Diss. ETH No. 24986

Strengthening Authentication and Integrity in Web Applications

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

NIKOLAOS KARAPANOS

Master of Science in Computer Science,
ETH Zurich

born on 05.07.1983

citizen of Greece

accepted on the recommendation of

Prof. Dr. Srdjan Čapkun, examiner
Prof. Dr. David Basin, co-examiner
Prof. Dr. Kenny Paterson, co-examiner
Prof. Dr. Carmela Troncoso, co-examiner
Prof. Dr. Moti Yung, co-examiner

2018

I learned that you can fail at what you don't love,
so you might as well do what you love.
— Jim Carrey

Abstract

Web applications have become one of the most common ways of providing access to online information and services. People use their desktop or mobile browsers to surf the web and perform a wide range of functions, as well as store and access their ever growing personal digital data. The popularity, importance and versatility of web applications comes at a cost though, as they naturally attract the interest of malicious actors. Web application breaches expose sensitive data and incur significant financial and reputation losses to the affected parties. It is, therefore, necessary to adequately secure web applications, and shield them from malicious intent.

In this thesis we consider authentication on the web which constitutes a vital aspect of web application security. We consider three main pillars of authentication, namely client authentication, server authentication, and data authenticity and integrity protection.

In the first part, we propose Sound-Proof, a two-factor authentication scheme for web logins. Sound-Proof leverages short audio recordings to verify the proximity of the user's phone to the computer on which the login is taking place. Sound-Proof is transparent to the user, as it does not require any user-phone interaction, resembling the behavior of password-only authentication. It is, moreover, readily deployable, requiring no additional software or browser plugins on the user's computer.

In the second part, we consider TLS MITM attacks where the attacker impersonates the legitimate web server to the user, with the goal of impersonating the user to the server and compromise the user's account. We show why the recently proposed client authentication based on TLS Channel IDs, as well as client web authentication in general, cannot fully prevent such attacks. We then show how combining strong client authentication with the concept of server invariance can protect against such attacks. We design a novel mechanism called SISCA (Server Invariance with Strong Client Authentication) and show how it can be realized in practice.

In the third part, we propose Verena, a web application platform that provides end-to-end integrity guarantees, under full server compromise. In Verena web clients can verify the integrity of a webpage by verifying the results of queries on data stored at the server. Verena provides correctness, completeness and freshness for a common set of database queries by relying on a small trusted computing base. Verena enables a developer to specify an integrity policy for query results based on our notion of trust contexts and enforces this policy efficiently.

Zusammenfassung

Webanwendungen haben sich zu einem der häufigsten Zugangswege zu Informationen und Dienstleistungen entwickelt. Browser auf Desktops und Mobilgeräten werden genutzt um im Internet zu surfen und zahlreiche Aufgaben zu erfüllen. Damit sind sie das Zugangsmedium zu einer stetig wachsende Zahl von persönlichen, digital gespeicherten Daten. Die Popularität, Wichtigkeit und Vielseitigkeit von Webanwendungen kommt jedoch mit einem hohen Preis, da sie natürlicherweise kriminelles Interesse auf sich zieht. Schwachstellen in Webanwendungen setzen sensible Daten frei und führen zu signifikanten Kosten und Rufschädigungen für die involvierten Parteien. Es ist daher notwendig Webanwendungen angemessen zu sichern und sie vor schädlichen Absichten zu schützen.

In dieser Dissertation betrachten wir Authentifizierung im Internet, einen zentralen Bestandteil von Webanwendungssicherheit. Wir betrachten drei zentrale Säulen der Authentifizierung: Authentifizierung für Endgeräte, Authentifizierung für Server und Authentisierung sowie Integritätsschutz für Daten.

Im ersten Teil entwickeln wir Sound-Proof, eine Lösung für Zwei-Faktor-Authentifizierung beim Anmelden im Internet. Sound-Proof nutzt kurze Audioaufnahmen, um die räumliche Nähe von Telefon und Computer des Benutzers, an dem der Anmeldevorgang stattfindet, zu überprüfen. Sound-Proof ist unsichtbar für den Benutzer, weil es keine Benutzerinteraktion mit dem Telefon benötigt und sich daher ähnlich wie Passwort-gestützte Authentifizierung verhält. Weiterhin ist es einfach einzusetzen und erfordert keine zusätzliche Software oder Browsererweiterungen auf dem Computer des Benutzers.

Im zweiten Teil untersuchen wir TLS MITM-Angriffe bei denen der Angreifer aus Sicht des Benutzers die Identität des legitimen Webservers annimmt, um gegenüber des Servers als Benutzer auftreten zu können und so den Account des Benutzers zu übernehmen. Wir zeigen warum die kürzlich vorgeschlagene Authentifizierung von Endgeräten mit TLS Channel IDs, sowie die allgemeine Authentifizierung im Internet, solche Angriffe nicht vollständig verhindern kann. Dann zeigen wir wie starke Authentifizierung für Endgeräte mit der Invarianz des Servers kombiniert werden kann, um gegen solche Angriffe zu schützen.

Im dritten Teil stellen wir Verena, eine Webanwendungsplattform mit Ende-zu-Ende Integrität selbst im Falle von vollständiger Übernahme des Servers, vor. Bei Verena können Endgeräte die Integrität einer Webseite verifizieren, indem sie die Ergebnisse der Anfragen für Daten, die auf dem Server gespeichert sind, überprüfen. Basierend auf einem kleinen, vertrauten Kern, bietet Verena Korrektheit, Vollständigkeit und Aktualität für häufig gebrauchte Datenbankanfragen an. Verena erlaubt es Entwicklern Integritätsregeln für die Ergebnisse zu spezifizieren, die auf dem Prinzip von gemeinsamen Vertrauen basieren, und erlaubt es diese Regeln effizient durchzusetzen.

Sommario

Le applicazioni web sono diventate uno dei principali modi di accedere a informazioni e servizi online. La gente utilizza browser per computer o cellulari per navigare in internet e per salvare e accedere una quantità sempre crescente di dati personali digitalizzati. La popolarità, importanza e versatilità delle applicazioni web risulta però un'arma a doppio taglio poichè attirano l'interesse di individui malintenzionati. Le violazioni delle applicazioni web espongono dati sensibili e incorrono in perdite finanziarie e di immagine. Di conseguenza, è necessario rendere le applicazioni web sicure e protette da intrusi e accessi non autorizzati.

In questa tesi analizziamo, in generale, l'autenticazione sul web che costituisce un aspetto vitale per la sicurezza delle applicazioni web. In particolare, consideriamo tre aspetti cardine dell'autenticazione, ovvero autenticazione del client, del server, come anche dei dati e della protezione della loro integrità.

Nella prima parte proponiamo Sound-Proof, uno schema di autenticazione a due fattori per i login sul web. Sound-Proof sfrutta due brevi registrazioni audio per verificare la prossimità del cellulare dell'utente con il proprio computer, dove viene effettuato il login. Sound-Proof non richiede nessuna interazione dell'utente con il proprio cellulare, assomigliando, per esperienza, al semplice login con nome utente e password. Inoltre, Sound-Proof risulta facilmente utilizzabile poichè non richiede nessuna installazione di componenti aggiuntivi per il browser.

Nella seconda parte consideriamo attacchi di tipo TLS MITM, ovvero quando l'avversario impersona legittimamente il server web verso l'utente, con l'obbiettivo di impersonare l'utente nei confronti del server e così comprometterne l'account. Dimostriamo come l'autenticazione del client basata su "TLS Channel IDs", così come l'autenticazione del client più in generale, non previene questo genere di attacchi. In seguito dimostriamo come questi attacchi possono essere risolti unicamente combinando l'autenticazione client "strong" con il concetto di invarianza di server. Ideiamo un nuovo meccaniscmo, chiamato SISCA (Server Invariance with Strong Client Authentication), e dimostriamo come può essere realizzato in practica.

Nella terza ed ultima parte, proponiamo Verena, una piattaforma per applicazioni web che fornisce garanzie di integrità end-to-end, considerando un server completamente compromesso. Con Verena, i client web possono verificare l'integrità di una pagina web verificando i risultati di diverse query per i dati salvati sul server. Verena garantisce la correttezza, la completezza e la freschezza dei dati, per un insieme di query per database, basandosi su una piccola base di calcolo affidabile. Verena permette ad uno sviluppatore di specificare una politica di integrità per i risultati delle query basata sulla nostra nozione di contesti di affidamento (“trust contexts”); questa politica di integrità viene poi applicata da Verena efficientemente.

Acknowledgments

Pursuing my PhD at the System Security Group of ETH Zurich has been a unique life experience filled with many beautiful moments to remember, through which I learned a lot but also had lots of fun.

I would like to express my most sincere gratitude to my advisor Prof. Srdjan Čapkun for his support and guidance throughout my studies and for being a great mentor and, importantly, a great friend at the same time. All these years we have had an amazing collaboration and relationship, and I am sure it will continue to be the case for years to come.

I would also like to sincerely thank my co-advisors Prof. David Basin, Prof. Kenny Paterson, Prof. Carmela Troncoso and Prof. Moti Yung who accepted to be on my dissertation committee and dedicated their time to read and evaluate this thesis, as well as provide valuable feedback.

I am grateful to my co-authors, Alexandros Filios, Dr. Claudio Marforio, Prof. Raluca Ada Popa and Dr. Claudio Soriente, who contributed in making this thesis a reality. It has been an immense pleasure working with them, and I gained a lot through their experience and insights.

Many thanks go to Dr. Claudio Marforio and Ilias Rinis for their help in improving this thesis.

A special thanks goes to all of my fellow colleagues with whom I had a great time sharing office space throughout my studies: Prof. Arthur Gervais, Marco Guarnieri, Dr. Claudio Marforio, Dr. Claudio Soriente and Der-Yeuan Yu.

The System Security Group has been a great home and family all these years, and I would therefore like to extend my gratitude to everyone who has been part of this great family during the time I was part of it.

I am grateful to my girlfriend Kimily, for her love and support during the last stage of my PhD, especially while writing this thesis, as well as for her help in improving it. I am also grateful to Efstratia for supporting and bearing with me throughout my studies at ETH Zurich.

Finally, last but by no means least, I would like to deeply thank my parents and my brother for supporting me throughout my PhD years and my life in general, and for helping me become the person I am today.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The Web is one of the most popular ways of accessing online information on the Internet. Millions of people interact with multiple websites every day for a plethora of purposes, such as reading the news, accessing their personal web email, interacting with their social media accounts, performing financial transactions, accessing their medical records and so forth. Besides personal-oriented activities, such as the ones mentioned above, a substantial number of corporate functions are implemented on top of web applications, which company employees access daily at work. Online office applications, web-based thin clients and HR management tools are just some examples of business-related web applications.

Traditionally, this interaction involved using a web browser on a desktop or laptop computer. However, in recent years mobile devices have become the most popular way of browsing the web [210]. This trend has been driving companies to provide mobile-friendly versions of their websites in order to offer a smooth browsing experience on mobile platforms, as well.

Given the ubiquity and range of functionality that web applications offer nowadays, their security is arguably a significant aspect that cannot be overlooked. Sensitive private user or corporate information is at risk and can be exposed if an attacker manages to gain unauthorized access in a web application. According to a recent report [4, 219], web application attacks have soared significantly in the last years and nowadays constitute the largest source of attack resulting in successful data breaches.

The prevalence of web applications, as well as their nature, i.e., they are often public facing and accessible over the Internet, makes them an attractive target to attackers. Contributing to this fact is the complexity of the web application software stack. Modern Web 2.0 sites offer a rich user experience, through a variety of client- and server-side technologies, which nevertheless significantly increase the software complexity as well as the exposed attack surface. Given the above, it is not surprising that there is a wide range of attack vectors (e.g., [212]) through which malicious actors can try to compromise a web application and the online accounts and data of its users.

One of the important aspects and fundamental building blocks in the security of web applications is *authentication*. Authentication on the web comprises three pillars. On the one hand, *client authentication* allows the web server to verify the identity of a user, such that only the intended user can access his online account and data and thereby deny any unauthorized and malicious access. On the other hand, *server authentication* ensures that users are interacting with the intended website, for example the e-banking application of their bank, instead of a fraudulent one. Going one step further, *data integrity protection and authentication* can ensure that the users of a web application will be able to retrieve correct and authentic information while interacting with the web application, even if an adversary has managed to fully compromise the server.

In this thesis we focus on web application security and authentication. In particular, we look into the three authentication pillars, namely client authentication, server authentication and data authenticity. In each pillar we explore existing challenges and open problems and advance the state of the art by proposing novel solutions that strengthen authentication and improve the security of web applications. While *security* is, of course, an important design goal, we also aim for our proposals to be feasible and practical. For a proposed security solution to be practical, it should be as user-friendly as possible and easy to deploy, without requiring significant changes in the existing infrastructure. These features combined can encourage adoption and people can therefore benefit from the security that the solution brings. In other words, *usability* and *deployability* are another two essential design goals which we strive to achieve throughout our work. In particular, we highlight the interaction between these three goals and show how the overall security can be improved by striking the appropriate balance.

In the first part of this thesis we investigate client authentication in web applications. Password-based authentication is almost ubiquitous when it comes to authenticating users on the web. While two-factor authentication is a promising solution to the reduced security of passwords, the adoption rates are significantly low when two-factor authentication is optional [165]. We propose a novel two-factor authentication solution which removes interaction between the user and his mobile phone, which acts as the second-factor device. We show how such a usable and easily deployable solution can potentially improve the overall security by fostering wider user adoption, such that more online accounts can benefit from the increased security which two-factor authentication offers.

In the second part of this thesis we focus on web server authentication and the security of the current Certificate Authority (CA) trust model. We look into Transport Layer Security (TLS) Man-In-The-Middle (MITM) attacks, where the attacker is able to successfully impersonate the server to the user, with the intention of compromising the user's online account and data. We show and explain why a recently proposed solution to this problem, based on strong client authentication, fails to actually prevent such attacks. We then come up with a novel solution that leverages the concept of server invariance which, when combined with strong client authentication, can resist MITM attacks.

Finally, in the third part of this thesis we look into data authenticity and integrity protection for web applications. We discuss how integrity of information can be crucial and more important than confidentiality in certain types of applications. We investigate the case where a web application server has been fully compromised by the attacker. Such an attacker can tamper with the data and query computation results and thus serve corrupted webpages to the user. We then present a web application framework that provides end-to-end integrity guarantees for the legitimate users of the web application even in the face of such a powerful adversary.

1.1 Part I: Web Client Authentication

The majority of web applications employ password-based authentication in order to authenticate their users and grant them access to their online accounts. Nevertheless, passwords are susceptible to a number of attacks such as phishing and password database compromise. As a matter of fact, billions of user passwords, in either plaintext or hashed form, are publicly known to have leaked through website data breaches [215]. To make the problem even worse, users are known to often choose simple, easy to guess passwords and to reuse passwords across websites. Consequently, a breach on one website can lead to the leakage of the user's password, which can in turn lead to the compromise of his online accounts on other independent websites, assuming that the same password is reused.

Two-factor authentication protects users and their online accounts even if passwords are leaked. Most users, however, prefer password-only authentication. One reason why two-factor authentication is so unpopular is the extra steps that the user must complete in order to log in [36, 86, 233]. Currently deployed two-factor authentication mechanisms require the user to interact with his phone to, for example, copy a verification code to the browser. Two-factor authentication schemes that eliminate user-phone interaction exist but require additional software, which poses an obstacle to deployment.

We propose Sound-Proof, a usable and deployable two-factor authentication mechanism. Sound-Proof does not require interaction between the user and his phone. In Sound-Proof the second authentication factor is the proximity of the user's phone to the device being used to log in. The proximity of the two devices is verified by comparing the ambient noise recorded by their microphones. Audio recording and comparison are transparent to the user, so that the user experience is similar to that of password-only authentication. Sound-Proof can be easily deployed as it works with current phones and major browsers without plugins. We build a prototype for both Android and iOS. We provide empirical evidence that ambient noise is a robust discriminant to determine the proximity of two devices both indoors and outdoors, even if the phone is in a pocket or purse. We conduct a user study designed to compare the perceived usability of Sound-Proof with Google 2-Step Verification. Participants ranked Sound-Proof as more usable and appreciated the speed and seamless experience that it offers in comparison to a code-based two-factor authentication solution. The majority of the users would be willing to use Sound-Proof even for scenarios in which two-factor authentication is optional.

