it resides in and organization name it is registered to. While IP address is a more precise identifier than AS number, the latter is a more stable ID for hosts with dynamic IP addresses and is useful as a potential cross-check when verifying client location (see further comment regarding this in Section 3.3.6).

(b) **Geolocation.** The client's geographical location may be determined via several mechanisms. Modern web browsers typically expose APIs (e.g., via the navigator BOM object, cf. vector 1[a]) by which a website may request user permission to obtain current location (via, e.g., onboard GPS hardware, cellular triangulation, WiFi access point information, or user-supplied information). Network-based mechanisms [137] include WHOIS lookups based on IP address, inference based on routing data, and geolocation involving delay-based measurements [2].

(c) **Active TCP/IP stack fingerprinting.** Differences across network links and OS TCP/IP implementations allow devices to be fingerprinted by sending them carefully-crafted TCP/IP probes and analyzing response packet header fields (e.g., RTT, TCP initial window size) or link characteristics (e.g., MTU, round-trip delay). This method is browser-independent, and can be used on any Internet-accessible host. Nmap [116] is a popular scanning tool that includes host discovery, port scanning, and an OS detection feature that sends various probe packets and applies heuristics from its built-in database to differentiate thousands of systems. As this sends special probe packets to the client, i.e., is *active fingerprinting*, it can trigger firewall and IDS alerts due to its common reconnaissance use by attackers.

(d) **Passive TCP/IP stack fingerprinting.** A less intrusive (less powerful) technique, *passive fingerprinting*, sniffs existing network communication but uses heuristics similar to active fingerprinting to identify hosts; p0f [210] is an example of such a tool. Passive approaches are more suitable fingerprinting vectors, as header analysis of existing web traffic requires no new packets that may be flagged as intrusive.

(e) **Protocol fingerprinting.** Protocol-level fingerprinting can be applied to higher-level protocols to differentiate versions or configurations of browser software or libraries. For example, the server may record HTTP header fields sent by clients,

e.g., user-agent string, list of acceptable languages and character encodings, and the user-configurable DoNotTrack parameter. Moreover, the browser's TLS library can be fingerprinted using its ClientHello packet from the handshake sequence that negotiates protocol parameters [31]; information of relevance includes client TLS version, supported ciphersuites (and their order of presentation), compression options, and list of extensions (and associated parameters such as elliptic curve parameters).

(f) **DNS resolver.** A web server may determine which DNS resolver a client is using—for many clients, the default DNS resolver is configured by the user's ISP (which typically should not respond to queries originating from outside the ISP's network), but a minority of users may switch to other DNS resolvers, e.g., run by Google or OpenDNS. One approach involves a server sending the client browser a document containing a reference to a randomly-generated subdomain under a domain for which the authoritative DNS server is under control of the website owner [184]. When the client attempts to resolve the subdomain, the website's DNS server receives the request from the client's DNS resolver and can associate the randomly-generated subdomain with the client for which it was originally generated.

(g) **Clock skew.** TCP timestamps can be passively analyzed to measure client clock skew—a measure (e.g., in microseconds per second) of the rate at which the client's clock deviates from the true time [104].

(h) **Counting hosts behind NAT.** For clients behind a NAT, the number of hosts behind the NAT contributes to a device fingerprint. Bellovin [17] first proposed counting hosts behind a NAT by passively analyzing the IPv4 ID field (used for fragment reassembly); Kohno et al. [104] proposed the use of clock skew measurements to differentiate hosts behind a NAT. These techniques may be augmented with upper layer information, e.g., by including in the fingerprint the IDs of other users who access accounts from the same IP address.

(i) **Ad blocker detection.** While ad blocker detection can be done client-side with JavaScript as in vector 3(c), it can also be done server-side by monitoring incoming HTTP requests to detect if the client has requested the fake ad.

3.3 Device Fingerprinting Vectors: Desirable Properties

Augmenting authentication with device fingerprinting requires considering various characteristics, which we now explore through a comparative analysis. Some characteristics may influence device detection accuracy or effectiveness: for behavioral advertising applications that use device fingerprinting to track user browsing habits (not our focus), it may be sufficient to identify a unique individual device correctly 80% or 90% of the time; such accuracy is rarely sufficient in authentication contexts (our focus). We discuss these properties below, with a comparative summary in Table 3.2.

To maximize accuracy, device fingerprints employ multiple fingerprinting vectors. All vectors in Section 3.2 can be freely combined, limited by possible impact on device/user experience (only a few vectors, as noted below, have non-negligible resource costs), and server cost for fingerprint verification. Vectors from Categories 1 to 3 involve client-side JavaScript (and thus incur non-zero, but typically small, client overhead). To help compare vectors, and provide information of use in combining them, estimates of the distinguishability provided by each vector are included below and in Table 3.2.

3.3.1 Stability

Virtually all components of a device fingerprint are subject to change, but some (e.g., time zone for desktop machines or for mobile devices of users who rarely travel) may change much less frequently than others (e.g., browser version). A significant change in device fingerprint may require the verifying server (if device fingerprinting is used to augment password authentication) to temporarily fall back to a less convenient but more reliable user authentication mechanism, such as a one-time passcode sent over SMS or e-mail. Thus it is preferable from a usability perspective to employ vectors that provide fingerprints that are stable over time, i.e., with a sufficient number of component vectors stable at a given time. Overall fingerprint stability can be improved by combining multiple vectors and implementing a scoring mechanism that allows a subset of vectors to change. In the empirical study by Eckersley [56], an algorithm correctly linked a device's old fingerprint with the updated fingerprint in

		Citation	Stability	Repeatability	Low Resource Use	Spoofing Resistance	Client Passiveness	Distinguishing Info.	Overlaps With
Category 1: Browser-provided information	1a) Major software and hardware details	[56]	◑	●	●	○	○	○○○●○	1d; 2e,f,g; 3a,c; 4b,e
	1b) WebGL information	[199]	●	●	●	○	○	○●○○○†	2a
	1c) System time and clock drift	[57]	●	●	●	○	○	○●○○○†	
	1d) Battery information	[151]	◑	●	◑	○	○	○●○○○†	1a
	1e) Evercookies	[4]	●	●	●	●	○	○○○○●	
	1f) WebRTC	[32]	●	●	●	○	○	○○●○○†	
	1g) Password autofill	*	●	●	●	○	○	●○○○○	
Category 2: Inference based on device behaviour	2a) HTML5 canvas fingerprinting	[132]	◑	●	●	○	○	○○●○○	1b
	2b) System performance	[131]	●	◑	○	○	○	○●○○○†	
	2c) Hardware sensors	[23, 47]	●	◑	◑	◑	○	○○●○○	
	2d) Scroll wheel fingerprinting	[148]	●	◑	●	○	○	●○○○○†	
	2e) CSS feature detection	[194]	◑	●	●	○	○	●○○○○	1a
	2f) JavaScript standards conformance	[138]	◑	●	●	○	○	●○○○○	1a
	2g) URL scheme handlers	[92]	●	●	●	○	○	●○○○○	1a
	2h) Video RAM detection	*	●	◑	○	○	○	●●○○○†	
	2i) Font detection	[146]	●	●	●	○	○	○●●○○	
	2j) Audio processing	[59]	●	●	●	○	○	○○●○○†	
Category 3: Extensions and plugins	3a) Browser plugin fingerprinting	[56, 113]	●	●	●	○	○	○●●○○	1a
	3b) Browser extension fingerprinting	[131]	●	●	●	○	○	●●○○○	4i
	3c) System-fingerprinting plugins	[146]	●	●	●	◑	○	○○○●○	1a
Category 4: Network- and protocol-level techniques	4a) IP address	–	◑	●	●	●	●	○○○●○	
	4b) Geolocation	[137]	◑	●	●	◑	◑	○●●●○	1a
	4c) Active TCP/IP stack fingerprinting	–	◑	●	●	◑	◑	○●●○○†	4d
	4d) Passive TCP/IP stack fingerprinting	–	◑	●	●	◑	●	○●○○○	4c
	4e) Protocol fingerprinting	[31]	◑	●	●	○	●	○○○●○†	1a
	4f) DNS resolver	[32]	◑	◑	●	◑	◑	○●●○○	
	4g) Clock skew	[104]	●	◑	●	●	●	○○●○○	
	4h) Counting hosts behind NAT	[17]	○	◑	●	◑	●	○●○○○	
	4i) Ad blocker detection	–	●	●	●	○	●	●○○○○	3b

Table 3.2: Classification and comparative summary of device fingerprinting vectors. Citation sources provide discussion or original proposal; asterisk denotes unawareness of previous literature proposing the vector; dash indicates vector commonly known (but no specific research paper, to our knowledge, that gives focus to it). Filled circle indicates that vector (row) provides benefit (column); half or empty circle indicates that benefit is partially provided or not provided, respectively. Distinguishing Info. uses a 5-dot scale, from left to right: VeryLow, Low, Medium, High, and VeryHigh, with coarse mapping to bit-ranges 0-2, 3-6, 7-14, 15-30, and ≥31; † denotes Requires Further Study (RFS). The final column notes vectors that overlap in the information that they provide.

65% of devices with 99.1% accuracy. If such an algorithm were used in the context of authentication, a high-security application might minimize false accepts at the expense of a higher false reject rate. This trade-off may be different in a context where security is less important than user convenience.

The majority of individual vectors from Categories 1 to 3 are relatively stable, with some exceptions:

- 1(a) Major software and hardware details may occasionally change if the user upgrades their OS, switches browsers, installs or removes browser plugins, etc.
- 1(d) Battery information will change, e.g., as battery capacity degrades with age.
- 2(a) HTML5 canvas rendering details may change with browser, OS, or graphics driver updates.
- 2(e) CSS feature detection and 2(f) JavaScript standards conformance may change with browser updates.

In contrast, vectors from Category 4 are generally less stable:

- 4(a) IP address, 4(b) Geolocation (depending on granularity), and 4(f) DNS resolver (unless manually configured) change as users log in from different locations. These vectors are more stable for desktop than mobile devices.
- 4(c, d) TCP/IP stack fingerprinting varies with routing changes that affect, e.g., round-trip delay and number of hops between client and server. If a user logs in from a different network location, additional changes would be observable in, e.g., the MTU of the network link, or responses to probe packets due to different firewall rules.
- 4(e) Protocol fingerprinting will vary with updates to the browser or shared (e.g., SSL/TLS) libraries.
- 4(h) Counting hosts behind a NAT is unstable over time, as devices may enter and leave a network.

Exceptions are 4(i) Ad blocker detection and 4(g) Clock skew (TCP timestamps reflect CPU clock skew, which is relatively stable [104]). Even vectors unstable over longer periods may remain useful for applications requiring identification across short periods, e.g., throughout one day, hour, or session.

3.3.2 Repeatability

We define *repeatability* as the property whereby a vector generates the same result if the software, hardware, and network configurations of a device are unchanged (whereas *stability* primarily concerns changes in device configuration). Repeatability is challenging for vectors measuring device performance, e.g., CPU, GPU, or network throughput, since performance varies if clients simultaneously perform other tasks. Most vectors discussed are repeatable, with these exceptions:

- 2(b) System performance may vary based on whether the device is burdened with other tasks. For mobile devices, it can also depend on temperature, as mobile chipsets scale back clockspeeds at high temperature.
- 2(c), 4(g) Temperature can affect hardware sensor data [47] and clock skew [139].
- 2(d) Scroll wheel fingerprinting requires that the user uses their scroll wheel. If the user does not do so on every visit to a web page, the vector is not repeatable.
- 2(h) Available VRAM will vary based on how much is currently in use by the device.
- 4(f) The DNS resolver used may vary due to load balancing of DNS resolvers.
- 4(h) Vectors counting hosts behind a NAT may vary depending on the presence of other devices on the network.

Fingerprinting vectors that are not reliably repeatable (i.e., not 100% repeatable, but repeat often enough) may still be useful, if many vectors are used in the overall device fingerprint and an appropriate scoring mechanism is used as per Section 3.3.1, where some vectors are allowed to change.

3.3.3 Resource Use and Latency (Overhead)

Fingerprinting vectors that require more system resources (e.g., CPU cycles, system memory, or I/O) incur performance costs (and reduce battery life, for mobile devices). This is of less concern for websites that fingerprint devices only once per

authentication process, and more for websites aiming to detect session hijacking by fingerprinting repeatedly throughout a session.

Most Table 3.2 vectors have low client-side overhead, requiring processing time in the milliseconds range, but a few may require time on the order of seconds: 1(d) Battery information, 2(b) System performance, 2(c) Hardware sensors, and 2(h) Video RAM detection. While 2(b) consumes CPU cycles and 2(h) GPU memory, 1(d) and 2(c) only require time to collect sufficient data (i.e., they introduce latency, but do not consume system resources) and thus rate half circles.

3.3.4 Spoofing Resistance

As per Section 3.1, attackers will try to spoof a device fingerprint that resembles the target. For vectors from Categories 1 to 3, the browser runs client-side JavaScript and returns output to the server; this makes it easy for attackers to spoof a (guessed or intercepted) response,[7] with three exceptions:

- 1(e) Evercookies are stateful (violating our pure fingerprinting definition), and can store global identifiers using, e.g., Flash cookies and HTML5 local storage, which are protected by Same-Origin Policy (SOP [128]). Rather than *spoofing*, evercookies may be stolen and replayed by M5 attackers; but since this requires exploiting a vulnerability, we grant a filled circle in Table 3.2.
- 2(c) While hardware sensors can be spoofed by obtaining a copy of the sensor data from the client and replaying it, obtaining such data in the first place may require user cooperation. For example, the web browser may ask for the user's permission before accessing the microphone.
- 3(c) Information available only through specialized plugins, e.g., hardware identifiers, may require more effort for attackers to obtain if the plugin is designed to communicate only with the website by which it was installed.

In comparison, Category 4 vectors are more resistant to spoofing (but none are completely immune). The following vectors are rated spoofing-resistant in Table 3.2:

[7] For example, credit card fraudsters use the FraudFox toolkit [65, 102], using a heavily modified Mozilla Firefox and Adobe Flash, to spoof various fingerprintable attributes such as OS version, screen resolution, and list of fonts. It includes a feature for capturing device fingerprints of phished users.

- 4(a) IP source address spoofing is possible due to insufficient deployment of source address validation across the Internet [21], and is often used to mount Denial-of-Service (DoS) attacks. However, using a spoofed source address to establish two-way communication with a host is difficult, as the host will always send response packets to the spoofed address.

- 4(g) Clock skew fingerprinting has been proposed for identifying wireless sensors in mesh networks [88] and access points in 802.11 wireless LANs [95]; both techniques rely on timestamps used in the respective MAC protocols. Arackaparambil et al. [10] showed that clock skew can be spoofed, but doing so introduces irregularities detectable through analysis, and that using a smaller 801.11 beacon frame transmission interval (typically 100ms) causes the irregularities to be more pronounced and thus more easily detectable. Clock skew measurements derived from TCP timestamps would be more coarse-grained compared to MAC-protocol-level timestamps, but may nonetheless be difficult to spoof undetectably.

With the exception of 4(i) Ad blocker detection, and 4(e) Protocol fingerprinting, which can be spoofed fairly easily by using the same browser and library versions as the target device, the remaining Category 4 vectors are graded as partially resistant to spoofing (half-circle) in Table 3.2:

- 4(b) Techniques like delay-based location verification [2] require attacker access to a proxy situated close to the victim to spoof the victim's location. Geolocation relying on information reported by a browser location API is more easily spoofed (but falls in Category 1).

- 4(c,d) Spoofing an OS TCP/IP stack can be done by manipulating TCP/IP behaviour with various off-the-shelf tools [19]. Network- or link-dependent measurements such as number of hops between client and server, round-trip delay, and MTU appear to require comparatively more effort to spoof.

- 4(f) If the client uses its ISP's DNS resolver (which typically responds only to DNS requests originating within the ISP's network), a successful attack requires access to a machine within that ISP's network.

- 4(h) The difficulty of spoofing the number of hosts behind a NAT varies based on the technique used; a simpler technique based on the IPv4 ID field [17] would be easier to spoof, whereas a clock skew based technique [104] would be more difficult (as discussed previously).

Resistance to spoofing may be increased by certain strategies. For example, with 2(a) HTML5 canvas fingerprinting, when a device is first associated with an account, the server may send multiple challenges (i.e., different sets of text and graphics to render), with the client returning corresponding results. The server may then randomly select a subset of stored challenges on each authentication attempt. Improving resistance of a vector to simple replay improves spoofing resistance, but a highly resourceful attacker might be able to configure attack software to mimic a target device for any given challenge, including for 2(a) as just noted. Different strategies may enhance the spoofing resistance of other vectors—e.g., for 2(b) System performance, a client puzzle [12] approach might be used, whereby the server sends cryptographic puzzles to the client and measures the time taken to receive a correct result. While more powerful devices might still spoof slower devices, such techniques raise the bar for attackers.

3.3.5 Client Passiveness

We classify a vector as *client-passive* if it can be used by the server without explicit cooperation or knowledge of the device. All vectors from Categories 1 to 3 require browsers to execute client-side JavaScript and return output to the server; these are not client-passive. All Category 4 vectors are client-passive to some degree, as none require explicit client cooperation. We elaborate in further comments.

- 4(b) Geolocation can be client-passive depending on mechanism, e.g., IP-based geolocation with table look-up is client-passive, but not vectors that require client-side JavaScript.
- 4(c) Active TCP/IP stack fingerprinting requires sending extra probe packets to the client to observe the response (or lack thereof). While sending probe packets does not require client cooperation, such packets may be detectable by client firewalls; probes crafted to appear part of regular HTTP traffic may

be considered client-passive. We thus grade this vector partially client-passive (half-circle in Table 3.2).

- 4(f) DNS resolver fingerprinting does not require client-side JavaScript, but on carefully inspecting webpage source code a client might suspect that the random-appearing domain name string was injected to learn the client's DNS resolver. Thus, we grade this vector partially client-passive (half-circle in Table 3.2).

All other vectors in Category 4 are graded client-passive; a device fingerprint can be constructed with these solely by inspecting existing HTTP traffic between server and client.

3.3.6 Distinguishing Information

To describe the granularity at which device fingerprinting (whether individual vectors, or a combination of vectors) can identify a device, we use the term *distinguishing information*. Distinguishability depends on the size of the (user) device space in question and the diversity of the underlying properties fingerprinted. Prior work (e.g., [56, 37, 59, 113]) has used Shannon entropy [174] to quantify the distinguishing information conveyed by device fingerprinting vectors. Shannon entropy is a measure of the average information conveyed by an event generated by a stochastic source modelled by a random variable. It is calculated using the formula

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_b P(x_i), \tag{3.1}$$

where $H(X)$ is the entropy (typically measured in bits) of the random variable X, $P(x_i)$ is the probability of an event x_i occurring, n is the number of possible events that may be generated by X, and b is the base of the logarithm (the base is 2 when bits are the unit of measure). For any fixed n, $H(X)$ is maximized when every x_i is equiprobable (i.e., a uniform probability distribution). In the case of device fingerprinting, X is a random variable where each event x_i represents a distinct device fingerprint, $P(x_i)$ is the probability that a user visiting a website (in this case, of the verifying server) will have the device fingerprint x_i, and n is the number of distinct device fingerprints (note that multiple devices may share the same device fingerprint

x_i). Using Shannon entropy to estimate the distinguishing information conveyed by device fingerprinting vectors poses two major challenges, discussed below: (1) the relationship between sample size and the resulting probability distribution, and (2) device fingerprint distributions encountered by different websites may differ from each other and from the fingerprint distribution of web-enabled devices as a whole.

Sample Size and Probability Distribution: Analyzing Individual Vectors

Shannon entropy is determined by an established probability distribution, which we are not in position to describe for device fingerprinting. Prior studies have collected device fingerprints from tens or hundreds of thousands of devices (e.g., [56, 37, 113]) from volunteer users, and used the collected fingerprints to calculate sample probabilities. For example, 100 devices sharing the same fingerprint x_1, out of a sample size of 10,000 devices, would result in the probability $P(x_1) = 0.01$. The aforementioned studies used these sample probabilities to calculate the entropy of the resulting sample distribution, using Equation 3.1. However, the device fingerprints collected by these studies have not been demonstrated to reflect the statistical distribution of device fingerprints among the overall population of web-enabled devices—for example, 10% of the AmIUnique [113] dataset comes from desktop Linux-based systems, which is about fives times higher[8] than the Linux desktop marketshare, thus indicating a more technologically-savvy pool of volunteers.

Collecting device fingerprints from a larger pool of volunteers (e.g., tens of millions, rather than tens of thousands) could alter the calculated probability distribution in the following ways:

1. The larger pool may result in the same calculated probabilities x_i as the smaller (original) pool. In this case, the entropy calculated using Equation 3.1 would be unchanged. This outcome would be most probable if the smaller sample of device fingerprints was already sufficiently large to be representative of the distribution of device fingerprints among the entire population of web-enabled devices.

[8] As of the time of writing, based on data from Net Marketshare: https://www.netmarketshare.com/.

2. The larger pool may result in different calculated probabilities x_i when compared to the smaller (original) pool. In this case, there are two possible outcomes:

   (a) The larger pool may have a more uniform (i.e., less skewed) distribution of device fingerprints when compared to the smaller (original) pool. In other words, different device fingerprints x_i would be relatively more equiprobable. In this case, the entropy calculated using Equation 3.1 would increase.

   (b) The larger pool may have a more skewed distribution of device fingerprints, where some device fingerprints x_i occur much more frequently than others. In this case, the entropy calculated using Equation 3.1 would decrease.

As can be seen in Equation 3.1, entropy depends not only on the skewness of the probability distribution, but also on the number of possible outcomes n. In all cases described above, n is likely to increase with the number of device fingerprints collected—this is because in practice, there will always be a chance of encountering a new device fingerprint that has not been previously observed. However, as the sample size of the collected device fingerprints grows, n will grow much more slowly, and device fingerprints x_i with very few occurrences will have negligible impact on the entropy calculated using Equation 3.1 (since the corresponding $P(X_i)$ will be near-zero).

Due to the lack of an established mechanism for determining or ensuring that the statistical distribution of a device fingerprint dataset matches with the real-world distribution of devices, it is difficult to draw accurate conclusions by comparing entropy values calculated using different datasets (especially when datasets differ in the number of device fingerprints collected). Different sample sizes may result in different calculated entropies, for the reasons discussed immediately above. For this reason, a normalized formula for entropy has been used [113, 37] to compare different datasets, and has been calculated by aforecited work as

$$H_{norm}(X) = \frac{H(X)}{H_N} \tag{3.2}$$

$$= \frac{H(X)}{log_2(n)}, \tag{3.3}$$

where H_N represents what $H(X)$ would be if all n values of X were equiprobable. This normalized formula yields a value between 0 and 1; it does not reflect the number of possible events n, but only the uniformity of the probability distribution. Therefore, it has been used to compare the relative uniformity of the device fingerprint distributions obtained by datasets using the *same* fingerprinting vectors. This measure does not reflect relative differences in distinguishing information offered by different vectors: any vector that yields equiprobable values would result in a normalized entropy value of 1 (irrespective of whether the vector yields, e.g., 2 equiprobable values, or 100 equiprobable values). Moreover, outliers have a non-negligible impact on $H_{norm}(X)$, since every outlier increases the value of n by 1, thereby increasing H_N and decreasing H_{norm}. In contrast, the impact of outliers on $H(X)$ is negligible. We therefore argue that $H_{norm}(X)$ is not guaranteed to be a robust measure of relative uniformity between different datasets, since larger datasets are more likely to have more outliers.

The takeaway from this discussion is that there are major limitations associated with widely-used methods for quantifying the distinguishing information offered by different fingerprinting vectors, and in comparing different datasets.

Sample Size and Probability Distribution: Analyzing Combined Vectors

Global Probability Distribution vs. Per-Website Probability Distributions

Even if the distinguishing information of device fingerprinting vectors could be calculated based on a probability distribution that accurately reflects the overall distribution of web-enabled devices, the utility of such a figure may in many cases be limited. This is because the device distributions observed by different websites may vary significantly. For example, if the 1(c) System time or 4(b) Geolocation vectors are used by a website that targets users from a specific country (e.g., a website for

a government service), the distribution of fingerprints would be much more skewed when compared to a website that targets all users worldwide (e.g., a social media website). Further work is required to develop more useful baseline measurements that are more directly applicable to a wider range of websites.

Comparative Analysis

In what follows, we give best-effort estimates of distinguishability, to help in comparing and selecting vectors to combine. Our analysis is informed in part by external studies, and in part by qualitative analyses of each vector's distinguishability. Despite the limitations associated with Shannon entropy and device fingerprint collection methodology, the numerical estimates given in bits should be interpreted as Shannon entropy values based on probability distributions that reflect the general population of web-enabled devices—currently, we believe that this is the most useful basis for comparison. The values provided may change in the future if more extensive datasets are collected or if the fingerprinted features evolve based on widespread changes among web-enabled devices (e.g., the deprecation of Adobe Flash). Table 3.2 summarizes the ratings, and its caption gives the scale and coarse mapping for the estimates below; RFS denotes Requires Further Study.

- 1(a) Major software and hardware details is graded High, as it includes attributes such as user-agent string, list of plugins, screen resolution, and presence of localStorage and sessionStorage (~10 bits, ~15 bits, ~4.83 bits, and ~2.12 bits respectively as per [56]), in addition to other attributes.
- 1(b) WebGL information is estimated Low (RFS), based on our preliminary experimentation.
- 1(c) System time depends on time zone and daylight saving time; 24 possible values would yield a maximum of ~4.65 bits, but in practice the distribution of users across time zones will be skewed, e.g., ~3 bits as per [56]. Clock skew can also be measured, but this RFS, so we grade this vector overall as Low (RFS).
- 1(d) Battery information (RFS, but) is estimated Low for battery life, assuming that web-connected devices largely have battery life in the range of 4-10

hours of active use; using current charge level would provide more distinguishability for within-session fingerprinting.

- 1(e) Evercookies are graded VeryHigh; the server can save a globally unique identifier on the device.
- 1(f) WebRTC is graded Medium (RFS), as most consumer routers assign IP addresses in the 192.168.0.x range, thus these have upper bound ~8 bits.
- 1(g) Password autofill and 4(i) Ad blocker detection are graded VeryLow; each are binary values.
- 2(a) HTML5 canvas fingerprinting is graded Medium, as it gives ~8.6 bits of distinguishing information, per [113].
- 2(b) System performance (RFS, but) is estimated Low due to limited granularity at which system performance can be measured within the constraint of a few seconds.
- 2(c) Hardware sensors is graded Medium; Bojinov [23] collected accelerometer data from over 3000 devices, calculating the entropy of the distribution to be ~7.5 bits.
- 2(d) Scroll wheel fingerprinting yields a binary value to distinguish scroll wheel from touchpad, but can also potentially detect differences in OS scroll speed settings; VeryLow (RFS).
- Vectors 2(e), 2(f), 2(g) are yet to be reported in empirical studies, but we estimate VeryLow; these vectors serve to distinguish between different browser vendors and versions, for which the distribution is likely skewed towards the most recent versions.
- 2(h) Video RAM is estimated VeryLow to Low (RFS). Many common configurations offer limited (say 2-3 bits of) choice, e.g., 1GB, 2GB, 4GB. (However this is complicated by many devices having shared system/video memory.)
- 2(i) Font detection is graded Low to Medium. Enumerating all fonts via the Flash plugin yields 7-14 bits [56, 113], with the spread largely attributable to substantially lower diversity of fonts on mobile devices. JavaScript font detection is unordered and cannot do a full enumeration (it tests using a list of known fonts).

- 2(j) Audio processing (RFS, but) is estimated Medium, since it is conceptually analogous to 2(a) [59].
- 3(a) Browser plugin fingerprinting includes full system font enumeration in addition to leakage of more granular system information (e.g., kernel version). However, mobile devices do not support plugins, and desktop web browsers are moving instead to the extension model [113], so we grade this Low to Medium.
- 3(b) Browser extension fingerprinting is graded VeryLow to Low, based on the expectation that few users install many browser extensions, aside from highly technical users. This may be too conservative, as some extensions (if installed) provide more distinguishing information, e.g., a NoScript whitelist.[9]
- 3(c) System-fingerprinting plugins can contribute considerable distinguishing information, since they have less restricted access to the underlying OS and hardware (see Section 3.2), compared to JavaScript. We grade it High.
- 4(a) IP addresses can often serve as global identifiers (but not always [38], e.g., due to NAT, proxying, and in some cases rapid address changes). We grade this vector High.
- 4(b) Geolocation can give considerable distinguishability, depending on the granularity of the specific geolocation mechanism. We grade this Low to High (high variability).
- 4(c,d) Based on the p0f [210] MTU and TCP flag signature lists, we estimate Low for 4(d) Passive TCP/IP stack fingerprinting, and Low to Medium for 4(c) Active TCP/IP stack fingerprinting; active probing is more powerful (RFS).
- 4(e) Protocol fingerprinting includes the list of HTTP headers (~4.36 bits [113]), the values corresponding to certain headers such as user-agent string, HTTP accept headers (resp. ~10 bits and ~6 bits [56]), and DoNotTrack (up to 1 bit). It can also be inferred whether cookies are enabled or disabled. SSL fingerprinting RFS. Thus we give an overall grade of High (RFS).
- 4(f) DNS resolver is analogous to geolocation, but less granular. We grade this Low to Medium.

[9] Published studies [56, 113], while skewed to technical users, did not specifically look at extensions beyond ad blockers. Technical users might improve privacy by, e.g., disabling Flash; using many browser extensions counters this.

- 4(g) Clock skew is graded Medium, as Kohno et al. [104] collected clock skew data for several thousand devices, calculating the entropy of the distribution to be ~6 bits.
- 4(h) Counting hosts behind a NAT is graded Low; for household users, there is likely little variation in the number of hosts behind the NAT that will communicate with the web server, e.g., 1 to 16 devices (it would be higher for enterprise devices).

3.4 Role of Device Fingerprinting in Augmenting Authentication

To determine the most appropriate role for device fingerprinting in web authentication, we first discuss why it is unsuitable as a sole authentication mechanism. An idealized[10] form of device fingerprinting within the context of authentication might have the following properties:

P1: Each device has a unique fingerprint that can be associated with a user's account.

P2: Fingerprints obtained at different times from the same device are either identical; or linkable, i.e., can be determined with high confidence to be from the same device; or if changed to the extent of being unlinkable (e.g., due to major changes in software or hardware configuration), a backup mechanism such as e-mail or SMS-based recovery is in place to allow the user to re-associate a device with the target account.

P3: One of the following two properties is present:

i) Fingerprints are released only to legitimate websites to which the user intends to authenticate.

ii) It is difficult for an attacker, even with full knowledge of the device's hardware and software configuration, to spoof that device.

If the above requirements could be met, device fingerprinting alone could be used for account authentication. However, from Section 3.3, these requirements appear

[10] This characterization is impractical, but represents a highly favourable scenario for a web application wishing to accurately identify devices by fingerprints.

unreachable at present. It nonetheless remains possible to use device fingerprinting to strengthen authentication via an additional dimension, and importantly, without increasing the user burden—as opposed to, e.g., trying to improve password strength by forcing a heavier burden on users through a more complex password policy.

Device fingerprinting in this context has two use cases, pursued in subsections below: a) augmenting start-of-session authentication (cf. models M1-M4 of Section 3.1); and b) maintaining authentication throughout a session, to stop hijacking of authenticated sessions (related to model M5).

3.4.1 Authentication at Start of Session

When used as an added authentication dimension alongside passwords (or another primary authentication method), to authorize a session the server requires both the correct primary response and matching client fingerprint data. The server must thus have persistent access to data sufficient to verify the fingerprint on later authentication requests. The relevant time frame over which a fingerprint must be stable thus spans multiple sessions. This increases the importance of property P2 above—if the device configuration changes to the extent that an evolved fingerprint is no longer verifiable, a backup mechanism is needed to re-associate device and account.[11]

While resource use is relevant in the scenario of start-of-session authentication, a fingerprint need only be collected once at the start of the session; thus resource use is not a major barrier to arbitrary combinations of Table 3.2 vectors.