1.2 Part II: Web Server Authentication

More and more websites nowadays, even those who provide a service that is not security critical, support HTTPS (HTTP over TLS). Whereas HTTP is insecure and does not provide any confidentiality, integrity and server authentication guarantees, HTTPS is able to provide all of these properties through TLS. The established way in which web servers are authenticated to the client is the so called CA trust model. The browsers that people use for surfing the web come preconfigured with a list of trusted CAs. Any certificate issued by one of these CAs for a particular domain name is automatically trusted by the browser. The websites use these certificates in order to authenticate themselves to the browser during the TLS handshake phase. Although this model has worked well for the most part, there have been a number of significant attacks enabling attackers to get valid certificates for domains they do not control and mount TLS MITM attacks. Consequently, people have started to question the security of the CA trust model. As we will see later on in this thesis, this has been a very active area of research, with researchers trying to come up with new solutions and proposals to enhance or even completely substitute the current CA trust model.

In this thesis, we consider TLS MITM attacks in the context of web applications, where the attacker is able to successfully impersonate the legitimate server to the user, with the goal of impersonating the user to the server and thus compromising the user's online account and data. We describe in detail why the recently proposed client authentication protocols based on TLS Channel IDs [56], as well as client web authentication in general, cannot fully prevent such attacks.

Nevertheless, we show that strong client authentication, such as Channel ID-based authentication, can be combined with the concept of server invariance, a weaker and easier to achieve property than server authentication, in order to protect against the considered attacks. We specifically leverage Channel ID-based authentication in combination with server invariance to create a novel mechanism that we call SISCA: Server Invariance with Strong Client Authentication. SISCA resists user impersonation via TLS MITM attacks, regardless of how the attacker is able to successfully achieve server impersonation. We analyze our proposal and show how it can be integrated in today's web infrastructure.

1.3 Part III: Data Authenticity and Integrity in Web Applications

Arguably, data confidentiality of sensitive information is one of the basic properties we want to achieve through the use of secure protocols in web applications, such as HTTPS. Nowadays, our lives are almost completely digitalized and accessible through a variety of web services that we use. A lot of people are privacy sensitive and would not wish for their private financial assets, medical information, VOIP calls, email, friends on social media, or even their search queries to leak to unauthorized third parties. Privacy is a fundamental human right, and by encrypting all communication between a browser and a web server, we can thwart network-based eavesdroppers from accessing our private online information.

Nevertheless, one should not neglect that the integrity of the information which we access on web applications is equally as important, especially for certain classes of applications. Web applications rely on web servers to protect the integrity of sensitive information. However, an attacker gaining access to web servers can tamper with the data and query computation results, and thus serve corrupted webpages to the user. Violating the integrity of the webpage can have serious consequences, affecting application functionality and decision-making processes. Worse yet, data integrity violation may affect physical safety, as in the case of medical web applications which enable physicians to assign treatment to patients based on diagnostic information stored at the web server.

In this thesis we propose Verena, a web application platform that provides end-to-end integrity guarantees against attackers that have full access to the web and database servers. In Verena, a client's browser can verify the integrity of a webpage by verifying the results of queries on data stored at the server. Verena provides strong integrity properties such as freshness, completeness, and correctness for a common set of database queries by relying on a small trusted computing base. In a setting where there can be many users with different write permissions, Verena allows a developer to specify an integrity policy for query results based on our notion of *trust contexts*, and then enforces this policy efficiently. We implemented and evaluated Verena on top of the Meteor framework. Our results show that Verena can support real applications with modest overhead.

1.4 Related Publications

The work presented in this thesis is based on the following publications, co-authored during my doctoral studies at ETH Zurich.

- Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Čapkun. Verena: End-to-End Integrity Protection for Web Applications. In *Proc. 37th IEEE Symposium on Security and Privacy (S&P)*, 2016.

- Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Čapkun. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound. In *Proc. 24th USENIX Security Symposium*, 2015.

- Nikolaos Karapanos and Srdjan Čapkun. On the Effective Prevention of TLS Man-In-The-Middle Attacks in Web Applications. In *Proc. 23rd USENIX Security Symposium*, 2014.

1.5 Thesis Outline

In this thesis we focus on the security of web applications and in particular in web authentication, which we divide into three pillars; *client authentication*, *server authentication* and *data authenticity and integrity protection*. In the first part of the thesis we look into web client authentication and propose a secure, usable and deployable two-factor authentication mechanism. In the second part we investigate the problems of the current CA trust model for authenticating web servers. We come up with an attack on a recently proposed solution that attempts, but fails, to fully prevent TLS MITM attacks. We also develop a new concept and proposal for preventing such attacks. Finally, in the third part of the thesis we look at the importance of data authenticity and integrity in web applications and propose a new framework for achieving end-to-end data integrity guarantees, even under full server compromise. A detailed outline of the thesis follows.

In **Chapter 2** we briefly introduce today's web technologies and the typical web application architecture. We give an overview of the most common ways in which web servers and clients are authenticated and describe the security model of modern web browsers. For further background information relevant to each part of the thesis, we refer the reader to the respective background sections of each part. Moreover, the related work sections in each respective part, provide more information about the state of the art in each authentication pillar which we investigate in this thesis.

Part I: Web Client Authentication

Chapter 3 introduces the first part of the thesis, which focuses on web client authentication. In **Chapter 4** we give an overview of the the research on two-factor authentication, as well as currently deployed solutions. In **Chapter 5**, we present Sound-Proof, a two-factor authentication mechanism for web applications that leverages the ambient sound to verify that the user's phone, which acts as the second-factor device, is in close proximity to the computer from which the user is trying to log in. The key design goals of Sound-Proof are usability, and deployability, without requiring any changes on the browser.

Part II: Web Server Authentication

Chapter 6 introduces the second part of this thesis, which focuses on web server authentication. In **Chapter 7** we give an overview of the current research as well as deployed systems that enhance the security of the server authentication on the web. In **Chapter 8** we present an attack, which we call Man-In-The-Middle-Script-In-The-Browser (MITM-SITB), against a recently proposed solution to prevent TLS MITM attacks based on the concept of TLS Channel IDs. We then continue to propose SISCA, which is based on the concept of server invariance and show how it can prevent such attacks.

Part III: Data Authenticity and Integrity in Web Applications

Chapter 9 introduces the third part of this thesis, which focuses on data integrity protection in web applications. **Chapter 10** provides a short introduction to Authenticated Data Structures, which we make use of in our work. In **Chapter 11** we present Verena, the first web application platform which provides end-to-end data integrity protection for the users of a web application, even under full server compromise.

Chapter 12 concludes this thesis by presenting the closing remarks of the work presented.

Chapter 2
Background on Web Applications

In this chapter we briefly introduce the reader to a few basic and important aspects of web applications and web application security. As mentioned in Chapter 1, web applications are complex ecosystems, featuring a large number of different standards, protocols, frameworks, software implementations, browser vendors and so forth. Consequently, we omit many details and try to present the reader with information and concepts that are important in understanding and following the research topics and ideas that are discussed in the rest of this thesis.

We note that this chapter serves as a brief introduction to basic web application concepts and is intended for readers who are interested in this thesis but lack familiarity with foundation concepts of web applications. We encourage readers who are already familiar with these concepts to skip this chapter.

The rest of this chapter is organized as follows. Section 2.1 contains an overview of the architecture of web applications and presents some of the most important components and technologies in modern web applications. In Section 2.3 we describe the basic notions of the browser security model, which revolves around the fact that web browsers are used to access a multitude of websites, so it is essential that each website is isolated and cannot access data of other websites. Finally, in Section 2.2 we describe how common client (user) and server authentication is performed in web applications.

2.1 Web Application Architecture

Basic Components

Web applications, as shown in Figure 2.1, consist of a server component as well as a client component, which is typically a web browser. The user navigates the browser to a particular website, identified by a domain name. On a high level, the browser contacts the web server corresponding to this domain and asks to load the particular page. Through a series of HTTP requests issued by the browser and processed by the server, the browser fetches all the webpage resources and renders the page. Assuming an HTTPS-enabled web application, these HTTP requests and responses are taking place over a TLS connection between the browser and the server.

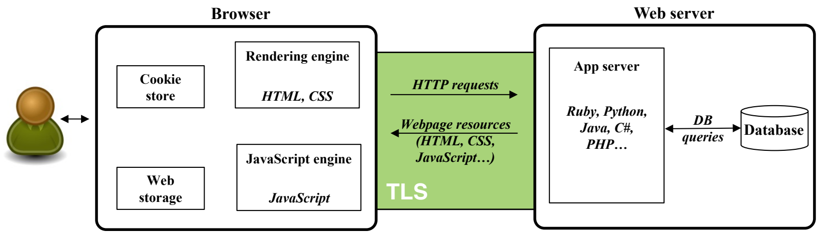


Figure 2.1: Web application architecture overview. A typical web application consists of both server-side and client-side code. Persistent data is stored on a backend database, but some application state can be stored on the client side in the form of cookies or within the browser's Web Storage. For HTTPS-enabled websites, the HTTP communication between the browser and the server occurs on top of TLS connections.

We emphasize the fact that in a web application two programs are running at the same time; the server-side code which processes and responds to incoming HTTP requests, and the client-side code which responds to user input. The client-side code is fetched by the browser from the server as part of loading the webpage, i.e., it is considered to be part of the webpage's resources.

On the server side there is a great variety of languages and frameworks used for creating web applications. Popular examples include Ruby and the Rails framework [183], Python and the Django framework [58], PHP, C# and Java. Nevertheless, any program, written in any programming language, which is able to respond to HTTP requests can act as the server-side part of a web application.

On the client side the landscape is different, with HTML, CSS and JavaScript being the languages used and understood by web browsers (excluding web plugins). Webpages, therefore, consist of HTML, CSS and JavaScript code, as well as other, mainly static, resources such as text, images and videos [144]. HTML (HyperText Markup Language) describes the layout and content of a webpage while CSS (Cascading Style Sheets) are used to describe the appearance of the content. JavaScript is the programming language that is executed within the web browser. It is powerful and can be used to create advanced, dynamic websites. The browser is responsible for parsing the webpage code, fetching all the resources that are required by the page, executing its JavaScript code and render the page to the user's screen.

Gradually, webpages shifted from consisting of mostly static HTML content to being fully dynamic through the use of JavaScript code. The page's JavaScript code communicates in the background with the server via AJAX [228] and WebSockets [70], and dynamically updates the page contents, by modifying the page's Document Object Model (DOM) [140], which is a tree-based, hierarchical representation of the entire page and its contents. As a result, while most of the page layout rendering was traditionally performed by the server-side code, a lot of modern web applications perform the page layout creation and/or manipulation using the client-side code, i.e., JavaScript. The extreme version of this, are so-called single-page applications [47], which load a single HTML page and dynamically update its contents while the user is interacting with the app. Single-page applications avoid page reloads and rely on background-based server communication (via AJAX and WebSocket), as well as HTML5 [97, 224] to provide a rich user-experience, similar to native desktop and mobile applications.

HTML5 is the latest version of HTML. The new features that it offers (for example, the ability to play and record audio and video and draw graphics), in conjunction with the use of JavaScript, help websites achieve a native-application like experience, as mentioned above. What is more, because of these capabilities, HTML5 has contributed to the decline of the various plugins which were used previously in webpages in order to deliver rich graphics and multimedia, such as Adobe Flash, Microsoft Silverlight and Java applets. These plugins have been often found to contain exploitable vulnerabilities [66, 96] that allow attackers to compromise the security of the user's browser, so phasing them out has been an important, security-related benefit of HTML5. As we will see in Part I, Sound-Proof, our proposed two-factor authentication mechanism, relies on WebRTC [85, 226], an HTML5 technology accessible via JavaScript, for recording audio in the browser, instead of a Flash plugin, greatly improving usability and security.

Server-side Data Storage and Access Control

The web application server typically communicates with a backend database system on which it stores the application's persistent data, such as user account information. Depending on the application requirements, the database system can be a relational database such as MariaDB [127], or a non-relational database, also known as NoSQL, such as MongoDB [137]. This differentiation mainly affects how stored data is organized within the database and consequently how it is retrieved through database queries issued by the web application server.

Among other things, this centralized way of storing data makes it easier to control access to it. The web server is responsible for authenticating the web application users (who access the application via their browsers) and only authorizing eligible users to read or modify certain pieces of data, based on the application's access control policy.

It is important to note that access control can only be enforced on the server side and not on the client. This is due to the fact that all client-side code and other webpage contents are fully accessible to the user and any client-side access control checks can be bypassed by a knowledgable user. Thus, the server is considered to be the trusted part of the application. Nevertheless, this assumption breaks once an attacker manages to compromise the server.

HTTP Cookies and Client-side Storage

Even though the application's state is stored on the server, some state can be stored on the client side, as well. The two typical methods for storing client-side information are HTTP cookies [16], Web Storage [227] and IndexedDB [225]. Cookies are small pieces of information which the server can set as part of its HTTP responses to the client's requests. The browser stores the server cookies in its cookie store and whenever it issues an HTTP request, it automatically attaches any cookies which were previously set by the server. Previously set cookies can be replaced by the server. Additionally, cookies expire, and thus deleted by the browser, either when the browser is closed (session cookies), or upon a predefined expiration date (persistent cookies). As we describe in Section 2.2.1, cookies play an important role in client authentication and session management, and their often incorrect handling (for example as Sivakorn et al. [196] show) by web applications can allow attackers to gain unauthorized access to user accounts.

Web Storage is an HTML5 mechanism that allows web applications to store data locally in the browser. It is essentially a key-value store and can be accessed through JavaScript. It offers the ability to store data that are either automatically deleted when the browser terminates, or persist indefinitely, and can be deleted explicitly via JavaScript. Similarly, IndexedDB is a JavaScript-based object-oriented database that allows indexed storage of structured data. IndexedDB is suitable for storing large amounts of data, while Web Storage is mainly useful for storing smaller amounts.

Overall, client-side storage mechanisms enable the web application to offload some of its state, or to cache it in order to access it faster, without incurring network communication overhead. We note that, the information stored in the client can be viewed and modified by the user. For pieces of information where this is undesired, web applications typically encrypt and sign it using a key known only to the server and send the encrypted version of it to the client for storage. This is for example often the case with cookies which encode information about the session of the user.

2.2 Authentication

Authentication in web applications is the main topic of this thesis. In this section we give a basic introduction to the common ways in which authentication is achieved in today's web applications. Each part of the thesis contains additional information, where necessary, in order to better describe and explain the topics and concepts of each respective part.

2.2.1 Client Authentication

Password-based Authentication Using HTML Forms

The most common way of authenticating users in web applications is through the use of passwords. When a user creates an account on a website, he chooses a username and a password. These constitute the user's credentials. In order to authenticate himself to the website and access his online account, he needs to submit his username and password. This is most commonly performed at the application level using HTML forms, also known as web forms.

The process is illustrated in Figure 2.2. Let us assume that a user has an online account at `www.example.com` and that he visits the website from a web browser not used before. The web server initially treats the user as a guest (unauthenticated user), and he is only able to access the public pages and information of the website. In order to access his account, he navigates the browser to the login area of the website. The browser renders the login web form of the website and transmits the supplied user credentials back to the server. The server verifies the credentials, and provided that they are valid, grants the user access to his account and private data.

The user is now authenticated and can navigate through the private areas and functions of the website pertaining to his own account, and according to his privileges. Nevertheless, HTTP is stateless, so the website needs a way to track the authenticated user across HTTP requests. This is achieved through the use of cookies. Upon processing the incoming HTTP request containing the user credentials, the web server creates a session for the user and sets a cookie in its response to the browser. This cookie

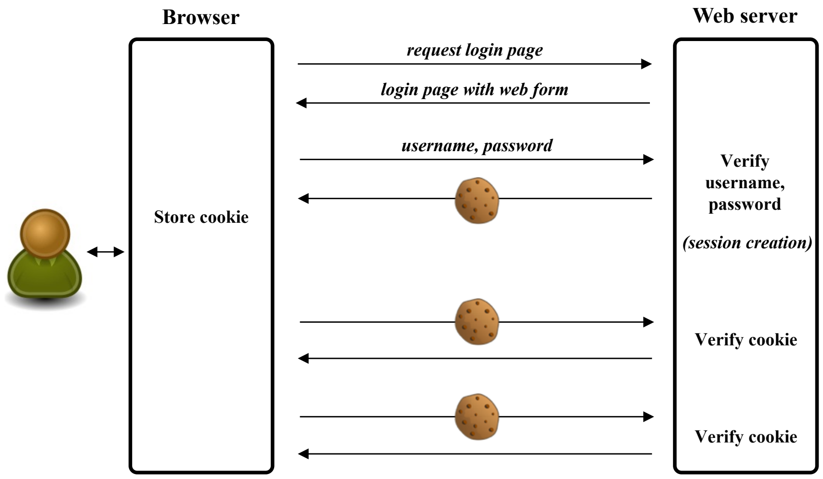


Figure 2.2: Password-based authentication using web forms. In order to login to his account, the user submits his credentials, i.e., username and password, using an HTML form presented by the website. Upon successful verification of the credentials, the server creates a new session for the user by setting a cookie in his browser. This cookie acts as the session identifier and token, and enables the server to identify the user session in subsequent requests.

contains a unique session identifier. Subsequent HTTP requests made by the browser will have this cookie attached in their headers, which allows the server to authenticate the user and track his session as he is interacting with the website. A cookie in such a case acts as an authentication token, and its compromise can allow an attacker to hijack the user's session and gain access to his account. It should be, therefore, sufficiently protected. In particular, it should be transmitted only via HTTPS, to prevent theft by network eavesdroppers [143] and it should be inaccessible to JavaScript, to prevent theft via Cross-Site Scripting (XSS) attacks [156, 157].

HTTP Authentication

Even though using HTML forms is by far the most common way for implementing password-based authentication, there exist alternatives. Namely, the HTTP protocol offers an authentication framework [138], and the most common HTTP authentication scheme is the so-called "Basic" authentication [176]. HTTP authentication lacks the flexibility which HTML forms offer to web applications (for example, HTML forms are customizable and their look and feel can be arbitrarily tailored, which is not the case for the HTTP authentication, browser-controlled credential input dialogue). Thus, it is rarely used.

TLS Client Authentication

Another client authentication mechanism that is available to web applications is TLS client authentication [55]. Typically, and as we describe in Section 2.2.2, only the server is authenticated during the establishment of the TLS connection between the browser and the web server. In TLS client authentication the browser authenticates itself (and consequently the user) to the server by using a certificate, during the TLS handshake. The certificate has to be signed by a CA that is trusted by the server. It contains the public key of the client and, as part of the TLS handshake protocol, the client proves possession of the corresponding private key to the server.

With TLS client authentication, the client is authenticated at the TLS protocol level, similar to where server authentication is performed. This is in contrast with password-based authentication, which takes place at the application level. Moreover, TLS client authentication does not suffer from the security weaknesses of password-based authentication. Nevertheless, it is cumbersome to use (requiring the user to migrate the certificates across machines, for example) and generally offers a bad user experience [34] and thus is not widely deployed.

In Part I of this thesis, we focus on two-factor authentication, which tries to solve the security weaknesses of the popular password-based authentication method.

2.2.2 Server Authentication

Certificates and Certificate Authorities

In Section 2.2.1 we mentioned TLS client authentication as an alternative, yet not so common way for authenticating clients in web applications. When looking at the server side, authentication using digital certificates is, nonetheless, the ubiquitous way for authenticating websites. A digital certificate binds an entity, with the ownership of a public key. In web applications, the entity is a website and it is associated with one or more domain names, e.g., `www.example.com`. The format of such certificates is specified by the X.509 standard [48].

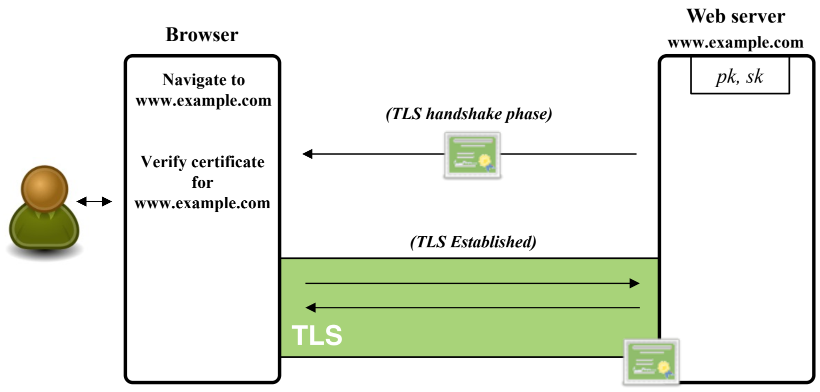


Figure 2.3: Web server authentication. The browser wants to connect to `www.example.com`. During the TLS handshake phase the server is authenticated by presenting a valid certificate for domain `www.example.com`. In order to be accepted by the browser, the certificate needs to be signed by a trusted CA.

A *Certificate Authority* (CA) is a trusted third party, whose task is to verify the ownership of a website's public key and create a certificate that binds the specified key with the domain name(s) of the website. The CA signs the certificate with its own private key. Let us assume a website with domain name `www.example.com`. The website generates a public/private key pair. Let *pk* be the public key and *sk* be the private key (also called secret key). The website then contacts a CA, in order to get its public key certified. During the verification process, the website proves to the CA ownership of the domain `www.example.com`, as well as *sk*. The CA then issues a certificate binding `www.example.com` and *pk* together. The certificate has an expiration date, and can also be revoked prematurely, for example if *sk* is deemed compromised, in which case the server would have to generate a new key pair and repeat the certificate issuance process.

Authentication Flow

The server's key pair $[pk, sk]$ and the CA-issued certificate can be used to authenticate the server. The authentication happens at the TLS connection level. It is illustrated in Figure 2.3 and works as follows. We consider the website `www.example.com`. The user navigates the browser to `https://www.example.com` and the browser performs a DNS lookup and attempts to connect using TLS with the server that answer at the IP address which was returned via the DNS lookup. During the TLS handshake the server presents its certificate to the browser and a proof that it possesses the secret key sk which corresponds to the pubic key pk which is contained in the certificate. The browser then performs a number of checks, including the following:

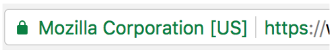
- The certificate is valid, i.e., it contains a valid signature by a CA which is trusted by the browser. The trust can be direct or indirect via a chain of trust. We explain this concept further below.
- The domain `www.example.com` is contained in the list of domains for which this certificate was issued.
- The server has access to the secret key sk. The browser uses the public key pk contained in the certificate and the server-supplied proof to verify this.
- The certificate has not expired and has not been revoked.

The browser accepts the certificate, deems the server as legitimate (provided that all checks were successful) and the TLS connection is established. From this point onward the HTTP requests issued by the browser to `www.example.com` will be performed on top of the encrypted TLS channel.

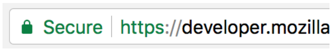
CA Trust Model

Website authentication is based on a public key infrastructure (PKI) where the keys of the web servers are vouched by the keys of the CAs. As mentioned above, in order for the browser to accept a certificate, it has to be signed by a CA that the browser trusts. The trust can be direct or indirect. In particular, web browsers are preconfigured to trust a fairly large number of, so-called, *root CAs*. Each root CA creates a self-signed certificate for its public key, the private counterpart of which, is used to issue certificates. These self-signed certificates, called root certificates, are embedded in the browser's certificate store. Any certificate signed by by the keys corresponding to the root certificates is trusted by the browser.

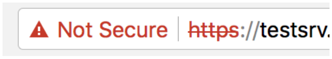
The root CAs are able to appoint intermediate CAs, by issuing them a special type of certificate that grants the ability to sign certificates. This means that intermediate CAs can use their private keys to issue certificates for websites, and, depending on the assigned privileges, other intermediate CAs. This creates chains of trust where, starting from a website certificate, called leaf certificate, there can be one or more levels of intermediate CAs that chain up to a root CA which is trusted by the browser.



(a) Valid EV Certificate



(b) Valid Certificate



(c) Invalid Certificate

Figure 2.4: Google Chrome (version 61) address bar security indicators of the validity of the web server's certificate. EV stands for Extended Validation and is a type of certification which requires stricter verification checks to be performed by a CA, thereby offering a higher level of trust about the legitimacy of the certified organization.

CAs are typically for-profit organizations i.e., they make profit from issuing certificates. Nevertheless, in recent years some non-profit organizations (e.g., [120]) have been issuing certificates to websites free of charge, a move that aims at overcoming financial obstacles and encouraging widespread adoption of HTTPS and eventual elimination of plain HTTP.

Given the above, it becomes obvious that there is a lot of trust placed in the current CA system. In particular, any CA trusted by browsers can issue a valid certificate for any domain in the world. As we discuss in Part II, this has led to a number of important attacks in recent years, and there is active research effort aiming in enhancing or completely substituting the current CA trust model, with a more secure alternative (e.g., [68, 111, 118, 234]).

Browser Indicators and User Involvement

When the user wishes to visit a particular website, it is the browser's task to authenticate the website, by validating the certificate which the latter presents. Browsers typically use security indicators to inform the user about the security of the connection. Figure 2.4 shows examples of the address bar security indicator for valid as well as invalid certificates.

One might consider that, if the browser fails to validate the server certificate, then this is an indication that there is an active attacker on the network, trying to perform a TLS MITM attack against the user by presenting an invalid certificate for the visited domain. Consequently the browser should forcibly always terminate the connection, thus keeping the user safe. However, in reality it is often the case that the validation fails, not due to an attack, but rather due to a benign reason, such as an expired certificate which the website operators neglected to renew, or some other misconfiguration on the server.

Because of this ambiguity, when the certificate verification fails, browsers tend to display prominent warning dialogues, informing the user about the risk of visiting the website. The user is consequently, given the option to click through the warning and let the connection go through. Users are known to indeed do so and click through TLS warnings, although the way in which the warning is communicated to the users can affect their behavior and help them make safer choices [3, 202].

In Part II of this thesis we focus on the issue of TLS MITM attacks, give an overview of the state-of-the-art proposals to mitigate the problem as well as discuss in detail our contributions to the topic.

2.3 Browser Security Model

The way in which web browsers operate and are used is almost unique compared to any other desktop or mobile Internet-based application. Namely, applications are normally designed to communicate with a single back-end service, which is specifically designed to serve the purposes of that application. In contrast, browsers are used to access an arbitrary number of different websites. In other words, while typical applications offer an isolated, dedicated access to a particular online service, web browsers offer access to and interaction with a multitude of services, i.e., web applications.

The above means that code and data from unrelated, mutually untrusted websites can co-exist and execute within the same browser. Co-existence and execution of different websites can be taking place simultaneously through multiple browser windows, tabs, or due to iframes (nested browsing contexts [139]) within a page, which the user is interacting with. It is therefore, essential to ensure that the code and data of each website is properly isolated from each other, in order to prevent . Browsers achieve this through an isolation mechanism called *same-origin policy*. Same-origin policy is an important mechanism pertaining to web application security and thus, we give an overview of how it works in the paragraphs that follow.

Same-origin Policy

The same-origin policy [142] is a set of restrictions that are enforced by the browser in order to prevent the code of a webpage to access the content of another webpage. As its name suggest the basic notion around

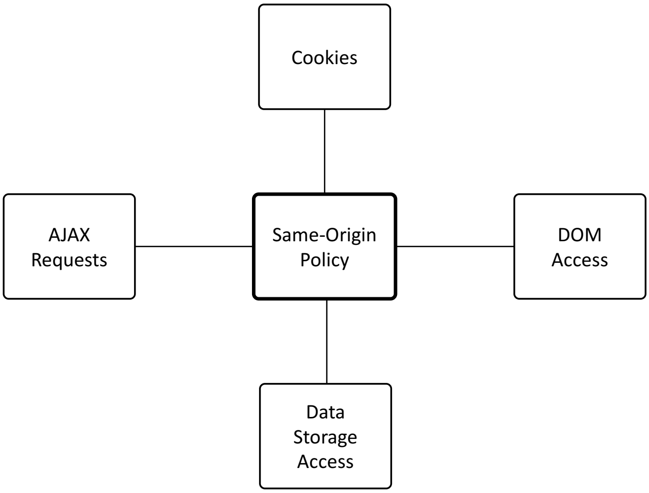


Figure 2.5: Functionalities affected by the same-origin policy. The same-origin policy enforces rules that limit data access, sharing and communication across different website origins with respect to all these four web functionalities.

which the policy is based, is called *web origin* [17], or origin for short. The origin, defined as the combination of the URI scheme, host name and port, is used by web browsers to set the scope of authority of a webpage. Webpages whose the above three URI components are the same are considered to be from the same origin. As an example, two pages from `https://www.example.com` belong to the same origin, while if one of the pages comes from `http://www.example.com` (different URI scheme) then its origin is different.

The general rule of the same-origin policy is that a webpage can access the contents of another webpage only if the two pages belong to the same origin. If two pages belong to the same origin, it is implied that they represent the same authority and thus accessing each other's content is not deemed as a security risk. However two pages coming from different origins are considered as representing different authorities and thus different security contexts and therefore the accessing each other page's, potentially sensitive data, should be prohibited. Figure 2.5 shows which webpage functionalities and features are governed by the same-origin policy restrictions. In particular:

- **DOM Access.** Webpages belonging to different origins cannot see or manipulate each other's DOM data via JavaScript or other means.
- **AJAX Requests.** A webpage cannot make AJAX requests to a different origin.
- **Data Storage Access.** JavaScript running within one origin cannot access data which was stored in the browser's Web Storage and IndexedDB stores by JavaScript belonging to a different origin.
- **Cookies.** A webpage can set a cookie for its own domain, or any parent domain. For example, if the domain of a page is `sub.example.com` then it can also set cookies for `example.com`. The browser will make a cookie set for a given domain available to any sub-domains, as well. However, any HTTP request to a given domain, regardless of which origin initiated it, will have the domain's cookies attached to it automatically by the browser. This is a subtle property than can be beneficial but can also lead to exploitable vulnerabilities, e.g., Cross-Site Request Forgery (CSRF) [155].

While the restrictions imposed by the same-origin policy are useful in most cases, there are legitimate circumstances where different origins need to communicate and interact with each other. Large websites that use multiple subdomains is a prominent example. For this reason web browsers support a variety of techniques [147] that allow the same-origin policy rules to be relaxed in a controlled manner. For example, Cross-Origin Resource Sharing [223] enables two origins to communicate using cross-origin AJAX requests, and cross-document messaging, using the `postMessage()` method, allows JavaScript code from different origins to exchange textual information with each other. Moreover, WebSockets are not restricted by the same-origin policy.

Browser Hardening

Web browsers are complex pieces of software, consisting of millions of lines of code. An attacker who manages to find an exploitable vulnerability, and can successfully lure victim users to a website which is under the attacker's control, can remotely deliver a payload that triggers the exploit and compromises the browser application. Web browser compromise can lead to a very powerful attack, called *Man-In-The-Browser* (MITB) [158]. MITB allows the attacker to completely bypass all browser security mechanisms, including the same-origin policy, and thus access local website code and data, present the user with false information, hijack the user's sessions to his online website accounts that he accesses through the browser, and perform harmful actions to these accounts.

For these reasons, modern web browsers incorporate a variety of defense-in-depth techniques [38, 175], such as sandboxing of the rendering engine and other critical components, automatic updates and known malicious site warnings, in order to mitigate the severity of exploitable vulnerabilities.

2.4 Summary

We have given an overview of the basic concepts around web applications and web application authentication. We hope that this will be useful in order to better understand the topics and ideas which are discussed in the three main parts of the thesis that follow.

Part I

Web Client Authentication

Chapter 3
Introduction

Password-based authentication is the most popular way of authenticating users on the web. In Chapter 2 we briefly described how password-based authentication works. In particular we overviewed the most common implementation which is based on customizable HTML forms and cookies. The user submits his credentials, i.e., username and password, to the website via an HTML form. Once he successfully authenticates, a cookie is used in order to authenticate and keep track of the user's session.

Given their popularity, it is implied that passwords are a convenient way for people to log in to their online accounts, and this is indeed true to a certain extend. For example, passwords require nothing to carry around. All a user has to do is to remember his password and type it in the browser, which only takes a few seconds. Nevertheless, having to remember passwords is one of the main reasons from which the insecurity of passwords stems.