Augmenting two-factor authentication. Client fingerprinting can augment two-factor authentication (by our terminology, "factors" typically involve user actions; "dimensions" like device fingerprinting ideally do not). Consider Google two-step verification [76], which requires users to log in with a username and password, followed by a six-digit SMS code sent in real-time to a mobile device. If the user chooses to "trust" the computer on which they have logged in, a cookie is saved by the browser to relieve the user of entering a verification code on subsequent authentication attempts from that computer for the next 30 days. This is a security trade-off made in favour

[11] Likewise, a backup mechanism is needed if the user logs in from a new device that the server does not yet recognize. The backup mechanism would thus indicate to the verifying server that the new device is being used by the legitimate user, so that its device fingerprint may be recognized on subsequent authentication attempts.

of usability, since an attacker obtaining the cookie bypasses the second factor. Device fingerprinting could be integrated into this scheme in at least three ways:

(a) When users submit a password, the server can validate the device fingerprint before sending the SMS code. If fingerprint verification fails, the server may require additional authentication tasks and/or send the user an alert (e.g., via e-mail)—improving security in the event that the device used for receiving the SMS codes is stolen.

(b) When users submit a password, the server can require both a matching device fingerprint and a "trusted" cookie to bypass the second authentication factor (e.g., 6-digit SMS code)—improving security in case of cookie theft.

(c) Advertisers use device fingerprinting to restore tracking cookies after users clear them. This suggests a third application: if a user clears browser cookies and later attempts to re-authenticate, the server may recognize, by a device fingerprint, that the user previously designated the device as "trusted", and allow it to skip the second factor (i.e., authenticate by username-password alone). However, this approach appears to be a bad idea: users may clear their cookies precisely so that the server "forgets" their device for security reasons, e.g., on a device shared by multiple users.

3.4.2 Authentication Throughout a Session (Continuous Authentication)

In typical password-based web authentication (Section 3.4.1), upon receiving a username and password the server returns a browser session cookie, allowing the client to maintain its authenticated state by including the cookie in subsequent HTTP requests; the cookie, as a bearer token, replaces the password. Thus, an adversary obtaining the cookie (e.g., by device theft, cross-site scripting, interception due to a man-in-the-middle (MITM) attack or improperly configured HTTPS [106]) can submit authenticated requests without password knowledge.

Some websites record the user IP address when initiating an authenticated session, and check that any incoming HTTP requests containing a session cookie originate from the same address [40]. Such address binding of session cookies enhances session

security—if the address cross-check fails, the server can terminate the session, and optionally alert the user and lock down the account pending additional authentication steps. Since the client sends the session cookie alongside each HTTP request, the server ideally validates the source IP address on each request. This may impact usability, e.g., user IP addresses often change in mobile environments; here it may help to use multiple fingerprinting vectors.

Since modern webpages contain many resources, visiting a single page generates many HTTP requests; thus in this use case, client-passive vectors are critical. For *fully* client-passive vectors (including those in 4(b) that are client-passive, e.g., IP-based geolocation), the server can extract relevant fingerprint data from existing traffic flow, and can thus validate them on each HTTP request; this is not so for partially client-passive 4(c) or 4(f), which require extra network traffic.

Non-client-passive vectors require the browser to perform certain operations and generate fingerprint output for inclusion in each HTTP request. If the server uses fixed vectors, and which do not involve time-varying challenges, and the device configuration does not change, this fingerprint remains static for the duration of the user viewing a single webpage. Thus recomputing this fingerprint prior to each HTTP request is redundant for legitimate clients (and might impact user experience if vectors are high overhead); but, static fingerprints allow replay attacks by M5 attackers.

The SmartAuth framework [162] addresses replay attacks by hashing the device fingerprint at the client with a counter before sending to the server. This is ineffective against M5 attackers employing XSS to steal session cookies, as XSS can also steal the plaintext device fingerprint and counter. Such attackers can resume the session on their own machine and continue generating valid device fingerprint hashes, e.g., incrementing the counter and recomputing the hash. An attack script may collect additional information about the target device to allow device spoofing in the event that the server dynamically adjusts its fingerprinting by, e.g., following the strategy employed by Unger et al. [194] of collecting a different subset of attributes each time the device is fingerprinted. M5, our most powerful attack model, grants session hijacking ability, and is more difficult to defend against.

In summary, for throughout-session fingerprinting:

1. Device fingerprints should be validated by the server for every HTTP request.

2. Advantages of client-passive vectors include (i) obscuring the server's fingerprinting strategy to the attacker;[12] and (ii) eliminating client resource burdens.

3. Periodically varying the fingerprinting challenge (see Section 3.1) improves spoofing resistance against attackers that intercept client-generated device fingerprints. However, forcing clients to regularly recompute device fingerprints increases resource usage costs.

4. Spoofing-resistance strategies that involve varying the format of client-generated device fingerprints (e.g., using counters and hashes, or varying the fingerprinted attributes used) can improve security but may be of limited effectiveness against advanced M5 attackers.

5. Vectors with higher spoofing resistance (see Table 3.2) provide stronger authentication assurances.

3.4.3 Account Recovery

When a fallback authentication mechanism is used for account recovery, the strength of user authentication offered by a web application is limited by the weaker of the two schemes (i.e., the primary mechanism and fallback mechanism) offered; cf. Section 2.1.3. Therefore, it is important for the fallback mechanism to offer security assurances comparable to the primary mechanism. Device fingerprinting can be used to augment fallback authentication mechanisms, similarly to how they can augment primary authentication mechanisms. Moreover, since fallback authentication schemes are used less frequently than primary schemes, they are less constrained by usability requirements. This allows a trade-off whereby more secure device fingerprints (i.e., offering higher distinguishing information and difficulty of spoofing) can be collected with the help of user involvement (e.g., using vectors that require users to perform specific actions, such as granting permission to obtain sensor readings).

Cao et al. [37] demonstrate that many fingerprinting vectors (especially those that rely on identifying hardware features) are cross-browser, meaning that the information obtained by them is not specific to any particular browser implementation. The need

[12] While this temporarily increases spoofing resistance, recall our base assumption that this ultimately becomes known.

for spoofing-resistance, and the more relaxed usability constraints for fallback authentication mechanisms, together motivate out-of-band (i.e., outside of the browser) collection of device fingerprints using trusted computing technologies. For example, Intel SGX [90] (Software Guard Extensions) can be used to execute signed code in a secure enclave on user devices, allowing websites to cryptographically validate that device fingerprints are generated by trusted code. During the account recovery process, this would require websites to supply a signed executable application that users could download and run to generate a device fingerprint, which the website could then compare with device fingerprints that were collected from users' devices during prior successful authentication attempts.

3.5 Concluding Remarks and Recommendations

Our classification and analysis is informed and validated by previous studies as cited earlier [56, 113, 23, 59, 210, 104] and our own experimentation with 19 fingerprinting vectors. One conclusion from these is that combining essentially any subset of vectors in Table 3.2 appears feasible. Combining multiple vectors into a device fingerprint of course affects the properties discussed (see Sections 3.3.1 and 3.3.2). Distinguishability is expected to increase, but does not in all cases—e.g., if a smartphone's model number is in the user-agent string, screen resolution adds no further distinction, as all smartphones of one model have a given resolution. While an attacker may try to statistically guess some components of a device fingerprint—e.g., spoofing the most common screen resolutions—components such as hardware sensor calibration may be completely random and thus difficult to guess efficiently.

While we cannot give precise quantitative guidelines, combining more vectors tends to improve spoofing resistance, or at least raise the bar. However, we provide some general guidance for combining vectors. A naive strategy for combining vectors may involve a simple voting scheme, where a minimum of n out of m vectors must match for the overall device fingerprint to be considered a match. However, we believe that a more effective strategy would be to divide the set of vectors into multiple subsets, such that each subset consists of vectors that are most closely correlated. Thus, the impact of a single user action that changes the device fingerprint (e.g., upgrading the browser, or logging in from a new location) would be more closely confined to a

single subset. For example, OS-specific vectors (e.g., OS version and some TCP/IP stack fingerprinting vectors) could form a subset; vectors relying on hardware-level manufacturing variation (e.g., clock skew and hardware sensors) could form a subset; browser-specific vectors (browser version, JavaScript standards conformance, CSS feature detection) could form a subset; and network- or location-related vectors could form other subsets. Statistical analysis of a device fingerprint dataset representative of the user base of a given website would help in determining the best division of vectors into subsets. A two-stage voting scheme could then be used: first, each subset would be assigned a collective match or mismatch, by a voting scheme between the component vectors in the individual subset; then, a voting scheme between the subsets would assign a match or mismatch for the overall device fingerprint. The voting scheme can be tuned to enhance usability by reducing the likelihood that legitimate users will be locked out or forced to fall back to a less convenient fallback authentication scheme. For example, users who lose their device, but attempt to log in from the same location using a different device on the same LAN, could be permitted access. The reverse (logging in from the same device from a different location) could also be permitted. We also recommend that, when possible, different subsets should use specialized criteria to determine whether the fingerprint is a match—e.g., in a subset of browser-specific vectors, a browser version string and presence of JavaScript/CSS features that indicate a newer browser version (with respect to the browser version that was last recorded for that user) should still be considered a match, whereas an older version should be considered a mismatch (we validate this by the reasoning that modern browsers typically apply updates automatically).

Further exploration may involve a more advanced quantitative analysis of overall device fingerprint diversity and appropriate mechanisms for enabling users to associate multiple devices with their account. New mechanisms may also be developed to better cope with vectors that lack sufficient repeatability. One approach that we recommend is to employ techniques that can determine the repeatability of individual fingerprint vectors collected from a device, and exclude those vectors in instances where the values are deemed unrepeatable. For example, clock skew fingerprinting would be less repeatable over a network link with high jitter (i.e., high variation in latency), which is trivial to measure. Determining the repeatability of fingerprint vectors such

as system performance may require developing new mechanisms (e.g., a mechanism may be developed to detect the CPU load on a device, and subsequently determine whether the measured system performance would be repeatable).

We again emphasize that the fingerprinting mechanisms discussed herein require no new user interaction and thus impose no additional usability burdens on users; given increasing attention to usability, this strongly motivates the use of device fingerprinting to augment user authentication.

Chapter 4

Framework for Evaluating Mimicry Resistance of Web Authentication Schemes

The many password alternatives for web authentication proposed over the years, despite having different designs and objectives, predominantly rely on the knowledge of some secret. This motivates us, herein, to provide the first detailed exploration of the integration of a fundamentally different element of defense into the design of web authentication schemes: a *mimicry-resistance* dimension. We analyze web authentication mechanisms with respect to new properties related to mimicry resistance, and in particular evaluate *invisible* techniques that provide some mimicry resistance (unlike those relying solely on static secrets), including device fingerprinting schemes, physically unclonable functions (PUFs), and a subset of Internet geolocation mechanisms.

4.1 Introduction

A challenge in providing sufficient security guarantees for web authentication, i.e., *user-to-web* and *device-to-web* (versus *user-to-device*), is that the security of many schemes, including those relying on *something-you-have* or *something-you-are*, requires the ability to protect a secret. For example, a physical biometric such as a fingerprint can be captured in transit and replayed by an attacker without possession of the physical fingerprint, resulting in security properties similar to other stored secrets such as passwords. The reliance of many schemes on an element of secrecy is reflected in the Usability-Deployability-Security (UDS) evaluation framework [25], where eight of nine security properties assess a scheme's resilience against the exposure of a secret.

We revisit the process of compromising an account from an attacker's perspective, now viewing it as a two-stage process involving both exposure and mimicry. Exposure refers to the capture of information that enables account access, such as a password,

session cookie, or a cryptographic key; mimicry refers to the imitation of a legitimate user's (or user device's) actions or associated attributes with the aim of generating an authentication response that would be accepted as legitimate by the server (e.g., spoofing a user's geographic location if location-based authentication [51] is used). Accordingly, a scheme's authentication token may resist attacks by both: resisting exposure and resisting mimicry.[1] We investigate the little-studied mimicry-resistance dimension in web authentication, first defining a suitable set of criteria and then ranking schemes across a continuum of three classes of resistance to mimicry, as detailed in Section 4.3. Mimicry-resistant authentication schemes have been proposed to displace password authentication in the *user-to-device* context (see, e.g., analysis of mimicry attacks on *user-to-device* authentication [101]). In contrast, our analysis herein of *user-to-web* authentication schemes, including several previously evaluated by Bonneau et al. [25, p.11] under the UDS framework, finds that most offer little to no resistance to mimicry.

To construct a more comprehensive evaluation framework for authentication schemes, we augment the existing UDS security properties, which concentrate on exposure resistance, with new properties measuring mimicry resistance. We leverage the UDS framework security properties to systematically rate authentication schemes across an additional continuum ranging from lowest to highest resistance to exposure, and use these as orthogonal axes (exposure and mimicry) to plot a two-dimensional chart. Along both dimensions, our evaluation reflects the scalability of attacks required to defeat a scheme: schemes plotted closer to the origin can be defeated by attacks that can be scaled with ease (such as online guessing) to break a large number of accounts; defeating schemes plotted further from the origin requires more targeted attacks (such as device theft) that are highly unscalable.

The lack of mimicry-resistant schemes among those previously evaluated under UDS [25] motivates us to select and evaluate a representative set of techniques for reinforcing web authentication that demonstrate the benefits of mimicry resistance. We evaluate techniques that fall under four approaches, namely device fingerprinting (FP) [56], Internet geolocation [54], Physically Unclonable Functions (PUFs) [209],

[1]The term *mimicry* has been used previously in the context of intrusion detection systems, referring to the attacker's ability to mimic legitimate traffic [203]. We use the term in the web authentication context.

and One Time Passwords (OTPs) [136]; some variations of these offer resistance to mimicry attacks, and/or are also *invisible*, in that they do not require any user effort to configure or use (see Section 4.2). We supplement the UDS framework with two usability properties and four security properties (Table 5.1). Under this revised framework, we evaluate techniques which fall under the four aforementioned approaches when combined with passwords, and find that invisible and mimicry-resistant schemes combined with passwords provide significantly higher resistance to attack. This constitutes an initial step towards identifying mimicry-resistant web authentication schemes that can enhance security with minimal usability penalties.

In summary, the following contributions are made in this chapter:

- Investigating the mimicry-resistance dimension in web authentication, including ranking schemes under three sub-classes of mimicry resistance.
- Analyzing relatively new "invisible" techniques, and evaluating their degree of mimicry resistance when used for web authentication.
- Constructing a comprehensive evaluation framework, which includes (a) a two-dimensional chart combining the exposure- and mimicry-resistance dimensions, to visually reflect the ability of a scheme to resist scalable attacks; (b) an augmented UDS framework (in Chapter 5).

The remainder of this chapter is organized as follows. Section 4.2 provides background. Section 4.3 introduces the mimicry-resistance dimension in web authentication, and plots a representative subset of authentication schemes onto an Exposure-Mimicry two-dimensional space. Section 4.4 offers insights gained from the analysis.

4.2 Background and Context

We briefly provide background and define terms for modes of authentication (Figure 4.1) and review related work on the evaluation of authentication schemes.

The standard method of user authentication on the web is for the user to type a password into a web form, which is submitted over the Internet for the web server to verify. We refer to this as *user-to-web* (see Figure 4.1) authentication—even though

the password is physically entered into the user's local device, the verifier is a remote server (cf. local vs. remote authentication, Section 2.1.1). The device used for authentication, which we call the *access device* herein, is a passive token[2] conveyor, and does no checking on behalf of the web server. In contrast, when a user authenticates to their smartphone, it is *user-to-device*. Some authentication schemes that appear to be *user-to-web* are actually *two-stage* authentication schemes that combine a *user-to-device* scheme and a *device-to-web* scheme; for example, a mobile payment app may authenticate the user via a biometric (e.g., fingerprint or iris scan), which unlocks a locally-stored cryptographic key used by the mobile device to authenticate to the remote server.

4.2.1 Invisible Authentication

As discussed in Section 2.1.1, implicit and zero-effort authentication reduce user burden by, e.g., measuring and using users' biometric attributes or physiological behaviors for authentication, without requiring any deliberate user effort. Many such schemes have been proposed, and are typically used for *user-to-device* authentication; the accuracy and resistance to mimicry of some of these schemes has been evaluated [101]. In contrast, herein we consider web authentication schemes, thus focusing on *user-to-web* and *device-to-web* authentication.

We also make the distinction between *implicit* and *invisible* schemes. The former refers to schemes that do not require extra user effort during login, but do require some initial user effort for setup; those are generally *user-to-device* schemes that authenticate the user based on their behaviour as measured through various sensors (e.g., accelerometer, swipe patterns), where the initial user effort is often downloading an app, or calibrating input sensors. On the other hand, we define *invisible* schemes to be *device-to-web* authentication schemes requiring no user involvement at all, neither during set-up nor login. Note that not all *device-to-web* schemes are invisible, as some require a user to carry out an action; for example, in a *device-to-web* scheme that fingerprints the access device's accelerometer while at rest [23], users may need to place their device on a flat surface.

[2]We use the terms *token* (by default indicating a digital token) and *credential* interchangeably. Hardware tokens will be explicitly identified.

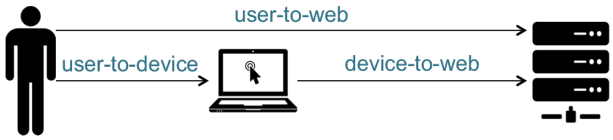

Figure 4.1: A comparison between *user-to-device*, *device-to-web*, and (direct) *user-to-web* authentication.

4.3 The Mimicry-Resistance Dimension

User authentication typically relies on *something-you-know* (i.e., some secret the user knows), *something-you-have*, or *something-you-are*. Biometric-based *user-to-web* authentication mechanisms (i.e., *something-you-are*) are similar in server-side implementation to *something-you-know*, since biometric data is (preferably) stored as a digital secret. Since the transmission path from the user's biometric sensor (e.g., fingerprint reader) to the authentication server is often untrusted, or at least less so than the path from the sensor to an authenticating application in *user-to-device* authentication, an attacker may defeat authentication by simply replaying an exposed secret (e.g., a fingerprint). Exposure can occur through, e.g., guessing or capture. Most *something-you-know* and *something-you-are* web authentication schemes offer limited resistance to mimicry, since exposure of a user secret typically allows attackers to trivially defeat the scheme.

User authentication via *something-you-have* requires verifying servers to both authenticate the hardware token *and* to verify user possession of the hardware token. Hardware tokens are generally electronic devices (e.g., a USB OTP token or smartphone) that can be authenticated by the server using, e.g., cryptographic techniques.[3] Because authenticating these hardware tokens will almost always rely on secrets, *something-you-have* often boils down to a *something-you-know*, i.e., something *the hardware token* knows. Most known variations of hardware authenticator tokens are defeated once that secret is exposed [25], e.g., by capture or theft, again providing little to no resistance to mimicry.

[3]Physically Unclonable Functions (PUFs) do not hide a key, but still have a component that may be exposed albeit harder to reproduce/mimic. See Section 4.3.3 for further discussion.

4.3.1 The Exposure-Mimicry Duality

Defeating web authentication typically requires two actions: exposing a token and mimicking certain behavior. That behavior is any action the legitimate user (or device) normally performs while authenticating to a service; for passwords, that behavior is trivially mimicked by simply submitting a static string. A scheme's resistance to compromise thus depends on its ability to independently resist *exposure* and *mimicry*. To evaluate a scheme's resistance across these orthogonal components, we construct a two-dimensional space in Figure 4.2 and plot various web authentication schemes on it. A marker (i.e., dot) represents a scheme's authentication token. Height along the y-axis indicates resilience to exposure; distance from the origin along the x-axis indicates resilience to mimicry. Schemes placed on the chart are explained in Section 4.3.3.

Vertical axis

The y-axis of Figure 4.2 is split into three segments: V1-Negligible-resistance (i.e., guessable), V2-Guess-resistant, and V3-Leak-resistant. Guessing a credential is the easiest form of exposure. Digital theft (*leak* henceforth) is generally easier than physical theft in that (1) leaks can often be arranged at scale, e.g., by phishing, and (2) leaks can be by remote access, e.g., an Internet-facing authentication server is subject to attack from anywhere in the world (an adversary need not be in physical proximity).

V1. Within the lower-vertical segment, a credential requiring more guesses is placed higher. The guessability of a scheme's credential may depend on several factors. A very weak password, for example, could be easier to guess than a randomly-generated PIN. Unless stated otherwise, schemes are plotted according to their strength in typical scenarios (e.g., passwords are assumed to be user-chosen, which limits their resilience to guessing). Guessing attacks are assumed to follow common guessing strategies for *trawling attacks* (capturing as many accounts as possible, often by guessing the most common password across all accounts, then moving down a list of candidate passwords).

V2. The middle-vertical segment comprises schemes with secrets impractical to guess, such as cryptographic keys or randomly-generated passwords with sufficient

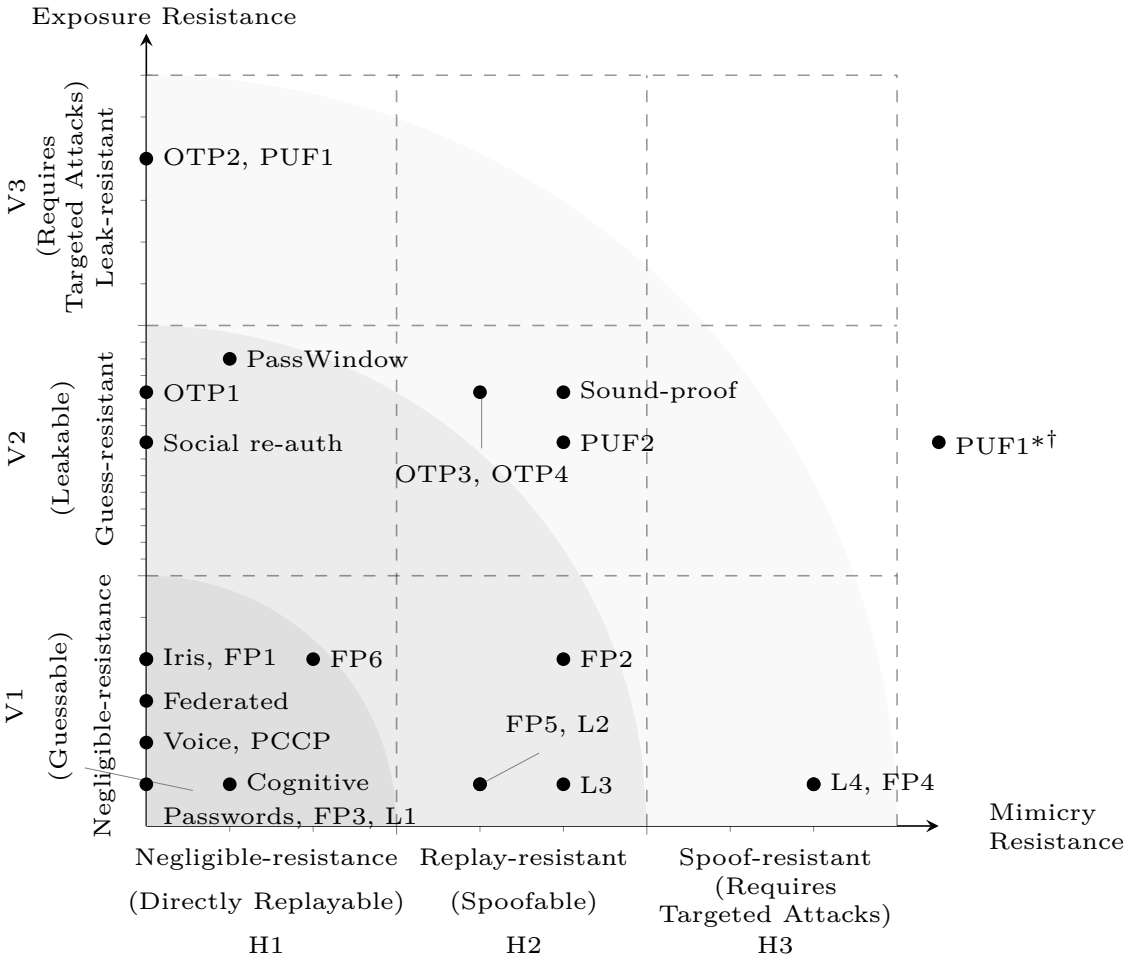

Figure 4.2: Exposure resistance and mimicry resistance as two dimensions to rate authentication-scheme resistance to compromise. Along both axes, distance from the origin reflects the scalability of attacks required to defeat a scheme: schemes plotted closer to the origin (darker regions) can be defeated by attacks that can be scaled with ease (such as online guessing) to break a large number of accounts; defeating schemes plotted further from the origin requires more targeted attacks (such as device theft) that are highly unscalable. Section 4.3.3 gives detailed explanations of different classes of One Time Passwords (OTPn), Device fingerprinting (FPn), Geolocation (Ln) techniques, and Physically Unclonable Functions (PUFn). †See inline discussion at the end of Section 4.3.3 (p. 80) on the placement of PUF1*.

length to withstand offline guessing attacks. Those may still be leaked (e.g., client-side malware). Malware may either (1) steal login credentials for later user impersonation, or (2) remain dormant until a user logs into their account, after which attacker-created transactions can be authorized by the malware on the victim user's device [119]. Herein we are interested in evaluating an authentication scheme's resilience to impersonation, not to malicious authorization. Defending against the latter may require mechanisms such as out-of-band authorization for sensitive transactions (which may be independent from the method of user authentication being used).

We consider within this segment four sub-levels, based on the number of sources from which leaks may occur: the human user, the browser, the user device (e.g., laptop or smartphone), and a public server whose compromise defeats authentication; e.g., a trusted third party of the main authentication server, or a party that also stores the same credential (e.g., a same password used across multiple websites). A scheme subject to leaks from all four sources is placed at the lowest of V2's four sub-levels; one subject to leaks from any three of the four places it second-lowest, and so on. A scheme not subject to leaks from any of these four sources is placed in V3.

V3. Schemes in the upper-vertical segment are those resilient to exposure by digital theft (e.g., leak or capture)—thus mostly physical tokens of some sort. Compromising these requires targeted attacks (most commonly physical theft) involving physical proximity to specifically pre-identified users. A scheme's vertical position within V3 varies with vulnerability to theft. A smartphone for example, relatively small in size and carried around more often, is easier to steal than a desktop PC—the latter may require physical break-in to an office and effort to conceal the escape.

Sorting rationale. The intuition behind arranging the three segments in the above manner follows logically from the trawling attack strategy of maximizing the number of accounts broken into per unit of attack effort. Assuming, for example, that a website does not throttle online password guessing, a good attack strategy is to try guessing passwords. If online guessing fails, an attacker often moves to digital theft (V2 segment on Figure 4.2), e.g., stealing credentials via XSS attacks, phishing, or client-side malware. Since random passwords may be leaked but not efficiently guessed, they are harder to expose. Weak passwords or PINs, for example, can be

guessed and often also captured,[4] and are rated lower in resistance to exposure. If all forms of leaks (digital theft) fail, the attacker is left with physically stealing, e.g., a smartphone or hardware authentication token. These attacks also become gradually harder to scale (for an adversary) in the aforementioned order.

Relationship to UDS framework. As summarized in Table 4.1, the vertical axis in Figure 4.2 addresses the S (Security) benefits in the UDS framework, excluding *Unlinkable* and *Requiring-Explicit-Consent*.[5] The UDS framework did not intend to provide an overall summary rating for a scheme, as that would require subjective weighting of the usability, security, and deployability benefits (which may be context-dependent). In contrast, we use our sorting rationale (as discussed above) in conjunction with the UDS security benefits to rate a scheme's overall resistance to exposure; for example, if a scheme only fails to provide *Resilient-to-Unthrottled-Guessing* but provides the remaining eight security properties, it is placed near the bottom of the vertical axis within V1 (despite having a virtually full row of security bullets). Passwords fail to provide *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing* and are therefore placed in V1, despite providing *Resilient-to-Physical-Theft* (which corresponds to our highest exposure-resistance category, segment V3).

Schemes are plotted along the vertical axis (see Section 4.3.3) based on their security benefits offered,[6], using the following criteria (also summarized in Table 4.1):

1. Schemes subject to guessing attacks (i.e., not offering both S3 and S4) are placed in V1.
2. Schemes offering S3 and S4, but lacking any of S1, S5-7, or S9, are placed in V2.
3. Schemes offering all of the above benefits are placed in V3.

[4] An example where a password can be guessed but not leaked would be challenge-response schemes where the password is never typed on the keyboard nor stored anywhere (neither on the server nor any client device), such that the user computes the response from the challenge in their head, or offline using a calculator. That would be resilient to phishing, theft, malware, leaks from verifiers, requires no trusted third parties, and possibly physical observation.

[5] These two security benefits from the original UDS framework do not share the focus of Figure 4.2 on exposure resistance. *Unlinkable* is a privacy benefit and *Requiring-Explicit-Consent* relates to malicious authorization, e.g., a malicious RFID-based card reader embedded in a sofa that authorizes a transaction without user knowledge [25].

[6] For schemes already evaluated in the original UDS paper [25] we use the benefits evaluated therein.

Table 4.1: Original UDS security benefits [25] evaluating schemes by susceptibility to exposure, and relation to new framework. This table lists the properties a scheme must offer to move to the next-higher vertical segment in Figure 4.2; i.e., both S3 and S4 must be offered to be placed in V2; all of S1, S3-S7 and S9 must be offered to be placed in V3. The benefits are listed in descending order of the scalability of carrying out the corresponding attack; e.g., guessing attacks (S3) are highly scalable, but physical theft (S8) is highly unscalable.

UDS Security Property	Segment
S3. *Resilient-to-Throttled-Guessing* S4. *Resilient-to-Unthrottled-Guessing*	Lower (V1)
S1. *Resilient-to-Physical-Observation* S5. *Resilient-to-Internal-Observation* S6. *Resilient-to-Leaks-from-Other-Verifiers* S7. *Resilient-to-Phishing* S9. *No-Trusted-Third-Party**	Middle (V2)
S2. *Resilient-to-Targeted-Impersonation*† S8. *Resilient-to-Physical-Theft*	Upper (V3)

*Additional third parties increase chances of leaks from public servers. From the adversary's perspective, this property is similar to *Leaks-from-Other-Verifiers*. See inline for details.

†S2 encompasses a variety of targeted attacks, e.g., deceiving a human acting as a trusted party as in social re-authentication [29], locating a user's password written on a post-it note, or lifting fingerprints from a doorknob [25]. The success of these attacks requires attackers to be in physical or logical proximity (e.g., social engineering attacks against social re-authentication) to victims. The difficulty in scaling such targeted attacks is similar to attacks involving physical theft (S8). These differ from more scalable attacks such as educated guessing based on information scraped *en-masse* from social media—these are covered by the more general guessing attacks (S3 and S4).

Table 4.2: Three new UDS-type security properties (given in Section 5.1) directly related to mimicry. A scheme must achieve these to move further right across the horizontal segments in Figure 4.2. Example: Both M2 and M3 must be achieved to be placed in H3.

New UDS Security Properties	Segment
M2. *Resilient-to-Delayed-Replay*	Left-most (H1)
M3. *Resilient-to-Immediate-Replay*	Middle (H2)
M4. *Resilient-to-Spoofing*	Right-most (H3)

Horizontal axis

The x-axis (mimicry resistance) reflects burdens on the attacker to mimic the verifier-expected behavior, *after* exposing a scheme's credential. Unfortunately, the security of most authentication schemes proposed to date rely on some form of resisting exposure, with compromise complete once an underlying secret token is revealed. This includes passwords, *user-to-web* physical biometrics (see Section 4.2), and poorly managed/generated private keys [85]. If leaking a credential does not allow an attacker access by direct replay of the credential, the scheme exhibits some degree of mimicry resistance, and gains horizontal distance from the origin in Figure 4.2.

We split the horizontal axis into three segments: H1-Negligible-resistance (i.e., easily replayable), H2-Replay-resistant, and H3-Spoof-resistant. In Section 5.1, we expand the UDS framework by six new benefits; three of these benefits (M2, M3, and M4) correspond to the aforementioned horizontal segments (H1, H2, and H3) as defined precisely in Table 4.2.

H1. If an attacker can simply submit or replay a credential after exposing it, the scheme provides no resistance to replay and thus gains no horizontal distance along the mimicry-resistance dimension (i.e., lies on the vertical axis). For example, replaying a captured password is trivial; thus, passwords lie directly on the vertical axis in Figure 4.2.

In some cases, there is no clear-cut answer as to whether exposing information directly helps the attacker gain access. For example, answers to personal knowledge questions based on recent account activity (e.g., *Who was the last person you emailed?*) likely remain unchanged for a few hours. Capturing an answer allows the attacker to login only within that window. Such schemes are given some horizontal

distance from the origin, but remain within H1. Another example is password expiration policies [211, 42] forcing resets every, e.g., one hour, which would provide some mimicry resistance (albeit highly unusable), and placed within H1 on Figure 4.2.

Some challenge-response schemes are also candidates for H1; whether or not a captured response directly allows the attacker to compromise the account is conditional on whether the captured response remains valid. Schemes where the set of challenge-response pairs is finite and relatively small, or when capturing a handful of challenge-response pairs suffices for defeating authentication, would thus fall within H1. Cognitive schemes like Weinshall [207], and some challenge-response schemes such as PassWindow [157], are prominent examples. Other challenge-response schemes are placed in H2.

H2. After a credential is leaked, schemes that either require additional attacker actions beyond simply replaying a string or conducting a straightforward operation (e.g., cryptographic signing), or where attackers have a limited time window (e.g., < 2 mins) in which the credential can be used, are considered replay-resistant and placed within H2. An example is DNS resolver fingerprinting (FP5 in Figure 4.2), which determines the DNS resolver used by the client (see Section 4.3.3); knowing the resolver's address is insufficient to defeat authentication, as the attacker must also be able to use it to resolve domain names.

One-time password (OTP) over SMS is another example scheme placed within H2, as it enables the attacker to compromise the account only if the user has not already used the OTP. An attacker that captures the OTP in clear text [69] must use it before the legitimate (victim) user. Additionally, the server may set a 2-minute time window where an OTP token is valid for usage, and expires afterwards. Such schemes would be placed within H2 on Figure 4.2.

H3. We place within H3 any scheme that requires additional equipment and/or systems (e.g., hardware chip manufacturers, or large scale distributed botnets) to mimic the behavior that the server measures and expects from the legitimate user. An example is robust location verification (see L4 in Section 4.3.3).

Because spoof-resistant schemes rely on measurements of various phenomena (e.g., a user behavior or habit), the measurement process must typically allow some degree of tolerance to account for (1) imperfections in the measuring apparatus, or (2) the

phenomena's natural instability over time.

For (1), consider an example location-based authentication scheme [197], where a server grants access to a user only if the user is at an expected geographic location. Ideally, the user's location would be identified to such a high granularity that no two human beings could exist at the same {latitude, longitude, altitude} coordinates.

For (2), again using location-based authentication as an example, a user's daily commute [94] from one geographic location to another (e.g., work to home) makes it impractical to grant account access only when the user is geographically present in, e.g., a 1 m^2 area, even if the geolocation mechanism being used allows for such high accuracy. Geolocating users with courser granularity (e.g., city-level) would thus seem more practical (e.g., to avoid false rejects) for generic location-based authentication purposes.

The degree of "fuzziness" (i.e., lack of precision) introduced while measuring a user's behavior in spoof-resistant schemes is thus likely to result in *false accepts*, i.e., falsely accepting another user as the intended one. In practice, some websites use IP-address based location look-ups [158] to implement location-aware authentication, which returns locations at the city or state level. Thus, all attackers physically present in the user's city (or even a legitimate user mistyping the username with no malicious intent, i.e., if geolocation is used as a stand-alone authentication scheme) are falsely accepted.

Note the distinction between false accepts (a result of imprecision) and mimicry resistance (a result of resistance to attack): A scheme that uses high-precision GPS co-ordinates (e.g., within a radius of $1m$) reported by the user browser (e.g., L1, Section 4.3.3) may offer low false accept rate, but offers no mimicry resistance since attackers can use browser extensions that report forged coordinates.

False accepts are not directly reflected in the Mimicry-Resistant dimension (though we capture them in the evaluation framework in Section 5.1). They are indirectly captured however by a scheme's position vertically. This is because resilience to guessing attacks and resilience to false accepts are both related to the size and distribution of the credential space (e.g., cryptographic keys with sufficient length and chosen uniformly at random are not subject to guessing attacks and false accepts).

4.3.2 Interpreting Attack Scalability

Figure 4.2 illustrates the relative resilience of schemes with respect to trawling attacks, i.e., attacks aiming to maximize the number of accounts broken into. Schemes further from the origin are less scalable to attack, i.e., they require more targeted attacks and therefore impose a higher cost on the attacker in terms of time, effort, and money, as a function of the total number of accounts targeted for attack. Schemes closer to the origin are subject to more scalable attacks, i.e., a greater number of accounts broken into at lower cost.

Along the vertical (exposure-resistance) axis, schemes most vulnerable to scalable attacks are those where the attacker can guess user credentials (V1). Attacks against schemes requiring information theft (e.g., via software vulnerabilities or malware) are less scalable (V2), and schemes requiring physical device theft (V3) are the least scalable (i.e., most costly) to attack.

Along the horizontal (mimicry-resistance) axis, attacks against schemes where an exposed secret can be used as-is by the attacker (see Section 4.3.1) are most scalable; attack scalability decreases for schemes that increase the attacker burden (in terms of hardware costs or control of network infrastructure) for mimicking account-holders beyond a simple replay attack (see Table 4.2). Since our evaluation is qualitative, it is not intended that absolute distance from the origin be a quantitative measure suitable for comparing schemes—however, the resilience of two schemes (i.e., scalability of the attacks that a scheme can withstand) can be compared if they share one of the two co-ordinates (i.e., identical x- or y-coordinates).

4.3.3 Scheme Placement on Figure 4.2

We use the mapping method of Section 4.3.1 ("Relationship to UDS framework") to position a representative subset of schemes along the exposure-resistance axis using their previously assessed (cf. [25, p.11]) security benefits. Upon evaluating the mimicry resistance of these schemes, we find they offer little to no resistance to mimicry as we explain below.

PassWindow [157] is a visual crypto technique whereby the user overlays a transparency card on the screen to visually read and enter a 4-digit pass-code, which proves

possession of the user's card. It was shown [142] that observing 20–30 challenge/response pairs can leak the card's secret; we thus place the scheme within H1. Likewise, cognitive schemes, including GrIDsure [96], Weinshall [207], Hopper Blum [87], and Word Association [180] are placed in H1 since the schemes can be compromised after few observations (e.g., Weinshall can be compromised after about seven observations [68]).

In social re-authentication [29], a trusted friend can vouch for a user who, e.g., has lost their credentials. An attacker may capture a vouch code (e.g., via malware on a trusted friend's device) and simply replay it to authenticate as the victim; we thus place it in H1.

Federated schemes, including OpenID [164], Microsoft Passport [105], Facebook Connect, and BrowserID [82], offer no mimicry resistance if password authentication is used, and we thus place them in H1. Their position on the chart would change if the identity provider utilizes mimicry-resistant schemes. Persuasive Cued Click-Points (PCCP) [41] and biometric schemes on Figure 4.2 (Iris and Voice) are also placed in H1, since captured biometric samples are typically replayable [25].

We explore and discuss example schemes which do provide mimicry resistance, and position them along the two axes (the remaining schemes on Figure 4.2). To avoid redundancy, we only explain the horizontal position of these schemes in this section; see Section 5.1 for their vertical position.

Geolocation (selected methods)

We review four broad classes of Internet geolocation techniques that can be used for location-based authentication, placing a marker for each class on Figure 4.2 to rate their ability to resist compromise. Location-based authentication typically requires server-side storage of the user's expected location (e.g., city-level, nation/state level, or latitude/longitude coordinates, depending on the geolocation method employed) in plain text, hashed, or cloaked [46] to a certain degree to preserve users' privacy. Any of the four classes of geolocation methods described below may be used by a verifying server to obtain and/or verify the user's current location at the time of authentication, and grant access if the user is verified to be at the location expected by the server.

L1: GPS and WPS. GPS and WiFi Positioning System (WPS) geolocation are commonly used in practice. WPS uses multi-lateration based on the signal strengths between the device and nearby WiFi access points with known locations. These techniques are usually selected by the user's browser in the W3C geolocation API [159], whereby the browser obtains the coordinates from the device's GPS driver, or from a location provider after submitting a list of nearby WiFi access points and their signal strengths to the location provider. L1 gains no depth across the mimicry-resistance dimension (see Figure 4.2), since techniques rely on browser-reported information that can be substituted and replayed by an attacker, so knowledge of the user's location is sufficient to break authentication.

L2: IP-address based tabulation. Tabulation-based geolocation service providers such as `Maxmind`[7] and `ipinfo`[8] maintain lookup tables, which map IP address blocks to cities and countries, possibly through publicly available information such as IP address registries (e.g., *whois*[9]) and the geography of IP address allocation. Geolocation based on IP address table lookups is unreliable due to outdated entries [158], and is evadable through use of middleboxes and virtual private networks (VPNs) [137]. L2 is rated slightly more resilient to compromise than forgeable GPS/WPS coordinates, and is placed within H2; it is not in H3 since even an attacker unable to forge source IP addresses may use public HTTP proxies or bots in close proximity to the user.

L3: Measurement-based geolocation. In this class, network measurements, such as Round-Trip Times (RTTs), are conducted from a set of landmarks (e.g., cloud-based or CDN servers) to the target user, and are then mapped to geographic distances using a pre-calibrated delay-to-distance function. The user's location is estimated through multi-lateration, relative to the landmarks' locations. Examples include Spotter [110] and CBG [81]. Other proposals have suggested mapping the network topology for higher accuracy [114]. To date, measurement-based geolocation can achieve an accuracy on the order of a few tens of meters. Manipulating L3 requires more advanced techniques than simply submitting forged coordinates, but can be achieved with enough knowledge of the network topology and the landmarks/verifiers

[7] https://www.maxmind.com/
[8] http://ipinfo.io/
[9] https://www.whois.net/

being used [1, 67]; they are limited to H2.

L4: Robust location verification. Some techniques are designed to verify location information obtained by other Internet geolocation techniques (e.g., L1-L3, above), and/or are designed to be resilient to common adversarial manipulation tactics. The result of a preliminary geolocation is treated as an asserted location (analogous to a username asserting identity), to be verified by a measurement-based proof (analogous to proof of knowledge of a secret). Examples of such schemes include Client Presence Verification (CPV) [2] (see Section 5.1) and Trusted Platform Module (TPM)-supported GPS drivers [154]; the former cryptographically protects network delay measurements used for verifying location assertions, and the latter communicates coordinates securely. We call such techniques robust location verification and rate them as spoof-resistant (H3) because manipulation requires attackers to expend more effort than simply reporting a false location; successfully spoofing legitimate client locations requires using specialized proxy machines (for attacks that "co-locate" using a machine nearby to the victim) or GPS satellite signal-spoofing devices [193].

Device Fingerprinting

Device fingerprinting refers to techniques by which a server collects information on a device's hardware/software configuration for the purpose of identification [56]. From the 29 methods surveyed in Chapter 3, we derive six representative categories for evaluation.

FP1: System parameters/preferences. This class includes software and hardware information about the device to be authenticated, provided by the web browser's JavaScript API (e.g., the navigator and windows JavaScript BOM objects), such as operating system version, screen resolution, time zone, system language, and supported WebGL capabilities. FP1 lies on the y-axis, since it can simply be mimicked by replaying the information.

FP2: Audio and canvas challenge/response. This class includes two techniques that fingerprint the client's graphics and audio subsystems, respectively. HTML5 canvas fingerprinting renders a variety of text and graphics in an HTML5 canvas on the client's browser, which results in subtly different images (e.g., due to

differences in anti-aliasing or font smoothing) depending on the graphics driver and hardware on the device being fingerprinted. Audio processing fingerprinting leverages the HTML AudioContext API that provides real-time frequency- and time-domain analysis of audio playback, mainly used for creating audio visualizations. Playing the same sound on different devices results in subtly different waveforms, depending on the sound driver and hardware. To improve resistance to replay attacks, these two techniques can be used in a challenge-response scheme, where the server can store the client's responses to many different challenges [33], and are thus placed within H2.

FP3: Hardware sensors. This class, suitable to fingerprint smartphones, leverages the inherent variation in the manufacturing and factory calibration of typical smartphone sensors, such as accelerometer and speaker-microphone systems. Similar to FP1, FP3 lies on the y-axis since information can simply be replayed.

FP4: Clock skew. The server uses TCP timestamps to measure the clock skew of the device being fingerprinted, which differs across devices due to manufacturing variation. FP4 is placed within H3, since clock skew spoofing attacks are highly sophisticated; e.g., Arackaparambil et al. [10] show how timestamp manipulation can be detected to identify rogue wireless LAN access points. However, clock-skew spoofing over a WAN connection is not well-studied, and therefore this rating may change subject to further study and experimentation.

FP5: DNS resolver. The server determines a client's DNS resolver(s) by presenting to the client a page that contains a reference to a randomly-generated (nonexistent) subdomain, triggering a client DNS lookup; as a result the server will receive a DNS query from the client's resolver. The DNS resolver IP address serves as a fingerprint. FP5 provides partial replay-resistance but can be defeated via the use of proxies, similar to L2. In some cases, organizations run their own DNS servers, which resolve domain names only to machines within the department's network. To use a victim's DNS resolver, an attacker would need to be able to resolve domain names using the organization's private DNS server. Since identifying the server is not enough (and is easier than using the organization's DNS), FP5 is placed within H2.

FP6: Protocol-based fingerprinting. This class includes schemes that glean information from network-, transport-, and application-layer protocol fields. For example, the TLS library of the client device can be fingerprinted using the Client Hello

packet received during the handshake sequence, which includes information such as the device's supported TLS version, supported ciphersuites (and their order of presentation), compression options, and list of supported extensions (along with associated parameters such as elliptic curve parameters). FP6 lies within H1, since it is susceptible to mimicry, but attacks are less scalable than simply replaying a static string; OS- or library-level modifications may be required.

OTP Schemes

One-time password (OTP) schemes generate short-lived credentials, often used as a second factor alongside conventional passwords. Depending on implementation (four follow), an attacker may aim to capture either the seed or a challenge-response pair.

OTP1: OTP mobile apps. This class (e.g., Google Authenticator) generates OTPs to be manually typed into a user's access device, using a combination of a locally-stored shared secret and either the current time (TOTP [136]) or a counter (HOTP [135]). With malware on the device, the attacker can capture the locally-stored shared secret; because this directly enables the attacker to compromise the account, OTP1 provides no resistance to mimicry, and is placed on the vertical axis of Figure 4.2. Note: A variant of this class is a mobile app that stores a public-private key pair, e.g., as used by Duo Security [55] and Google Prompt (a more recent iteration of Google two-step verification [9]). When the user types in their password, the server sends a cryptographic challenge to the mobile app. The mobile app then requests user consent, via simply tapping a button, to send the cryptographic response to the server to complete the user authentication process. The advantage of this approach compared to OTP1 is that it is more efficient to use. The disadvantages are that the mobile app requires an Internet connection, and there is a chance of the user accidentally consenting to a malicious authentication attempt.

OTP2: OTP USB tokens. This class (e.g., FIDO U2F keys [147]) is similar to OTP1, but requires less effort since the user can press a button on the hardware token to automatically enter the OTP into a browser window. Assuming hardware tokens can resist malware, it becomes relatively challenging for an attacker to capture challenge-response pairs. Similar to OTP1, the attacker can thus target the seed, i.e., hardware token theft. This gives the attacker direct account compromise, and

therefore is placed on the vertical axis (i.e., no horizontal distance from the origin) in Figure 4.2.

OTP3: SMS OTP. The server sends a randomly-generated OTP (i.e., no reliance on a shared secret/seed) to the user via SMS. Contrary to the previous two OTP classes, the seed here is only stored on the server, which makes it harder to capture. The attacker can however capture an OTP in transit, e.g., by exploring weaknesses in the cellular network [69], but will be required to use it before it expires (and before the user uses it). Since the time window for a successful attack is limited, OTP3 is placed further (horizontally) from the origin, in H2 on Figure 4.2.

OTP4: E-mail OTP. The server sends a randomly-generated OTP (again no reliance on a shared secret/seed) to the user via e-mail. Because an attacker can capture an OTP, e.g., via malware on the user's machine, its mimicry resistance is similar to OTP3.

PUFs

Physically Unclonable Functions (PUFs) are hardware modules (typically manufactured into silicon-based chips) that leverage an underlying unique physical structure to generate challenge-response pairs; ideal PUFs are impossible to clone, since the unique structure of each individual PUF results from manufacturing variation [209]. For our evaluation herein, we assume the use of a challenge-response protocol (as described by Yu et al. [209]) in conjunction with PUFs that are built into the user's device (as opposed to, e.g., a USB-based token).

PUF1: Strong PUFs. These theoretically generate an endless supply of challenge-response pairs, allowing verifying servers to store any number of challenge-response pairs (e.g., at user account creation) to be used for user authentication. Several implementations of strong PUFs are available [168]; e.g., some PUFs generate challenge-response pairs by shining a laser on a scattering object at selected angles and points of incidence [153]. A major challenge in developing strong PUFs is to avoid susceptibility to model-building attacks that collect challenge-response pairs and apply machine-learning algorithms to build a mathematical model of the PUF [50]. Since strong PUFs resist such attacks, the only security property that PUF1 lacks from Table 4.1 is *Resilient-to-Physical-Theft*, thereby placing PUF1 in

V3; since device theft allows an attacker to directly impersonate the user (i.e., defeat authentication), PUF1 offers no mimicry resistance for this type of attack.

PUF2: Weak PUFs. These can only generate a limited number of challenge-response pairs. They are suitable for authentication when augmented with appropriate mechanisms, e.g., when restricted to only responding to challenges sent from a single trusted verifier over an authenticated channel [209]; this limits an attacker's ability to exhaustively capture all possible challenge/response pairs to mount a mimicry attack. PUF devices are commercially available, e.g., for device authentication and cryptographic key storage [91]. PUF2 is placed within H2—it lacks *Resilient-to-Internal-Observation* from Table 4.1, but a successful mimicry attack requires building a mathematical model of the PUF only after capturing a sufficient number of challenge/response pairs (e.g., via man-in-the-middle); however, since every PUF is unique, the mathematical model would be device-specific.

As discussed later in Section 4.4, schemes can be plotted in Figure 4.2 using multiple markers that reflect resilience against different classes of attacks. To illustrate this, we plot an additional marker labelled PUF1* to depict the mimicry resistance of PUF1 against model-building attacks. Since strong PUFs are resilient to model-building, PUF1* is placed outside of the plot axes (thus reflecting the impracticality of such an attack). This serves to illustrate that while PUF1 does not offer any mimicry resistance in the event of device theft, it is immune to model-building mimicry attacks.

Sound-Proof

Sound-Proof [99] is an authentication scheme that determines whether the user's smartphone and access device (i.e., another device from which the user is authenticating to access their online account) are in close proximity. The access device records ambient sound from the microphone and transmits it to the smartphone (via the Internet), which compares with the sound recorded by its own microphone. A match indicates that both devices are in an identical noise environment, and therefore likely in close proximity, resulting in the smartphone sending a cryptographically-signed assertion to the web server to approve the user authentication.

We include Sound-Proof in our evaluation as it provides a degree of mimicry resistance if a trusted channel is established between the smartphone and web server,

e.g., by means of a TPM (via a trusted execution environment and secure key storage). A TPM-augmented Sound-Proof would address attacks where, e.g., the attacker can steal the key via root-privileged malware and use it to sign assertions. Note that while Sound-Proof was originally proposed and evaluated as a second factor alongside passwords, we place it on Figure 4.2 as a single-factor scheme (without passwords)—Section 4.4 explains our rationale.

Although a TPM-based implementation may render it impractical for an attacker to steal the Sound-Proof app's cryptographic key from the user's smartphone and thus gain permanent access to the account, the attacker may still use a malicious app on the user's device to relay recorded audio. For example, the attacker can reproduce the ambient sound from the environment in which the user's smartphone is currently located, e.g., by leveraging malware on the user's smartphone or a nearby device to record and relay the ambient sound to the attacker. Sound-Proof is thus placed in H2 because this process is more sophisticated than simply replaying a static token. It is not placed in H1, since repeated (physical or internal) observations do not seem to help the attacker gain permanent access—the authors show [99] that the scheme is resilient to guessing attacks even when the attacker knows the user's environment (e.g., by using typical background noise from a Starbucks coffee shop), and that attacks are also made more difficult by requiring the sound files recorded by both devices to be timestamped (with NTP-synchronized clocks).

4.4 Further Insights

We briefly discuss further insights from our analysis of mimicry resistance, and the relative strengths of schemes plotted in Figure 4.2.

Multi-factor authentication. Figure 4.2 should be used to evaluate individual authentication schemes. To evaluate a multi-factor scheme, the scheme must be broken down to its constituent factors, each represented individually as an independent marker. Two independent markers can then be combined as a two-factor scheme as follows: the (x,y) position of the resultant (combined) scheme is the greater of both x-values and both y-values. That position should however be carefully interpreted. For example, although combining L4 with OTP2 would result in a marker in {H3,V3} (see Figure 4.2), an attacker capable of physically stealing the device to defeat OTP2

will already be geographically co-located with the user, and thus need not expend any additional effort to defeat L4.

Lack of schemes in top-right corner of Figure 4.2. None of the schemes analyzed have strong resistance to both mimicry and exposure—see the three empty squares in the top-right ({H2,V3}, {H3,V2}, {H3,V3}). Schemes in this region would strongly resist scalable attacks. This motivates combining complementary schemes, as discussed immediately above.

Bands versus markers. Instead of marker representation, schemes on Figure 4.2 could be represented using bands (i.e., lines/curves or shaded areas) or multiple markers. For example, passwords could be represented by a vertical band from V1 (user-chosen passwords) to somewhere in V2 (system-assigned passwords that are resilient to guessing attacks, but still subject to leaks). Such representation can help identify how different implementations of a scheme may alter security. We chose the simpler representation as it is easier to interpret, and to avoid cluttering the chart with intersecting bands.

Chapter 5

Comparative Evaluation of Web Authentication Schemes with a Mimicry-Resistance Dimension

The primary contribution of this chapter is the augmentation of the original 25 usability, deployability, and security properties (benefits) of the UDS framework [25] with six new benefits relevant to schemes that are invisible and offer mimicry resistance. We use the augmented UDS framework herein for the first detailed exploration of the benefits of combining mimicry-resistant web authentication techniques with ubiquitous password-based *user-to-web* authentication.

The remainder of this chapter is organized as follows. Section 5.1 evaluates relatively little-explored schemes found to have mimicry resistance, using a modified UDS framework with new properties addressing the mimicry-resistance dimension. Section 5.2 analyzes benefits when the aforementioned schemes are combined with passwords. Section 5.3 concludes and offers recommendations.

5.1 Comparative Evaluation

In Table 5.1, we evaluate selected classes of invisible and mimicry-resistant authentication schemes (selected baseline schemes are also included for comparison). We augment the original 25 usability, deployability, and security properties (benefits) of the UDS framework [25] with six new properties relevant to schemes that are invisible and have mimicry resistance. The original UDS properties (see Appendix A) and our new properties defined below are italicized when referenced herein. The new properties are now described:

U9. No-False-Rejects is a usability benefit concerning authentication failures resulting from *system* error (e.g., due to measurement error). False rejects arise due to *fuzzy* or non-binary matching functions employed by some mimicry-resistant and/or invisible authentication schemes, and are thus relevant to penalize (by withholding this benefit) schemes that suffer from false rejects (e.g., measurement-based Internet

geolocation methods). This differs from *Infrequent-Errors* (U7, Appendix A), wherein authentication may fail due to *user* action (e.g., incorrectly typing a password) or attempts to authenticate under unusual circumstances (e.g., from unexpected locations).

U10. Easy-to-Change-Credentials is a usability benefit for schemes where a user may easily change credentials (e.g., in event of a server database leak). Intuitively, since invisible schemes require no user action upon login or account set-up (see Section 4.2), these schemes rely on remotely (cf. *device-to-web*) observing habitual user/device attributes and/or behaviours, and transparently verifying these upon login. Changing these credentials (regardless of the reason for changing) is likely to impose user burden, i.e., changing physical habits and/or habitual behaviors. This new property (U10) allows appropriate penalization for such schemes. Note that in contrast, *Easy-Recovery-From-Loss* (U8) reflects how easily a user can recover from a credential loss (e.g., forgotten password). Credential loss requires a fallback mechanism to verify the user's identity; changing credentials does not, since the user remains in possession of valid credentials. *Easy-to-Change-Credentials* is inherent to the authentication mechanism itself, whereas *Easy-Recovery-From-Loss* may also depend on the fallback mechanism.

M1. No-False-Accepts is a mimicry-related (hence, the 'M' index) security benefit of schemes that have a sufficiently large credential space and/or measurement precision such that two different sets of credentials (e.g., iris scans from two different individuals) are always distinguishable. False accepts may include both non-malicious users and attackers mistaken for legitimate users, e.g., due to close proximity in a geolocation scheme, or a device fingerprint similar (within a margin of error) to that of a legitimate user. Similarly to false rejects, the fuzzy nature of many mimicry-resistant schemes explored in Section 4.3 also introduces the possibility of false accepts, justifying the importance of this new property (M1). Note that false accepts exclude attacks on the integrity of the authentication system (these are covered by other security properties), such as manipulating delay measurements to spoof a location or tampering with client-side code to spoof a device fingerprint. For example, a location-based scheme lacks this benefit if it is susceptible to colocation attacks, i.e., where the attacker travels and colocates himself with the user in a highly targeted attack.

M2. Resilient-to-Delayed-Replay is a security benefit of schemes in which credentials are not static, but change relatively slowly, e.g., personal knowledge questions based on the user's account activity, such as recent transactions. Such non-static credentials, when compromised, limit the duration for which the attacker retains account access. Schemes with horizontal distance from the origin in Figure 4.2 (i.e., placed in H1, H2, or H3) offer this benefit.

M3. Resilient-to-Immediate-Replay is a security benefit of challenge-response based schemes where the server issues a new challenge per authentication; thus an attack simply capturing and replaying a static string fails. OTP schemes offer this benefit, since the credentials expire either upon first use or within a short time frame (e.g., 2 minutes), thereby substantially limiting the window for successful replay. Schemes within H2 or H3 in Figure 4.2 offer this benefit.

M4. Resilient-to-Spoofing is a security benefit of schemes that leverage measurement techniques (e.g., hardware or network-based) that are impractical for an attacker to defeat at scale. For example, CPV [2] is *Resilient-to-Spoofing*, since the measurements cannot be manipulated to make an attacker appear to be in a different location (i.e., that of the victim user). Only schemes in H3 of Figure 4.2 offer this benefit.

Table 5.1 cells corresponding to benefits M2, M3, and M4 are populated based on the analysis in Section 4.3.3 (and visualized in Figure 4.2). The following sections evaluate the schemes with respect to the remaining benefits.

5.1.1 Impact of Account Recovery on Security

In practice, authentication schemes are paired with backup mechanisms to help users recover account access if they lose their credentials. The recovery mechanism should not be easier to defeat than the primary scheme, but may sacrifice usability since it is used less frequently.

For conventional password-based authentication, e-mail based password reset is the standard recovery mechanism. Its security relies on the implicit assumption that the user's e-mail account is at least as well-secured as any systems that rely on it for password reset. It is ideally expected that users choose a stronger password for primary e-mail accounts; security-conscious users may also use two-factor authentication.

Table 5.1: Evaluation of geolocation, device fingerprinting, OTP, and PUF schemes as stand-alone authentication schemes and in combination with passwords. Password authentication is included as a baseline; OTP schemes are widely used in combination with passwords; Sound-Proof [99] is a two-factor scheme that also provides some mimicry resistance. • denotes the scheme provides the corresponding benefit (column); ○ denotes partial benefit; an empty cell denotes absence of benefit. ** denotes framework properties introduced herein.

	Scheme	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors (User errors)*	*Easy-Recovery-from-Loss*	***U9. No-False-Rejects (System errors)**	***U10. Easy-to-Change-Credentials**	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Physical-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*	***M1. No-False-Accepts (System errors)**	***M2. Resilient-to-Delayed-Replay**	***M3. Resilient-to-Immediate-Replay**	***M4. Resilient-to-Spoofing**
		Usability										Deployability						Security														
–	P1: Web passwords (PW)			•		•	•	○	•	•	•	•	•	•	•	•	•		○						•	•	•	•	•			
Geolocation	L1: GPS and WPS	•	•	•	•	•	•	•	○	•	•	•	•		•	•	•								•	○	○	○				
	L2: IP-address based tabulation	•	•	•	•	•	•	•	○	•		•	•		•	•	•								•			○		•	○	
	L3: Measurement-based	•	•	•	•	•	•	○	○	•		•	•		•	○	•								•	•		○		•	•	
	L4: Robust location verification	•	•	•	•	•	•	○	○	•		•	•		•		•								•	•		○		•	•	•
Device FP	FP1: System parameters/prefs.	•	•	•	•	•	•	•	•	•		•	•		•	○	•	○	○	•						•		○				
	FP2: Audio+Canvas Chal./Resp.	•	•	•	•	•	•	•	•	•		•	•		•	○	•	○	○	•						•		○		•	•	
	FP3: HW Sensors	•	•	•	•	•	•	•	•	•		•	•		•		•	•	•							•		○				
	FP4: Clock skew	•	•	•	•	•	•	•	•	•		•	•		•		•	•	•							•		○		•	•	•
	FP5: DNS Resolver	•	•	•	•	•	•	•	•	•		•	•		•	○	•	○	○							•		○		•	○	
	FP6: Protocol-based	•	•	•	•	•	•	•	•	•		•	•		•	○	•	○	○							•		○		•		
OTP	OTP1: OTP mobile app	•	•	○		•		•		•	•	○	•		•	•	•	•	•	•	•		•	•		•	•	•	•			
	OTP2: OTP USB token	•	•			•		•		•	•	○			○	•	•	•	•	•	•	•	•	•		•	•	•	•			
	OTP3: SMS OTP	•	•	○		•		•	○	•	•	○			•	•	•	•	○	•	•		•	•		•	•		•	•	•	○
	OTP4: E-Mail OTP	•	•	•		•		•	•	•	•	○	•		•	•	•	•	•	•	•		•	•	•	•	•	○	•	•	•	○
PUFs	PUF1: Strong PUF	•	•		•	•	•	•		•	•	•						•	•	•	•	•	•	•		•		•	•			
	PUF2: Weak PUF	•	•		•	•	•	•		•		•					•	•	•	•	•		•	•					•	•	•	
–	Sound-Proof [99]	•	○	○	•	•	•	•		○	•	•	•		•		•	○		•	•		•	•		•		•	○	•	•	
	Combining passwords (PW) with secondary scheme																															
Geo.+PW	L1: GPS and WPS			•		•	•	○	•	•		•	•		•	•	•		○						•	○	•	○	•			
	L2: IP-address based tabulation			•		•	•	○	•			•	•		•	•	•		○						•		•	○	•	•	○	
	L3: Measurement-based			•		•	○	○	•			•	•		•	○	•		○						•	•	•	○	•	•	•	
	L4: Robust location verification			•		•	○	○	•			•	•		•		•		○						•	•	•	○	•	•	•	•
Device FP+PW	FP1: System parameters/prefs.			•		•	•	○	•			•	•		•	○	•	○	○	•					•	•	•	○	•			
	FP2: Audio+Canvas Chal./Resp.			•		•	•	○	•			•	•		•	○	•	○	○	•					•	•	•	○	•	•	•	
	FP3: HW Sensors			•		•	•	○	•			•	•		•		•	•	•						•	•	•	○	•			
	FP4: Clock skew			•		•	•	○	•			•	•		•		•	•	•						•	•	•	○	•	•	•	•
	FP5: DNS Resolver			•		•	•	○	•			•	•		•	○	•	○	○						•	•	•	○	•	•	○	
	FP6: Protocols			•		•	•	○	•			•	•		•	○	•	○	○						•	•	•	○	•	•		
OTP+PW	OTP1: OTP mobile app			○		•		○		•	•	○	•		•	•	•	•	•	•	•		•	•	•	•	•	•	•			
	OTP2: OTP USB token					•		○		•	•	○			○	•	•	•	•	•	•	•	•	•	•	•	•	•	•			
	OTP3: SMS OTP			○		•		○	○	•	•	○			•	•	•	•	○	•	•		•	•	•	•	•		•	•	•	○
	OTP4: E-Mail OTP			•		•		○	•	•	•	○	•		•	•	•	•	•	•	•		•	•	•	•	•	○	•	•	•	○
PUFs	PUF1 + PW					•	•	○		•	•	•						•	•	•	•	•	•	•	•	•	•	•	•			
	PUF2 + PW					•	•	○		•		•					•	•	•	•	•		•	•	•		•		•	•	•	
–	Sound-Proof + PW [99]			○		•	•	○		○	•	•	•		•		•	○	○	•	•		•	•	•	•	•	•	•	•	•	

E-mail based recovery can also be used for the geolocation and device fingerprinting schemes evaluated herein; alternatives such as SMS-based OTP are also suitable. For example, in a geolocation-based scheme, a user wishing to login from a new location could be e-mailed a link through which they could confirm their new location.