For a password to be secure, it must be sufficiently long and high-entropy in order to resist guessing attacks. However, people find it hard to remember a long, random password. Moreover, people maintain many accounts across a variety of websites so it is inconvenient and hard having to remember so many of them. Thus, in order to make password-based authentication convenient, people tend to choose small, easy to remember passwords, and reuse the same passwords across many different websites [52]. This significantly reduces the security of passwords. In recent years billions of passwords are known to have been leaked [215] through website database breaches(e.g., [31, 59, 101, 119]. Even in cases where the leaked passwords are properly stored using a secure password hashing algorithm, such as bcrypt [170] and PBKDF2 [105], attackers are still able to retrieve low-entropy passwords via offline brute force, or dictionary-based attacks. Due to password reuse, a user's password leaked from a particular website can be used to compromise the user's accounts on other websites on which the same password is used.

In this thesis we focus on two-factor authentication (2FA) which is a technique that can be employed in order to augment the poor security of password-based authentication. As its name suggests, the main idea of two-factor authentication is that the user is required to present two different authentication factors before been granted access to his account. The first authentication factor is something that the user *knows*, which in the case of password-based authentication is his password. The second authentication factor can be something that the user *has*, i.e., a device acting as the second-factor token or something that the user *is*, i.e., a biometric trait of the user, such as his fingerprint. Typically, in web applications the second authentication factor is a token which the user has to prove possession of to the server. As we discuss in Chapter 4, second-factor tokens can be implemented as dedicated hardware devices or as applications that can be installed on the user's smartphone or wearable device.

In currently deployed 2FA schemes, the user has to perform an extra step in order to prove possession of the second-factor token. In other words, they require user interaction [10, 65, 83, 240], for example to type a verification code, received via SMS or from an application, into the browser. Users have to find their smartphones, unlock them, open the application or wait for the SMS message to arrive and then copy the code in order to log in. This constitutes a departure from the default user behavior of password-only authentication. A recent study [165] has found that only 6.4% of users actually use 2FA for their Gmail accounts. While the situation is different for websites where two-factor authentication is mandatory (e.g., for banking institutions), it looks similar to the Gmail case for other websites that offer 2FA as an optional feature. For example, less that 1% of the Dropbox users have 2FA enabled [31]. Although 2FA solutions that do not require user interaction have been proposed, they leverage system resources that are not currently available to web browsers, such as bluetooth [51] or direct access to the wireless network card of the system [191].

From the above it becomes clear that although security plays an important role, usability and deployability are equally as important. In order for a secure solution to be used at large scale so that the masses can benefit from it, it has to be both usable and deployable. In our work we take into consideration the usability and deployability aspects of 2FA.

After surveying related work on two-factor authentication in Chapter 4, we introduce Sound-Proof in Chapter 5. To the best of our knowledge, Sound-Proof is the first both usable and deployable two-factor authentication solution, i.e., it requires no user interaction and is immediately deployable with today's available technologies. Sound-Proof provides 2FA for web logins by checking the proximity of the user's smartphone to the computer from which he is logging in. We perform such a check by comparing audio which is simultaneously recorded by the microphones of the smartphone and the computer. If the two audio samples match, the user is successfully logged into the system. Sound-Proof is completely transparent to the user, making the login process similar to password-only authentication in terms of user experience. We evaluate Sound-Proof in different scenarios, both indoors and outdoors, and show its applicability as well as its security against an attacker that is not co-located with the victim (as for most 2FA schemes). In a user-study, we compare Sound-Proof with Google 2-Step Verification, and show that users rank Sound-Proof considerably higher in terms of its usability. Similarly, participants appreciate the reduced time that Sound-Proof imposed on the overall login procedure, compared to the code-based approach. Finally, as Sound-Proof requires access to only the microphone on both devices it is deployable on major browsers without requiring the installation of a plugin, as well as on Android and iOS smartphones.

Chapter 4
Related Work

The problem with password-only authentication systems is that their security stems from the strength of user passwords. Recent password databases leaks [31, 59, 101, 119] confirm the fact that users tend to choose weak passwords and also that users tend to reuse passwords across websites [52].

Researchers have analyzed the problem extensively in recent years. A case study on Google accounts analyzed the main ways through which attackers manage to steal user credentials and compromise their online accounts [213]. The study found credential leakage through password database breaches to be the most common way of credential theft, followed by phishing.

A plethora of proposals that aim at increasing the security of passwords have been proposed. In general, solutions focus on client-side increased password strength [53, 88, 181, 237] and server-side password protection through the use of cryptographic tools [42, 104].

Two-factor authentication enhances the security of passwords by complementing password authentication with an additional authentication factor. In the rest of this chapter we review related work in the area of two-factor authentication.

4.1 Two-Factor Authentication for the Web

Hardware Tokens. Solutions based on hardware tokens come in a variety of forms, such as the RSA SecurID [182] as well as the more recent USB dongles, like the ones offered by Yubico [240], which conform with the FIDO U2F [72] open authentication standard. In the case of the RSA SecurID token a one-time code is displayed on the token, which is refreshed periodically, e.g., every 30 seconds. The user is asked to type the currently displayed code into the login form of the website.

The most recent hardware tokens which come in the form of USB dongles (e.g., Yubikey [240]) are easier to use but require the token to be physically connected to the computer used to login (i.e., through a USB connection). The user must press a button (or simply touch the token) upon login, in order to prove the intention of the user that is actually performing the login attempt.

Hardware token-based solutions have the disadvantage of requiring the user to carry and interact with the token. Cost of deployment is another consideration. In the case of standardized solutions, e.g., FIDO U2F [72], one token can be used for authentication with any compliant service provider. The browser and the system through which the user is performing the login must support this new hardware and the specification. Currently, only Google Chrome supports FIDO U2F natively, and Mozilla Firefox supports it through the use of a plugin.

In [30] the authors propose an additional type of factor, complementary to the existing ones which consist of something that you know, something that you have and something that you are. According to their proposal, the fourth authentication factor is somebody you know. In particular, they suggest leveraging the user's social network as the fourth factor. This additional factor can be used to vouch for a user, for example in situations where some of the other factors are not available.

Software Tokens. As smartphones became popular and ubiquitous, 2FA solutions based on software tokens have emerged. These solutions come in the form of a mobile app that the user installs on his smartphone or wearable device.

One of the most representative examples of a software token on mobile phones is Google 2-Step Verification (2SV) [83]. It is a 2FA solution that uses one-time codes, similar to the RSA SecurID hardware token. Google 2SV comes in two variants, namely as an application on the mobile phone that generates time-based one-time codes, as well as without the use of an application, in which the code is delivered to the user's phone via SMS. The user has to read the verification code from the phone and type it into the browser when logging in. The same mechanism, based on time-based one-time codes is implemented by various application vendors such as 1Password [98] and Authy [10].

Research proposals leverage the use of smartphones as 2FA software tokens in various different ways. For example, in [185] the authors suggest using the smartphone for provide cues to the user while he interacts with a graphical password on his computer. Namely, the user gets the necessary instructions from the smartphone in order to click the correct parts of the graphical password for the login attemt to be deemed successful. Solutions like this radically change the traditional password-only authentication experience.

Duo Push [61], Encap Security [65] and other similar solutions simplify the user interaction with his phone, by sending a push notification to the user's smartphone and prompting him to approve the current login attempt. The login approval dialogue prompt contains contextual information about the particular login attempt, for example, the username, the service being accessed and the IP address together with geoIP location information. The user is given the option to approve or reject the attempt. A similar approach can also be used to confirm user online actions, such as financial transactions. An example of this is the use of the Transakt application [67] which prompts the user to authorize online purchases made with his credit card.

Hardware/Software Token Comparison. Software tokens are considered more usable compared to hardware ones, as the user does not have to carry an extra device with him. Rather, he only has to carry his smartphone or wearable device which, arguably, he carries with him most of the time. On the other hand, software tokens require the user to possess a smartphone or wearable device. Nevertheless, software tokens typically mean decreased or even non-existing deployment costs for the service providers, compared to hardware tokens which need to be purchased and shipped to the customers (although sometimes the provider may choose to shift the hardware token deployment costs to the users). As far as security is concerned, hardware tokens are considered more secure that software ones, since the former run on dedicated hardware while the latter run on general purpose mobile platforms which can be more easily infected with malware, even remotely. Depending on the type of malware, this can lead to full compromise of the second factor.

Although software tokens are generally considered more usable, as mentioned above, both hardware and software tokens still require the user to interact with the token in some way in order to successfully complete the login attempt. This interaction adds extra burden to the user and has an impact on the usability of all these schemes, hardware and software ones alike.

4.1.1 Reduced-Interaction 2FA

Various online service providers (e.g., Google, Facebook, Github, Microsoft, Apple, Dropbox and many more) offer one or more of the standard 2FA solutions we described above (both hardware-based as well as software-based). In these cases, 2FA is typically offered as an optional security mechanism to the users in order to better protect their accounts. Nevertheless, according to recent research [36, 165], 2FA mechanisms witness very low adoption rates when offered as an optional mechanism. At the same time, various banking institutions are using such 2FA mechanisms in order to enhance the security of their e-banking access and online transactions. However, in these cases 2FA is typically a mandatory requirement.

Arguably, one of the reasons due to which 2FA faces low adoption rates when optional is its poor usability. Users are accustomed to the password-only authentication experience, and thus, anything that moves away from that experience is considered cumbersome and faces adoption difficulties [86, 233]. Existing 2FA solutions in particular require the extra step of user interaction with his second factor device, as described previously. Researchers have identified this issue and have proposed novel 2FA techniques that try to minimize the interaction between the user and his second factor device, in order to increase the usability of 2FA. We discuss such solutions in the next paragraphs

Short-range Radio Communication. PhoneAuth [51] is a 2FA proposal that leverages unpaired Bluetooth communication between the browser and the phone, in order to eliminate the user to phone interaction. The fact that the Bluetooth communication is unpaired does away with the need to pair the user's smartphone with every computer from which he attempts to log in to his account. The Bluetooth channel enables the server (through the browser) and the phone to engage in a challenge-response protocol which provides the second authentication factor. Similarly, [163] and [191] also leverage Bluetooth communication between the browser and the phone.

These schemes require the browser to expose a Bluetooth API that is currently not available on any browser. A specification to expose a Bluetooth API in browsers has been proposed by the Web Bluetooth Community Group [231]. It is unclear whether the proposed API will support the unauthenticated RFCOMM or similar functionality which is required to enable seamless connectivity between the browser and the phone. However, if the Bluetooth connection is unauthenticated, an adversary equipped with a powerful antenna may connect to the victim's phone from afar [236] and login on behalf of the user, despite 2FA. Distance-bounding protocols [172] can prevent such range-extension attacks. Nevertheless, today's phones and computers are not equipped with the appropriate hardware to run such protocols.

Authy [10], besides the code-based solution mentioned previously, is another approach that allows for seamless 2FA using Bluetooth communication between the computer and the phone. In this case, the code is transferred automatically from the phone to the browser via the Bluetooth channel. In order to enable this capability, nevertheless, extra software must be installed on the user's computer.

As an alternative to Bluetooth, the browser and the phone can communicate over WiFi [191]. This approach only works when both devices are on the same network. Shirvanian et al., [191] use extra software on the computer to virtualize the wireless interface and create a software access point (AP) with which the phone needs to be associated. The user has to perform this setup procedure every time he uses a new computer to log in. Their solution also requires a phone application listening for incoming connections in the background, which is currently not possible on iOS.

Finally, the browser and the phone can communicate over NFC. NFC hardware is not commonly found in commodity computers, and current browsers do not expose APIs to access NFC hardware. Moreover, similar to Bluetooth communication, the NFC communication range can be extended if an adversary uses an appropriate directional antenna [54, 90]. Furthermore, a solution based on NFC would not completely remove the user to phone interaction because the user would still need to hold his phone close to the computer.

We note that 2FA mechanisms that employ direct communication between the browser and the phone may provide additional security against remote attacks. For example, the phone can detect if the user tries to login on a phishing website and block the attempt [51, 163]. The scheme in [191] further resists offline dictionary attacks against compromised hashed password databases. Nevertheless, none of such solutions can be deployed for the reasons we discussed above.

Location Information. As an alternative approach when the user is logging in, the server can check if the computer and the phone are co-located by comparing their GPS coordinates. GPS sensors are available on all modern phones but are rare on commodity computers. If the computer from which the user logs in has no GPS sensor, it can use the geolocation API exposed by some browsers [146]. Nevertheless, information retrieved via the geolocation API may not be accurate, for example when the device is behind a VPN or it is connected to a large managed network (such as enterprise or university networks). Furthermore, geolocation information can be easily guessed by an adversary. For example, assume the adversary knows the location of the victim's workplace and uses that location as the second authentication factor. This attack is likely to succeed during working hours since the victim is presumably at his workplace.

Near-ultrasound. SlickLogin [207] minimizes the user-phone interaction transferring the verification code from the computer to the phone using near-ultrasounds. The idea is to use spectrum frequencies that are non-audible for the majority of the population but that can be reproduced by the speakers of commodity computers (> 18kHz). Using non-audible frequencies accommodates for scenarios where users may not want their devices to make audible noise. Due to their size, the speakers of commodity computers can only produce highly directional near-ultrasound frequencies [184]. Near-ultrasound signals also attenuate faster, when compared to sounds in the lower part of the spectrum (< 18kHz) [8, 91]. With Slick-Login, the user must ensure that the speaker volume is at a sufficient level during login. Also, login will fail if a headset is plugged into the laptop. Finally, this approach may not work in scenarios where there is in-band noise (e.g., when listening to music or in cafes) [91]. We also note that a solution based on near-ultrasounds may result unpleasant for young people and animals that are capable of hearing sounds above 18kHz [180].

Other Sensors. A 2FA mechanism can combine the readings of multiple sensors that measure ambient characteristics, such as temperature, concentration of gases in the atmosphere, humidity, and altitude, as proposed in [192]. These combined sensor modalities can be used to verify the proximity between the computer through which the user is trying to login and his phone. However, today's computers and phones lack the hardware sensors that are required for such an approach to work.

4.1.2 Biometrics

All the previously surveyed 2FA solutions make use of a token (either hardware or software) as a second authentication factor. The user has to prove to the server that is in possession of this token. Biometrics can be used as an alternative authentication factor, replacing the token, or even password-based authentication as a whole. Biometrics rely on a physical property or trait of the user, which is presumed to be unique among the user base.

Fingerprints are one of the most popular biometric traits to authenticate a user [174]. In recent years the use of fingerprint authentication have seen widespread use through the integration of fingerprint sensors in mobile devices. On a high level fingerprint authentication works as follows. During the one-time registration phase the user provides the fingerprint of one or more of his fingers by placing them a few times on the reader, in order to train the system. After this, during the login phase the user places again one of his registered fingers on the reader and the newly captured scan is compared with the training fingerprint data. Fingerprint authentication is susceptible to forgery attacks. In particular, it has been shown that it is relatively easy to recreate a person's fingerprint by extracting it from an object he has previously touched [40, 41]. Advanced fingerprint sensors employ techniques that perform additional checks for the "liveness" of the scanned fingerprint, nevertheless similar attacks work against these scanners, too.

A variety of other traits are used in biometric systems such as iris [69], voice, face [151] and palm veins [75, 126]. As further, more recent examples, researchers have explored the use of a person's unique body electrical transmission [173] or eye movements [62] as biometric traits. The bone conduction of sound through the user's skull has been proposed as a way for authenticating people using eyewear devices [187]. Similar to fingerprint authentication, all these proposals rely on a reference value of the user's trait that is obtained during a training phase, which is subsequently matched against fresh scans of the trait when the user attempts to authenticate himself. Depending on the use case, the reference value is stored either on a remote server, or locally on the user's device.

Behavioral biometrics, is another type of biometric authentication, in which the user is authenticated based on the way he behaves and interacts with the system. Examples of behavioral biometric traits include the user's keyboard typing pattern[190], mouse movements [103] and touchscreen interaction [74]. The user's behavior is constantly monitored by the system which only grants him access as long as the observed behavior matches the one that the user registered into the system during the training phase.

Overall, biometrics typically offer a convenient way of authenticating the user. However, they can be hard to keep secret and stolen biometrics can have lifelong security implications as they cannot be reset, like passwords or other credentials.