5.1.2 Evaluation of Stand-alone Schemes

Table 5.1 summarizes our evaluation of four main categories of schemes: geolocation, device fingerprinting, OTPs and PUFs as both stand-alone authentication schemes and in combination with passwords, based on the augmented UDS criteria from Section 5.1. Password authentication is included as reference; OTP schemes are included as they are widely used in combination with passwords. While additional schemes could have been included, those selected were chosen as examples to demonstrate the comparative framework. Each row in the table corresponds to an authentication scheme, and each column to a benefit; a cell with a bullet represents a benefit offered by the scheme, an empty circle represents a benefit partially provided, and an empty cell indicates that the benefit is not provided.

Web Passwords

For the new properties, web passwords provide *No-False-Rejects* since a correctly-typed password will never be rejected, *Easy-to-Change-Credentials* since passwords can be easily changed, and *No-False-Accepts* since an exact match is required. However, passwords lack *Resilient-to-Delayed-Replay*, *Resilient-to-Immediate-Replay* and *Resilient-to-Spoofing*, since it is trivial to replay a captured password.

Location-based Schemes

Since L1-L4 in Table 5.1 are invisible to the user, they provide most of the usability benefits. L3 (measurement-based) and L4 (location verification) may sometimes take longer than conventionally considered convenient, thus providing only a partial *Efficient-to-Use* benefit. Additionally, all but L1 (GPS/WPS) may miscalculate the location and falsely reject users in some cases, and therefore do not fulfill *No-False-Rejects*. Although L1 (GPS/WPS) may sometimes result in a small error in location calculation, the calculated location will generally remain in the same city,

and thus gets a bullet. None of L1-L4 are *Easy-to-Change-Credentials*, since changing credentials would require the legitimate user to change their location.

For deployability, all of L1-L4 are *Accessible*; they do not require any explicit user action. They are *Negligible-Cost-Per-User*, since the infrastructure expense is independent from the number of users being served. They lack *Server-Compatible* as they require server-side changes, but are *Browser-Compatible* (no client-side changes needed). L3 is partially *Mature* since there are indications that it is being used in practice [141]; L4 is not *Mature*. Finally, *Non-Proprietary* variations for all of L1-L4 are available.

L1-L4 are largely similar to each other in terms of security properties offered, with the exception of *Replay-Resistance* and *Spoofing-Resistance*. Due to the small guessing space, L1-L4 are not *Resilient-to-Throttled-Guessing-Attempts* and *Resilient-to-Unthrottled-Guessing-Attempts*. Since location information must be stored server-side, they are susceptible to the same exposure threats as passwords, namely *Resilient-to-Physical-Observation*, *Resilient-To-Internal-Observation*, *Resilient-to-Leaks-from-Other-Verifiers*, and *Resilient-to-Phishing*. *No-Trusted-Third-Party* is provided by L3-L4 if the website runs their own geolocation infrastructure (e.g., deploys their own verifier servers), but not by L2 (IP address tabulation) which typically rely on a third-party service provider, and partially by L1 since GPS does not require a third-party service whereas WPS may. None of L1-L4 are fully *No-False-Accepts*, since other users (both legitimate users and attackers) could be residing near the legitimate user, and thus could be indistinguishable to the server. They are not *Resilient-to-Targeted-Impersonation*, since they are susceptible to targeted colocation attacks. They are not *Requiring-Explicit-Consent*, since they are invisible to the user—except for L1, since a GPS/WPS location request may trigger a browser permission prompt to the user, but if the user allows the browser to remember the preference, no future prompts are displayed. They are all partially *Unlinkable*, since leaking server-stored user location information may narrow the search space for linking together multiple accounts across websites (the attacker may still need to collect additional information).

Device fingerprinting

FP1-FP6 deliver most usability benefits, since they are invisible to the user. However, as significant changes in the device configuration (e.g., software or hardware upgrade) may substantially change a device fingerprint, they do not provide *No-False-Rejects*. They are *Easy-Recovery-from-Loss* since e-mail based recovery can be used, as with password authentication. They are not *Easy-to-Change-Credentials*, since changing a device fingerprint may require the user to obtain a different device.

For deployability, device fingerprinting schemes are *Accessible*, since they do not require any explicit user action. They are *Negligible-Cost-Per-User*, since the cost of implementation is essentially independent of the number of users. They are not *Server-Compatible*, as server-side implementation is required, but are *Browser-Compatible*. As device fingerprinting has been used for anti-fraud applications [192, 123], but not widely for user authentication, we consider FP1, FP2, FP5, and FP6 partially *Mature*; FP3 and FP4 are not, as they have been demonstrated academically but are not used in practice to our knowledge. FP1-FP6 are generally available via *Non-Proprietary* implementations.

Many of the security properties are shared across FP1-FP6. They lack *No-False-Accepts*, since users that own identical devices (or share the same device) may be indistinguishable from each other (and for techniques such as clockskew, the overall credential space is not large enough to rule out collisions). They lack *Resilient-to-Unthrottled-Guessing-Attempts*, since none of the data collected thus far to our knowledge indicates that the overall distribution of device fingerprints would offer distinguishability of more than about 30 bits [56, 113, 23, 59, 104, 7]. They lack *Resilient-to-Internal-Observation* and *Resilient-to-Phishing*, since an attacker may collect device information by running their own device fingerprinting scripts via XSS attacks or phishing websites.[1] They lack *Resilient-to-Leaks-from-Other-Verifiers*, since a user's device fingerprint will be similar across websites (varying only based on the particular fingerprinting techniques that each website uses, and the method of storage used), thereby leaking information that can be used to attack users' accounts on different

[1] Collecting device information helps the attacker, but alone is not enough to defeat mimicry-resistant schemes such as L4.

websites. They are each individually partially *Unlinkable* since there may well be multiple users with colliding device fingerprints—the likelihood of this may diminish substantially when combining multiple techniques, however. *Requiring-Explicit-Consent* is not provided, since device fingerprinting is invisible to the user.

The remaining security properties differ across FP1-FP6, as follows. FP3 and FP4 are *Resilient-to-Physical-Observation* and *Resilient-to-Targeted-Impersonation*, since they rely on device-specific manufacturing variations (even across devices of the same model) and therefore can only be determined via measurement; the remaining schemes only partially fulfill these properties since an attacker that visually observes a user's device may obtain or mimic the same device model. FP1 and FP2 have been shown to provide enough distinguishing information to provide *Resilient-to-Throttled-Guessing-Attempts*—to our knowledge the remainder have not, but may collectively provide it if combined together [7].

OTP Schemes

For usability, OTP1-OTP4 are *Memorywise-Effortless* since they do not require the user to memorize anything; they are *Scalable-for-Users* since they allow the user to set up multiple accounts without impacting usability. OTP4 (e-mail) is *Nothing-to-Carry*, since the user just needs to be able to access their e-mail account, whereas OTP1 (mobile app) and OTP3 (SMS) are partially *Nothing-to-Carry*, assuming users are typically in possession of their mobile phones at all times; OTP2 (USB token) does not fulfill this property. All schemes are *Easy-to-Learn*, but *Inefficient-to-Use* since they require the user to type an extra code. They all provide *Infrequent-Errors* since typos are much less likely with a short 6-digit numerical code, and *No-False-Rejects*. OTP1 and OTP2 lack *Easy-Recovery-From-Loss* since the secret is stored on the device/token; OTP3 partially provides it since there is no secret stored on the phone; OTP4 provides it, since a back-up e-mail address can be used. They all offer *No-False-Rejects*, since a valid OTP will not be rejected. *Easy-to-Change-Credentials* is provided in full by all; a user that needs to switch to a new OTP authenticator could, e.g., login with their old authenticator, and then register the new one.

For deployability, all of OTP1-OTP4 are partially *Accessible*, as blind users would require screen reading software to read the OTP code. Only OTP1 and OTP4 are

Negligible-Cost-per-User, since SMS incurs a per-message cost to the server,[2] and USB tokens have a per-user cost. All schemes except for OTP2 (USB tokens) are *Browser-Compatible*; in addition to requiring browser support (at this time, only Google Chrome supports FIDO U2F), OTP2 tokens require a hardware interface (e.g., USB-A, USB-C, NFC) that is compatible with the user's devices—while a variety of hardware tokens are available, they typically only possess one or two interfaces. All of OTP1-OTP4 are *Mature* and with *Non-Proprietary* implementations available.

For security, all of OTP1-OTP4 are *No-False-Accepts*, *Resilient-to-Physical-Observation*, *Resilient-to-Throttled-Guessing*, and *Resilient-to-Unthrottled-Guessing*. OTP1-OTP2 and OTP4 are *Resilient-to-Targeted-Impersonation*, but OTP3 only partially provides this benefit—attackers may transfer the victim's phone number to a new device via a social engineering attack on the user's mobile operator [161]; the ease of conducting such an attack is subject to the security measures put in place by the mobile operator. They are *Resilient-to-Leaks-from-Other-Verifiers* and *Resilient-to-Phishing*. Only OTP2 (USB tokens) is *Resilient-to-Internal-Observation*—OTP1 is susceptible to theft of the shared seed via malware, and OTP3/OTP4 are susceptible to malware- and network-based [69] capture attacks.[3] Only the e-mail mechanism (OTP4) is *Resilient-to-Physical-Theft*. They all provide *No-Trusted-Third-Party* and *Requiring-Explicit-User-Consent*. OTP1 and OTP2 are *Unlinkable* since a unique key is used for each website; phone numbers (OTP3) are linkable across sites; e-mail addresses (OTP4) are partially *Unlinkable*, since users may freely create different e-mail aliases.

PUFs

For usability, PUF1 and PUF2 lack *Nothing-to-Carry* and *Easy-Recovery-From-Loss*, since they are tied to the device. PUF1 is *Easy-to-Change-Credentials* since virtually endless challenge/response pairs can be generated; PUF2 is not, since the hardware

[2]Bulk SMS rates can be on the order of a penny each (varying by country). The total cost will vary based on whether the user base is small or in the millions, and on average login frequency.

[3]Since the UDS framework combines both network- and device-based eavesdropping/interception into a single property (*Resilient-to-Internal-Observation*), OTP1 and OTP3-OTP4 share the same y-coordinate in Figure 4.2, though OTP3 should be lower since it appears to be susceptible to a larger subset of network-based interception attacks [172].

needs replacement if subjected to a model-building attack. All other usability properties are fulfilled, since no user effort is required.

For deployability, PUF1 and PUF2 are not *Negligible-Cost-per-User*, since hardware would need to be deployed for each user, and they are not *Server-Compatible*, *Browser-Compatible*, or *Mature*. *Non-Proprietary* designs for PUF2 are available [209], but not for PUF1 (to our knowledge).

For security, assuming a large enough space of challenge-response pairs, PUF1 and PUF2 fulfill *No-False-Accepts*, *Resilient-to-Throttled-Guessing*, and *Resilient-to-Unthrottled-Guessing*. PUF1 provides *Resilient-to-Internal-Observation* and *Resilient-to-Phishing* since challenge/response pairs can be exposed without consequence; PUF2 does not provide the former, since it is susceptible to model-building attacks, but it provides the latter since it only responds to challenges originating from a verifier server. Both are *Resilient-to-Leaks-from-Other-Verifiers*. PUF1 and PUF2 are neither *Resilient-to-Physical-Theft* nor *Requiring-Explicit-Consent*. PUF1 has *No-Trusted-Third-Party* and *Unlinkable* since every website can store a separate set of challenge-response pairs, but PUF2 lacks these properties since it typically requires a single server to verify challenge-response pairs (see Section 4.3.3).

Sound-Proof

For usability, Sound-Proof provides *Memorywise-Effortless* since there is no secret for the user to memorize. It is partially *Scalable-for-Users*, since only a single device is required, but it appears that under the current architecture each service requires a separate app. It partially provides *Nothing-to-Carry*, since users with a smartphone will typically carry it with them at all times. It is *Easy-to-Learn*, *Efficient-to-Use*, and *Infrequent-Errors* as (aside from installing a smartphone app at setup time) it does not require any user effort to use. It partially provides *No-False-Rejects* and *No-False-Accepts*—the false accept and false reject rates are tunable via a threshold value, and the authors show that the intersection between the two (the equal error rate) is about 0.02%. It lacks *Easy-Recovery-From-Loss*, as the consequence of losing the phone is similar to that of OTP1. For deployability, it lacks *Server-Compatible* and *Mature*.

For security, Sound-Proof partially provides *Resilient-to-Physical-Observation*,

since being able to record sound in the vicinity of the user can break the scheme (it also lacks *Resilient-to-Targeted-Impersonation*, for this reason). It is *Resilient-to-Throttled-Guessing-Attempts* and *Resilient-to-Unthrottled-Guessing-Attempts*. It is not *Resilient-to-Internal-Observation* or *Resilient-to-Theft*, since the scheme can be broken by malware on the phone or by device theft. It is *Resilient-to-Leaks-From-Other-Verifiers* (since there is no static secret to be stored), *Resilient-to-Phishing*, and *No-Trusted-Third-Party*. It is not *Requiring-Explicit-Consent* since it requires no user action. It is *Unlinkable*, since different services can use their own apps on the user's smartphone.

5.2 Evaluation of Combined Schemes

The evaluation above naturally suggests that schemes offering some degree of mimicry resistance may complement passwords by adding a new security dimension; and invisible schemes such as device fingerprinting and geolocation do so without further burdening users. None of the latter (invisible) schemes, however, seem suitable as a sole mechanism for *user-to-web* authentication because either the invisibility aspect makes them limited to *device-to-web* authentication (see Section 4.2), or their security space is so small that they lack the ability to uniquely identify users (e.g., suffer from false accepts). This motivates exploring the resultant benefits when these schemes are combined with passwords. This is also useful since some password usability drawbacks might be ameliorated in the medium to long-term by use of browser-based and/or stand-alone password managers.

When combining two authentication schemes, both sets of credentials should be correct for the user to gain access. This then strengthens security. Ideally, the implementation should limit any partial feedback, which might be beneficial to an attacker using a divide-and-conquer strategy to independently defeat each scheme individually. When password authentication is combined with an invisible scheme, if the correct password is provided but the supporting invisible scheme fails, the server can immediately fall back to a backup scheme such as e-mail OTP. Otherwise, allowing retry authentication attempts benefits attackers, but not legitimate users (except in cases of system error where a retry might succeed, since invisible credentials are not entered by the user). In the event of an attack, this serves to notify the user that

their password has been compromised; otherwise, it gives the user an opportunity to reset their supporting authentication mechanism (e.g., by registering a new location or new device).

In Table 5.1, a combined authentication scheme as described above only inherits the intersection of the usability and deployability benefits, and the union of the security benefits—except for *No-Trusted-Third-Party* and *Unlinkable*.

The bottom half of Table 5.1 evaluates the schemes from the top half when used in combination with passwords. The evaluation suggests that either geolocation or device fingerprinting can be combined with passwords to improve security properties, namely *Resilient-to-Physical-Observation*, *Resilient-to-Throttled-Guessing*, *Resilient-to-Delayed-Replay*, *Resilient-to-Immediate-Replay*, and/or *Resilient-to-Spoofing*. To further improve security, geolocation can be combined with one or more device fingerprinting schemes. While combining schemes, select those that maximize the resulting overall benefits, e.g., combining L4 with FP2 and/or FP4. As a downside, this would increase the probability of false rejects, assuming the failure of any scheme causes authentication failure. PUFs provide many security benefits, suggesting that they could be sufficient as a sole authentication mechanism. However, their major drawback is that they lack most deployability benefits (in the web authentication context).

The usability drawbacks observed for the combined schemes in Table 5.1 are in *Efficient-to-Use* for schemes that introduce a perceptible delay in authenticating the user, *No-False-Rejects* primarily due to variations in measurement or fuzzy matching functions, and *Easy-to-Change*. The deployability drawbacks are in *Server-Compatible* and *Mature*, for schemes that have not yet been widely used in practice.

5.3 Concluding Remarks and Recommendations

The primary focus of this work is to provide the first systematic and in-depth treatment of mimicry resistance as an additional dimension of security for *user-to-web* and *device-to-web* authentication schemes, which has received virtually no scrutiny whatsoever. Segmenting the mimicry-resistance dimension into levels of replayability and spoofability helps differentiate the ability of various schemes to resist mimicry attacks. Evaluating schemes using this new dimension, we found that most web

authentication schemes have limited to no mimicry resistance as they rely fundamentally on the knowledge of some secret that (once exposed) can be directly used by the attacker for fraudulent authentication. As validation for the usefulness of this work, we make the argument that the recommendations and insights offered herein were only possible from consideration of this new dimension that we capture in our comparative evaluation framework. Moreover, our work highlights the advantages of utilizing mimicry-resistant techniques in web authentication, motivating further exploration into invisible and mimicry-resistant techniques, and providing directions on how to explore and evaluate the ability of schemes to resist scalable attacks. Below, we summarize the most significant insights and recommendations drawn from this work.

Benefits of Employing Mimicry-Resistant and Invisible Schemes. Our evaluation demonstrates that there are security advantages in augmenting *user-to-web* authentication schemes with *device-to-web* schemes (especially *device-to-web* schemes with some degree of mimicry resistance). The resulting combination offers greater security, with minimal impact on usability if the scheme is also invisible. In multi-factor authentication, combining highly usable schemes often enhances the resultant (combined) benefits even when each individual scheme offers relatively few additional security benefits (i.e., security benefits from Table 5.1 on p. 86 that evaluate resilience against various exposure and mimicry attacks); this is because security benefits are often additive (complementary). Among the new benefits defined in our evaluation framework to incorporate different properties associated with mimicry resistance, benefits M2, M3, and M4 in Table 5.1 directly reflect the level of effort that an attacker must expend to defeat an authentication scheme after the exposure of a secret.

Among the schemes evaluated in Table 5.1, we find that mimicry resistance is offered by some variations of device fingerprinting, several OTP-based schemes, robust Internet geolocation methods, and certain variations of Physically Unclonable Functions (PUFs). A challenging obstacle in reinforcing password authentication with additional schemes is doing so while minimizing or entirely avoiding deployability or usability penalties [86]. As such, our framework also includes two new usability benefits (U9: *No-False-Rejects* and U10: *Easy-to-Change-Credentials* in Table 5.1)

relevant to *device-to-web* and mimicry-resistant schemes. Geolocation and device fingerprinting in particular offer substantial deployability advantages, and are highly usable since they are *invisible* in that they require no user involvement, both for setup (i.e., registration) and for login. We therefore recommend augmenting password authentication with one or more schemes that are both invisible and offer mimicry resistance—among the schemes evaluated in Table 5.1, L3, L4, FP2, FP4, FP5, and FP6 are strong contenders.

Limiting Partial Feedback with Invisible Schemes. As discussed in Section 5.2, we recommend that when password authentication (or an alternative primary authentication scheme such as OTP) is augmented with an invisible scheme, any partial feedback should be limited upon failed login attempts. Since invisible schemes are *device-to-web* by definition, the credentials are provided by the device and are therefore not prone to user error such as typos. Successful validation of a password or OTP, but failed validation of an invisible scheme such as device fingerprinting, indicates either system error or an attacker that has broken the primary authentication scheme. In either case, the website should immediately revert to a fallback authentication scheme (whose security is no weaker than the primary scheme, as per p. 97).

Attack Scalability as a Criteria for Scheme Selection. Plotting authentication schemes along both exposure-resistance and mimicry-resistance dimensions (Figure 4.2 on p. 66) gives a novel representation of attack scalability, where a scheme's ability to resist scalable attacks is conveyed by its distance from the origin. The new framework thus allows for a more comprehensive evaluation of web authentication schemes by their ability to resist both exposure and mimicry, and helps visualize relative resistance to scalable attacks, represented as a function of both components. We recommend that attack scalability (i.e., the difficulty of scaling an attack to break a large number of accounts) be used as a key criteria by websites for evaluating the security offered by authentication schemes, since we believe it best reflects the overall resilience of web accounts to compromise. In contrast, evaluating a scheme's resilience to targeted attacks (i.e., targeting one or a few individuals, as opposed to a large number of accounts as aforementioned) requires careful consideration of numerous specific scenarios that depend on each individual user; for example, despite

the fact that schemes relying on a hardware token are generally considered to be stronger than password authentication, and despite the fact that stealing large numbers of hardware tokens is a highly unscalable attack, an individual user's hardware token may still be easily stolen by a malicious roommate in a highly targeted attack. Therefore, determining appropriate defenses against targeted attacks depends on the circumstances and needs of each individual user, and recommendations should be tailored accordingly (beyond the scope of the recommendations herein).

Selection of Fallback Authentication Schemes. While multi-factor authentication typically requires all factors to pass in order to grant access, account recovery schemes provide an alternative authentication path that attackers can exploit to bypass the primary authentication mechanism. Therefore, while combining multiple factors strengthens security, we recommend that the security level of any alternative attack path (e.g., account recovery mechanisms) be no weaker than the primary scheme. Figure 4.2 is a helpful tool for choosing a fallback authentication scheme, as schemes plotted close to each other offer similar resilience to scalable attacks. Further insight on fallback schemes is offered in the discussion on FIDO UAF in Section 6.5.

Future Directions for Developing New Mimicry-Resistant Schemes. An interesting avenue for future work (motivated by our discussion on Robust Location Verification and Sound-Proof in Section 4.3.3) is to identify and explore further measurement-based techniques that require no user effort while providing mimicry resistance when a trusted *device-to-web* channel can be established, e.g., by means of a TPM. In particular, some authentication schemes that offer mimicry resistance for *user-to-device* authentication (see Khan et al. [101]) may be augmented with TPM (by providing cryptographically-signed touchscreen sensor readings for, e.g., a swipe gesture recognition scheme) to offer mimicry resistance for *user-to-web* authentication as well. Moreover, an example scheme analogous to FIDO UAF (see Section 6.1.1) wherein each device is provisioned with a manufacturer-specific attestation key and can generate on-device signing keys (in this case, for signing sensor readings as aforementioned) could potentially occupy the top-right corner of Figure 4.2, as it would require the attacker to both steal the user's device and mimic the user. Note that this

example scheme would be a *user-to-web* scheme (and *not* a two-stage scheme involving a *user-to-device* and *device-to-web* component), since the user input is validated by the server, with the TPM functioning to defend against forged input.

Chapter 6

Comparative Analysis of SSO Schemes

Single Sign-On (SSO) systems have the potential to strengthen web authentication while largely retaining the benefits of conventional password-based authentication. We analyze in depth the state-of-the art of web SSO through the lens of usability, deployability, security, and privacy, to construct a taxonomy and evaluation framework based on different SSO architecture design properties and their associated benefits and drawbacks.

SSO is an umbrella term for schemes that allow users to rely on a single master credential to access a multitude of online accounts. We categorize SSO schemes across two broad categories: federated identity systems (FIS) and credential managers (CM). FIS establishes a means of communicating user identity across administrative domains; this allows users to authenticate to an Identity Provider (IdP) that can communicate proofs of user identity to Service Providers[1] (SPs), thereby granting users access to SP services without needing to re-authenticate. CM-based SSO stores SP-specific credentials (e.g., passwords or cryptographic keys) and automatically uses them to authenticate to SPs on behalf of users; CMs are typically protected by a single master credential such as a password or a hardware token containing a cryptographic key (e.g., smart card). We classify both FIS- and CM-based SSO schemes into more granular subcategories, while identifying benefits and drawbacks associated with different approaches. While SSO has long been used in enterprise networks to enable users to access network services and applications with a single set of credentials (e.g., Kerberos [143]), we focus specifically on SSO designed for web authentication.

Our primary contributions in this chapter are summarized as follows:

1. While prior work has explored various aspects of SSO security, this is to our knowledge the first to perform a comprehensive analysis and comparison of a broad range of SSO systems proposed and/or deployed within the last decade,

[1] Often also referred to as Relying Parties (RPs).

including two hardware-based SSO schemes that are undergoing large-scale deployment, FIDO UAF [117] (as of this writing, currently being standardized as the W3C Web Authentication API [22]) and Mobile Connect [79].

2. We identify different design properties of SSO schemes and develop a taxonomy built on a categorization scheme across the design properties. We also develop a corresponding evaluation framework that highlights benefits and drawbacks of SSO schemes based on their design properties, and analyze a representative subset of 14 SSO schemes under this framework.

3. We identify trade-offs between different design goals, by identifying how various SSO schemes can be augmented with existing techniques to achieve specific benefits while forgoing others. Such trade-offs allow SSO schemes to be tailored to different needs and scenarios.

The remainder of this chapter is organized as follows. Section 6.1 provides an overview of the SSO protocols evaluated. Section 6.2 presents the new classification and evaluation framework, and Section 6.3 provides a detailed evaluation of each scheme. Section 6.4 discusses insights from the evaluation, and Section 6.5 concludes and provides recommendations.

6.1 Overview of SSO Protocols

To give technical context for our comparative analysis, we give a simplified overview of the schemes analyzed. Pointers to in-depth overviews are referred to by citations herein.

6.1.1 Credential Managers

As cited in Section 2.4.1, CM-based schemes seem to gain wider acceptance from users when they are already built into their browser or OS. This motivates our analysis of Firefox Sync [134] (password-based CM) and FIDO UAF [117] (public-key based CM). We also analyze Impostor [156], since its properties help demonstrate our taxonomy.

Firefox Sync

The Firefox password manager saves user-created passwords (an updated interface will offer a random password generator [133]) when typed into a website, and offers to automatically enter them on subsequent visits. Users may also manually view their saved passwords through a graphical interface. The password database is stored locally in a file, which the Firefox Sync [134] protocol synchronizes across users' devices. Below, we give a simplified overview of two versions of this protocol.

Firefox Sync 1.5 locally generates a symmetric key to encrypt the user password database. The key is stored locally, and the encrypted password database is uploaded to the user's Sync account, accessed with a user name and password that is only typed in once per device during the initial Sync setup [205]. After setting up Sync on one device, the symmetric key can be transferred to another user device over a secure connection initiated via a "pairing" process that first establishes a shared session key. The pairing process uses J-PAKE [83] to display a short code on one device, which the user types into the second device. User devices remain synchronized by downloading the latest copy of the encrypted password database from the Sync server, and uploading any updated contents if necessary. Sync 1.5 was superseded by Sync 2.0, since many users did not understand the purpose of the pairing process, and expected to access their password database from any device using only their Sync password. Users were generally unaware of the existence of the encryption key and of the option to print it out for backup purposes, and consequently many users lost access to their passwords if they only owned a single device that they replaced without first backing up their key.

Firefox Sync 2.0 [206] locally derives two symmetric keys from the user's Sync password using PBKDF2 [98] and HKDF [107]. One key is used for authenticating to the user's Sync account (thereby not revealing the Sync password to the server), and the second key is used to encrypt the password database before uploading it to the server. This eliminates the need for device pairing; setting up a new device only requires the user to enter their password during Sync setup. Although iterated hashing is used both on the client side (to generate the authentication key) and on the server side (for storing a hashed version of the authentication key) to slow down offline attacks, the strength of the encryption key is still dependent on the user-chosen Sync

password (in contrast to Sync 1.5, where the encryption key is randomly generated).

By default, both versions of Firefox Sync store passwords unencrypted on the client system. Users may optionally select an offline master password (separate from the synchronization password), from which a key is derived to encrypt the password database on-disk. Chrome Sync [71], which employs a synchronization protocol similar to that of Firefox Sync 2.0, uses the OS's built-in credentials manager (which typically encrypts its contents with users' login passwords) to store passwords client-side.

FIDO UAF

FIDO UAF [117] (Universal Authentication Framework), currently being standardized as the W3C Web Authentication API [22], uses client-side UAF-enabled software or hardware authenticators to authenticate users to SPs via public-key cryptography. UAF authenticators are typically hardware modules built into end-user devices, but they may also be software-based. When users register a UAF device with an SP account (each device must be individually registered with each SP), the UAF client generates an SP-specific asymmetric key pair (private keys are stored on-device; corresponding public key certificates, which are either self-signed or signed using an attestation key discussed below, are sent to SPs). Local authentication, such as a PIN or biometric, is used to "unlock" the FIDO authenticator and initiate the cryptographic challenge-response protocol to authenticate users to SPs.

Hardware-based UAF authenticators with secure key storage capabilities can be certified by the FIDO Alliance and provisioned with a signed attestation certificate containing metadata about the device (e.g., device manufacturer, method of local user authentication used). SPs can validate the certificate for a higher degree of user identity assurance, since hardware-based UAF clients can protect against key theft to a much higher degree than software-based UAF clients. Attestation certificates can be revoked in the event that vulnerabilities are found (e.g., which may allow the extraction of private keys from the device) in a hardware UAF client, to aid SPs in phasing out support for vulnerable hardware.

Impostor

Impostor [156] is a proxy-based CM that stores users' passwords on a remote IdP. Passwords are automatically submitted to SPs that the user visits through the IdP proxy. Its design goal was to provide a means of using an untrusted machine without exposing any long-term secret to it. Therefore, the authors suggest that the user-to-IdP authentication be done through a challenge-response based scheme, e.g., a hardware one-time-password (OTP) token (which we assume in our analysis).

6.1.2 Federated Identity Systems

An identity federation consists of one or more member IdPs and one or more member SPs such that users can authenticate to member SPs through member IdPs. The authentication process consists of a user-to-IdP authentication stage, followed by an IdP-to-SP identity assertion to convey the user's identity information (Section 6.2.3 outlines different methods for doing so) and assure the SP that the user has been authenticated by the IdP. FIS protocols can have proprietary or open specifications—all protocols discussed herein have open specifications. Some schemes that we evaluate are protocols (such as OpenID 2.0, OAuth 2.0, and OpenID Connect), and others are specific implementations of protocols, such as Mobile Connect (a proprietary implementation of OpenID Connect) and Shibboleth (a free implementation of SAML2). The set of schemes we evaluate were selected to best highlight the features of our taxonomy and evaluation framework.