4.2 Summary

Two-factor authentication is a promising technology for overcoming the security limitations of password-based authentication on the web. A large number of diverse solutions are currently deployed in the real world or have been proposed by research. Nevertheless, all these solutions see little or no adoption for the vast majority of cases in which 2FA is offered as an optional security feature.

We argue that one of the main reasons for this is the impact of the 2FA solutions on usability, as they diverge from the typical password-only experience which users are used to. Consequently, as part of our research we try to remove this limitation and propose a 2FA scheme which feature a user experience that is similar to password-only authentication with the hope that this will increase user adoption.

Chapter 5
Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound

Software tokens on modern phones are replacing dedicated hardware tokens in two-factor authentication (2FA) mechanisms. Using a software token, in place of a hardware one, improves deployability and usability of 2FA. For service providers, 2FA based on software tokens results in a substantial reduction of manufacturing and shipping costs. From the user's perspective, there is no extra hardware to carry around and phones can accommodate software tokens from multiple service providers.

Despite the improvements introduced by software tokens, most users still prefer password-only authentication for services where 2FA is not mandatory [36, 165]. This is probably due to the extra burden that 2FA causes to the user [86, 233], since it typically requires the user to interact with his phone.

Recent work [51, 191] improves the usability of 2FA by eliminating the user-phone interaction. However, those proposals are not yet deployable as their requirements are not met by today's phones, computers or browsers.

In this chapter, we focus on both the usability and deployability aspect of 2FA solutions. We propose *Sound-Proof*, a two-factor authentication mechanism that is transparent to the user and can be used with current phones and with major browsers without any plugin. In Sound-Proof the second authentication factor is the proximity of the user's phone to the computer being used to log in. When the user logs in, the two devices record the ambient noise via their microphones. The phone compares the two recordings, determines if the computer is located in the same environment, and ultimately decides whether the login attempt is legitimate or fraudulent.

Sound-Proof does not require the user to interact with his phone. The overall user experience is, therefore, close to password-only authentication. Sound-Proof works even if the phone is in the user's pocket or purse, and both indoors and outdoors. Sound-Proof can be easily deployed since it is compatible with current phones, computers and browsers. In particular, it works with any HTML5-compliant browser that implements the WebRTC API [85], which is currently being standardized by the W3C [226]. When Sound-Proof was originally published as an academic paper, only Google

Chrome, Mozilla Firefox and Opera were supporting WebRTC. At the time of writing this thesis, Microsoft has adopted WebRTC with its Edge browser, and Apple Safari supports it since version 11. We anticipate that other browsers, including mobile browsers, will adopt it soon.

Similar to other approaches that do not require user-phone interaction nor a secure channel between the phone and the computer (e.g., [51]), Sound-Proof is not designed to protect against targeted attacks where the attacker is co-located with the victim and has the victim's login credentials. Our design choice favors usability and deployability over security and we argue that this can edge for larger user adoption.

We have implemented a prototype of Sound-Proof for both Android and iOS. Sound-Proof adds, on average, less than 5 seconds to a password-only login operation. This time is substantially shorter than the time overhead of 2FA mechanisms based on verification codes (roughly 25 seconds [232]). We also report on a user study we conducted which shows that users prefer Sound-Proof over Google 2-Step Verification [83].

Contributions. We focus on two-factor authentication for web application logins and make the following contributions.

- We propose Sound-Proof, a novel 2FA mechanism that does not require user-phone interaction and is easily deployable. The second authentication factor is the proximity of the user's phone to the computer from which he is logging in. Proximity of the two devices is verified by comparing the ambient audio recorded via their microphones. Recording and comparison are transparent to the user.
- We implement a prototype of our solution for both Android and iOS. We use the prototype to evaluate the effectiveness of Sound-Proof in a number of different settings. We show that Sound-Proof works even if the phone is in the user's pocket or purse and that it fares well both indoors and outdoors.
- We conducted a user study to compare the perceived usability of Sound-Proof and Google 2-Step Verification. Participants ranked the usability of Sound-Proof higher than the one of Google 2-Step Verification, with a statistically significant difference. More importantly, we found that most participants stated that they would use Sound-Proof even if 2FA were optional.

The rest of this chapter is organized as follows. Section 5.1 details our assumptions and goals. Section 5.2 provides an overview on audio similarity techniques. We present Sound-Proof in Section 5.3 and its prototype implementation in Section 5.4. Section 5.5 evaluates Sound-Proof, while Section 5.6 reports on our user study. We discuss limitations and ways to further improve Sound-Proof in Section 5.7. Section 5.8 reviews related work and Section 5.9 concludes Part I and discusses interesting future research directions.

5.1 Assumptions and Goals

5.1.1 System Model

We assume the general settings of the commonly used password-based authentication on the web, which we briefly described in Section 2.2.1. The user has a username and a password to authenticate to a web server. The server, in addition, implements a 2FA mechanism that uses software tokens on phones.

The user navigates his browser to the server's login webpage and enters his username and password. The server verifies the validity of the password and challenges the user to prove possession of the second authentication factor.

5.1.2 Threat Model

We assume a remote adversary who has obtained the victim's username and password via phishing, leakage of a password database, or via other means. His goal is to authenticate to the server on behalf of the user. In particular, the adversary visits the server's webpage and enters the username and password of the victim. The attack is successful if the adversary convinces the server that he also holds the second authentication factor of the victim.

We further assume that the adversary cannot compromise the victim's phone. If the adversary gains control of the platform where the software token runs, then the security of any 2FA scheme reduces to the security of password-only authentication. Also, the adversary cannot compromise the victim's computer. The compromise of the computer allows the adversary to mount a Man-In-The-Browser (MITB) attack [158] and hijack the victim's session with the server, therefore defeating any 2FA mechanism. Moreover, while 2FA would prevent an attacker from logging in from a different computer, a MITB attack can enable the attacker to steal the user's session management cookie (which is used by the server to identify the user's session following successful authentication – see Section 2.2.1) and use it on another computer to access the user's account.

We do not address targeted attacks where the adversary is co-located with the victim. 2FA mechanisms that do not require the user to interact with his phone cannot protect against targeted, co-located attacks. For example, if 2FA uses unauthenticated short-range communication [51], a co-located attacker can connect to the victim's phone and prove possession of the second authentication factor to the server. We argue that targeted, co-located attacks are less common than non-selective, remote attacks. Furthermore, any 2FA mechanism may not warrant protection against powerful or sufficiently skilled attackers who can place themselves in close proximity with the victim. For example, if 2FA uses verification codes, a determined attacker may gain physical access to the phone or read the code from a distance [11, 12, 171].

We do not consider TLS Man-In-The-Middle adversaries. As we show in Part II, client authentication is not sufficient to defeat MITM attacks in the context of web applications. We also do not address active phishing attacks where the attacker lures the user into visiting a phishing website and relays the stolen credentials to the legitimate website in real-time. Such attacks can be thwarted by having the phone detect the phishing domain [51, 163]. This requires short-range communication between the phone and the browser. However, seamless short-range communication between the phone and the browser is currently not possible. (At the time of writing this thesis, this may be about to change [231]).

5.1.3 Design Goals

Our design goals for a novel two-factor authentication solution for web applications are the following.

- *Security.* The 2FA mechanism should enhance the security of password-only authentication, against the common, remote attacks. In particular, if an attacker manages to steal the user's password, he should still not be able to login as the user and compromise his account. We stress, as mentioned in our threat model, this does not include targeted, co-located attacks.

- *Usability.* Users should authenticate using only their username and password as in password-only authentication. In particular, users should not be asked to interact with their phone — not even to pick up the phone or take it out of a pocket or purse.

- *Deployability.* The 2FA mechanism should work with common smartphones, computers and browsers. It should not require additional software on the computer or the installation of browser plugins. A plugin-based solution limits the usability of the system because (*i*) a different plugin may be required for each server, and (*ii*) the user must install the plugin every time he logs in from a computer for the first time. The mechanism should also work on a wide range of smartphones. We therefore discard the use of special hardware on the phone like NFC chips or biometric sensors.

5.2 Background on Sound Similarity

The problem of determining the similarity of two audio samples is close to the problem of audio fingerprinting and automatic media retrieval [39]. In media retrieval, a noisy recording is matched against a database of reference samples. This is done by extracting a set of relevant features from the noisy recording and comparing them against the features of the reference samples. The extracted features must be robust to, for example, background noise and attenuation. Bark Frequency Cepstrum Coefficients [87], wavelets [14] or peak frequencies [230] have been proposed as robust features for automatic media retrieval. Such techniques focus mostly on the frequency domain representation of the samples because they deal with time-misaligned samples. In our scenario, we compare two quasi-aligned samples (the offset is less than 150ms) and we therefore can also extract relevant information from their time domain representations.

In order to consider both time domain and frequency domain information of the recordings, we use one-third octave band filtering and cross-correlation.

One-third Octave Bands. Octave bands split the audible range of frequencies (roughly from 20Hz to 20kHz) in 11 non-overlapping bands where the ratio of the highest in-band frequency to the lowest in-band frequency is 2 to 1. Each octave is represented by its center frequency, where the center frequency of a particular octave is twice the center frequency of the previous octave. One-third octave bands split the first 10 octave bands in three and the last octave band in two, for a total of 32 bands. One-third octave bands are widely used in acoustics and their frequency ranges have been standardized [208]. The center frequency of the lowest band is 16Hz (covering from 14.1Hz to 17.8Hz) while the center frequency of the highest band is 20kHz (covering from 17780Hz to 22390Hz). In the following we denote with $B = [lb - hb]$ a set of contiguous one-third octave bands, from the band that has its central frequency at lbHz, to the band that has its central frequency at hbHz.

Splitting a signal in one-third octave bands provides high frequency resolution information of the original signal, while keeping its time-domain representation.

Cross-correlation. Cross-correlation is a standard measure of similarity between two time series. Let $x,\ y$ denote two signals represented as n-points discrete time series. As an example, 16-bit PCM audio signals have each sample value range between -32768 to +32767. For simplicity we assume both series to have the same length. The cross-correlation $c_{x,y}(l)$ measures their similarity as a function of the lag $l \in [0, n-1]$ applied to y:

$$c_{x,y}(l) = \sum_{i=0}^{n-1} x(i) \cdot y(i-l)$$

where $y(i) = 0$ if $i < 0$ or $i > n-1$.

To accommodate for different amplitudes of the two signals, the cross correlation can be normalized as:

$$c'_{x,y}(l) = \frac{c_{x,y}(l)}{\sqrt{c_{x,x}(0) \cdot c_{y,y}(0)}}$$

where $c_{x,x}(l)$ is known as auto-correlation.

The normalization maps $c'_{x,y}(l)$ in $[-1,1]$. A value of $c'_{x,y}(l) = 1$ indicates that at lag l, the two signals have the same shape even if their amplitudes may be different; a value of $c'_{x,y}(l) = -1$ indicates that the two signals have the same shape but opposite signs. Finally, a value of $c'_{x,y}(l) = 0$ shows that the two signals are uncorrelated.

If the actual lag between the two signals is unknown, we can discard the sign information and use the absolute value of the maximum cross-correlation $\hat{c}_{x,y}(l) = \max_l(|c'_{x,y}(l)|)$ as a metric of similarity $(0 \leq \hat{c}_{x,y}(l) \leq 1)$. The computation overhead of $c_{x,y}(l)$ can be decreased by leveraging the cross-correlation theorem and computing $c_{x,y}(l) = F^{-1}(F(x)^* \cdot F(y))$, where $F()$ denotes the discrete Fourier transform and the asterisk denotes the complex conjugate.

5.3 Sound-Proof Architecture

The second authentication factor of Sound-Proof is the proximity of the user's phone to the computer being used to log in. The proximity of the two devices is determined by computing a similarity score between the ambient noise captured by their microphones. For privacy reasons we do not upload cleartext audio samples to the server. In our design, the computer encrypts its audio sample under the public key of the phone. The phone receives the encrypted sample, decrypts it, and computes the similarity score between the received sample and the one recorded locally. Finally, the phone reports to the server whether to accept or reject the login, based on the result of the similarity score function. Note that the phone never uploads its recorded sample to the server. Communication between the computer and the phone goes through the server. We avoid short-range communication between the phone and the computer (e.g., via Bluetooth) because it requires changes to the browser or the installation of a plugin.

Computing the similarity score on the phone offers two benefits. First, user privacy is protected as the phone audio sample is never uploaded to the server. This implies that a potentially curious server cannot spy on the user by arbitrarily querying the phone pretending a Sound-Proof authentication is taking place and retrieving the recordings of the phone. Second, the computation of the similarity score is the most CPU-intensive operation of the Sound-Proof protocol, so offloading it to the mobile phone can avoid the server becoming overloaded when a large number of concurrent Sound-Proof authentications are taking place by multiple users.

5.3.1 Similarity Score

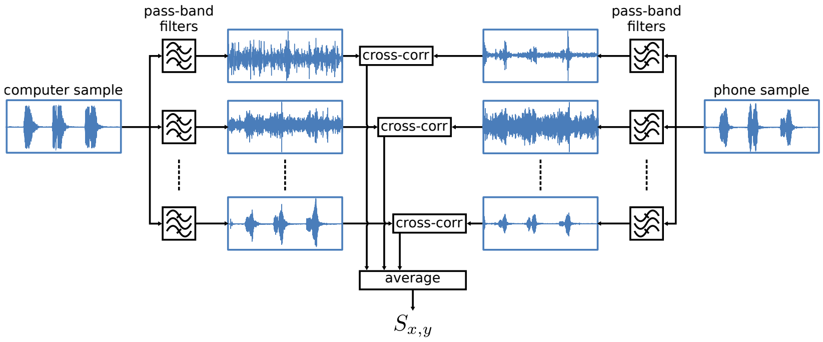


Figure 5.1: Block diagram of the function that computes the similarity score between two samples. The computation takes place on the phone. If $S_{x,y} > \tau_C$ and the average power of the samples is greater than τ_{dB}, the phone judges the login attempt as legitimate.

Figure 5.1 shows a block diagram of the function that computes the similarity score between the two recorded samples. Each audio signal is input to a bank of pass-band filters to obtain n signal components, one per each of the one-third octave bands that we take into account. Let x_i be the signal component for the i-th one-third octave band of signal x. The similarity score is the average of the maximum cross-correlation over the pairs of signal components $x_i,\ y_i$:

$$S_{x,y} = \frac{1}{n}\sum_{i=1}^{i=n} \hat{c}_{x_i,y_i}(l)$$

where l is bounded between 0 and ℓ_{max}.

5.3.2 Enrollment and Login

Similar to other 2FA mechanisms based on software tokens, Sound-Proof requires the user to install an application on his phone and to bind the application to his account on the server. This one-time operation can be carried out leveraging existing techniques to enroll software tokens, e.g., using a QR code image which is generated by the server and scanned using the application on the phone. We assume that, at the end of the phone enrollment procedure, the server receives the unique public key of the application on the user's phone and binds that public key to the account of that user.

Figure 5.2 shows an overview of the login procedure. The user points the browser to the URL of the server and enters his username and password. The server retrieves the public key of the user's phone and sends it to the browser. Both the browser and the phone start recording through their local microphones for t seconds. During recording, the two devices synchronize their clocks with the server. When recording completes, each device adjusts the timestamp of its sample taking into account the clock difference with the server. The browser encrypts the audio sample under the phone's public key and sends it to the phone, using the server as a proxy. The phone decrypts the browser's sample and compares it against the one recorded locally. If the average power of both samples is above τ_{dB} and the similarity score is above τ_C, the phone concludes that it is co-located with the computer from which the user is logging in and informs the server that the login is legitimate.

The procedure is completely transparent to the user if the environment is sufficiently noisy. In case the environment is quiet, Sound-Proof requires the user to generate some noise, for example by clearing his throat.

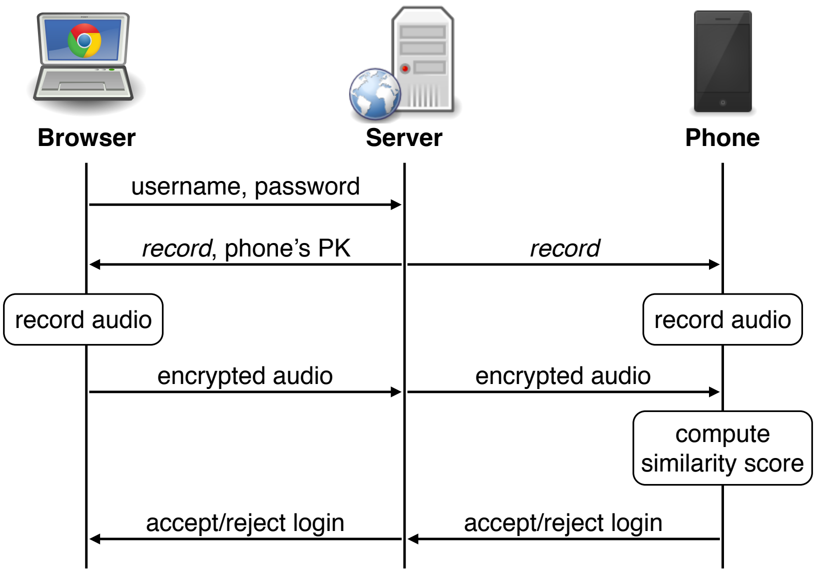


Figure 5.2: Sound-Proof authentication overview. During login, the phone and the computer record ambient noise with their microphones. The phone computes the similarity score between the two samples and returns the result to the server.

5.3.3 Security Analysis

Remote Attacks. The security of Sound-Proof stems from the attacker's inability to guess the sound in the victim's environment at the time of the attack.

Let x be the sample recorded by the victim's phone and let y be the sample submitted by the attacker. A successful impersonation attack requires the average power of both signals to be above τ_{dB}, and each of the one-third octave band components of the two signals to be highly correlated. That is, the two samples must satisfy $Pwr(x) > \tau_{dB}$, $Pwr(y) > \tau_{dB}$ and $S_{x,y} > \tau_C$ with $l < \ell_{max}$.

We bound the lag l between 0 and ℓ_{max} to increase the security of the scheme against an adversary that successfully guesses the noise in the victim's environment at the time of the attack. Even if the adversary correctly guesses the noise in the victim's environment and can submit a similar audio sample, the two samples must be synchronized with an error smaller than ℓ_{max}. We also reject audio pairs where either sample has an average power below the threshold τ_{dB}. This is in order to prevent an impersonation attack when the victim's environment is quiet (e.g., while the victim is sleeping).

Quantifying the entropy of ambient noise, and hence the likelihood of the adversary guessing the signal recorded by the victim's phone, is a challenging task. Results are dependent on the environment, the language spoken by the victim, his gender or age to cite a few. In Section 5.5 we provide empirical evidence that Sound-Proof can discriminate between legitimate and fraudulent logins, even if the adversary correctly guesses the type of environment where the victim is located.

Co-located Attacks. Sound-Proof cannot withstand attackers who are co-located with the victim. A co-located attacker can capture the ambient sound in the victim's environment and thus successfully authenticate to the server, assuming that he also knows the victim's password. Sound-Proof shares this limitation with other 2FA mechanisms that do not require the user to interact with his phone and do not assume a secure channel between the phone and the computer (e.g., [51]). Resistance to co-located attackers requires either a secure phone-to-computer channel (as in [10, 191]) or user-phone interaction (as in [61, 83]). However, both techniques impose a significant usability burden which would contradict our design goal of usability.

Rogue Third-party Servers. Assume a user who has an online account on two different websites, *A* and *B*, for which he uses the same credentials. If an attacker manages to compromise one of the web servers, say *A*, and remain resident on it, he is then able to successfully compromise the user's account on server *B*, even if it uses Sound-Proof, in the following way. When the user tries to login on his account at website *A*, the compromised server *A* can start the authentication procedure with the target server *B* (for which it knows the user's credentials) and relay the messages to the user's browser. The JavaScript code running in the user's browser is then fully controlled by the attacker and can be used to record an audio sample that will be similar to the sample recorded by the user's phone, which should presumably be in close proximity. This attack is not strictly an active phishing attack but it can be categorized as one. In particular, the attacker does not have to fool the victim that he is visiting a known domain but, by controlling a server to which the user authenticates to, effectively achieves a similar result. We argue that this is a powerful attack that would succeed against other 2FA solutions as well, such as Google 2-Step Verification or SMS-based approaches. This is due to the fact that the compromised website can fool the user into revealing his one-time-code to the attacker.

Forcing Predictable Phone Sounds. After Sound-Proof was originally published and by the time of writing this thesis, Shrestha et al. [193] proposed an attack against Sound-Proof, named *Sound-Danger*. Sound-Danger is based on the fact that the user's phone can be made to produce predictable sounds (e.g., ringer, notification, alarm or vibration sounds) and works as follows. An attacker manages to compromise the user's credentials through a database breach. The breached data, besides the credentials, is assumed to contain other user information, such as the user's phone number, with the help of which the attacker is able to contact the user's phone, e.g., by placing a regular or VoIP call or sending an SMS or other chat application message. Thus, at the time of attack, the attacker contacts the user's phone, which makes it produce a predictable sound which will be subsequently captured in the phone's Sound-Proof audio recording. The attacker submits a similar audio sample, in the place of the browser recording, making audio comparison to pass and thereby allowing the attacker to log in as the user.

We acknowledge this as a valid vulnerability that significantly increases the attacker's chances to correctly guess the ambient audio captured by the phone and thus defeat Sound-Proof. Nevertheless, we note that in practice it can be easily prevented or detected in various ways. For example, the Sound-Proof application can detect if the phone is emitting audio or is vibrating while the recording is taking places and reject the particular authentication attempt, thereby foiling the attack. Alternatively, on Android phones, it is possible to temporarily silence the phone's ringer while Sound-Proof is recording, although this is rather obtrusive and not preferred over the aforementioned detection-based approach. Finally, due to the way the iOS audio framework operates, it is not possible for the Sound-Proof application to record at the same time when the phone is playing any audio (such as ringing), which renders the attack infeasible on this platform.

5.4 Prototype Implementation

Our implementation works with multiple browsers. We tested it with Google Chrome (version 38.0.2125.111), Mozilla Firefox (version 33.0.2) and Opera (version 25.0.1614.68). We anticipate the prototype to work with different versions of these browsers, as long as they implement the `navigator.getUserMedia()` API of WebRTC. At the time of writing this thesis, we also tested and confirmed that our implementation works with Microsoft Edge (version 40.15063.0.0) and Apple Safari (version 11.0.1). We tested the phone application both on Android and on iOS. For Android, on a Samsung Galaxy S3, a Google Nexus 4 (both running Android version 4.4.4), a Sony Xperia Z3 Compact and a Motorola Nexus 6 (running Android version 5.0.2 and 5.1.1, respectively). We also tested different iPhone models (iPhone 4, 5 and 6) running iOS version 7.1.2 on the iPhone 4, and iOS version 8.1 on the newer models. The phone application should work on different phone models and with different OS versions without major modifications.

Web Server and Browser. The server component is implemented using the CherryPy [209] web framework and MySQL database. We use WebSocket [70] to push data from the server to the client. The client-side (browser) implementation is written entirely in HTML and JavaScript. Encryption of the audio recording uses AES-256 with a fresh symmetric key; the symmetric key is encrypted under the public key of the phone using RSA-2048. We use the HTML5 WebRTC API [85, 226]. In particular, we use the `navigator.getUserMedia()` API to access the local microphone from within the browser. Our prototype does not require browser code modifications, extensions or plugins.

Software Token. We implement the software token as an Android application as well as an iOS application. The mobile application stays idle in the background and is automatically activated when a push notification arrives. Push messages for Android and iOS use the Google GCM (Google Cloud Messaging) APIs [84] and Apple's APN (Apple Push Notifications) APIs [7] (in particular the silent push notification feature), respectively. Phone to server communication is protected with TLS.

Most of the Android code is written in Java (Android SDK), while the component that processes the audio samples is written in C (Android NDK). In particular, we use the ARM Ne10 library, based on the ARM NEON engine [9] to optimize vector operations and FFT computations. The iOS application is written in Objective-C and uses Apple's vDSP package of the Accelerate framework [6], in order to leverage the ARM NEON technology for vector operations and FFT computations. On both mobile platforms we parallelize the computation of the similarity score across the available processor cores.

Operations	Mean (ms)	Std.Dev.
Recording	3000	—
Similarity score computation	642	171
Cryptographic operations	118	15
Networking		
WiFi	978	135
Cellular	1243	209

Table 5.1: Overhead of the Sound-Proof prototype. On average it takes 4677ms (± 181ms) over WiFi and 4944ms (± 233ms) over Cellular to complete the 2FA verification.

Time Synchronization. Sound-Proof requires the recordings from the phone and the computer to be synchronized. For this reason, the two devices run a simple time-synchronization protocol (based on the Network Time Protocol [148]) with the server. The protocol is implemented over HTTP and allows each device to compute the difference between the local clock and the one of the server. Each device runs the time-synchronization protocol with the server while it is recording via its microphone. When recording completes, each device adjusts the timestamp of its sample taking into account the clock difference with the server.

Run-time Overhead. We compute the run-time overhead of Sound-Proof when the phone is connected either through WiFi or through the cellular network. We run 1000 login attempts with a Google Nexus 4 for each connection type, and we measure the time from the moment the user submits his username and password to the time the web server logs the user in. On average it takes 4677ms (± 181ms) over WiFi and 4944ms (± 233ms) over Cellular to complete the 2FA verification. Table 5.1 shows the average time and the standard deviation of each operation. The recording time is set to 3 seconds. The similarity score is computed over the set of one-third octave bands $B = [50\text{Hz} - 4\text{kHz}]$. (Section 5.5.1 discusses the selection of the band set.) After running the time-synchronization protocol, the resulting clock difference was, on average, 42.47ms (± 30.35ms).

5.5 Evaluation

Data Collection. We used our prototype to collect a large number of audio pairs. We set up a server that supported Sound-Proof. Two subjects logged in using Google Chrome[1] over 4 weeks. At each login, the phone and the computer recorded audio through their microphones for 3 seconds. We stored the two audio samples for post-processing. Login attempts differed in the following settings.

- *Environment:* an office at our lab with either no ambient noise (labelled as Office) or with the computer playing music (Music); a living-room with the TV on (TV); a lecture hall while a faculty member was giving a lecture (Lecture); a train station (TrainStation); a cafe (Cafe).
- *User activity:* being silent, talking, coughing, or whistling.
- *Phone position:* on a table or a bench next to the user, in the trouser pocket, or in a purse.
- *Phone model:* Apple iPhone 5 or Google Nexus 4.
- *Computer model:* Mac Book Pro "Mid 2012" running OS X10.10 Yosemite or Dell E6510 running Windows 7.

At the end of the 4 weeks we had collected between 5 and 15 login attempts per each setting, totaling 2007 login attempts (4014 audio samples).

5.5.1 Analysis

We used the collected samples to find the configuration of system parameters (i.e., τ_{dB}, ℓ_{max}, B, and τ_C) that led to the best results in terms of False Rejection Rate (FRR) and the False Acceptance Rate (FAR). A false rejection occurs when a legitimate login is rejected. A false acceptance occurs when a fraudulent login is accepted. A fraudulent login is accepted if the sample submitted by the attacker and the sample recorded by the victim's phone have a similarity score greater than τ_C, and if both samples have an average power greater than τ_{dB}.

[1]We used Google Chrome since it is currently the most popular browser [201]. We have also tested Sound-Proof with other browsers and have experienced similar performance (see Section 5.7).

To compute the FAR, we used the following strategy. For each phone sample collected by one of the subjects (acting as the victim), we use all the computer samples collected by the other subject as the attacker's samples. We then switch the roles of the two subjects and repeat the procedure. The total number of victim–adversary sample pairs we considered was 2,045,680.

System Parameters. We set the average power threshold τ_{dB} to 40dB which, based on our measurements, is a good threshold to reject silence or very quiet recordings like the sound of a fridge buzzing or the sound of a clock ticking. Out of 2007 login attempts we found 5 attempts to have an average power of either sample below 40dB and we discard them for the rest of the evaluation.

We set ℓ_{max} to 150ms because this was the highest clock difference experienced while testing our time-synchronization protocol (see Section 5.4).

An important parameter of Sound-Proof is the set B of one-third octave bands to consider when computing the similarity score described in Section 5.3.1. The goal is to select a spectral region that (*i*) includes most common sounds and (*ii*) is robust to attenuation and directionality of audio signals. We discarded bands below 50Hz to remove very low-frequency noises. We also discarded bands above 8kHz, because these frequencies are attenuated by fabric and they are not suitable for scenarios where the phone is in a pocket or a purse. We tested all sets of one-third octave bands $B = [x - y]$ where x ranged from 50Hz to 100Hz and y ranged from 630Hz to 8kHz.

We found the smallest Equal Error Rate (EER, defined as the crossing point of FRR and FAR) when using $B = [50\text{Hz}—4\text{kHz}]$. Figure 5.3 shows the FRR and FAR using this set of bands where the EER is 0.0020 at $\tau_C = 0.13$. We experienced worse results with one-third octave bands above 4kHz. This was likely due to the high directionality of the microphones found on commodity devices when recoding sounds at those frequencies [218]. Appendix D shows similar plots for all the band ranges B we tested starting from 50Hz to 100Hz and going up from 630Hz to 4kHz.

We also computed the best set of one-third octave bands to use in case usability and security are weighted differently by the service provider. For example, a social network provider may value usability higher than security. In particular, we computed the sets of bands that minimized $f = \alpha \cdot FRR + \beta \cdot FAR$, for $\alpha \in [0.1, \ldots, 0.9]$ and $\beta = 1 - \alpha$. Figure 5.4(b) shows the set of bands that provided the best results for each configuration of α and β. As before, we experienced better results with bands below

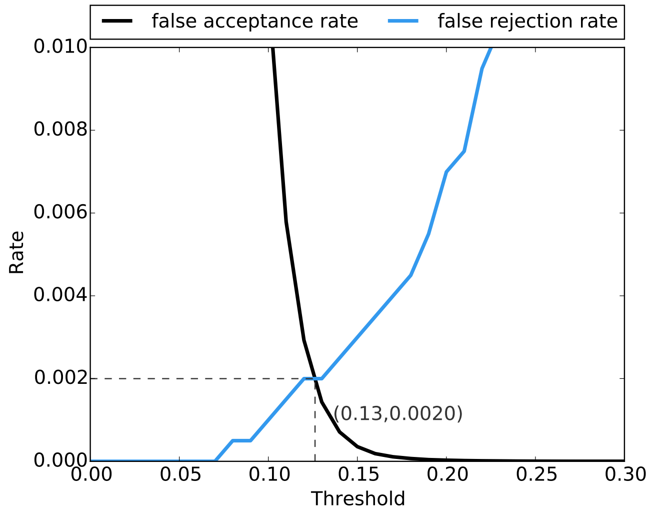


Figure 5.3: False Rejection Rate and False Acceptance Rate as a function of the threshold τ_C for $B = [50\text{Hz} - 4\text{kHz}]$. The Equal Error Rate is 0.0020 at $\tau_C = 0.13$.

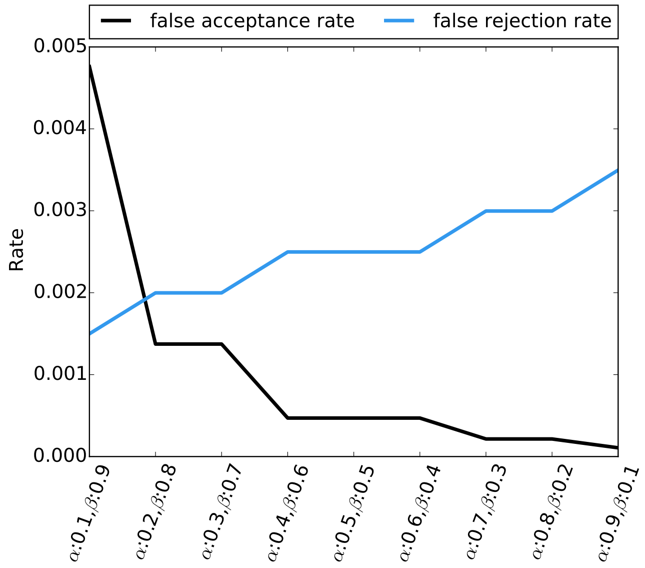
4kHz. Figure 5.4(a) plots the FRR and FAR against the possible values of α and β. We stress that the set of bands may differ across two different points on the x-axis.