Different organizations using the same protocol are not necessarily part of the same federation. Moreover, both IdPs and SPs may support SSO authentication via multiple protocols and be a member of multiple federations. Section 6.2.2 discusses different types of federations based on how IdP-SP associations are established.

Shibboleth

Shibboleth [175] is a popular open-source FIS implementation based on SAML [149], a highly flexible and extensible XML-based standard for exchanging identity information between federation members. SAML's high flexibility requires many protocol

message format details to be agreed upon between federation members, thereby reducing deployment scalability [187]. A high-level overview of the protocol is as follows:

1. The user visits an SP, and clicks the login link to initiate the authentication process.

2. A discovery process (see Section 6.2.2) takes place to discover the user's IdP, e.g., by presenting a list of supported IdPs to the user.

3. The user is redirected to their IdP. If an authenticated session does not already exist with the IdP, the user logs in.

4. The IdP generates an authentication response and attaches it to an HTTP POST request when redirecting the user back to the SP (while the authenticity and integrity of the authentication response is protected by TLS, it may also be signed by the IdP and encrypted with the SP's key as an additional layer of protection [176]).

5. The SP receives and validates the authentication response, and creates an authenticated session for the user.

OpenID 2.0

The OpenID 1.x [164] and 2.0 [163] family of specifications were designed to be a much simpler (but also less flexible) FIS scheme compared to SAML. The simpler protocol message format enables OpenID-based IdPs and SPs to communicate without requiring prior agreement on a large set of protocol parameters as was the case with SAML. The design philosophy of OpenID was to allow any domain owner to set up an IdP (users can therefore set up their own IdP if desired) and provide services to any SP without prior coordination. The OpenID user ID format is a URL (or XRI [48]) of a user profile page; the page contains metadata that points SPs to the IdP URL to which the user should be redirected for authentication. The high-level protocol flow is similar to that of Shibboleth as described above, but the IdP discovery step consists of retrieving the metadata from the user's supplied profile URL. OpenID SPs experimented [191, 169] with various user interface designs for obtaining users' OpenID identifiers—generally, users would select their IdP from a list of logos

of the most popular IdPs, and subsequently type in their user name (the SP could then determine the correct IdP URL corresponding to the user name); more advanced users could pick the option to manually enter their own URL instead (e.g., if they host their own IdP or use a lesser-known IdP).

OAuth 2.0

OAuth 1.0 [11] and 2.0 [84] enable users to authorize web applications to retrieve resources (e.g., photos or documents) from or perform actions (e.g., upload a new document) on a user account on a resource server without revealing their account password to web applications. However, OAuth can also be used as a FIS if the role of the resource server is to store identity information. A high-level overview of OAuth 2.0 as an authorization protocol is as follows:

1. The user visits an SP and initiates the authentication process, typically by clicking on a button from among a list of supported IdPs.

2. The user is redirected to the IdP for authentication, and authenticates to the IdP.

3. The IdP presents the user with a list of resources that the SP has requested permission to access. At the minimum, the SP requires access to the user's identity information. However, the SP may request additional permissions (e.g., accessing the user's contact list or permission to make social media posts on the user's behalf).

4. If the user approves the permissions requested by the SP, the IdP generates an authorization code and encodes it as a URI parameter when redirecting the user back to the SP.

5. The SP submits the authorization code to the IdP in exchange for an access token. Access tokens may be short-lived, when used by an SP to only verify a user's identity and establish an authenticated session, but may also be long-lived if SPs request persistent access to perform actions on the user's IdP account (e.g., periodically obtain an updated copy of the user's social media contacts) even while the user is not logged into the SP.

6. The SP queries the IdP to obtain a user identifier (e.g., e-mail address) that is associated with the access token; the SP then creates an authenticated session (i.e., initializes the server-side session, and provides a corresponding session cookie to the browser) for the account corresponding to the user identifier.

Unlike OpenID, OAuth requires SPs to register out-of-band with IdPs they wish to support. The registration typically requires the SP administrator to submit an online form on the IdP website and establish a shared secret (typically a randomly-generated string) for the SP to include when making API calls to the IdP. The primary purpose of the shared secret is to prevent unauthorized use of access tokens (in case of token theft), which provide access to user information and resources (albeit limited by the permissions granted by the user in step 3 above) stored by the IdP. OAuth 2.0 relies on TLS to protect the secrecy of the shared secret.

OpenID Connect

OpenID Connect [170] defines a standardized mechanism for exchanging identity information over OAuth 2.0. The high-level protocol overview is as described above for OAuth 2.0, except that to identify the user in the final step, the SP requests an ID token (in a standard JSON Web Token [97] format) from the IdP. The more strictly-defined protocol message format, compared to OAuth 2.0, facilitates building code libraries for SPs that can interoperate with multiple IdPs. OpenID Connect also includes a number of optional (but rarely used) features such as *dynamic client registration*, which allows SPs to automatically register with IdPs instead of going through an out-of-band process; this is intended to allow OpenID Connect to operate more similarly to OpenID 2.0, i.e., to allow any SP to request user authentication services from any IdP, without any prior coordination. OpenID Connect also defines a framework for establishing federated relationships; this allows co-operating IdPs to form a federation, and allows SPs to register with the federation (instead of with each individual IdP) to gain access to the services of all member IdPs.

Mobile Connect

Mobile Connect [79] is an OpenID Connect federation operated by the GSM Association; its IdPs consist of mobile network operators (MNOs) worldwide, and users

authenticate to IdPs using their mobile phones as hardware authenticator tokens. Mobile Connect redefines several optional API parameters as mandatory [78], such as the nonce and state parameters used for binding an access token to an HTTP session; the acr_values parameter is used by SPs to communicate a required Level of Assurance (LoA) to IdPs for user authentication on a 4-point scale, as defined by ISO/IEC and ITU-T [64, 93]:

- LoA1: Minimal confidence that user's identity is consistent across multiple authenticated sessions (e.g., using a device's MAC address). Not applicable to Mobile Connect.
- LoA2: Some confidence in user's asserted identity. Requires at least one authentication factor; typically implemented by Mobile Connect IdPs by sending an SMS one-time password (OTP) to their mobile phone (to verify possession), which the user types into the device (e.g., computer) they are authenticating from.
- LoA3: High confidence in user's asserted identity. Requires at least two authentication factors; typically implemented by Mobile Connect IdPs by requiring a user-to-device authentication mechanism (on the mobile device) such as a 4-digit PIN or a biometric before displaying the OTP that the user needs to type in.
- LoA4: Similar to LoA3, but requires in-person identity proofing (i.e., the user's online identity is associated with a real-world individual). Not currently supported by Mobile Connect.

Mozilla Persona

Mozilla Persona [82] was[2] designed to enable users to tie their online identities to their e-mail addresses, and for e-mail providers to fill the role of IdPs for their users. Through public-key cryptography, Persona IdPs delegate to users' browsers the responsibility of generating and sending identity assertions to SPs; this has the privacy

[2]Due to limited adoption, Mozilla discontinued internal development of the project in 2014 [34]. Nevertheless, it remains an interesting protocol to study due to its unique approach and the interest it received in the security community.

benefit that IdPs do not learn of the SPs that users visit. Similarly to OpenID 2.0, automatic discovery (i.e., a protocol mechanism allowing for SPs to request service from IdPs without any prerequisite human intervention such as out-of-band key establishment) is supported, allowing any Persona SP to request user authentication services from any Persona IdP. A high-level protocol overview is as follows:

1. The user visits an SP website and clicks on a "Log in with Persona" button, which pops up a new window in which the user enters their e-mail address.

2. The browser (using client-side JavaScript) determines whether the user's e-mail provider supports Persona by checking for the presence of a browserid file accessible from the e-mail provider's domain, at the location https://idp.domain/.well-known/browserid. This file includes the IdP's public key used for signing certificates, and the URL at which the IdP's users should be redirected for authentication.

3. The user is redirected to the URL specified by the browserid file, and authenticates to their IdP.

4. The browser (using client-side JavaScript) generates a public-private key pair and sends the public key to the IdP. The IdP returns a signed *user certificate*, containing the browser's public key.

5. The browser generates and signs an identity assertion for the SP that the user wishes to authenticate to. The signed assertion is sent to the SP, along with the user certificate from the previous step.

6. The SP verifies the IdP's signature (the IdP's public key can be obtained from the browerid file discussed in step 2) on the user certificate, and subsequently the browser's signature on the identity assertion.

7. The SP initiates an authenticated session for the user.

Due to limited adoption of Persona by e-mail providers, Mozilla introduced a *fallback IdP* to authenticate users whose e-mail providers did not support Persona. Users could create a Persona account on the Mozilla fallback IdP, by using their e-mail address as their user name. Upon account creation, the fallback IdP e-mails the

user a verification URL, which the user clicks to verify that they control the e-mail address. Creating the Persona account requires users to select a password, to be used for authentication in step 3 above, when the user is forwarded from an SP to the Mozilla fallback IdP URL for authentication.

For some e-mail providers that support OAuth 2.0, the Mozilla fallback IdP acted as an OAuth 2.0 bridge: instead of the user having to create a password for their Persona account and verifying possession of their e-mail address through a verification URL, users were instead redirected to their e-mail provider to authenticate via OAuth 2.0. In effect, the Mozilla fallback IdP acts as an SP to request an OAuth access token (see Section 6.1.2) from the user's e-mail provider. The fallback IdP then uses the access token to obtain the user's e-mail address from the e-mail provider, and compares it with the e-mail address provided by the user to verify the user's possession of the e-mail address. The benefit of this approach (compared to the default behaviour of the fallback IdP as described above) is that the user does not have to create a new password for use with Persona.

SecureKey Concierge

SecureKey Concierge [179] is a privacy-focused SSO system that enables IdPs to provide user authentication to SPs through an intermediary service that performs triple blinding: SPs are blinded from users' selected IdPs, IdPs are blinded from the SPs that users access, and SecureKey is blinded from any personally-identifying user information. SecureKey is used by various online services offered by the Government of Canada, and the approved list of IdPs currently consists of Canadian banking institutions. A high-level protocol overview is as follows:

1. The user visits an SP and initiates the authentication process by clicking on a link to authenticate through SecureKey Concierge.

2. The user is prompted to choose from a list of IdPs approved by SecureKey.

3. SecureKey forwards the user to their selected IdP for authentication.

4. Upon successfully authenticating the user, the IdP generates a meaningless-but-unique identifier (MBUN) for the user. The user is then redirected back to

SecureKey, along with the MBUN. The MBUN blinds SecureKey from users' real-world identities.

5. SecureKey internally maps the MBUN to an internal Persistent Anonymous Identifier (iPAI). The iPAI is used to generate an RP-specific (i.e., SP-specific) Persistent Anonymous Identifier (rpPAI) to forward back to the SP. Generating a unique rpPAI for each SP prevents colluding SPs from correlating the user's identity across accounts.

6. SecureKey redirects the user back to the SP, along with the rpPAI.

7. The SP initiates an authenticated session for the user.

The protocol flow as described above includes two authentication flows that are chained together:

1. SecureKey acts as an IdP when interacting (over either Shibboleth or OpenID Connect) with SPs. The SP forwards users to SecureKey for authentication, and when the process is completed the SP only receives an rpPAI in an authentication response message signed by SecureKey.

2. SecureKey acts as an SP when interacting (over Shibboleth) with the user's selected IdP: SecureKey forwards users to their selected IdP for authentication, and receives a signed authentication response containing the MBUN.

SAW

SAW [195] leverages e-mail addresses and the existing SMTP e-mail system to build a FIS. The protocol overview is as follows:

1. The user visits an SP and types in their e-mail address to initiate authentication.

2. The SP sends the user an e-mail containing a verification link to complete the authentication process. The link contains a string that combines a string generated by the SP with a string generated by the user's browser, thereby binding the verification link to the user's session.

3. The user clicks the link, and the SP initiates an authenticated session.

6.2 Classification and Evaluation Framework

To evaluate the SSO schemes discussed in Section 6.1, we first define 14 benefits (Section 6.2.1) related to usability, deployability, security, and privacy. We then classify schemes across 5 design properties (defined in Sections 6.2.2 through 6.2.6), together forming a taxonomy for SSO schemes; each of these sections define categories associated with the respective property, and conclude with an assessment of the benefits that can or cannot be provided by schemes in each category.

6.2.1 Benefits Provided

Below, we define 14 benefits relating to usability, deployability, security, and privacy aspects of SSO systems.

Usability

B1: Portable-Identity-Across-IdP. Users can change their IdP without updating their SP accounts (i.e., without having to reconfigure each of their SP accounts to point to their new IdP). This is a usability benefit, as users may wish to change their IdP due to, e.g., changes in policy or pricing from their existing IdP, wanting to benefit from features offered by another IdP, replacing a hardware authenticator, or wanting to host their own IdP. This benefit is partially provided by schemes where the user can only change to another vetted IdP (see B5: *No-IdP-Vetting* benefit) within the same federation.

B2: No-Device-Setup. No software or hardware configuration (e.g., generating or transferring cryptographic keys) is required by the user when authenticating from a new device.

B3: No-Hardware-Token-Required. Users do not need to carry around a hardware authenticator token. Schemes that restrict users to a single hardware authenticator, such as a mobile phone, lack this benefit. Schemes that allow IdPs to specify their own authentication mechanism (see Section 6.2.5 for more detailed discussion on user-to-IdP authentication) are assumed here to use conventional password-based authentication (unless stated otherwise), since this is the widespread

practice.

B4: Resilient-to-Temporary-Service-Outage. Users can continue to authenticate to SPs even during a temporary outage of a remote server providing an SSO service. As discussed further in this section, the type of SSO scheme determines the remotely-hosted service being relied upon, e.g., an IdP, discovery server, or synchronization server. This is also a deployability benefit, since it reduces the risk for SPs that they will lose user traffic in the event of an IdP outage.

Deployability

B5: No-IdP-Vetting. IdPs do not need to register with a federation operator, thereby facilitating deployment. This also provides users with a wider range of IdP choices, and even allows them to host their own IdP.

B6: No-SP-Sponsoring. SPs do not need to manually register with each individual IdP that they wish to support, thereby facilitating deployment.

B7: No-SP-Stored-User-Secret. The SP does not need to store any user secret, thereby facilitating SP implementation and eliminating the possibility of leaking secret credentials in the event of an SP server compromise.

Security

The following three properties (B8a, B8b, and B8c) reflect whether the ability of attackers to extract information from three different entities (user device, remote IdP, and SP, respectively) can be leveraged to impersonate the user. We exclude *active* attacks, i.e., involving attackers with persistent and full control over a user device, since this is an extremely challenging threat model for any authentication scheme to defend against. We also exclude session hijacking from our analysis, since SSO does not offer any inherent defense against such attacks—instead, defense against session hijacking can be offered by complementary mechanisms such as token binding (cf. Section 2.4).

B8a: Resilient-to-Client-Leaks. Attackers cannot defeat user authentication

by extracting (e.g., via malware) data from users' access devices, such as keystrokes or data stored on disk or memory.

B8b: Resilient-to-SP-Leaks. Attackers cannot defeat user authentication by extracting (e.g., via server-side software vulnerabilities) user-specific data from the SP, such as passwords or cryptographic authentication keys.[3]

B8c: Resilient-to-Third-Party-Leaks (e.g., IdP-leaks). Attackers cannot defeat user authentication by extracting (e.g., via server-side software vulnerabilities) user-specific data from a trusted remote server (e.g., an IdP, bridge IdP, or synchronization server), such as passwords or cryptographic authentication keys.

B9: Signals-Assurance-Level. This is a security benefit, whereby the IdP can convey to SPs the level of assurance (LoA) that it can provide in the user's identity. LoA is typically dictated based on the strength of the authentication mechanism used. For example, users authenticated with two-factor authentication would have a higher LoA than users authenticated with only a password.

B10: SPs-Can-Filter-IdPs. SPs can restrict the set of IdPs that they wish to support (via whitelist or blacklist), based on any criteria the SP chooses. For example, based on their needs, SPs may prefer to only support IdPs that offer certain security measures such as *Signals-Assurance-Level*, IdPs that use two-factor authentication, or IdPs that tie user identity to a real-world attribute such as a telephone number or physical address.

B11: No-Impersonation-by-Third-Party. The SSO scheme does not provide any remote server with the ability to impersonate a user. A scheme partially provides this benefit if impersonation attempts can be detected by the user. For example, with conventional password-based authentication with e-mail recovery, a malicious e-mail provider may force a password reset on the user's account to gain access, but this is detectable by users since it results in a denial-of-service.

Privacy

[3]Extraction of server-specific data, such as TLS keys that facilitate man-in-the-middle attacks against *all* users of the SP, are excluded. SSO does not offer any inherent defense against such attacks, and complementary mechanisms are needed.

B12: Private-Browsing. The IdP has no knowledge of the SPs that its users authenticate to. We also provide this benefit to schemes where the IdP is hosted on a device that is under the user's control.

B13: Unlinkable-Across-SPs. SPs are not provided with any user identifiers that are linkable across different SP accounts. When evaluating schemes for this benefit, we do not consider other mechanisms that colluding SPs may employ (e.g., browser fingerprinting [56], or collecting personally-identifiable information from the user such as their e-mail address) to find links between different SP accounts.

B14: No-Sharing-of-User-Data. Schemes that authorize SPs to access user information from IdPs do not offer this benefit. For example, OAuth 2.0 and OpenID Connect are widely used by major IdPs (namely Google and Facebook) to provide SPs with access to account information (e.g., contact lists and demographic information), and support long-term access tokens that allow SPs to access account information from IdPs even when the user is logged off.

6.2.2 IdP-SP Association Model

Figure 6.1 (p. 120) illustrates different models by which IdP-SP associations are established. Typically, each IdP is associated with its own namespace. For example, user *John Smith* from IdP_1 and user *John Smith* from IdP_2 are separate identities. Each user-IdP pair is expressed by a unique representation whose format is specified by the SSO protocol, e.g., OpenID 2.0 designates a unique URL for each user under the IdP's domain, and Mozilla Persona uses e-mail addresses under the IdP's domain. An IdP-SP association is characterized by an SP allowing users to identify themselves by their ownership or control over an account or resource under the IdP's namespace. Such an identification process requires a protocol that enables the user to prove their control over the IdP account. We describe six models of IdP-SP association below, and provide a summary in Table 6.1 (p. 120). For each model, we also specify whether the user identity namespace is managed by the user's IdP, a federation operator, or whether each SP manages its own user identity namespace (e.g., as is the case with credential manager schemes).

A1: Decentralized association (IdP-managed namespace). This encompasses "open" protocols in which any entity implementing the protocol can become

an IdP or an SP. For example:

(a) Mozilla Persona: The only requirements to become an IdP are to (1) own a domain name; (2) place a text file on the web server (as explained in Section 6.1.2) containing metadata such as the URL for the user authentication endpoint; and (3) assign user identifiers formatted as a standard e-mail address userid@domain.com (it is not required for the IdP to actually run an e-mail server). Any website can become a Persona SP by implementing the protocol as described in Section 6.1.2 (no registration with a central authority is required).

(b) OpenID 2.0: Similar to Persona, becoming an IdP requires only the implementation of the protocol as described in Section 6.1.2. No registration with any central authority is required, neither for IdPs nor for SPs. In OpenID 2.0, user identities are in the format of a URL (e.g., a profile page) over which the user proves their control by authenticating through the IdP that hosts the URL.

A1 protocols provide *No-IdP-Vetting* and *No-SP-Sponsoring*: Together, these benefits provide the deployability benefit that all SPs and IdPs can implement the protocol without requiring co-ordination. Users also have the freedom to either create an account on an existing IdP or to establish their own IdPs if they wish to do so.

However, A1 protocols lack *Signals-Assurance-Level:* In an implicit SP-IdP association model, SPs cannot determine users' means of authentication—even if IdPs were to provide the information, SPs cannot rely on it without a pre-established trust relationship.

A2: Explicit association (IdP-managed namespace). This is the most common form of web SSO today. SPs must establish an explicit association with each IdP that they wish to support, typically through a manual registration process that involves key exchange or the establishment of a shared secret. OAuth 2.0 and OpenID Connect work in this manner.

A2 protocols provide *No-IdP-Vetting*: Any entity can implement the protocol and become an IdP, thereby facilitating deployment.

However, A2 protocols lack *No-SP-Sponsoring*: Each SP must register with each IdP with which it would like to associate. This typically involves a manual procedure

through which the SP submits a form to establish certain parameters with the IdP, such as an application ID and a shared secret. Typically, a human administrator from the SP must manually submit the form, and the confirmation/approval by the IdP may or may not be automated. This manual process is a deployability drawback that results in SPs supporting a smaller set of IdPs, thereby raising the barrier to entry for potential SP users.

A3: Federated association (IdP-managed namespace). This refers to schemes whereby a central authority (e.g., in OpenID Connect terminology, the *federation operator*) maintains authoritative metadata as to which IdPs and SPs are part of the federation. Both IdPs and SPs must undergo a registration process to join the federation, but not explicitly with each other. Upon joining, SPs may rely on the authentication service of any IdPs that are part of the federation, without needing to manually pre-register with any of them. Member IdPs and SPs may be required to follow certain privacy- and security-related policies, e.g., with regards to storage of user data, or the method of authentication used. Different federations may be established between organizations based on some shared attributes (e.g., geographical area) or objectives (e.g., educational institutions). For example, CANARIE [36] operates a federation of research and educational institutions across Canada.

Currently, SAML (a popular implementation of which is Shibboleth) is the most popular protocol that falls within A3. OpenID Connect also has an optional federation component that is not implemented by any of the current major IdPs, namely Google, Microsoft, Yahoo, and Paypal—SPs must therefore manage independent associations with each IdP. Mobile Connect, administered by the GSM Association, is a recent extension of the OpenID Connect standard that implements a federated association model. Mobile phone service operators across the world that are members of the GSM Association are eligible to be IdPs within the federation. Therefore, once fully deployed, Mobile Connect SPs will be able to authenticate any user worldwide that has mobile phone service.

Aside from maintaining IdP and SP metadata, a principal role of the federation operator is to provide a **discovery service** through which users can be redirected to the appropriate IdP for authentication, e.g., SAML may ask the user to select their IdP manually when logging in, or Mobile Connect may determine the user's IdP

automatically by their mobile phone number.

A3 protocols lack *No-IdP-Vetting*, since the federation must be managed by an operating entity, with which IdPs must register. They also lack *No-SP-Sponsoring*, since SPs must manually register (and potentially abide by certain security policies) to become a member of the federation.

A3 protocols may offer *Signals-Assurance-Level*, if the federation operator imposes requirements for the type of authentication mechanisms used. For example, Mobile Connect grades the user's LoA on a 4-point scale, based on the authentication mechanism used.

A4: Bridged association (federation-operator-managed namespace). These are similar to A3 schemes in that SPs associate with IdPs via a federation operator. However, the federation operator acts as both a discovery service and an *IdP bridge*, in that the SP relies on it for verifying users' control over accounts under the namespace of a multitude of IdPs (as opposed to the SP communicating directly with IdPs). An example is the Mozilla Persona fallback protocol (as opposed to the main Mozilla Persona protocol, which falls within A1 as previously discussed), in which a fallback IdP performs this service in two ways: (1) via OAuth 2.0, for several major e-mail service providers that support it; or (2) by sending a verification code to the user's e-mail address to verify that they can receive the code. In other A4 schemes, such as SAW, the SP may implement its own IdP bridge—e.g., each SAW SP verifies user control of an e-mail address by e-mailing a verification code upon every authentication attempt. A4 schemes may be tailored towards a number of different goals, namely,

- Leveraging an existing protocol deployment (e.g., SMTP and OAuth 2.0, in the case of the Persona fallback protocol) to build a new identity protocol over it.

- Enhancing privacy by implementing the IdP bridge as an anonymizing proxy. For example, SecureKey Concierge is an A4 protocol that provides a "triple-blind" service, wherein SPs are blind to users' IdP information, IdPs are blind to users' SP information, and SecureKey itself only processes unique but anonymous identifiers for users [179].

- Allowing users to switch between IdPs within the federation without imposing

any burden on the SP. SecureKey Concierge provides this feature, by allowing their users to switch between different banking institutions.

Third-party namespace verification may be performed in a way such that it can be validated by multiple parties (e.g., both the IdP and SP). For example, Keybase [100] is a directory service that maps users' public keys to their social media identities (while Keybase is not an SSO service, its function relates to third-party namespace verification). Users post their public keys on their Keybase profile page, and post corresponding cryptographically signed statements on their online accounts (e.g., Twitter, Facebook, Github) to link their online identities. Since the cryptographically signed statements are publicly-accessible, they can be verified not only by Keybase but by anybody—this reduces the trust that must be placed in Keybase itself to verify the statements. In contrast, the Mozilla Persona fallback IdP ties user identity to control of an e-mail address, which involves sending the user an e-mail that only the user can access—a process that is not publicly verifiable.

A5: Non-attested credential manager (SP-managed namespace). These schemes provide repositories in which users save their SP-specific credentials, which may consist of e.g., user names along with passwords or cryptographic keys. The users' credentials serve as pseudonymous identifiers, and are not associated with control over any IdP-controlled namespace. Password managers are the most popular type of A5 protocols currently in use. We call A5 schemes non-attested, since SPs cannot cryptographically verify any information about the authentication mechanism between the user and IdP, including which IdP (if any) was used.

A5 schemes cannot provide *SPs-Can-Filter-IdPs* or *Signals-Assurance-Level*, since SPs have no means of determining the user's IdP. They provide *No-IdP-Vetting* and *No-SP-Sponsoring*, since users can use any A5 scheme without obtaining SP approval.

A6: Attested credential manager (SP-managed namespace). These schemes are similar to A5, but provide additional security guarantees. Through a federation authority, SPs may verify information such as the user's IdP and the type of user-to-IdP authentication used. For example, FIDO UAF uses hardware authenticator devices that act as users' IdPs by storing a unique cryptographic key pair for each SP website. Certified hardware authenticators possess an attestation key

certified by the FIDO Alliance, thereby providing assurance to SPs that the authenticator can securely store private keys in hardware (precluding theft by malware) and signalling the LoA based on the type of user-to-IdP authentication that is used (e.g., PIN or biometric).

A6 protocols can provide *Signals-Assurance-Level*, since IdPs undergo a certification process by the federation operator, which can then signal the LoA information to SPs.

To support increased device compatibility, FIDO UAF also supports software authenticators lacking the ability to protect cryptographic keys from malware, and therefore performing no attestation (however, this is explicitly signalled). FIDO UAF is therefore a hybrid system supporting both "strong" attested hardware authenticators certified by a central authority (i.e., A6 scheme that provides LoA signalling) and "weak" unattested software authenticators (i.e., A5 protocol). We analyze hardware and software authenticators separately, since they differ substantially in the benefits offered.

6.2.3 IdP: User Identity Conveyance Method

The following categories classify SSO schemes based on how IdPs convey user identity to SPs.

G1: IdP assertion. The IdP generates an identity assertion each time the user needs to access their SP. OAuth, OpenID Connect, Mobile Connect, and Shibboleth fall within this category. The IdP typically communicates the assertion to the SP through the browser via HTTP redirection, HTTP POST requests, and/or cross-origin message-passing between iframes with postMessage [129].

G2: Browser assertion. The IdP delegates the authority for identity assertion generation to the user's browser. Mozilla Persona falls within this category: The browser generates a cryptographic key pair to sign its own self-generated identity assertions. Identity assertions also contain an IdP signature over the browser's public key, thereby allowing SPs to validate assertions. The assertion is typically communicated from client-side code to the SP. Mozilla Persona was intended to eventually be built into the browser, but as an interim measure it used a client-side JavaScript library.

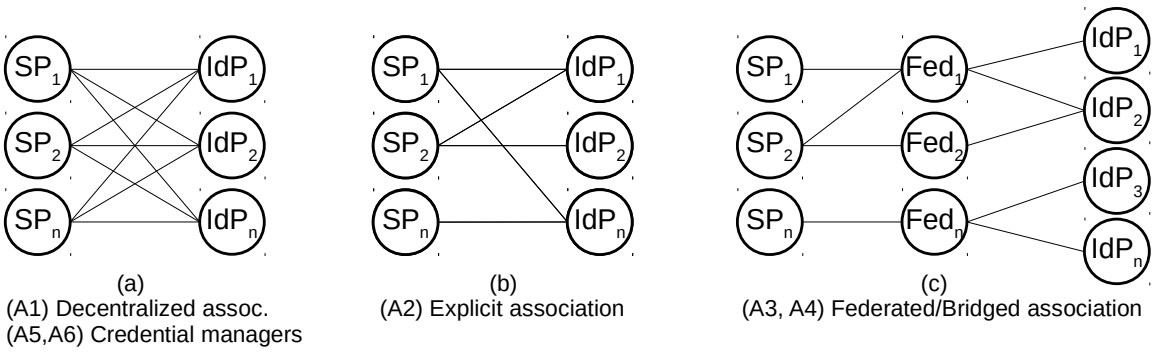

Figure 6.1: IdP-SP association models A1-A6, defined in Section 6.2.2. {A1,A5,A6} allow users to authenticate to any SP through any IdP; A2 requires IdPs and SPs to have a pre-established relationship, e.g., via a manual registration process; {A3, A4} allow users to authenticate to SPs through an IdP only if both the SP and IdP are federation members.

IdP-SP Association Model	Association Category	Example Scheme	NS Auth.: IdP	NS Auth.: Fed. Operator	NS Auth.: SP	Defining Characteristics
A1	(a)	OpenID 2.0	•			Any server implementing the protocol can become an IdP or SP.
A2	(b)	OAuth 2.0	•			SPs establish explicit associations with IdPs, e.g., by manual registration to establish shared secrets.
A3	(c)	Mobile Connect	•			A federation operator records which IdPs and SPs belong to this federation; SPs then rely on IdPs in the federation without having explicit relationships with them.
A4	(c)	SecureKey Concierge		•		Similar to A4, except that the federation operator is responsible not only for recording which IdPs and SPs belong to the federation, but also for conveying identity assertions to SPs on behalf of IdPs.
A5	(a)	Firefox Sync			•	Credential management schemes that store and automate the use of SP-specific user credentials.
A6	(a)	FIDO UAF (Attested)			•	Credential management schemes that can provide cryptographic attestation to SPs to provide security assurances, e.g., so the SP can verify the method of user-to-device authentication used.

Table 6.1: Summary of IdP-SP association models from Section 6.2.2. Association category (second column) corresponds to the three graphs from Figure 6.1. For each association model (row), a bullet in one of three columns indicates whether the user identity namespace (NS) authority is the user's IdP, a federation operator, or the SP.

G3: User-to-SP authentication. In this category, there is no vouching mechanism that can signal to the SP that the user has already authenticated itself to the IdP, and therefore the user must authenticate directly to the SP. The IdP may facilitate or may automate the process by which users authenticate to SPs (e.g., by storing and synchronizing the user's credentials across multiple devices, and automatically filling in forms to enter the credentials). All A5 protocols fall within this category. A4 protocols may also fall within this category, if the SP acts as its own IdP bridge (e.g., SAW).

G4: IdP-proxied authentication. In this case, the IdP serves as an HTTP proxy between the user and SP, which may convey identity information and/or credentials over a direct network connection to the SP. Thus, all traffic between users and SPs (both during authentication and any subsequent traffic during the authenticated session) flows through the IdP. Web proxies such as *EZproxy* [150] are commonly used by educational institutions to allow off-campus access of academic material—e.g., ieeexplore.ieee.org may be accessed through ieeexplore.ieee.org.edu-proxy.com, where edu-proxy.com is the URL of the institution's proxy. However, such proxies typically do not convey any identity information, since the SPs operate on IP address based access control—thus, any user visiting from within the institution's IP address range is granted access. We are not aware of any proxy-based SSO schemes used in practice, but they have been proposed in literature; Impostor [156] falls within this category.