Experiments in the remaining of this section were run with the configuration of the parameters that minimized the EER to 0.0020: $\tau_{dB} =$ 40dB, $\ell_{max} = 150$ms, $B = [50\text{Hz} - 4\text{kHz}]$, and $\tau_C = 0.13$.

5.5.2 False Rejection Rate

In the following we evaluate the impact of each setting that we consider (environment, user activity, phone position, phone model, and computer model) on the FRR. Figures 5.5 and 5.6 show a box and whisker plot for each setting. The whiskers mark the 5th and the 95th percentiles of the similarity scores. The boxes show the 25th and 75th percentiles. The line and the solid square within each box mark the median and the average, respectively. A gray line marks the similarity score threshold ($\tau_C = 0.13$) and each red dot in the plots denotes a login attempt where the similarity score was below that threshold (i.e., a false rejection).

Environment. Figure 5.5 shows the similarity scores for each environment. Sound-Proof fares equally well indoors and outdoors. We did not experience



(a) False Rejection Rate and False Acceptance Rate when usability and security have different weights.

	B	τ_c
$\alpha = 0.1,\ \beta = 0.9$	[80Hz − 2500Hz]	0.12
$\alpha = 0.2,\ \beta = 0.8$	[50Hz − 2500Hz]	0.14
$\alpha = 0.3,\ \beta = 0.7$	[50Hz − 2500Hz]	0.14
$\alpha = 0.4,\ \beta = 0.6$	[50Hz − 800Hz]	0.19
$\alpha = 0.5,\ \beta = 0.5$	[50Hz − 800Hz]	0.19
$\alpha = 0.6,\ \beta = 0.4$	[50Hz − 800Hz]	0.19
$\alpha = 0.7,\ \beta = 0.3$	[50Hz − 1000Hz]	0.2
$\alpha = 0.8,\ \beta = 0.2$	[50Hz − 1000Hz]	0.2
$\alpha = 0.9,\ \beta = 0.1$	[50Hz − 1250Hz]	0.21

(b) One-third octave bands and similarity score threshold.

Figure 5.4: Minimizing $f = \alpha \cdot FRR + \beta \cdot FAR$, for $\alpha \in [0.1, \dots, 0.9]$ and $\beta = 1 - \alpha$.

rejections of legitimate logins for the Music (over 432 logins), the Lecture (over 122 logins), and the TV (over 430 logins) environments. The FRR

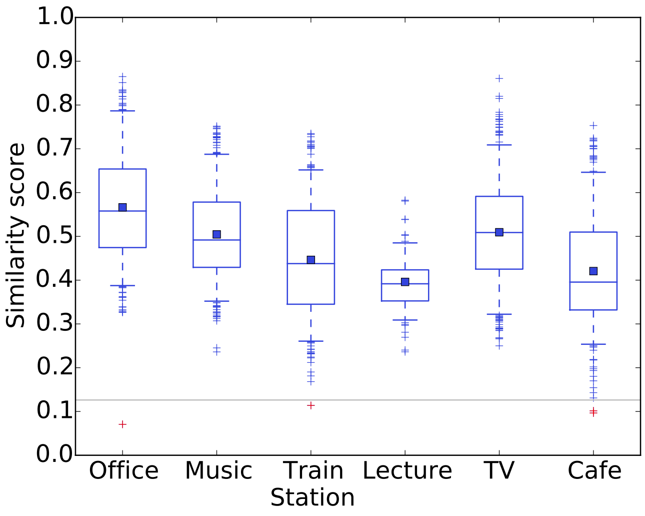


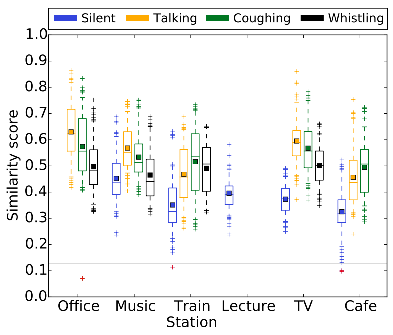
Figure 5.5: Impact of the environment on the False Rejection Rate.

was 0.003 (1 over 310 logins) for Office, 0.003 (1 over 370 logins) for TrainStation, and 0.006 (2 over 338 logins) for Cafe.

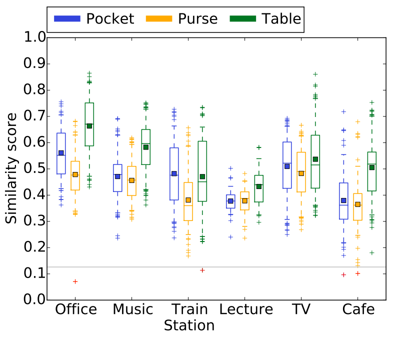
User Activity. Figure 5.6(a) shows the similarity scores for different user activities. In general, if the user makes any noise the similarity score improves. We only experienced a few rejections of legitimate logins when the user was silent (TrainStation and Cafe) or when he was coughing (Office). In the Lecture case the user could only be silent. We also avoided whistling in the cafe, because this may be awkward for some users. The FRR was 0.005 (3 over 579 logins) when the user was silent, 0.002 (1 over 529 logins) when the user was coughing, 0 (0 over 541 logins) when the user was speaking, and 0 (0 over 353 logins) when the user was whistling.

Phone Position. Figure 5.6(b) shows the similarity scores for different phone positions. Sound-Proof performs slightly better when the phone is on a table or on a bench. Worse performance when the phone is in a pocket or in a purse are likely due to the attenuation caused by the fabric around the microphone. The FRR was 0.001 (1 over 667 logins) with the phone on a table, 0.001 (1 over 675 logins) with the phone in a pocket, and 0.003 (2 over 660 logins) with the phone in a purse.

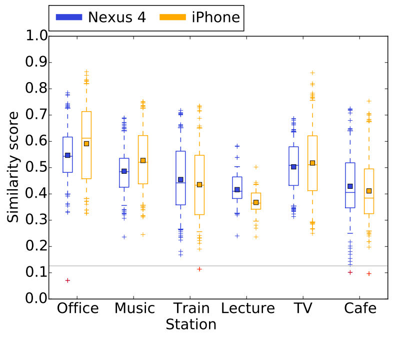
Phone Model. Figure 5.6(c) shows the similarity scores for the two phones. The Nexus 4 and the iPhone 5 performed equally good across all environ-



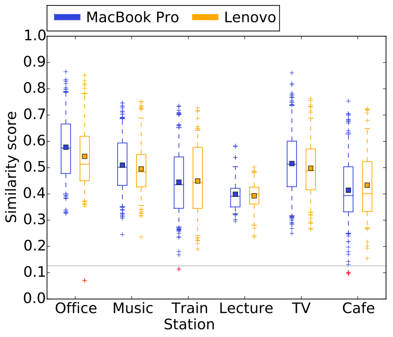
(a) User Activity



(b) Phone position



(c) Phone model



(d) Computer

Figure 5.6: Impact of user activity, phone position, phone model and computer model on the False Rejection Rate.

ments. The FRR was 0.002 (2 over 884 logins) with the iPhone 5 and 0.002 (2 over 1118 logins) with the Nexus 4.

Computer. Figure 5.6(d) shows the similarity scores for the two computers we used. We could not find significant differences between their performance. The FRR was 0.002 (3 over 1299 logins) with the MacBook Pro and 0.001 (1 over 703 logins) with the Dell.

Distance Between Phone and Computer. In some settings (e.g., at home), the user's phone may be away from his computer. For instance, the user could leave the phone in his bedroom while watching TV or working in another room. We evaluated this scenario by placing the computer close to the TV in a living-room, and testing Sound-Proof while the phone was away from the computer. For this set of experiments we used the iPhone 5 and the MacBook Pro. The average noise level by the TV was measured at 50dB. We tested 3 different distances: 4, 8 and 12 meters (running 20 login attempts for each distance). All login attempts were successful (i.e., FRR=0). We also tried to log in while the phone was in another room behind a closed door, but logins were rejected.

Discussion. Based on the above results, we argue that the FRR of Sound-Proof is small enough to be practical for real-world usage. To put it in perspective, the FRR of Sound-Proof is likely to be smaller than the FRR due to mistyped passwords (0.04, as reported in [112]).

5.5.3 Advanced Attack Scenarios

A successful attack requires the adversary to submit a sample that is very similar to the one recoded by the victim's phone. For example, if the victim is in a cafe, the adversary should submit an audio sample that features typical sounds of that environment. In the following we assume a strong adversary that correctly guesses the victim's environment. We also evaluate the attack success rate in scenarios where the victim and the attacker access the same broadcast audio source from different locations.

Similar Environment Attack. In this experiment we assume that the victim and the adversary are located in similar environments. For each environment, we compute the FAR between each phone sample collected by one subject (the victim) and all the computer samples of the other subject (the adversary). We then switch the roles of the two subjects and repeat the procedure. The FAR for the Music and the TV environments were 0.012 (1063 over 91960 attempts) and 0.003 (311 over 90992 attempts), respectively. The FAR for the Lecture environment was 0.001 (8 over 7242 attempts). When both the victim and the attacker were located at a train station the FAR was 0.001 (44 over 67098 attempts). The FAR for the Office environment was 0.025 (1194 over 47250 attempts). When both the victim and the attacker were in a cafe the FAR was 0.001 (32 over 56994 attempts).

The above results show low FAR even when the attacker correctly guesses the victim's environment. This is due to the fact that ambient noise in a given environment is influenced by random events (e.g., background chatter, music, cups clinking, etc.) that cannot be controlled or predicted by the adversary.

	False Acceptance Rate		
	SC-SP	SC-DP	DC-DP
TV channel 1	1	0.1	0.1
TV channel 2	1	1	0
TV channel 3	1	0	-
TV channel 4	1	0	-
Web radio 1	1	0	0.4
Web radio 2	0.1	0.8	0.8
Web TV 1	0	0	0
Web TV 2	0	0	0

Table 5.2: False Acceptance Rate when the adversary and the victim devices record the same broadcast media. SC-SP stands for “same city and same Internet/cable provider”, SC-DP stands for “same city but different Internet/cable providers”, DC-DP stands for “different cities and different Internet/cable providers”. A dash in the table means that the TV channel was not available at the victim’s location.

Same Media Attack. In this experiment we assume that the victim and the adversary access the same audio source from different locations. This happens, for example, if the victim is watching TV and the adversary correctly guesses the channel to which the victim’s TV is tuned. We place the victim’s phone and the adversary’s computer in different locations, but each of them next to a smart TV that was also capable of streaming web media. Since the devices have access to two identical audio sources, the adversary succeeds if the lag between the two audio signals is less than ℓ_{max}. We tested 4 cable TV channels, 2 web radios and 2 web TVs. For each scenario, we run the attack 100 times and report the FAR in Table 5.2. When the victim and the attacker were in the same city, we experienced differences based on the media provider. When the TVs reproduced content broadcasted by the same provider, the signals were closely synchronized and the similarity score was above the threshold τ_C. The FAR dropped in the case of web content. When the TVs reproduced content supplied by different providers, the lag between the signals caused the similarity score to drop below τ_C in most of the cases. The similarity score sensibly dropped when the victim and the attacker were located in different cities.

5.6 User Study

The goal of our user study was to evaluate the usability of Sound-Proof and to compare it with the usability of Google 2-Step Verification (2SV), since 2FA based on one-time codes is arguably the most popular. (We only considered the version of Google 2SV that uses an application on the user's phone to generate one-time codes.) We stress that the comparison focuses solely on the usability aspect of the two methods. In particular, we did not make the participants aware of the difference in the security guarantees, i.e., the fact that Google 2SV can better resist co-located attacks.

We ran repeated-measure experiments where each participant was asked to log in to a server using both mechanisms in random order. After using each 2FA mechanism, participants ranked its usability answering the System Usability Scale (SUS) [33]. The SUS is a widely-used scale to assess the usability of IT systems [15]. The SUS score ranges from 0 to 100, where higher scores indicate better usability.

5.6.1 Procedure

Recruitment. We recruited participants using a snowball sampling method. Most subjects were recruited outside our department and were not working in or studying computer science. The study was advertised as a user study to "evaluate the usability of two-factor authentication mechanisms". We informed participants that we would not collect any personal information and offered a compensation of CHF 20. Among all respondents to our email, we discarded the ones that were security experts and ended up with 32 participants.

Experiment. The experiment took place in our lab where we provided a laptop and a phone to complete the login procedures. Both devices were connected to the Internet through WiFi. We set up a Gmail account with Google 2SV enabled. We also created another website that supported Sound-Proof and mimicked the Gmail UI.

Participants saw a video where we explained the two mechanisms under evaluation. We told participants that they would need to log in using the account credentials and the hardware we provided. We also explained that we would record the keystrokes and the mouse movements (this allowed us to time the login attempts).

We then asked participants to fill in a pre-test questionnaire designed to collect demographic information. Participants logged in to our server using Sound-Proof and to Gmail using Google 2SV. We randomized the order in which each participant used the two mechanisms. After each login, participants rated the 2FA mechanism answering the SUS.

At the end of the experiment participants filled in a post-test questionnaire that covered aspects of the 2FA mechanisms under evaluation not covered by the SUS.

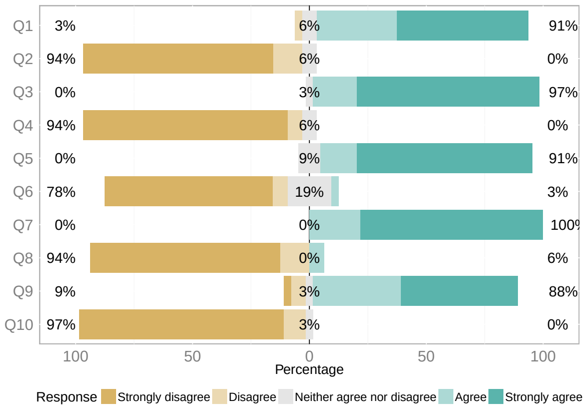
5.6.2 Results

Demographics. 58% of the participants were between 21 and 30 years old. 25% of the participants were between 31 and 40 years old. The remaining 17% of the participants were above 40 years old. 53% of the participants were female. 69% of the participants had a master or doctoral degree. 50% of the participants used 2FA for online banking and only 13% used Google 2SV to access their email accounts.

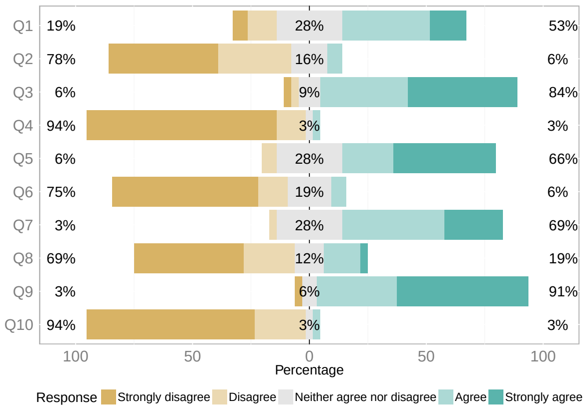
SUS Scores. The mean SUS score for Sound-Proof was 91.09 (±5.44). The mean SUS score for Google 2SV was 79.45 (±7.56). Figure 5.7(a) and Figure 5.7(b) show participant answers on 5-point Likert-scales for Sound-Proof and for Google 2SV, respectively. To analyze the statistical significance of these results, we used the following null hypothesis: "there will be no difference in perceived usability between Sound-Proof and Google 2SV". A one-way ANOVA test revealed that the difference of the SUS scores was statistically significant ($F(1,31) = 21.698$, $p < .001$, $\eta_p^2 = .412$), thus the null hypothesis can be rejected. We concluded that users perceive Sound-Proof to be more usable than Google 2SV. Appendix A reports the items of the SUS.

Login Time. We measured the login time from the moment when a participant clicked on the "login" button (right after entering the password), to the moment when that participant was logged in. We neglected the time spent entering username and password because we wanted to focus only on the time required by the 2FA mechanism. Login time for Sound-Proof was 4.7 seconds (±0.2 seconds); this time was required for the phone to receive the computer's sample and compare it with the one recorded locally. With Google 2SV, login time increased to 24.4 seconds (±7.1 seconds); this time was required for the participant to take the phone, start the application and copy the verification code from the phone to the browser.