G1 schemes are not *Resilient-to-Temporary-Service-Outage*, since SPs must obtain an IdP-generated identity assertion to establish an authenticated session. G2 schemes *may* partially provide this benefit, if the client possesses a relatively long-lived assertion-signing key; with Mozilla Persona, the expiration time of the client's signing key is set by the IdP and could be set to, e.g., 24 hours (an excessive expiration time poses additional impersonation risks if the signing key is stolen). G3 schemes *may* fully provide this benefit if the credentials are cached client-side—users may thus continue to use their locally-cached credentials, but would not be able to update/synchronize any of their devices with new or updated credentials for as long as the IdP remains unreachable. G4 schemes are not *Resilient-to-Temporary-Service-Outage*, since all communication between clients and SPs must be proxied through an IdP.

6.2.4 SP: User Identity Verification Method

The methods of conveying an identity assertion from an IdP to an SP can be categorized as follows:

C1: IdP Query. The SP must query the IdP to determine the validity of an authentication token presented by the user (e.g., to verify that the token was issued by the IdP, intended for use by the SP in question, and has not expired).

C2: Local Verification. The SP itself directly verifies that the user has correctly authenticated. This may be done if (1) the user authenticated directly to the SP (see G3), or (2) if the identity assertion is cryptographically signed by the IdP and can be verified by the SP.

The OpenID 2.0 specification supports both C1 and C2, but recommends C2 since it reduces the number of required protocol round trips [163]—C1 is called *direct validation*, and C2 is called *association mode*. Association mode requires a Diffie-Hellman key exchange to establish a shared key between the IdP and SP; the IdP uses the key to compute a MAC on identity assertions destined for the associated SP. Association mode prevents tampering of identity assertions via MITM attacks that may occur if direct validation is used. However, TLS also prevents tampering, and is recommended (but not required) by the specification.

Mozilla Persona supports only C2: IdPs are *stateless*, since they do not generate any identity assertions, and are not informed upon the generation of any identity assertions. Instead, the browser generates the identity assertion (see 6.2.3), which contains (1) the SP domain for which the assertion was generated, (2) the assertion's expiry time, (3) a client-generated cryptographic signature of the assertion, and (4) an IdP-generated signature of the client-generated signing key. The SP can validate assertions locally, since the only information required from the IdP is the IdP's public key (see Section 6.1.2), to be used for verifying the IdP-generated signature of the client's signing key. All other required information for validating the assertion (e.g., SP domain, assertion expiry time) is present within the assertion itself.

SAML and the OAuth 2.0 family of protocols (which includes OpenID Connect and Mobile Connect) are *stateful* protocols: Clients must either (1) obtain a bearer token from the IdP and send it to the SP, which must then query the IdP to determine the assertion's validity (known as *Implicit Grant* in OAuth 2.0); or (2) obtain a temporary

code from the IdP and send it to the SP, which must then exchange it for a token from the IdP (known as *Authorization Code Grant* in OAuth 2.0).

C2 protocols that are stateless can provide *Private-Browsing* (this includes all A5 and A6 protocols), since IdPs are not aware of the SPs with which users are associated. Protocols that are both C1 and A4 may also offer this benefit if blinding is used (see Section 6.2.2).

6.2.5 User to IdP Authentication Type

Single sign-on involves both a user-to-IdP component and an IdP-to-SP component—the latter was covered in Section 6.2.3. The user-to-IdP component can be classified into two general categories based on whether the IdP is local (i.e., a device under physical control of the user) or remote (i.e., a remote server). Evaluating specific user-to-web authentication schemes [25] is out of the scope of this chapter. However, we highlight the differences between different categories of single-factor and two-factor schemes, particularly for FIDO UAF and Mobile Connect. The following two categories are illustrated in Figure 6.2.

T1: Remote authentication. The user authenticates to a remote IdP, by means of a password or any other form of user-to-web authentication. This may involve one or more authentication factors:

(a) **Single factor.** Most currently-deployed IdPs that use single-factor authentication use conventional password-based authentication. One exception is Mobile Connect with LoA2, which validates the user's possession of their mobile phone. Depending on the implementation, this may require the user to type in an SMS OTP received on their phone, or the Mobile Connect authenticator application may display a popup on the phone for the user to confirm the authentication request by tapping an "OK" button.

(b) **Two factor.** Two-factor authentication can be further divided into two sub-categories, that we define as follows:

(i) **Two-step.** Two-factor authentication in its most popular form on the web

involves two steps, wherein the server verifies the user's password (*what-you-know*) in addition to their possession of a hardware token (*what-you-have*) such as a smartphone or USB dongle. Here, the two factors are independent from each other.

(ii) **Two-stage.** Mobile Connect with LoA3 requires that the mobile device (*what-you-have*) be protected by a local authentication mechanism such as a PIN or biometric. We call this two-stage, since the first authentication stage (e.g., entering a PIN) occurs on the device itself, with no participation from the IdP. Only the second stage (e.g., receiving an SMS OTP or tapping an "OK" button as described above) involves the IdP. Here, the second factor (stage) depends on the first.

The security offered by the first stage of two-stage authentication depends on the implementation of the local authentication stage. For example, a local authentication mechanism that relies on a secure boot mechanism and trusted execution environment to render a stolen device inoperable after ten consecutive failed authentication attempts would offer higher security than a device on which an attacker is given unlimited authentication attempts or can easily extract the plaintext secret from memory.

T2: Local authentication. The user authenticates to an IdP hosted on a local device under physical control of the user. This differs significantly from T1 schemes that use hardware authenticators (e.g., Mobile Connect or OpenID Connect with Google two-step verification) to authenticate to a remote IdP. Local authentication can be further classified as follows:

(a) **Device-possession only.** No user-to-device authentication is required, aside from device possession (typically implied for local authentication). Firefox Sync 1.5 and 2.0 are both in this category by default, since the password vault is stored on-disk in plaintext and can easily be extracted when in possession of the device. The two protocols differ in their cross-device synchronization mechanisms, which is distinct from the local user-to-IdP authentication mechanism discussed here.

(b) **Single factor.** A password or other user-to-device authentication scheme (e.g.,

smart card, PIN, biometrics) is used to authenticate to the on-device local IdP. FIDO UAF is in this category since it requires, e.g., a PIN or biometric to be used locally before the user can be authenticated to any remote SPs. Firefox Sync could be classified into this category if, e.g., the password vault is protected by a master password or by full-disk encryption. However, weaker mechanisms such as the user login mechanism in many commodity desktop operating systems that do not encrypt user data can be easily defeated—for example, an attacker in physical possession of the device could circumvent the user login (e.g., by removing the hard disk and reading its contents) to extract the unencrypted password vault from the device's storage. Therefore, similarly to the first stage of two-stage remote authentication as discussed above, the security offered by a single-factor local authentication scheme depends on implementation and device configuration details, e.g., use of secure boot and full-disk encryption.

Local authentication may use two or more factors in addition to the possession factor, but this is not typically used (we are not aware of any such scheme used in practice).

T1 offers *No-Device-Setup*, since users can authenticate to a remote IdP from any device; T2 cannot offer this benefit, since users must configure their new devices before they are able to authenticate to SPs through them.

Trusted Computing

Secure hardware key storage, and trusted execution environments [140] can be used to enhance the security of both T1 and T2 schemes. The primary benefit is the reduction of damage that can be done by an attacker that breaches the IdP (e.g., via a remote exploit, malware, or physical device theft)—both for IdPs hosted on a remote server or on a local device under the user's control. For context, Table 6.2 summarizes some attacks against user-to-IdP authentication along with corresponding defenses.

Another function of trusted computing in SSO is that it allows one party to make cryptographically-verifiable guarantees to another party. For example, through trusted execution, FIDO UAF attested hardware authenticators guarantee to SPs that the user has been authenticated on the local device via, e.g., a physical biometric. Mobile Connect can use the mobile phone's SIM card as a secure element for credential storage and as a trusted execution environment to authenticate the user, guaranteeing

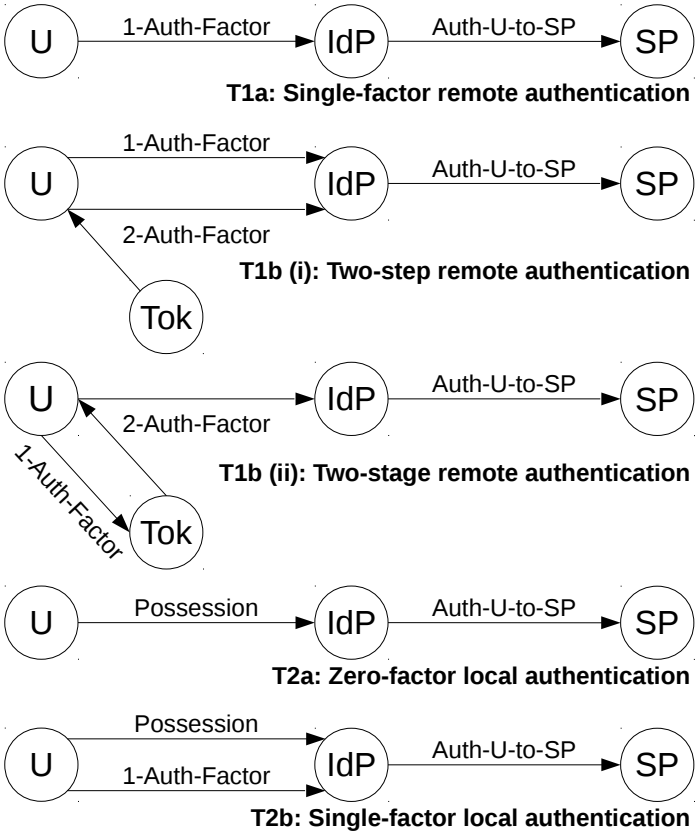

Figure 6.2: Authentication flow for single-factor and two-factor T1 (Remote authentication) and T2 (Local authentication) schemes. **U**, **IdP**, **SP**, and **Tok** represent the user, identity provider (remote or local), service provider, and hardware token, respectively.

Authentication Type	Possible Threat	Corresponding Mitigation
T1a. Single factor, remote IDP	Attacker may break first factor	Add second factor
T1b(i). Two-step, remote IdP	Attacker may breach remote IdP and steal server-side stored credentials, e.g., password hash (to perform offline guessing attack) and OTP seed	Use non-replayable credentials (e.g., public-key crypto)
		Use hardware security mechanisms to securely store replayable server-side credentials (e.g., use hardware security module to store symmetric key and use it to perform HMAC of passwords instead of a standard hash)
T1b(ii). Two-stage, remote IdP	Attacker in possession of hardware token may break first factor, since it is validated locally	Use hardware security on authentication token (e.g., trusted boot, trusted execution environment, encrypted storage) to protect authentication from local attacks
T2a. Device-possession only, local IdP	Device theft	Add a local authentication factor
T2b. Single factor, local IdP	Attacker may circumvent local authentication	Use hardware security on authentication token (e.g., trusted boot, trusted execution environment, encrypted storage) to protect authentication from local exploits
	Attacker may breach local IdP (e.g., via malware) and steal stored credentials to impersonate user	Use non-replayable credentials (e.g., public-key crypto) and protect them using secure hardware storage
		Ensure that replayable credentials are never exposed to any party other than the corresponding SP, e.g., by using mandatory access control to ensure that only the IdP has access to stored credential and only releases them to authorized SPs

Table 6.2: Threats and corresponding mitigations to discussed user-to-IdP authentication types. Single-factor user-to-device authentication (**T2b**) requires user possession of the device, and therefore provides security benefits similar to two-factor user-to-remote-IdP authentication (**T1b**).

a higher LoA [77]. SSO may also benefit directly or indirectly from other applications of trusted computing that enhance the security and privacy characteristics of online services. For example, a remote server with Intel SGX can guarantee [108] that the password entered by the user will be processed in a secure enclave and that only the CMAC (cipher-based MAC) will be stored server-side, thereby eliminating the possibility of offline attacks in the event that the password database is leaked. The secure messaging application Signal [122] uses SGX for private contact discovery, allowing clients to poll a secure enclave on the server to discover which of their contacts are also using Signal, without revealing the user's contact list to the server.

6.2.6 Multi-Device Usage Model

The means by which users can access their SP accounts from multiple devices is related to, but not fully determined by, the user-to-IdP authentication type (discussed above in Section 6.2.5). Schemes can be categorized by this property as follows.

M1: Device-independent. This includes remote authentication schemes (T1) that do not rely on a hardware authenticator device that the user needs to carry.

M2: Portable hardware token. This includes remote authentication schemes (T1) that use a portable hardware authenticator device, e.g., MobileConnect (both single-factor LoA2 and two-factor LoA3).

M3: Per-device authorization. This includes schemes in which each of the user's devices need to be individually authorized by the user's IdP (for T1 schemes) or the user's SPs (for T2 schemes). Mobile Connect (T1) could be extended to allow mobile network operators to authorize multiple SIM cards that users could install in their devices (e.g., laptops); this would result in a hybrid of M2 and M3, since users could authorize their devices equipped with a cellular modem, but would need to carry around their mobile phone to log in on non-equipped devices. FIDO UAF (T2) is an M3 scheme, since users that obtain a new FIDO-enabled device need to authorize the new device with each of their SPs.

M4: Device pairing. This includes local authentication schemes in which new user devices need to be authorized by (i.e., "paired" with) an existing user device. This could be achieved by, e.g., securely transferring a symmetric key from one device to another, in the case of Firefox Sync. A more complex mechanism may be to generate a new cryptographic key pair on the new user device, and authorizing the new key pair from the existing device (e.g., by signing the public key).

M1 cannot offer *Resilient-to-Client-Leaks*, but offers *No-Hardware-Token-Required*. M2 may offer *Resilient-to-Client-Leaks* but cannot offer *No-Hardware-Token-Required*. M3 may offer *Resilient-to-Client-Leaks* but M4 can only offer it if public-key cryptography is used and the private key cannot be extracted from the device. Neither M3 nor M4 can offer *No-Device-Setup*, since users cannot authenticate from a new device to their SP without first enrolling the device onto their SP accounts.

Scheme	IdP-SP Association Model	IdP: User Identity Conveyance Method	SP: User Identity Verification Method	User to IdP Authentication Type	Multi-Device Usage Model	B1: Portable-Identity-Across-IdP	B2: No-Device-Setup	B3: No-Hardware-Token-Required	B4: Resilient-to-Temporary-Service-Outage	B5: No-IdP-Vetting	B6: No-SP-Sponsoring	B7: No-SP-Stored-User-Secret	B8a: Resilient-to-Client-Leaks	B8b: Resilient-to-SP-Leaks	B8c: Resilient-to-Third-Party-Leaks	B9: Signals-Assurance-Level	B10: SPs-Can-Filter-IdPs	B11: No-Impersonation-by-Third-Party	B12: Private-Browsing	B13: Unlinkable-Across-SPs	B14: No-Sharing-of-User-Data
	Design Properties					Usability				Deploy.			Security						Privacy		
Firefox Sync 1.5 [205]	A5	G3	C2	T2a	M4	●		●	●	●	●				●			●	●	●	●
Firefox Sync 2.0 [206]	A5	G3	C2	T2a	M4	●	○	●	●	●	●				○			○	●	●	●
FIDO UAF (Attested) [117, 22]	A6	G1	C2	T2b	M3			●	●		●	●	●	●	●	●	●	●	●	●	●
FIDO UAF (Non-Attested) [117, 22]	A5	G1	C2	T2b	M3	●		●	●	●	●	●			●	●		●	●	●	●
Impostor [156]	A5	G4	C2	T1	M1		●			●	●		●							●	●
SAW [195]	A4	G3	C1	T1	M1	○	●	●		●	●	●		●			●				●
OAuth 2.0 [84]	A2	G1	C1	T1	M1		●	●		●		●		●			●				
OpenID 2.0 [163]	A1	G1	C2	T1	M1		●	●		●	●	●		●			●				●
OpenID Connect [170]	A2	G1	C1	T1	M1		●	●		●		●		●			●				
Mobile Connect [79]	A3	G1	C1	T1	M2		●				●	●	●	●	●	●	●			●	
Mozilla Persona [82]	A1	G2	C2	T1	M1		●	●	○	●	●	●		●			●		●		●
Mozilla Persona Fallback [82]	A4	G2	C2	T1	M1		●	●	○	●	●	●		●			●	†	●		●
Shibboleth [176]	A3	G1	C2	T1	M1		●	●			§	●		●		●	●			§	
SecureKey Concierge [179]	A4	G1	C2	T1	M1	○	●	●			●	●		●		●	●	†	●	●	●

Table 6.3: SSO schemes (rows) categorized across design properties (columns, left half) and evaluated across benefits (columns, right half) from Section 6.2. Bullets represent benefits provided; hollow circles represent partially-provided benefits; empty cells represent benefits not provided. †Susceptible to impersonation by either of *two* third parties. §Offered via optional protocol feature.

6.3 Evaluation of SSO Schemes

Here, we explain our evaluation (based on the benefits discussed in Section 6.2.1) and categorization (based on the design properties discussed from Sections 6.2.2 through 6.2.5) of each SSO scheme introduced in Section 6.1, as is summarized in Table 6.3.

Firefox Sync. We evaluate Firefox Sync 1.5 and 2.0 together, and indicate differences between the two versions in the benefits offered. Firefox Sync offers *Portable-Identity-Across-IdP*, since the user's device is the IdP, and transferring credentials to new devices is supported via a synchronization mechanism. Users can authenticate to SPs from any synchronized device without an additional hardware token (*No-Hardware-Token-Required*), even when the synchronization server is down (*Resilient-to-Temporary-Service-Outage* for authenticating to SPs, but not for updating or adding new passwords to the vault). Firefox Sync 1.5 does not offer *No-Device-Setup*, since users must "pair" any new devices with one of their existing devices by transferring their symmetric key for decrypting their password vault; Firefox Sync 2.0 partially offers this benefit, since the decryption key is derived from the user's password and must be entered during the device setup process.

Users' devices are not vetted by any central authority (*No-IdP-Vetting*) and SPs do not require any IdP-specific server-side modifications (*No-SP-Sponsoring*). However, SPs must still store and protect per-user secret credentials (*No-SP-Stored-User-Secret* not offered). *Signals-Assurance-Level* and *SPs-Can-Filter-IdPs* are not provided since SPs cannot distinguish between IdPs.

Firefox Sync 1.5 offers *No-Impersonation-by-Third-Party* since the password vault is locally encrypted on the user's device with a randomly-generated symmetric key; Firefox Sync 2.0 partially offers this benefit, since the vault encryption key is password-derived and is therefore subject to offline attack by the synchronization server. Firefox Sync 1.5 does not offer *Resilient-to-Client-Leaks*, since the vault encryption key is not stored in hardware and can be transferred to all the user's devices via the pairing process; neither does Firefox Sync 2.0, since it derives the key from a password, which may be captured by an attacker. Neither offer *Resilient-to-SP-Leaks*, since SPs must store user passwords server-side (hashed passwords may still be cracked via offline guessing attacks). Firefox Sync 1.5 offers *Resilient-to-Third-Party-Leaks*, since password vaults stored on the synchronization server are

encrypted with randomly-generated keys known only to clients, and therefore are infeasible to decrypt if leaked; Firefox Sync 2.0 partially offers this benefit, since it relies on password-derived keys, and therefore leaked password vaults may be subject to offline attack (with attack success depending on the strength of the user-chosen passwords).[4]

Firefox Sync offers *Private-Browsing*, *Unlinkable-Across-SPs*, and *No-Sharing-of-User-Data* since all authentication is performed through the user's device without any participation by a remote third-party server.

FIDO UAF. We evaluate the attested and non-attested FIDO UAF variants together, and indicate differences between the two versions in the benefits offered. UAF offers *No-Hardware-Token-Required* and *Resilient-to-Temporary-Service-Outage*: users can access their SP accounts from any of their UAF-enabled personal devices without dependence on any third-party remote servers, but users must first set up UAF on their devices with each individual SP (so *No-Device-Setup* is not offered). Unattested UAF offers *Portable-Identity-Across-IdP* since users can transfer their credentials to a new device, but attested UAF does not offer this benefit since keys are stored securely in the device's hardware and cannot be extracted.

Non-attested UAF offers *No-IdP-Vetting*, but attested UAF does not, since IdPs must have their attestation key signed by the FIDO Alliance. Both UAF variants offer *No-SP-Sponsoring* since SPs do not require any IdP-specific server-side modifications—SPs need only possess the FIDO Alliance root certificate to verify the validity of any FIDO-signed IdP attestation keys. Both variants offer *No-SP-Stored-User-Secret*, since SPs store users' public keys.

Attested UAF offers *Resilient-to-Client-Leaks*, since private keys are stored in hardware and cannot be extracted; non-attested UAF does not, since keys are stored in software and are thus susceptible to capture. Both variants offer *Resilient-to-SP-Leaks* and *Resilient-to-Third-Party-Leaks*, since verifying servers only store users' public keys (which are not secret). Attested UAF allows SPs to validate signed IdP attestation certificates, which indicate the method of authentication used (e.g., 4-digit PIN or biometric) and the hardware vendor and model number (allowing SPs to phase

[4]This grading depends on the assumption that passwords are not re-used across SPs. Password re-use results in the loss of the *Resilient-to-Third-Party-Leaks* benefit, since a leak from one SP can lead to account compromise on other SPs.

out support for hardware authenticators in which vulnerabilities have been found), thereby offering both *Signals-Assurance-Level* and *SPs-Can-Filter-IdPs*; non-attested UAF does not offer these benefits, since they are not certified by FIDO and therefore use self-signed attestation certificates, with the signing key stored unprotected in software. Both UAF variants offer *No-Impersonation-by-Third-Party*, since users' authentication keys are only stored on their own devices.

Both UAF variants offer *Private-Browsing*, *Unlinkable-Across-SPs*, and *No-Sharing-of-User-Data* since all authentication is performed through the user's device without any participation by a remote third-party server. To limit linkability between SP accounts, attested UAF must use an attestation certificate that is shared by at least 100,000 hardware devices (the same batch/revision of any hardware authenticator model shares the same attestation certificate).

Impostor. *Portable-Identity-Across-IdP* is not offered, since users cannot change IdPs without updating all of their SP accounts with their new identity. *No-Device-Setup* is offered but *No-Hardware-Token-Required* is not offered (assuming a hardware OTP token is used for user-to-IdP authentication; see Section 6.1.1). *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through their IdP.

No-IdP-Vetting is offered, since any domain owner can set up their own IdP. *No-SP-Sponsoring* is offered, since users can provide any e-mail address as their user name to SPs. *No-SP-Stored-User-Secret* is not offered, since SPs store user passwords.

Resilient-to-Client-Leaks is offered, since Impostor uses a challenge-response authentication protocol designed for use on untrusted access devices, without exposing any long-term credentials to them. *Resilient-to-SP-Leaks* is not offered, since SPs must store user passwords server-side (hashed passwords may still be cracked via offline guessing attacks). *Resilient-to-Third-Party-Leaks* is not offered, since passwords are stored in plaintext by IdPs and are susceptible to capture and reuse. *Signals-Assurance-Level* and *SPs-Can-Filter-IdPs* are not offered, since there is no SP-IdP communication aside from automated password submission. *No-Impersonation-by-Third-Party* is not offered, since IdPs are in possession of users' SP passwords (unless users self-host an IdP).

Private-Browsing is not offered, since users access all their SPs through the IdP

proxy. *Unlinkable-Across-SPs* and *No-Sharing-of-User-Data* are offered, since IdPs only communicate SP-specific user names and passwords.

SAW. SAW partially offers *Portable-Identity-Across-IdP*, since users can use e-mail forwarding on an interim basis when transitioning from one address to another (but all SPs must still be individually updated). *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming password-based authentication to the e-mail provider. *Resilient-to-Temporary-Service-Outage* is not offered, since e-mail service outage prevents users from receiving OTPs.

No-IdP-Vetting and *No-SP-Sponsoring* are offered since any domain owner can host a mail server and exchange e-mails with any other mail servers. *No-SP-Stored-User-Secret* is offered since SPs do not need to store any user secrets (only their e-mail addresses).

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is not offered, since the user-to-IdP authentication method is not signalled to SPs. *SPs-Can-Filter-IdPs* is offered, since SPs can whitelist or blacklist e-mail providers by domain name. *No-Impersonation-by-Third-Party* is not offered: a malicious mail server administrator can impersonate their own e-mail users.

Private-Browsing is not offered since IdPs can track the SPs that their users visit. *Unlinkable-Across-SPs* is not offered, assuming users use the same e-mail address across different SP accounts (users can theoretically use different e-mail aliases pointing to the same e-mail account to obtain this benefit, but we do not assume this to be the case due to its impracticality). *No-Sharing-of-User-Data* is offered since SMTP does not provide any user information.

OAuth 2.0 and OpenID Connect. The following evaluation applies to both OAuth 2.0 and OpenID Connect, since their differences are small enough in their

default configurations such that they do not differ in their benefits offered.[5] *Portable-Identity-Across-IdP* is not offered, since users cannot change IdPs without updating all of their SP accounts with their new identity. *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming conventional password authentication. *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through their IdP.

No-IdP-Vetting is provided, since any domain owner can set up their own IdP. *No-SP-Sponsoring* is not offered, since SPs must complete a manual registration process with each individual IdP. *No-SP-Stored-User-Secret* is offered, since SPs only store a {user ID, IdP ID} pair.

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is not offered. *SPs-Can-Filter-IdPs* is offered, since SPs must explicitly implement support for each IdP. *No-Impersonation-by-Third-Party* is not provided, since IdPs can impersonate their users.

Private-Browsing is not offered, since user authentication requires redirecting the user's browser between the IdP and SP. *Unlinkable-Across-SPs* is not offered, since it is not required for IdPs to assign unique pairwise (user-to-SP) pseudonymous identifiers to prevent identity correlation across SP accounts. *No-Sharing-of-User-Data* is not offered, since OAuth 2.0 and OpenID Connect based IdPs provide SPs with extensive access to user profile information.

OpenID 2.0. *Portable-Identity-Across-IdP* is not offered, since users cannot change IdPs without updating all of their SP accounts with their new identity. *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming conventional password authentication. *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through their IdP.

[5] As discussed in Section 6.1.2, OpenID Connect offers a more strictly-defined protocol for authentication (compared to OAuth 2.0), which results in less variation between implementations, thereby enhancing interoperability. It also offers optional features that can be implemented to, e.g., establish a federated association model as is done by Mobile Connect (which is therefore evaluated separately).

No-IdP-Vetting is provided, since any domain owner can set up their own IdP. *No-SP-Sponsoring* is provided, since user IDs are in the format of a globally-resolvable user profile URL via which SPs can dynamically discover the user's IdP and initiate the authentication process. *No-SP-Stored-User-Secret* is offered, since SPs only store user profile URLs.

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is not offered. *SPs-Can-Filter-IdPs* is offered, since SPs can filter IdPs by domain name. *No-Impersonation-by-Third-Party* is not provided, since IdPs can impersonate their users.

Private-Browsing is not offered, since user authentication requires redirecting the user's browser between the IdP and SP. *Unlinkable-Across-SPs* is not offered, since user IDs (i.e., profile URL) are the same across different SPs. *No-Sharing-of-User-Data* is offered, since OpenID 2.0 does not support the exchange of user profile information.

Mobile Connect. *Portable-Identity-Across-IdP* is not offered, since users lose access to all their SP accounts upon changing their mobile network operator—the GSMA aims to address this in the future, by allowing users to transfer their identity across IdPs if they keep the same phone number when changing service providers [80]. *No-Device-Setup* is offered since users can log into their SP accounts on any device as long as they are in possession of their mobile phone (thus *No-Hardware-Token-Required* is not offered). *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through their IdP.

No-IdP-Vetting is not provided, since IdPs must be a GSM mobile network operator. *No-SP-Sponsoring* is provided, since Mobile Connect provides a discovery service through which SPs are redirected to the correct IdP based on the user's mobile phone number. *No-SP-Stored-User-Secret* is offered, since SPs only store pseudonymous user identifiers provided by IdPs.

Resilient-to-Client-Leaks is offered, since authentication private keys are securely stored in users' SIM cards; *Resilient-to-SP-Leaks* is offered, since SPs do not store any

user authentication secret; and *Resilient-to-Third-Party-Leaks* is offered since IdPs do not store any user authentication secrets (only public keys). *Signals-Assurance-Level* is offered, since IdPs signal SPs with an LoA for each authenticated user. *SPs-Can-Filter-IdPs* is offered. *No-Impersonation-by-Third-Party* is not provided, since IdPs can impersonate their users.

Private-Browsing is not offered, since user authentication requires redirecting the user's browser between the IdP and SP. *Unlinkable-Across-SPs* is offered, since IdPs assign unique pairwise (user-to-SP) pseudonymous identifiers to prevent user identity correlation across SPs. *No-Sharing-of-User-Data* is not offered, since Mobile Connect allows IdPs to share user attributes.

Mozilla Persona. The following evaluation applies to both Mozilla Persona and Mozilla Persona Fallback, with any differences between the two variants (in terms of benefits offered) explicitly indicated. *Portable-Identity-Across-IdP* is not offered, since users cannot change IdPs without updating all of their SP accounts with their new identity. *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming conventional password authentication. *Resilient-to-Temporary-Service-Outage* is partially offered, subject to the browser certificate expiry time set by the IdP (typically 24 hours)—the user's browser can generate signed identity assertions that can be validated by SPs (if the SP has a cached copy of the IdP public key).

No-IdP-Vetting is offered, since any domain owner can set up their own IdP. Persona offers *No-SP-Sponsoring*, since users can provide any e-mail address as their user name to SPs. *No-SP-Stored-User-Secret* is offered, since SPs only store users' e-mail addresses.

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is not offered. *SPs-Can-Filter-IdPs* is offered, since SPs can filter IdPs by domain name. *No-Impersonation-by-Third-Party* is not offered, since IdPs (and also the fallback server, in the case of the Persona Fallback scheme) can impersonate their users.

Private-Browsing is offered, since identity assertions are generated locally on users'

browsers. *Unlinkable-Across-SPs* is not offered, since user IDs (i.e., e-mail address) are the same across different SPs. *No-Sharing-of-User-Data* is offered, since Persona does not support the exchange of user profile information.

Shibboleth. *Portable-Identity-Across-IdP* is not offered, since users cannot change IdPs without updating all of their SP accounts with their new identity. *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming conventional password authentication. *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through their IdP.

No-IdP-Vetting is offered, since any domain owner can set up their own IdP. *No-SP-Sponsoring* may be offered if SPs use a federation discovery service that allows users to pick from a list of member IdPs in the federation. *No-SP-Stored-User-Secret* is offered, since SPs only store a {user ID, IdP ID} pair.

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is offered, since the underlying SAML protocol allows IdPs to signal an LoA for each authenticated user (enforcement of the feature depends on the federation authority). *SPs-Can-Filter-IdPs* is offered, since SPs can choose which IdPs to support. *No-Impersonation-by-Third-Party* is not offered, since IdPs can impersonate their users.

Private-Browsing is not offered, since user authentication requires redirecting the user's browser between the IdP and SP. *Unlinkable-Across-SPs* may be offered, since Shibboleth optionally supports IdP-assigned unique pairwise (user-to-SP) pseudonymous identifiers to prevent user identity correlation across SPs. *No-Sharing-of-User-Data* is not offered, since Shibboleth supports the transfer of user profile information.

SecureKey Concierge. *Portable-Identity-Across-IdP* is partially offered, since SecureKey maintains a unique pseudonymous identifier for each user that is mapped to the user's IdP of choice, and users may switch to another IdP within the federation at any time. *No-Device-Setup* and *No-Hardware-Token-Required* are offered, assuming conventional password authentication. *Resilient-to-Temporary-Service-Outage* is not offered, since users cannot authenticate to SPs without first authenticating through

their IdP.

No-IdP-Vetting is not provided, since IdPs must be approved by SecureKey. *No-SP-Sponsoring* is offered, since SecureKey can add or remove support for IdPs without requiring any server-side updates to SPs. *No-SP-Stored-User-Secret* is offered, since SPs only store pseudonymous user identifiers provided by SecureKey.

Resilient-to-Client-Leaks is not offered (assuming password authentication); *Resilient-to-SP-Leaks* is offered, since SPs do not store any user authentication secret; and *Resilient-to-Third-Party-Leaks* is not offered since hashed passwords stored by IdPs are susceptible to exposure and offline attack, resulting in attacker access to users' SP accounts. *Signals-Assurance-Level* is offered. *SPs-Can-Filter-IdPs* is offered, since SecureKey allows SPs to choose the set of IdPs that they wish to allow. *No-Impersonation-by-Third-Party* is not offered, since both SecureKey and users' IdPs can impersonate their users.

Private-Browsing, *Unlinkable-Across-SPs* and *No-Sharing-of-User-Data* are offered: users are indirectly forwarded from SPs to IdPs via SecureKey, which only exchanges blinded identifiers (see Section 6.1.2) to offer privacy from IdPs and prevent user identity correlation across SPs.

6.4 Discussion

Among the schemes analyzed, none possesses the technical features necessary to offer all the benefits discussed. This is to be expected, since each design property discussed necessitates some trade-offs between benefits. Moreover, even for schemes that are able to offer certain benefits, conflicting interests between the stakeholders involved (namely users, SPs, and IdPs) may result in the benefits not being offered—e.g., OpenID Connect supports the use of pseudonymous identifiers, which provides *Unlinkable-Across-SPs*, but major IdPs such as Google do not implement this feature [72]. This motivates a discussion to examine the scenarios under which each individual scheme offers the most appropriate combination of benefits.

Below, in Sections 6.4.1 and 6.4.2 we consider different priorities from users' and SPs' perspectives for authentication to high- and medium-value accounts (low-value accounts are less relevant to this discussion, since they do not require any of the more complex solutions as discussed herein). In Section 6.4.3 we discuss the threat of user

impersonation by rogue insiders at IdPs, and provide two examples to illustrate the complexity trade-offs associated with mitigating this threat.

6.4.1 High-Value SP Accounts

User's perspective. For high-value SPs, such as banking or government services, the threat of impersonation and/or data collection by IdPs (which relates to *No-Impersonation-by-Third-Party* and *Private-Browsing*) is an important concern from users' perspectives. In other words, *high-value SP accounts require more protection against potential IdP misbehaviour*. This motivates the use of CM-based schemes, such as Firefox Sync or FIDO UAF, that manage user credentials on the user's own device.

SP's perspective. If relying on third-party IdPs, high-value SPs may have concerns about IdPs following necessary security requirements (e.g., maximum number of authentication attempts before account lock-out, use of additional authentication factors), increasing the importance of *SPs-Can-Filter-IdPs* and *Signals-Assurance-Level*. However, *No-Impersonation-by-Third-Party* would be a concern not only for users as discussed above, but also for SPs. This leaves SPs with the following alternatives that they may pursue:

1. **Credential vault.** SPs can encourage users to follow good password management practices, including the use of a password manager, but this is difficult to enforce. Many high-value SPs (including many major banks) encourage their customers (and sometimes have made it mandatory [109]) to install client-side software such as IBM Trusteer Rapport (currently installed on several hundreds of thousands of systems [6]), to defend against client-side threats such as phishing or malware. However, numerous online comments by users indicate that such tools (which may also be extended to, e.g., verify the presence of a suitable password manager on the user's device) often degrade client-side performance and stability. FIDO-certified UAF authenticators present a new opportunity for such institutions to enforce client-side security requirements while being less intrusive and more usable for end-users.

2. **FIS with strong IdP vetting.** High-value SPs that prefer a FIS scheme to

delegate authentication to third-party IdPs must coordinate extensively (e.g., by putting in place appropriate auditing mechanisms) to compensate for the lack of the *No-Impersonation-by-Third-Party* benefit. SecureKey Concierge is an example of such a scheme (with millions of users [173]), which has extensive technical requirements for IdPs, currently consisting of large Canadian financial institutions. Partnering with financial institutions as IdPs has the unique advantage of benefiting from their fraud detection mechanisms—compromise in banking credentials leading to fraudulent transactions is likely to be caught quickly (either by automated means, or by users who notice transactions they did not authorize) and would result in the bank issuing new credentials for users. This provides additional protection to SPs against long-term compromise of accounts.

6.4.2 Medium-Value SP Accounts

User's perspective. For medium-value accounts, such as online shopping or blogging platforms, users may be more concerned about SP misbehaviour such as spamming or misuse of profile information obtained from IdPs [191]. In other words, *more protection is required against SP misbehaviour* (as opposed to IdP misbehaviour; cf. Section 6.4.1), thereby increasing the importance of the *Unlinkable-Across-SPs* and *No-Sharing-of-User-Data* benefits. Protection against IdP misbehaviour is likely a lower priority to users for medium-value SP accounts: the majority of such SPs already communicate extensively with users by e-mail (e.g., sending invoices and other detailed account information), and the vast majority of such SPs already use e-mail based password reset [28]. Therefore, users already place extensive trust in their e-mail providers when managing their SP accounts. This motivates the use of e-mail providers as IdPs for medium-value SP accounts.

Deployment of FIS schemes that allow users to use their e-mail addresses as their online identity and that minimize sharing of user data are scarce, e.g., Mozilla Persona was not adopted by either IdPs or SPs. Moreover, even if such a scheme were to be widely deployed, it may eventually be superseded by a scheme that is less privacy-friendly, e.g., major OpenID 2.0 IdPs migrated [70] their users to OpenID Connect (which enables more extensive sharing of user profile information), thereby forcing

users to either accept the change or to discontinue use of their existing SP accounts and create new SP accounts. Such scenarios also raise the issue of the lack of *Portable-Identity-Across-IdP*, which anchors users to their IdPs; another example is if a user uses their mobile network operator (MNO) as their IdP (e.g., via Mobile Connect), it will be more cumbersome to switch their mobile plan to a different MNO, since the user will lose access to all their SP accounts. Considering all these drawbacks, we believe the optimal solution from a user's point of view is the same as for high-value accounts: CM-based schemes such as Firefox Sync or FIDO UAF.

SP's perspective. SPs have an incentive to push users towards FIS schemes, to reduce the effort required in the sign-up process (e.g., the user is not required to select a new password) and to gain access to users' social media circles. The latter benefit is especially appealing to smaller SPs that have limited resources to do their own collection of user data. Medium-value SPs are more likely to choose the schemes that maximize the size of their user base, and therefore many SPs that offer the option for FIS-based SSO also offer the option for users to create a "conventional" account by selecting a user name and password.

6.4.3 Detecting Impersonation by IdP

As discussed above in Section 6.4.2, many SPs allow users to reset their passwords by e-mail. This theoretically would allow e-mail providers to impersonate users by performing a password reset—however, this would cause the user to lose access to their account, and is therefore a detectable impersonation attempt. Detectability of IdP impersonation may provide a sufficient deterrent from doing so. Deterrence as a tool to hold trusted third-parties accountable can be found in other contexts as well, such as certificate transparency [111] for TLS certificate authorities (CAs).[6] An important distinction between the CA model and the FIS model is that CAs can generally issue certificates for *any* domain, whereas IdPs can only impersonate their own users (but not users of other IdPs). However, since the vast majority of the FIS market share is held by a small number of IdPs, a rogue IdP can still have a major impact; this motivates the deployment of countermeasures, or at least deterrence or accountability

[6]Certificate transparency requires TLS CAs to publicize every certificate that they digitally sign, enabling detection of maliciously-issued certificates.

measures, against impersonation. Below, we discuss how two of the schemes analyzed can be augmented with existing tools to offer detection of impersonation by IdPs. The trade-off associated with this benefit is that it requires SPs to maintain additional state information, a shared secret, or out-of-band channel (i.e., independent from the IdP) with the user.

Augmenting SAW

When users log into an SP with SAW, the SP e-mails an OTP to users. An administrator from within an e-mail provider could therefore impersonate any of their own users by initiating an authentication process to an SP and intercepting the OTP e-mailed from the SP to the user. A possible deterrent to this threat would be for the SP to use client-side cookies (and with TLS token binding [160] for added security against cookie theft) to recognize whether or not it is the first time that the user has logged in from a browser; if not, the SP can send an out-of-band notification to the user informing them (or requesting confirmation, for added security) that their account has been accessed from a new machine. The out-of-band notification could be sent via, e.g., a secondary e-mail address, SMS, a push notification directly to the user's browsers on their existing devices (via the W3C Push API [20]), or instant messaging to users' social media accounts.

Augmenting Mobile Connect

One of the available LoA3 authentication options in Mobile Connect involves using the Mobile Signature Service (MSS) [60] standard to generate and store an asymmetric key pair on the user's SIM card, to be used for authenticating the user's device to the mobile network operator (MNO) [78]. When users log into an SP via Mobile Connect, they are first redirected to their IdP (i.e., their MNO), which initiates an authentication process whereby a confirmation prompt is received on the user's mobile device, and the user's response (signed using the key stored on the SIM) is sent back to the IdP. The IdP then generates an access token and redirects the user back to their SP, completing the standard OpenID Connect protocol flow as discussed in Section 6.1.

An alternative Mobile Connect design offering resistance to impersonation by

IdPs could leverage the cryptographic capabilities of SIM cards (currently only used for *device-to-IdP* authentication, as described above) to implement a *device-to-SP* cryptographic challenge-response protocol. A possible implementation would be for the user device to generate unique site-specific cryptographic key pairs (or use a signature scheme that uses a single key as a seed to generate unique site-specific keys [89, 35]), similarly to FIDO UAF. However, IdPs could sign users' public keys, to attest to SPs that users' signing keys are generated on an IdP-authenticated SIM. If a user loses their SIM, the IdP could issue the user a new SIM; this would require the user's device to generate new keys for each SP on their new SIM. A disadvantage of this approach is that it requires a mechanism for key revocation, whereby IdPs inform SPs that users have lost their devices so that SPs can invalidate users' old keys and accept the newly-generated keys. An advantage of this alternative approach is that the key revocation and regeneration process enables IdPs to facilitate account recovery in the event that users lose their mobile phones (in contrast to schemes such as FIDO UAF, in which IdPs cannot facilitate account recovery, and which thereby require SPs to implement their own recovery mechanisms). With this alternative design, user impersonation by rogue IdPs would require revocation of targeted users' SP-specific keys, which would be easily detectable by users (i.e., they would lose access to their accounts, similar to the scenario of a rogue e-mail provider performing a password reset).

6.4.4 Configuration for Device-Based (T2) Schemes

As discussed above in Section 6.4.3, countermeasures to IdP impersonation may require additional protocol and implementation complexity for IdPs and SPs. On the other hand, SSO is also achievable through user-to-device (T2) authentication mechanisms, which eliminates the requirement for users and SPs to trust remote IdPs for authentication (T1); however, this requires additional client-side configuration effort. As demonstrated in Table 6.3, no device-based (T2) CM scheme (A5, A6) fully offers *No-Device-Setup*.

Developing usable device configuration mechanisms can be a challenge. For example, a very high proportion of Firefox Sync 1.5 (see Section 6.1.1) users had only set up a single device—a practice which would result in those users losing all their

passwords if they were to lose their device or had to re-install their operating system (assuming, as revealed by a high volume of user complaints [205], that most of those users had not backed up their password database encryption key). Moreover, many users did not understand the purpose of "pairing" (i.e., transferring the encryption key to) new devices. On the other hand, Firefox Sync 2.0 simplifies configuration by using a password-derived encryption key, which improves usability but introduces weakness against both online and offline guessing attacks.

FIDO UAF also requires configuration for new devices. However, since the cryptographic keys cannot be extracted and transferred to new devices, users must "enroll" their new device individually on each SP account. For example, Microsoft's SP developer guide [208] for application developers suggests that new devices can be enrolled by asking the user to authenticate via a password and a second factor such as an SMS or e-mailed OTP, after which the new device can generate a new cryptographic key pair and send the corresponding public key to the SP. While it is too early to tell with certainty what authentication mechanism the majority of SPs will use when enrolling new UAF devices, relying on password-based authentication would nullify most of the security and usability benefits of using UAF, since: (1) attackers would focus on breaking the password authentication option instead of breaking UAF authentication, and (2) users who already use a password manager to automatically fill in strong passwords would see no usability or security benefit in enrolling any UAF devices to their SP accounts.

6.5 Concluding Remarks and Recommendations

Our classification and analysis of 14 SSO schemes offers a big-picture overview of the state-of-the-art of web SSO; the usefulness of the criteria identified in our evaluation framework is validated by their ability to distinguish the strengths and weaknesses of different architectural solutions. Moreover, our framework serves as a decision-making tool to select an appropriate SSO scheme under the given circumstances by matching the set of needs and requirements with a scheme that offers the corresponding benefits. To our knowledge, this work is also the first in-depth treatment (since the taxonomy of Pashalidis and Mitchell [155]) presenting a combined comparative evaluation framework for CM and FIS schemes, which both address similar goals.

Below, we summarize some important insights and recommendations drawn from our comparative analysis in this chapter.

While none of the 14 schemes emerges as a clear winner for all use cases, based on our analysis we believe that password managers (i.e., CM-based SSO) such as Firefox Sync 2.0 emerge as the overall best scheme that offers users improved security over conventional password authentication, with minimal usability impact. While this may be the overall best option for users wishing to improve their online security, there are no widely-accepted mechanisms by which SPs can require their users to use a password manager. In situations where the SP must enforce higher security guarantees (and where the SP has the resources to do so), other schemes have seen some success; e.g., with sufficient co-ordination, large institutions such as governments, banks, and telecommunications companies can form partnerships to enable stronger authentication while imposing little usability burden on users, as illustrated by the success of a number of public-private partnerships around the world such as SecureKey in Canada, GOV.UK Verify in the UK [49, 73], and BankID in Sweden and Norway [165].

FIDO UAF offers promising security benefits, but necessitates SPs to offer users a backup authentication mechanism for enrolling new devices. In practice, we believe this will result in a wide variation in the overall security and usability benefits gained (based on the backup mechanism used). For example, Microsoft's recommendation [208] that SPs may fall back to password authentication may result in many FIDO UAF SPs offering little to no security advantages over password authentication (even usability may suffer, if users become more prone to forgetting their passwords due to infrequent use). Therefore, we believe that FIDO UAF in its current form only offers meaningful improvements to security if certain guidelines are followed: SPs should eliminate password-only authentication for enrolling new devices and instead rely on a more secure (but potentially less usable) mechanism, such as requiring newly-enrolled devices to be authorized from a device that is already enrolled. An account recovery scheme offering sufficient security must also be used—recall our justification from Section 5.3 that authentication schemes offering greater security must also be paired with account recovery schemes offering greater security, to deny attackers an alternative attack path with weaker security. FIDO UAF in particular

may also result in more frequent use of account recovery compared to other schemes—recall that UAF requires each device to be enrolled with each individual SP, making it highly likely that users may have many SP accounts with only a single device enrolled; device loss in such cases would require use of the recovery mechanism. In such cases, account recovery may require the availability of technical support staff who can verify the identity of the user in-person or by telephone—such level of service would be unrealistic to expect from most free online services, but may be possible for organizations such as educational or financial institutions.

We highlight that certain trade-offs must be made when designing an SSO scheme, based on the benefits being prioritized. Moreover, different stakeholders (namely users, IdPs, and SPs) may have conflicting priorities. For example, as discussed in Section 6.4.3, *No-Impersonation-by-Third-Party* is beneficial to users but more complex for IdPs and SPs to implement. We also observe that protocols offering "niche" benefits to users, such as OpenID 2.0 (offering the freedom to users to use any IdP at any SP) or Mozilla Persona (offering the ability to browse privately from IdPs) have not survived the market. Instead, OAuth based protocols have emerged as the most dominant, which may be due in part to their higher suitability towards providing access to users' social media data (a drawback for users, an appeal to SPs and IdPs).

Chapter 7

Discussion and Conclusion

Web authentication continues to be a challenge for all stakeholders involved, and for users especially. In 2015, the average user in the United States had 130 online accounts [30]; this number was projected to grow to 207 accounts by the year 2020, based on a historical annual growth rate of about 14%. It is unrealistic and unreasonable to expect that users will select and memorize a strong and unique password for each of their online accounts. As outlined in Chapter 1, this thesis pursued three goals to simultaneously address both the security and usability challenges that arise due to password fatigue and other disadvantages of password authentication as discussed. Below, we revisit these goals and discuss insights gained.

G1. We identified, developed, and evaluated device fingerprinting mechanisms for use alongside passwords, and offered guidance on their use, to enhance the security of password-based web authentication. The primary usability and deployability advantages of device fingerprinting when compared to other supplementary authentication mechanisms (e.g., single-use SMS codes) is that they do not require any user interaction, client-side software installation, or specialized client-side hardware.

G2. We expanded on the concept of mimicry resistance, a dimension that has thus far been largely overlooked in the study and design of web authentication schemes. We developed a comprehensive framework for evaluating the mimicry resistance of web authentication schemes and provided guidance on how to combine multiple schemes alongside password authentication to maximize the benefits gained.

G3. We performed a comprehensive analysis and evaluation of a broad range of SSO schemes, which reduce password fatigue by allowing users to access multiple services through a single master credential (most commonly, a master password). While a wide range of SSO schemes have been developed, prior to the work in this thesis there was a lack of clarity regarding the important distinctions between them and their practical implications. In this thesis, we provided clarity on which benefits are

offered by different schemes, which benefits are more desirable to different stakeholders (namely users, SPs, and IdPs), and which benefits are better suited to different environments or use-case scenarios (e.g., medium-value vs. high-value accounts).

We note that our goals and corresponding contributions as summarized above are complementary. The first two goals (G1 and G2) relate to enhancing the security of web authentication through the use of additional authentication mechanisms that do not impose a usability cost. The third goal (G3) relates to reducing the number of passwords that users are required to memorize. These two approaches can be used together to improve security. In fact, more widespread adoption of SSO (G3) can help accelerate adoption of new authentication mechanisms (G1 and G2), since SPs can rely on IdPs to implement mechanisms that SPs may not otherwise have the expertise to implement themselves.

7.1 Further Insights and Future Directions

Below, we highlight further insights gained throughout our work and identify promising directions for future work.

7.1.1 Continuous Authentication

The primary focus of this thesis is on start-of-session authentication, as opposed to continuous authentication (refer to Section 2.1 for the differences between these two approaches). However, Chapter 3 also discussed the use of device fingerprinting for continuous authentication throughout a session, to defend against session-hijacking attacks. We briefly summarize insights on the cause of session hijacking, countermeasures, and future work for continuous authentication on the web.

Session Hijacking

Session hijacking is a threat not only to conventional password authentication, but to all forms of web authentication used in practice. This is due to the ubiquitous reliance on HTTP session cookies to maintain authenticated sessions. Reliance on session cookies stems from the fact that HTTP is a stateless protocol—in other words, each HTTP request sent by a web browser to a server is processed independently from all

others. Session cookies enable web browsers to locally store an identifier (typically randomly-generated, to prevent guessing attacks) to include in each HTTP request as proof that an authenticated session has already been established for the user. In the absence of a session identifier, the web server processing the HTTP request would require the user to re-authenticate (e.g., by typing in their password). Session hijacking is enabled by the theft of session cookies by attackers (typically via cross-site scripting [152] or man-in-the-middle[1] attacks); this allows attackers to generate arbitrary HTTP requests (e.g., to perform unauthorized transactions) containing the session identifier of an existing authenticated session to bypass user authentication.

Session Hijacking Defenses

Session hijacking can be prevented by ensuring that the following two requirements are met:

1. All protocol messages should be exchanged between client and server over a cryptographically secure connection, to prevent the capture of any session state information that can be used to hijack the session.

2. Proof of user authentication should only be valid for the duration of the secure connection in which the user was authenticated.

While HTTPS (HTTP over TLS) ensures that the first requirement is met, it does not ensure that the second is met—instead of limiting the validity of proof-of-user-authentication (i.e., the session cookie) to a single secure connection,[2] most effort has focused on making it more difficult (but still possible) for attackers to steal session cookies. ChannelID token binding (discussed in Section 2.4.1) seeks to fulfill the second requirement by cryptographically binding session cookies to the TLS session in which they are created. However, as of the time of writing, standardization is still ongoing and the protocol is currently only supported by Google Chrome [43].

Future work could further explore how device fingerprinting can be used to defend against session hijacking (by fulfilling the second requirement above), both as a more

[1] This includes several categories of attacks that involve relaying or eavesdropping on the communication (e.g., via a wireless access point under the control of an attacker) between the client and server.

[2] Note that a TLS connection is not limited to a single browsing session, since the same connection can be resumed when users visit the same website at a later time [171].

immediately-applicable countermeasure that is compatible with all current browsers, and to investigate whether additional benefits can be provided if used alongside ChannelID.

Future Work on Continuous Authentication

In addition to device fingerprinting, other approaches can be investigated for continuous authentication on the web. Chapter 2 discussed behavioural biometrics, which is outside the scope of this thesis but underexplored for web authentication. Moreover, continuous authentication can be used not only to protect against session hijacking attacks, but also to lock out attackers in the event of, e.g., device theft or credential theft. For example, Google has developed a number of basic heuristics to detect attempted attacker takeover of user accounts [130]. Note that these heuristics are based on attacker behaviour that occurs *after* the start-of-session authentication. For example, a commonly flagged behaviour pattern is for attackers to log into an e-mail account, send phishing e-mails to all the user's contacts, and create an e-mail filter to delete all incoming e-mails containing the word "hacked" (in case any contact attempts to inform the user that their account has been hacked). Such mechanisms are complementary to the work in this thesis and therefore offer an additional layer of security, warranting further exploration.

7.1.2 Further Development of Device Fingerprinting

Various avenues for further study on device fingerprinting were identified throughout this thesis. For example, some fingerprinting vectors identified, such as clock skew fingerprinting, should be further studied to determine the feasibility of mimicry attacks and possible defenses. The proposed vector for measuring the amount of GPU memory, could be extended to detect additional GPU hardware parameters; it could also be used as the basis of a challenge-response protocol to improve mimicry resistance.

There are also practical issues that need to be addressed, such as how device fingerprints should be stored server-side—storing a hash of the device fingerprint only allows servers to check for an exact match, but storing detailed information in plaintext may raise privacy concerns. Techniques used for biometric template storage or other implicit authentication schemes may offer further insights that can

be applied to device fingerprint storage. Other practical issues include determining the ideal granularity level to use for different fingerprinting vectors if an exact match is not required (e.g., for geolocation).

7.1.3 Device Fingerprinting Through Native Applications

In addition to web applications (the focus of this thesis), device fingerprinting can also be integrated into native applications[3] for various purposes. For example, as discussed in Section 3.4.3, account recovery can be done through device fingerprinting applications built using trusted computing technologies. Device fingerprinting performed by such applications would offer greater mimicry resistance compared to browser-based device fingerprinting, but would be less convenient for users (a reasonable trade-off for account recovery), since users would be required to download and run an application on their machine.

Device fingerprinting can also be used by native smartphone applications for bilateral authentication, as is done with Sound-Proof [99]. For example, the verifying server may determine whether the user's smartphone is connected to the same network as the access device, or may geolocate both the smartphone and access device to determine that they are in close proximity. Such fingerprinting vectors may be used independently or in conjunction with existing techniques such as Sound-Proof to offer greater security.

7.1.4 Improved Single Sign-On Architectures

In Section 6.4.3, we provided high-level descriptions of design changes to two SSO schemes (SAW and Mobile Connect) to make IdP impersonation of users detectable. Further exploration could identify other design changes that can gain different benefits for other SSO schemes, and the associated trade-offs (if any). For example, Intel SGX [90] (or other trusted computing technologies) may have useful applications to SSO protocols, particularly for privacy-related benefits (B12-B14, Section 6.2.1) since it allows clients to verify the integrity of code that is being executed server-side.

Future SSO schemes could be designed to offer new benefits that are not yet offered

[3] A native application is one that runs directly on the host operating system, as opposed to running within another application such as a web browser.

by any current schemes. For example, none of the schemes evaluated in Chapter 6 offer a mechanism for secure account sharing. Account sharing is a challenge even with conventional password-based authentication: while password-sharing is a discouraged practice, users often have no other choice if they wish to share their account with others.

Bibliography

[1] A. Abdou, A. Matrawy, and P. C. van Oorschot. Accurate manipulation of delay-based Internet geolocation. In *ACM AsiaCCS*, pages 887–898, 2017.

[2] A. Abdou, A. Matrawy, and P. C. van Oorschot. CPV: Delay-based location verification for the Internet. *IEEE TDSC*, 14(2):130–144, 2017.

[3] A. Abdou and P. C. van Oorschot. Server location verification (SLV) and server location pinning: Augmenting TLS authentication. *ACM Trans. Privacy and Security (TOPS)*, 21(1):1:1–1:26, January 2018.

[4] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *ACM CCS*, pages 674–689, 2014.

[5] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the web for fingerprinters. In *ACM CCS*, pages 1129–1140, 2013.

[6] E. Aharoni, R. Peleg, R. S., and T. Salman. Identifying malicious activities from system execution traces. *IBM Journal of Research and Development*, pages 5:1–5:7, 2016.

[7] F. Alaca and P. C. van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *ACM ACSAC*, pages 289–301, 2016.

[8] M. Alsaleh, M. Mannan, and P. C. van Oorschot. Revisiting defenses against large-scale online password guessing attacks. *IEEE TDSC*, 2012.

[9] R. Amadeo. Google makes it much easier to use 2FA on your account. https://arstechnica.com/gadgets/2016/06/googles-new-two-factor-authentication-system-tap-yes-to-log-in/, June 2016. Accessed: 2018-04-29.

[10] C. Arackaparambil, S. Bratus, A. Shubina, and D. Kotz. On the reliability of wireless fingerprinting using clock skews. In *ACM WiSec*, pages 169–174, 2010.

[11] M. Atwood, R. M. Conlan, B. Cook, L. Culver, K. Elliott-McCrea, L. Halff, E. Hammer-Lahav, B. Laurie, C. Messina, J. Panzer, et al. OAuth core 1.0. http://oauth.net/core/1.0/, December 2007. Accessed: 2018-04-29.

[12] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In *Security Protocols Workshop*, pages 170–177. Springer, 2000.

[13] M. D. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet and Web Information Systems*, 2011.

[14] D. Balfanz. FIDO TechNotes: Channel binding and FIDO. https://fidoalliance.org/fido-technotes-channel-binding-and-fido/, 2016. Accessed: 2018-04-29.

[15] D. Balfanz and R. Hamilton. Transport layer security (TLS) channel IDs. Internet-Draft (IETF), June 2013. https://tools.ietf.org/html/draft-balfanz-tls-channelid-01. Accessed: 2018-04-29.

[16] L. Ballard, F. Monrose, and D. P. Lopresti. Biometric authentication revisited: Understanding the impact of wolves in sheep's clothing. In *USENIX Security Symp.*, 2006.

[17] S. M. Bellovin. A technique for counting NATted hosts. In *ACM SIGCOMM Workshop on Internet Measurement*, pages 267–272. ACM, 2002.

[18] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, and B. Aboba. WebRTC 1.0: Real-time communication between browsers. https://www.w3.org/TR/webrtc/, 2016. Accessed: 2016-04-25.

[19] D. B. Berrueta. A practical approach for defeating Nmap OS-fingerprinting. https://nmap.org/misc/defeat-nmap-osdetect.html, 2003. Accessed: 2016-09-01.

[20] P. Beverloo, M. Thomson, M. van Ouwerkerk, B. Sullivan, and E. Fullea. Push API. https://www.w3.org/TR/push-api, 2017. Accessed: 2018-04-29.

[21] R. Beverly, R. Koga, and K. Claffy. Initial longitudinal analysis of IP source spoofing capability on the Internet. ISOC whitepaper, 2013.

[22] V. Bharadwaj, H. Le Van Gong, D. Balfanz, A. Czeskis, A. Birgisson, J. Hodges, M. B. Jones, R. Lindemann, and J. Jones. Web authentication: An API for accessing public key credentials. https://www.w3.org/TR/webauthn, 2018. Accessed: 2018-04-29.

[23] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh. Mobile device identification via sensor fingerprinting. In *arXiv preprint arXiv:1408.1416 [cs.CR]*, 2014.

[24] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *IEEE Symp. Security & Privacy*, pages 538–552. IEEE, 2012.

[25] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symp. Security & Privacy*, pages 553–567, 2012.

[26] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. Passwords and the evolution of imperfect authentication. *Communications of the ACM*, 58(7):78–87, 2015.

[27] J. Bonneau, M. Just, and G. Matthews. What's in a name? Evaluating statistical attacks on personal knowledge questions. In *Financial Cryptography and Data Security*, pages 98–113, 2010.

[28] J. Bonneau and S. Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.

[29] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor authentication: Somebody you know. In *ACM CCS*, pages 168–178, 2006.

[30] T. L. Bras. Online overload – it's worse than you thought. https://blog.dashlane.com/infographic-online-overload-its-worse-than-you-thought/, 2015. Accessed: 2018-01-29.

[31] L. Brotherston. Stealthier attacks and smarter defending with TLS fingerprinting. https://n0where.net/tls-fingerprinting/, 2015. Accessed: 2016-03-11.

[32] BrowserLeaks.com. https://www.browserleaks.com, 2011. Accessed: 2016-05-25.

[33] E. Bursztein, A. Malyshev, T. Pietraszek, and K. Thomas. Picasso: Lightweight device class fingerprinting for web clients. In *ACM SPSM*, pages 93–102, 2016.

[34] D. Callahan. Transitioning Persona to community ownership. https://web.archive.org/web/20141205135721/http://identity.mozilla.com/post/78873831485/transitioning-persona-to-community-ownership, 2014. Accessed: 2017-12-21.

[35] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks*, volume 2576 of *LNCS*, pages 268–289. Springer, 2003.

[36] CANARIE. Identity and access management: CAF. https://www.canarie.ca/identity/. Accessed: 2018-04-29.

[37] Y. Cao, S. Li, and E. Wijmans. (Cross-)browser fingerprinting via OS and hardware level features. In *NDSS*, 2017.

[38] M. Casado and M. J. Freedman. Peering through the shroud: The effect of edge opacity on IP-based client identification. In *NSDI*, 2007.

[39] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague. OAuth demystified for mobile application developers. In *ACM CCS*, pages 892–903, 2014.

[40] G. Chen. Convenience over safety: How authentication cookies compromise user account security on the web. http://randomwalker.info/advising/undergraduate/chen-independent-work.pdf, 2014. Accessed: 2016-05-01.

[41] S. Chiasson, E. Stobert, A. Forget, R. Biddle, and P. C. van Oorschot. Persuasive cued click-points: Design, implementation, and evaluation of a knowledge-based authentication mechanism. *IEEE TDSC*, 9(2):222–235, 2012.

[42] S. Chiasson and P. C. van Oorschot. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography*, 77(2-3):401–408, 2015.

[43] Chrome Platform Status. Token binding. https://www.chromestatus.com/feature/5097603234529280, June 2017. Accessed: 2018-01-29.

[44] M. D. Corner and B. D. Noble. Zero-interaction authentication. In *Proc. 8th Annu. Int. Conf. Mobile Computing Networking*, pages 1–11, 2002.

[45] L. Daigle. WHOIS protocol specification. RFC 3912 (IETF), September 2004.

[46] M. L. Damiani, C. Silvestri, and E. Bertino. Fine-grained cloaking of sensitive positions in location-sharing applications. *IEEE Pervasive Computing*, 10:64–72, 2011.

[47] A. Das, N. Borisov, and M. Caesar. Tracking mobile users through motion sensors: Attacks and defenses. In *NDSS*, 2016.

[48] P. Davis, N. Sakimura, M. Lindelsee, and G. Wachob. Extensible resource identifier (XRI) syntax v2.0. https://www.oasis-open.org/committees/download.php/15376/xri-syntax-V2.0-cs.html, 2005. Accessed: 2018-04-29.

[49] Deloitte. A blueprint for digital identity: The role of financial institutions in building digital identity. World Economic Forum, 2016.

[50] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. *Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible?*, pages 451–475. Springer, 2014.

[51] D. E. Denning and P. F. MacDoran. Location-based authentication: Grounding cyberspace for better security. *Computer Fraud & Security*, 1996(2):12–16, 1996.

[52] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *USENIX Security Symp.*, 2012.

[53] M. Dietz and D. S. Wallach. Hardening Persona – improving federated web login. In *NDSS*, 2014.

[54] Z. Dong, R. D. W. Perera, R. Chandramouli, and K. P. Subbalakshmi. Network measurement based modeling and optimization for IP geolocation. *Computer Networks*, 56(1):85–98, 2012.

[55] Duo Security. Secure two-factor authentication app. https://duo.com/product/trusted-users/two-factor-authentication/duo-mobile. Accessed: 2018-04-29.

[56] P. Eckersley. How unique is your web browser? In *Proc. Privacy Enhancing Technologies Symp.*, pages 1–18, 2010.

[57] O. Eisen. Method and system for identifying users and detecting fraud by use of the Internet. US Patent, Dec. 2010. US 7853533 B2.

[58] O. Eisen. Systems and methods for detection of session tampering and fraud prevention. US Patent, Apr. 2012. US 8151327 B2.

[59] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *ACM CCS*, pages 1388–1401, 2016.

[60] ETSI. TR 102 203 – mobile commerce (M-COMM); mobile signatures; business and functional requirements, 2003.

[61] D. Fett, R. Kusters, and G. Schmitz. An expressive model for the web infrastructure: Definition and application to the BrowserID SSO system. In *IEEE Symp. Security & Privacy*, pages 673–688, 2014.

[62] D. Fett, R. Küsters, and G. Schmitz. The web SSO standard OpenID Connect: In-depth formal security analysis and security guidelines. In *IEEE Computer Security Foundations Symp. (CSF)*, pages 189–202, 2017.

[63] D. Fifield and S. Egelman. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security*, volume 8975 of *LNCS*, pages 107–124. Springer, 2015.

[64] I. O. for Standardization. ISO/IEC 29115: Information technology – security techniques – entity authentication assurance framework, 2013.

[65] FraudFox. http://www.wickybay.xyz/2016/01/fraudfox.html, 2016. Accessed: 2016-05-25.

[66] D. M. Freeman, M. Dürmuth, and B. Biggio. Who are you? A statistical approach to measuring user authenticity. In *NDSS*, 2016.

[67] P. Gill, Y. Ganjali, B. Wong, and D. Lie. Dude, where's that IP?: Circumventing measurement-based IP geolocation. In *USENIX Security Symp.*, pages 241–256, 2010.

[68] P. Golle and D. Wagner. Cryptanalysis of a cognitive authentication scheme. In *IEEE Symp. Security & Privacy*, 2007.

[69] D. Goodin. Thieves drain 2FA-protected bank accounts by abusing SS7 routing protocol. https://arstechnica.com/security/2017/05/thieves-drain-2fa-protected-bank-accounts-by-abusing-ss7-routing-protocol/, May 2017.

[70] Google. Migrating from OpenID 2.0 to OpenID Connect. https://developers.google.com/identity/protocols/OpenID2Migration, 2016. Accessed: 2018-04-29.

[71] Google Chrome Help. WebRTC 1.0: Real-time communication between browsers. https://support.google.com/chrome/answer/165139?hl=en. Accessed: 2017-05-23.

[72] Google Identity Platform. OpenID Connect. https://developers.google.com/identity/protocols/OpenIDConnect, 2018. Accessed: 2018-04-29.

[73] GOV.UK Verify. How GOV.UK Verify works. https://www.gov.uk/government/publications/introducing-govuk-verify/introducing-govuk-verify, 2017. Accessed: 2018-04-29.

[74] P. A. Grassi, M. Garcia, and J. Fenton. Digital identity guidelines. NIST Special Publication 800-63-3, June 2017.

[75] P. A. Grassi, J. P. Richer, S. K. Squire, J. L. Fenton, E. M. Nadeau, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, and M. F. Theofanos. Digital identity guidelines: Federation and assertions. NIST Special Publication 800-63-3C, June 2017.

[76] E. Grosse and M. Upadhyay. Authentication at scale. *IEEE Security & Privacy*, 11(1):15–22, Jan. 2013.

[77] GSM Association. CPAS04 authenticator options, November 2015.

[78] GSM Association. Official document PDATA.01 - OpenID Connect Mobile Connect profile. https://www.gsma.com/latinamerica/wp-content/uploads/2016/06/techdoc-MC-OpenID_Connect_Mobile_Connect_Profile-1.pdf, 2015. Accessed: 2018-04-29.

[79] GSM Association. Mobile Connect API. https://developer.mobileconnect.io/mobile-connect-api, 2016. Accessed: 2018-04-29.

[80] GSM Association. User lifecycle best practices. https://developer.mobileconnect.io/user-lifecycle-best-practices, 2016. Accessed: 2018-04-29.

[81] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of Internet hosts. *IEEE/ACM Trans. Networking*, 14(6):1219–1232, 2006.

[82] M. Hanson, D. Mills, and B. Adida. Federated browser-based identity using email addresses. In *W3C Workshop on Identity in the Browser*, 2011.

[83] F. Hao and P. Ryan. J-PAKE: Authenticated key exchange without PKI. *Transactions on Computational Science XI*, pages 192–206, 2010.

[84] D. Hardt. The OAuth 2.0 authorization framework. RFC 6749 (IETF), October 2012.

[85] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security Symp.*, pages 205–220, 2012.

[86] C. Herley and P. C. van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012.

[87] N. J. Hopper and M. Blum. Secure human identification protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 52–66. Springer, 2001.

[88] D.-J. Huang, W.-C. Teng, C.-Y. Wang, H.-Y. Huang, and J. M. Hellerstein. Clock skew based node identification in wireless sensor networks. In *IEEE GLOBECOM*, 2008.

[89] IBM Research Zurich. Identity mixer—a cryptographic algorithm to protect your privacy. https://www.zurich.ibm.com/identity_mixer/howitworks.html. Accessed: 2018-04-28.

[90] Intel Corp. Intel software guard extensions programming reference. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014. Accessed: 2018-04-29.

[91] Intrinsic ID. https://www.intrinsic-id.com/, 2017. Accessed: 2017-05-14.

[92] IT Security Solutions. Fingerprinting browsers using protocol handlers. http://itsecuritysolutions.org/2010-03-29_fingerprinting_browsers_using_protocol_handlers/, 2010. Accessed: 2016-04-25.

[93] ITU-T. X.1254 – Cyberspace security – identity management, 2012.

[94] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit authentication for mobile devices. In *USENIX HotSec*, 2009.

[95] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Trans. Mobile Computing*, 9(3):449–462, March 2010.

[96] R. Jhawar, P. Inglesant, N. Courtois, and M. A. Sasse. Make mine a quadruple: Strengthening the security of graphical one-time PIN authentication. In *IEEE Network and System Security (NSS)*, pages 81–88, 2011.

[97] M. Jones, J. Bradley, and N. Sakimura. JSON web token (JWT). RFC 7519 (IETF), May 2015.

[98] B. Kaliski. PKCS #5: Password-based cryptography specification version 2.0. RFC 2898 (IETF), September 2000.

[99] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun. Sound-proof: Usable two-factor authentication based on ambient sound. In *USENIX Security Symp.*, pages 483–498, 2015.

[100] Keybase. Keybase documentation. https://keybase.io/docs, 2017. Accessed: 2018-04-29.

[101] H. Khan, A. Atwater, and U. Hengartner. A comparative evaluation of implicit authentication schemes. In *Research in Attacks, Intrusions and Defenses*, volume 8688 of *LNCS*, pages 255–275. Springer, 2014.

[102] J. Kirk. This tool makes it easier for thieves to empty bank accounts. PCWorld: http://www.pcworld.com/article/2872372/this-tool-may-make-it-easier-for-thieves-to-empty-bank-accounts.html, 2015. Accessed: 2016-05-25.

[103] Z. Kleinman. Politician's fingerprint 'cloned from photos' by hacker. http://www.bbc.com/news/technology-30623611, 2014. Accessed: 2018-01-04.

[104] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE TDSC*, 2(2):93–108, 2005.

[105] D. P. Kormann and A. D. Rubin. Risks of the Passport single signon protocol. *Computer Networks*, 33(1):51–58, 2000.

[106] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *NDSS*, 2015.

[107] H. Krawczyk and P. Eronen. HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869 (IETF), May 2010.

[108] K. Krawiecka, A. Kurnikov, A. Paverd, M. Mannan, and N. Asokan. SafeKeeper: Protecting web passwords using trusted execution environments. *WWW*, 2018.

[109] B. Krebs. A closer look at Rapport from Trusteer. https://krebsonsecurity.com/2010/04/a-closer-look-at-rapport-from-trusteer/, 2010. Accessed: 2018-04-29.

[110] S. Laki, P. Mátray, P. Hága, T. Sebők, I. Csabai, and G. Vattay. Spotter: A model based active geolocation service. In *IEEE INFOCOM*, pages 3173–3181, 2011.

[111] A. Langley, E. Kasper, and B. Laurie. Certificate transparency. RFC 6962 (IETF), June 2013.

[112] P. Laperdrix, B. Baudry, and V. Mishra. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *ESSoS*, 2017.

[113] P. Laperdrix, W. Rudametkin, and B. Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE Symp. Security & Privacy*, 2016.

[114] D. Li, J. Chen, C. Guo, Y. Liu, J. Zhang, Z. Zhang, and Y. Zhang. IP-geolocation mapping for moderately connected Internet regions. *IEEE Trans. Parallel and Distributed Systems, TPDS*, 24(2):381–391, 2013.

[115] Z. Li, W. He, D. Akhawe, and D. Song. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security Symp.*, pages 465–479, 2014.

[116] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[117] S. Machani, R. Philpott, S. Srinivas, J. Kemp, and J. Hodges. FIDO UAF architectural overview. https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-overview-v1.0-ps-20141208.html, December 2014. Accessed: 2016-07-07.

[118] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich. SoK: Single sign-on security – an evaluation of OpenID Connect. In *IEEE EuroS&P*, 2017.

[119] M. Mannan and P. C. van Oorschot. Leveraging personal devices for stronger password authentication from untrusted computers. *Journal of Computer Security*, 19(4):703–750, 2011.

[120] S. Mare, A. Molina-Markham, C. Cornelius, R. Peterson, and D. Kotz. Zebra: Zero-effort bilateral recurring authentication. In *IEEE Symp. Security & Privacy*, pages 705–720, 2014.

[121] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Smartphones as practical and secure location verification tokens for payments. In *NDSS*, 2014.

[122] M. Marlinspike. Technology preview: Private contact discovery for Signal. https://signal.org/blog/private-contact-discovery/, September 2017. Accessed: 2018-04-29.

[123] Maxmind Developer Site. Device tracking add-on for minFraud and proxy detection services. http://dev.maxmind.com/minfraud/device/, 2017. Accessed: 2017-05-14.

[124] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symp. Security & Privacy*, pages 413–427, 2012.

[125] D. McCarney. Password managers: Comparative evaluation, design, implementation and empirical analysis. Master's thesis, Carleton University, 2013.

[126] A. M. McDonald and L. F. Cranor. A survey of the use of Adobe Flash local shared objects to respawn HTTP cookies. *ISJLP*, 7:639–687, 2011.

[127] MDN web docs. WebGL. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed: 2015-11-23.

[128] MDN web docs. Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy, 2018. Accessed: 2018-04-29.

[129] MDN web docs. Window.postMessage(). https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage, 2018. Accessed: 2018-04-29.

[130] G. Milka. Anatomy of account takeover. In *USENIX Enigma*, 2018.

[131] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In *Web 2.0 Security & Privacy*, 2011.

[132] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Web 2.0 Security & Privacy*, 2012.

[133] Mozilla. Lockbox FAQ. https://mozilla-lockbox.github.io/lockbox-extension/faqs/, 2017. Accessed: 2018-04-29.

[134] Mozilla Corporation. Use bookmarks, tabs and passwords across devices. https://www.mozilla.org/en-US/firefox/sync/, 2018. Accessed: 2018-04-28.

[135] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. HOTP: An HMAC-based one-time password algorithm. RFC 4226 (IETF), December 2005.

[136] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-based one-time password algorithm. RFC 6238 (IETF), May 2011.

[137] J. A. Muir and P. C. van Oorschot. Internet geolocation: Evasion and counterevasion. *ACM Computing Surveys*, 42(1), 2009.

[138] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, and E. Weippl. Fast and reliable browser identification with JavaScript engine fingerprinting. In *Web 2.0 Security & Privacy*, 2013.

[139] S. J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *ACM CCS*, pages 27–36, 2006.

[140] S. J. Murdoch. Introduction to trusted execution environments. http://sec.cs.ucl.ac.uk/users/smurdoch/talks/rhul14tee.pdf, 2014. Accessed: 2018-04-29.

[141] Nanjee. http://www.nanjee.net/, 2017.

[142] S. Nettle, S. O'Neil, and P. Lock. PassWindow: A new solution to providing second factor authentication. *VEST Corporation*, 2009.

[143] C. Neuman, S. Hartman, and K. Raeburn. The Kerberos network authentication service (v5). RFC 4120 (IETF), July 2005.

[144] A. J. Nicholson, M. D. Corner, and B. D. Noble. Mobile device security using transient authentication. *IEEE Trans. Mobile Computing*, 5(11):1489–1502, November 2006.

[145] N. Nikiforakis, W. Joosen, and B. Livshits. PriVaricator: Deceiving fingerprints with little white lies. In *WWW*, 2015.

[146] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symp. Security & Privacy*, pages 541–555, 2013.

[147] Nitrokey. USB dongle auth list. http://www.dongleauth.info/dongles/, 2017. Accessed: 2017-05-19.

[148] J. C. Norte. Advanced Tor browser fingerprinting. http://jcarlosnorte.com/security/2016/03/06/advanced-tor-browser-fingerprinting.html, Mar. 2016. Accessed: 2016-03-11.

[149] OASIS. OASIS standards. https://www.oasis-open.org/standards, 2017. Accessed: 2017-12-19.

[150] OCLC. EZproxy features. http://www.oclc.org/en/ezproxy/features.html, 2018. Accessed: 2018-03-12.

[151] L. Olejnik, G. Acar, C. Castelluccia, and C. Diaz. The leaking battery: A privacy analysis of the HTML5 Battery Status API. *IACR Cryptology ePrint Archive*, 2015.

[152] OWASP. Cross-site scripting (XSS). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), 2018. Accessed: 2018-04-29.

[153] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[154] S. Park, J. N. Yoon, C. Kang, K. H. Kim, and T. Han. Tgvisor: A tiny hypervisor-based trusted geolocation framework for mobile cloud clients. In *IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 99–108, 2015.

[155] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In *Proc. 8th Australasian Conf. Inform. Security Privacy*, pages 249–264, 2003.

[156] A. Pashalidis and C. J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. In *GLOBECOM*, December 2004.

[157] PassWindow. Two factor authentication cards for secure online login. https://www.passwindow.com, August 2017.

[158] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. IP geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[159] A. Popescu. Geolocation API specification. *World Wide Web Consortium, Candidate Recommendation CR-geolocation-API-20100907*, 2010.

[160] A. Popov, D. Balfanz, A. Langley, and J. Hodges. The token binding protocol version 1.0. Internet-Draft (IETF), October 2017. https://tools.ietf.org/html/draft-ietf-tokbind-protocol. Accessed: 2018-04-29.

[161] N. Popper. Identity thieves hijack cellphone accounts to go after virtual currency. https://www.nytimes.com/2017/08/21/business/dealbook/phone-hack-bitcoin-virtual-currency.html, August 2017. Accessed: 2018-04-29.

[162] D. Preuveneers and W. Joosen. SmartAuth: Dynamic context fingerprinting for continuous user authentication. In *ACM SAC*, pages 2185–2191, 2015.

[163] D. Recordon and B. Fitzpatrick. OpenID authentication 2.0. http://openid.net/specs/openid-authentication-2_0.html, December 2007. Accessed: 2018-04-29.

[164] D. Recordon and D. Reed. OpenID 2.0: A platform for user-centric identity management. In *Proc. ACM Workshop on Digital Identity Management*, pages 11–16, 2006.

[165] D. Rennie. Creating test environments with the private sector. https://identityassurance.blog.gov.uk/2017/02/03/creating-test-environments-with-the-private-sector/, 2017. Accessed: 2018-04-29.

[166] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos. Progressive authentication: Deciding when to authenticate on mobile phones. In *USENIX Security Symp.*, pages 301–316, 2012.

[167] D. Roesler. STUN IP address requests for WebRTC. https://github.com/diafygi/webrtc-ips, 2015. Accessed: 2016-04-25.

[168] U. Ruhrmair and D. E. Holcomb. PUFs at a glance. In *Design, Automation and Test in Europe (DATE)*, pages 1–6. IEEE, 2014.

[169] E. Sachs. Usability research on federated login. https://sites.google.com/site/oauthgoog/UXFedLogin, 2008. Accessed: 2018-04-29.

[170] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect core 1.0. http://openid.net/specs/openid-connect-core-1_0.html, 2014. Accessed: 2018-04-29.

[171] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) session resumption without server-side state. RFC 5077 (IETF), January 2008.

[172] B. Schneier. NIST is no longer recommending two-factor authentication using SMS. https://www.schneier.com/blog/archives/2016/08/nist_is_no_long.html, August 2016.

[173] SecureKey. The Government of the Northwest Territories simplifies authentication with SecureKey Concierge. https://securekey.com/blog/government-northwest-territories-simplifies-authentication-securekey-concierge/, 2014. Accessed: 2018-04-29.

[174] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[175] Shibboleth Consortium. Shibboleth consortium – privacy preserving identity management. https://www.shibboleth.net, 2017. Accessed: 2017-12-19.

[176] Shibboleth Wiki. Shibboleth concepts: Flows and config. https://wiki.shibboleth.net/confluence/display/CONCEPT/FlowsAndConfig, 2017. Accessed: 2018-04-29.

[177] D. Silver, S. Jana, E. Chen, C. Jackson, and D. Boneh. Password managers: Attacks and defenses. In *USENIX Security Symp.*, pages 449–464, 2014.

[178] S. Sivakorn, I. Polakis, and A. D. Keromytis. The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In *IEEE Symp. Security & Privacy*, 2016.

[179] SK-UN117. Trust framework—SecureKey Concierge in Canada. http://securekey.com/wp-content/uploads/2015/09/SK-UN117-Trust-Framework-SecureKey-Concierge-Canada.pdf, September 2015. Accessed: 2018-04-29.

[180] S. L. Smith. Authenticating users by word association. *Computers & Security*, 6(6):464–470, 1987.

[181] P. Snyder, C. Taylor, and C. Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *ACM CCS*, pages 179–194, 2017.

[182] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[183] J. Spooren, D. Preuveneers, and W. Joosen. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proc. European Workshop on System Security*, 2015.

[184] Stack Overflow. Is it possible to detect visitor DNS server? http://stackoverflow.com/questions/10721731/is-it-possible-to-detect-visitor-dns-server. Accessed: 2015-10-27.

[185] W. Stallings and L. Brown. *Computer Security: Principles and Practice*. Pearson Education, 2 edition, 2012.

[186] E. Stobert and R. Biddle. The password life cycle: User behaviour in managing passwords. In *SOUPS*, 2014.

[187] S.-T. Sun. *Towards Improving the Security and Usability of Web Single Sign-On Systems*. PhD thesis, University of British Columbia, 2013.

[188] S.-T. Sun and K. Beznosov. The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems. In *ACM CCS*, pages 378–390, 2012.

[189] S.-T. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov. A billion keys, but few locks: The crisis of web single sign-on. In *New Security Paradigms Workshop*, pages 61–72, 2010.

[190] S.-T. Sun, K. Hawkey, and K. Beznosov. Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4):465–483, 2012.

[191] S.-T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov. What makes users refuse web single sign-on? An empirical investigation of OpenID. In *SOUPS*, 2011.

[192] The 41st Parameter. The 41st parameter announces new real-time product as world's first standard for online PC identification. http://www.the41.com/buzz/announcements/41st-parameter-announces-new-real-time-product-worlds-first-standard-online-pc-0, July 2006. Accessed: 2017-05-14.

[193] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun. On the requirements for successful GPS spoofing attacks. In *ACM CCS*, pages 75–86, 2011.

[194] T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E. Weippl. SHPF: Enhancing HTTP(S) session security with browser fingerprinting. In *Proc. 8th Int. Conf. Availability, Reliability and Security*, pages 255–261, Sept. 2013.

[195] T. W. van der Horst and K. E. Seamons. Simple authentication for the web. In *Proc. 3rd Int. Conf. Security Privacy Commun. Networks*, pages 473–482, 2007.

[196] T. Van Goethem, W. Scheepers, D. Preuveneers, and W. Joosen. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *Engineering Secure Software and Systems*, volume 9639 of *LNCS*, pages 106–121. Springer, 2016.

[197] P. C. van Oorschot and S. G. Stubblebine. On identity theft and a countermeasure based on digital uniqueness and location cross-checking. In *Financial Cryptography and Data Security*, 2005. See also extended version: Technical report TR-05-12, School of Computer Science, Carleton University, Canada.

[198] T. E. Varghese, J. B. Fisher, S. L. Harris, and D. B. Durai. System and method for fraud monitoring, detection, and tiered user authentication. US Patent, Mar. 2011. US 7908645 B2.

[199] V. Vasilyev. fingerprintjs2. https://github.com/Valve/fingerprintjs2, 2015. Accessed: 2016-05-25.

[200] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy. FP-STALKER: Tracking browser fingerprint evolutions. In *IEEE Symp. Security & Privacy*, 2018.

[201] Visa. See how Visa's new mobile location confirmation works. https://usa.visa.com/visa-everywhere/innovation/visa-mobile-location-confirmation.html, 2018. Accessed: 2018-01-04.

[202] D. Wagner and R. Dean. Intrusion detection via static analysis. In *IEEE Symp. Security & Privacy*, pages 156–168, 2001.

[203] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *ACM CCS*, pages 255–264, 2002.

[204] R. Wang, Y. Zhou, S. Chen, S. Qadeer, D. Evans, and Y. Gurevich. Explicating SDKs: Uncovering assumptions underlying secure authentication and authorization. In *USENIX Security Symp.*, pages 399–314, 2013.

[205] B. Warner. CloudFlare meet-up: Brian Warner from Mozilla talks cryptography. https://www.youtube.com/watch?v=G16rOGmpBUc, 2014. Accessed: 2017-05-23.

[206] B. Warner. Firefox Accounts/Sync Protocol. https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol, 2015. Accessed: 2017-05-23.

[207] D. Weinshall. Cognitive authentication schemes safe against spyware. In *IEEE Symp. Security & Privacy*, 2006.

[208] Windows Dev Center. Windows Hello. https://docs.microsoft.com/en-us/windows/uwp/security/microsoft-passport, 2017. Accessed: 2018-04-29.

[209] M.-D. M. Yu and S. Devadas. Pervasive, dynamic authentication of physical items. *Communications of the ACM*, 14(6), 2017.

[210] M. Zalewski. p0f v3. http://lcamtuf.coredump.cx/p0f3/. Accessed: 2015-10-26.

[211] Y. Zhang, F. Monrose, and M. K. Reiter. The security of modern password expiration: An algorithmic framework and empirical analysis. In *ACM CCS*, pages 176–186, 2010.

[212] Y. Zhou and D. Evans. SSOScan: Automated testing of web applications for single sign-on vulnerabilities. In *USENIX Security Symp.*, pages 495–510, 2014.

Appendices

Appendix A

Appendix A: UDS Properties

The following definitions of UDS properties are included essentially verbatim from Bonneau et al. [25] for reader convenience.

A.1 Usability benefits [25]

U1 *Memory-wise-Effortless*: Users of the scheme do not have to remember *any* secrets at all. We grant a *Quasi-Memory-wise-Effortless* if users have to remember *one* secret for everything (as opposed to one per verifier).

U2 *Scalable-for-users*: Using the scheme for hundreds of accounts does not increase the burden on the user. As the mnemonic suggests, we mean "scalable" only from the user's perspective, looking at the cognitive load (e.g., having to remember a distinct secret for every verifier is a cognitive burden, but so is having to select the relevant verifier out of a linear menu with hundreds of them), not from a system deployment perspective, looking at allocation of technical resources.

U3 *Nothing-to-Carry*: Users do not need to carry an additional physical object (electronic device, mechanical key, piece of paper) to use the scheme. A *Quasi-Nothing-to-Carry* is awarded if the physical object is one that they'd carry everywhere all the time anyway, such as their cellphone, but not if it's their computer (including tablets).

U4 *Manually-Effortless*: The authentication process does not require manual (as opposed to cognitive) user effort beyond, say, pressing a button. Schemes that don't offer this benefit include those that require typing, scribbling or performing a set of motions. We grant *Quasi-Manually-Effortless* if the user's effort is limited to speaking, on the basis that even illiterate people find that natural to do.

U5 *Easy-to-Learn*: Users who don't know the scheme can figure it out and learn it without too much trouble, and then easily recall how to use it.

U6 *Efficient-to-Use*: The time the user must spend for each authentication is acceptably short. The time required for setting up a new association with a verifier, although possibly longer than that for authentication, is also reasonable.

U7 *Infrequent-Errors*: The task that users have to perform to log in usually succeeds when performed by a legitimate and honest user. In other words, the scheme isn't so hard to use or unreliable that genuine users are routinely rejected. (We could view this benefit as "low false reject rate". In many cases the scheme designer could make the false reject rate lower by making the false accept rate higher. If this is taken to an extreme we count it as cheating, and penalize it through a low score in some of the security-related benefits.)

U8 *Easy-Recovery-from-Loss*: A user can quickly regain access if the token is lost or the credentials forgotten. We do not grant the benefit when loss-resilience is obtained by having an authority revoke the lost credential and issue a new one, as that's almost always available as the ultimate recourse (except perhaps for biometrics). This benefit essentially indicates whether the scheme offers convenient backups or secondary recovery schemes.

A.2 Deployability benefits [25]

D1 *Accessible*: Users who can use passwords are not prevented from using the scheme by disabilities or other physical (not cognitive) conditions. (Ideally one would just say: "the scheme is usable by everyone, regardless of disabilities". However we feel that, for any given scheme, it is always possible to come up with a disability or condition that would exclude a category of people, and that therefore no scheme would be able to offer this benefit. We therefore choose to award the benefit to schemes that do at least as well as the incumbent that is de facto accepted today, despite the fact that it too isn't perfect. An alternative to using passwords as the baseline could be to base the metric on the ability of the scheme to serve a defined percentage of the population of potential users.)

D2 *Negligible-Cost-per-User*: The total cost per user of the scheme, adding up the costs at both ends (any devices required at the prover's end and any share of the equipment and software required at the verifier's end), is negligible. The scheme must be a plausible choice for startups with no per-user revenue.

D3 *Server-Compatible*: At the verifier's end, the scheme is compatible with text-based passwords. Providers don't have to change their existing authentication setup to support the scheme.

D4 *Browser-Compatible*: Users can use the scheme with any current standards-compliant web browser and no additional software. Schemes fail to provide this benefit if they require the installation of plugins or any kind of software whose installation requires administrative rights. Schemes offer *Quasi-Browser-Compatible* if they rely on very recent web standards that are not yet available on all the major browsers. Schemes may still offer the full benefit if they require JavaScript or other active content that any current standards-compliant browser supports out of the box. This benefit is the dual of the previous one: at the prover's end, the scheme is compatible with text-based passwords and users don't have to change their client to support the scheme.

D5 *Mature*: The scheme has already been implemented and deployed on a large scale for actual authentication purposes rather than for research. Indicators to consider for granting the full benefit may also include whether the scheme has undergone user testing, whether the standards community has published documents covering the scheme, whether open-source projects implementing the scheme exist, whether anyone other than the implementers has adopted the scheme, the amount of literature on the scheme and so forth.

D6 *Non-Proprietary*: Anyone can implement or use the scheme for any purpose without having to pay royalties to anyone else. The relevant techniques are generally known, published openly and not protected by patents or trade secrets.

A.3 Security benefits [25]

S1 *Resilient-to-Physical-Observation*: An attacker cannot impersonate a user after observing them authenticate (observation attacks may include shoulder surfing, filming the keyboard, recording the sound of keystrokes, or visualizing the temperature of the keys on the keypad after they were touched) one or more times. We grant *Quasi-Resilient-to-Physical-Observation* if the scheme could be broken only by repeating the observation more than, say, 20 times.

S2 *Resilient-to-Targeted-Impersonation*: It is not possible for an acquaintance (or skilled investigator) to impersonate a specific user by exploiting knowledge of that user's personal details (date of birth, names of relatives etc.). Personal knowledge questions are the canonical scheme which fails on this point.

S3 *Resilient-to-Online-Guessing*: An attacker whose rate of guessing is constrained by a policy at the server cannot successfully guess the secrets of a significant fraction of the users. To give a quantitative example, we might grant this property if an attacker constrained to, say, 10 guesses per account per day, could only compromise fewer than 1% of accounts. This is meant to penalize schemes in which it is frequent for user-chosen secrets to be selected from a small and well-known subset.

S4 *Resilient-to-Offline-Guessing*: An attacker whose rate of guessing is constrained only by available computing resources cannot successfully guess the secrets of a significant fraction of the users. We might for example grant this benefit if an attacker capable of attempting up to 2^{40} or even 2^{64} guesses per account could still only reach fewer than 1% of accounts. This is meant to penalize schemes where the space of credentials is not large enough to withstand brute force search (including dictionary attacks, rainbow tables and related brute force methods smarter than raw exhaustive search, if credentials are user-chosen secrets).

S5 *Resilient-to-Internal-Observation*: An attacker cannot impersonate a user by intercepting the user's input from inside the user's device (e.g., by key-logging malware) or eavesdropping on the cleartext communication between prover and verifier (assuming that the attacker can also defeat TLS, perhaps through the

CA, if it is used). This rewards challenge-response schemes and penalizes those that reuse static secrets and thus are not replay-resistant. This benefit assumes that general-purpose devices like software-updatable personal computers and smartphones may contain malware, but that hardware devices dedicated exclusively to the scheme can be made malware-free.

S6 *Resilient-to-Leaks-from-Verifier*: Nothing that a verifier could possibly leak can help an attacker impersonate the user to another verifier. This penalizes schemes where insider fraud at one provider, or a successful attack on one back-end, endangers the user's accounts at other sites.

S7 *Resilient-to-Phishing*: An attacker that simulates a valid verifier cannot collect credentials that can later be used to impersonate the user to the actual verifier. This is meant to penalize schemes where the phishers make victims authenticate to lookalike sites and later use the harvested credentials against the genuine sites. It is not meant to penalize schemes vulnerable to more sophisticated real-time man-in-the-middle or relay attacks, in which the attackers have one connection to the victim prover (pretending to be the verifier) and simultaneously another connection to the victim verifier (pretending to be the prover).

S8 *Resilient-to-Theft*: If the scheme uses a physical object for authentication, the latter cannot be used for authentication by another person who gains possession of it.

S9 *No-Trusted-Third-Party*: The scheme does not rely on a trusted third party (other than the prover and the verifier) who could, if compromised, compromise the prover's security or privacy.

S10 *Requiring-Explicit-Consent*: The authentication process cannot be started without the consent of the user. This is both a security and a privacy feature (for a counterexample, think of wireless RFID-based credit cards: a rogue card reader embedded in the sofa could debit your card through your wallet without your knowledge).

S11 *Unlinkable*: Different verifiers cannot determine whether the same user is authenticating to both. This is a privacy feature.

Further details on the UDS framework are available from Bonneau et al. [25], from which these definitions were reproduced verbatim.