Failure Rates. We did not witness any login failure for either of the two methods. We speculate that this may be due to the priming of the users



(a) SUS answers for Sound-Proof



(b) SUS answers for Google 2SV

Figure 5.7: Distribution of the answers by the participants of the user study. We show the results for the System Usability Scale (SUS) of Sound-Proof (a) and Google 2-Step Verification (b). Percentages on the left side include participants that answered “Strongly disagree” or “Disagree”. Percentages in the middle account for participants that answered “Neither agree, nor disagree”. Percentages on the right side include participants that answered “Agree” or “Strongly agree”.

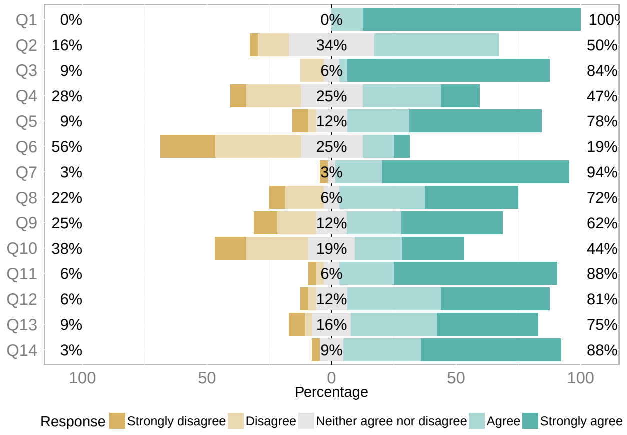


Figure 5.8: Distribution of the answers to the Post-test questionnaire. Percentages on the left side include participants that answered "Strongly disagree" or "Disagree". Percentages in the middle account for participants that answered "Neither agree, nor disagree". Percentages on the right side include participants that answered "Agree" or "Strongly agree".

right before the experiment, when we explained how the two methods work and that Sound-Proof may require users to make some noise in quiet environments.

Post-test Questionnaire. The post-test questionnaire was designed to collect information on the perceived quickness of the two mechanisms (Q1–Q2) and participants willingness to adopt any of the schemes (Q3–Q6). We also included items to inquire if participants would feel comfortable using the mechanisms in different environments (Q7–Q14). Figure 5.8 shows participants answers on 5-point Likert-scales. The full text of the items can be found in Appendix B.

All participants found Sound-Proof to be quick (Q1), while only 50% of the participants found Google 2SV to be quick (Q2). If 2FA were mandatory, 84% of the participants stated that they would use Sound-Proof (Q3) and 47% stated that they would use Google 2SV (Q4). In case 2FA were optional the percentage of participants willing to use the two mechanisms dropped to 78% for Sound-Proof (Q5) and to 19% for Google 2SV (Q6). Similar to [36, 165], our results for Google 2SV suggest that users are likely not to use 2FA if it is optional. With Sound-Proof, the difference in user acceptance between a mandatory and an optional scenario is only 6%.

We asked participants if they would feel comfortable using either mechanism at home, at their workplace, in a cafe, and in a library. 95% of the participants would feel comfortable using Sound-Proof at home (Q7) and 77% of the participants would use it at the workplace (Q8). 68% would use it in a cafe (Q9) and 50% would use it in a library (Q10). Most participants (between 91% and 82%) would feel comfortable using Google 2SV in any of the scenario we considered (Q11–Q14).

The results of the post-test questionnaire suggest that users may be willing to adopt Sound-Proof because it is quicker and causes less burden, compared to Google 2SV. In some public places, however, users may feel more comfortable using Google 2SV. In Section 5.7 we discuss how to integrate the two approaches.

The post-test questionnaire allowed participants to comment on the 2FA mechanisms evaluated. Most participants found Sound-Proof to be user-friendly and appreciated the lack of interaction with the phone. Appendix C lists some of the users' comments.

5.7 Discussion

Software and Hardware Requirements. Similar to any other 2FA solution based on software tokens, Sound-Proof requires an application on the user's phone. Sound-Proof, however, does not require additional software on the computer and seamlessly works with any HTML5-compliant browser that implements the WebRTC API. Google Chrome, Mozilla Firefox, Microsoft Edge and Apple Safari, already support WebRTC. Sound-Proof needs the phone to have a data connection. Moreover, both the phone and the computer where the browser is running must be equipped with a microphone. Microphones are ubiquitous in phones, tablets and laptops. If a computer such as a desktop machine does not have an embedded microphone, Sound-Proof requires an external microphone, like the one of a webcam.

Other Browsers. Section 5.5 evaluates Sound-Proof using Google Chrome. We have also tested Sound-Proof with Mozilla Firefox and Opera. Each browser may use different algorithms to process the recorded audio (e.g., filtering for noise reduction), before delivering it to the web application. The WebRTC specification does not yet define how the recorded audio should be processed, leaving the specifics of the implementation to the browser vendor. When we ran our tests, Opera behaved like Chrome. Firefox audio processing was slightly different and it affected the performance of our prototype. In particular, the Equal Error Rate computed over the samples collected while using Firefox was 0.012. We speculate that a better Equal Error Rate can be achieved with any browser if the software token performs the same audio processing of the browser being used to log in.

Privacy. The noise in the user's environment may leak private information to a prying server. As described in Section 5.3, in our design the audio recorded by the phone is never uploaded to the server. A malicious server can also access the computer's microphone while the user is visiting the server's webpage. This is already the case for a number of websites that require access to the microphone. For example, websites for language learning, Gmail (for video-chats or phone calls), live chat-support services, or any site that uses speech-recognition require access to the microphone and may record the ambient noise any time the user visits the provider's webpage. All browsers we tested ask the user for permission before allowing a website to use `getUserMedia`. Moreover, browsers show an alert when a website triggers recording from the microphone. Providers are likely not to abuse the recording capability, since their reputation would be affected, if users detect unsolicited recording.

Quiet Environments. Sound-Proof rejects a login attempt if the power of either sample is below τ_{dB}. In case the environment is too quiet, the website can prompt the user to make any noise (by, e.g., clearing his throat, knocking on the table, etc.). Moreover, Sound-Proof can be augmented with near-ultrasonic emission of a verification code, similar to SlickLogin [207]. Thus, if the environment is too silent for Sound-Proof to work, and assuming that the volume of the user's computer speakers is sufficient, then near-ultrasonic emission could be used to transfer the verification code from the computer to his phone, without user-phone interaction.

Fallback to other Types of 2FA. Sound-Proof can be combined with interactive 2FA mechanisms based on one-time codes, like Google 2SV [83], or login notification approvals received on the user's phone, like Duo Push [61]. For example, the webpage can employ Sound-Proof as the default 2FA mechanism, but give to the user the option to authenticate by approving the login on his phone, or entering a one-time code. This may be useful in cases where the environment is quiet and the user feels uncomfortable making any noise. Login based on one-time codes is also useful when the phone has no data connectivity (e.g., when roaming).

Failed Login Attempts and Throttling. Sound-Proof deems a login attempt as fraudulent if the similarity score between the two samples is below the threshold τ_C or if the power of either sample is below τ_{dB}. In this case, the server may request the two devices to repeat the recording and comparison phase. After a pre-defined number of failed trials, the server can fall-back to an interactive 2FA mechanism based on one-time codes or login approvals. The server can also throttle login attempts in order to prevent "brute-force" attacks and to protect the user's phone battery from draining.

Login Evidence. Since audio recording and comparison is transparent to the user, he has no means to detect an ongoing attack. To mitigate this, at each login attempt the phone may vibrate, light up, or display a message to notify the user that a login attempt is taking place. The Sound-Proof application may also keep a log of the login attempts. Such techniques can help to make the user aware of fraudulent login attempts. Nevertheless, we stress that the user does not have to attend to the phone during legitimate login attempts.

Continuous Authentication. Sound-Proof can also be used as a form of continuous authentication. The server can periodically trigger Sound-Proof, while the user is logged in and interacts with the website. If the recordings of the two devices do not match, the server can forcibly log the user out. Nevertheless, such use can have a more significant impact on the user's privacy, as well as affect the battery life of the user's phone.

Alternative Devices. Our 2FA mechanism uses the phone as a software token. Another option is to use a smartwatch and we plan to develop a Sound-Proof application for smartwatches based on Android Wear and Apple Watch. We speculate that smartwatches can further lower the false rejection rate because of the proximity of the computer and the smartwatch during logins.

Logins from the Phone. If a user tries to log in from the same device where the Sound-Proof application is running, the browser and the application will capture audio through the same microphone and, therefore, the login attempt will be accepted. This requires the mobile OS to allow access to the microphone by the browser and, at the same time, by the Sound-Proof application. If the mobile OS does not allow concurrent access to the microphone, Sound-Proof can fall back to 2FA based on login approvals or one-time codes.

Comparative Analysis. We use the framework of Bonneau et al. [28] to compare Sound-Proof with Google 2-Step Verification (Google 2SV), with PhoneAuth [51], and with the 2FA protocol of [191] that uses WiFi to create a channel between the phone and the computer (referred to as FBD-WF-WF in [191]). The framework of Bonneau et al. considers 25 "benefits" that an authentication scheme should provide, categorized in terms of usability, deployability, and security. Table 5.3 shows the overall comparison. The evaluation of Google 2SV in Table 5.3 matches the one reported by [28], besides the fact that we consider Google 2SV to be non-proprietary.

Usability: No scheme is scalable nor it is effortless for the user because they all require a password as the first authentication factor. They are all "Quasi-Nothing-to-Carry" because they leverage the user's phone. Sound-Proof and PhoneAuth are more efficient to use than Google 2SV because they do not require the user to interact with his phone. They are also more efficient to use than FBD-WF-WF, because the latter requires a non-negligible setup time every time the user logs in from a new computer. All mechanisms incur some errors if the user enters the wrong password (Infrequent-Errors). All mechanisms also require similar recovery procedures if the user loses his phone.

Deployability: Sound-Proof, PhoneAuth, and FBD-WF-WF score better than Google 2SV in the category "Accessible" because the user is asked nothing but his password. The three schemes are also better than Google 2SV in terms of cost per user, assuming users already have a phone. None of the mechanisms is server-compatible. Sound-Proof and Google 2SV are the only browser-compatible mechanisms as they require no changes to current browsers or computers. Google 2SV is more mature, and all of them are non-proprietary.

Security: The security provided by Sound-Proof, PhoneAuth, and FBD-WF-WF is similar to the one provided by Google 2SV. However, we rate Sound-Proof and PhoneAuth as not resilient to targeted impersonation, since a targeted, co-located attacker can launch the attack from the victim's environment. FBD-WF-WF uses a paired connection between the user's computer and phone, and can better resist such attacks.

5.8 Related Work

In chapter 4 we discussed alternative approaches to 2FA. In the following we review related work that leverages audio to verify the proximity of two devices.

Halevi et al., [89] use ambient audio to detect the proximity of two devices to thwart relay attacks in NFC payment systems. They compute the cross-correlation between the audio recorded by the two devices and employ machine-learning techniques to tell whether the two samples were recorded at the same location or not. The authors claim perfect results (0 false acceptance and false rejection rate). They, however, assume the two devices to have the same hardware (the experiment campaign used two Nokia N97 phones). Furthermore, their setup allows a maximum distance of 30 centimeters between the two devices. Our application scenario (web authentication) requires a solution that works (*i*) with heterogeneous devices, (*ii*) indoors and outdoors, and (*iii*) irrespective of the phone's position (e.g., in the user's pocket or purse). As such, we propose a different function to compute the similarity of the two samples, which we empirically found to be more robust, than what proposed in [89], in our settings.

Truong et al., [216] investigate relay attacks in zero-interaction authentication systems and use techniques similar to the ones of [89]. They propose a framework that detects co-location of two devices comparing features from multiple sensors, including GPS, Bluetooth, WiFi and audio. The authors conclude that an audio-only solution is not robust to detect co-location (20% of false rejections) and advocate for the combination of multiple sensors. Furthermore, their technique requires the two devices to sense the environment for 10 seconds. This time budget may not be available for web authentication.

The authors of [189] use ambient audio to derive a pair-wise cryptographic key between two co-located devices. They use an audio fingerprinting scheme similar to the one of [87] and leverage fuzzy commitment schemes to accommodate for the difference of the two recordings. Their scheme may, in principle, be used to verify proximity of two devices in a 2FA mechanism. However, the experiments of [189] reveal that the key derivation is hardly feasible in outdoor scenarios. Our scheme takes advantage of noisy environments and, therefore, can be used in outdoor scenarios like train stations.

5.9 Summary and Future Work

Although there is an ongoing effort to replace insecure password-based authentication, it is likely that it will remain the most common way of authenticating users on the web in the near future. Two-factor authentication is an effective mechanism that can complement password-based authentication to prevent attackers from accessing users' accounts and data. Nevertheless, deployed solutions have seen little adoption as users find it cumbersome to change their behavior when authenticating to a website.

We proposed Sound-Proof, a two-factor authentication mechanism for web logins that does not require the user to interact with his phone. In our solution the second authentication factor is the vicinity of the user's smartphone to the computer from which he is logging in. In particular two simultaneous recordings of the surrounding ambient audio are performed on the two devices and compared to test for their proximity. Sound-Proof is deployable today and works with major browsers. The user does not have to interact with his smartphone upon login and we have shown how our system works even if the phone is the user's pocket or purse as well as in a wide variety of environments. In comparison to Google 2-Step Verification, the participants in our user study found Sound-Proof to be more usable. More importantly, the majority said that they would use Sound-Proof for online services for which two-factor authentication is optional. We see the possibility to foster large-scale adoption of two-factor authentication for the web with a solution that is both usable and deployable today.

5.9.1 Future Work

With Sound-Proof we presented a two-factor authentication solution that is transparent to the user as he logs into a website. We now discuss interesting directions for future research.

Audio Comparison. Our audio comparison algorithm is based on cross-correlation. We perform some optimizations to make it more suitable to our needs such as filtering out lower and higher frequency bands. While this approach has shown to give good results in the experimental evaluation that we performed in multiple environments, we believe that there is room for improvement. Most audio comparison frameworks have seen research in order to perform a fast and accurate lookup of a short audio sample in a large samples dataset (e.g., to find a short and noisy recording of a song in a music catalog). Our use case is different and different comparison techniques can be further researched to improve the accuracy without hindering usability.

An approach that would be worthwhile researching is to apply machine learning techniques to our comparison algorithm. We envision two ways of performing this. First, machine learning could be used to better tweak the various parameters of the proposed comparison algorithm, such as the frequency band selection and the threshold. This fine tuning of the parameters would be based on the nature and the audio features of the audio samples under comparison, thus better adapting the comparison algorithm to each audio pair.

Second, machine learning could be used to compare the audio samples based on their extracted features, instead of comparing them directly using cross-correlation. This would also have the advantage that only the extracted features of the browser audio sample would have to be transferred, instead of the sample itself, thus minimizing the network overhead.

Regardless of the chosen machine learning approach, further research would be required into which features can be extracted and used from the audio samples while preserving the accuracy and security of Sound-Proof.

Security Guarantees. In this work we presented an empirical evaluation of our proposed solution. We collected a large number of samples in various environments and showed the robustness of Sound-Proof balancing security and usability. Future work can focus on understanding the physical properties of the audio samples that can be recorded with current platforms, evaluate them in terms of their entropy, and fully gauge the adversary's probability to successfully produce an audio sample that would match the user's one.

User Studies. We acknowledge the limitations of our small-scale user study, which focused on the usability aspects of Sound-Proof. A possible direction for future research is to perform further user studies in the field of two-factor authentication. In particular it would be interesting to understand how users actually interact with the second authentication factor and if, or how, they perceive the security benefits of two-factor authentication. Future user studies would enable researchers to propose solutions that better suit the needs of end users and possibly enable faster and more widespread adoption.

	Usability								Deployability						Security										
Scheme	Memorywise-Effortless	Scalable-for-Users	Nothing-to-Carry	Physically Effortless	Easy-to-Learn	Efficient-to-Use	Infrequent-Errors	Easy-Recovery-from-Loss	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-Theft	No-Trusted-Third-Party	Requiring-Explicit-Consent	Unlinkable
Sound-Proof			S		Y	Y	S	S	Y	Y		Y		Y	S		Y	Y		S	Y	Y	Y	Y	
Google 2SV			S		Y	S	S	S	S	S		Y	Y	Y	S	S	Y	Y		Y	Y	Y	Y	Y	
PhoneAuth			S		Y	Y	S	S	Y	Y				Y	S		Y	Y		Y	Y	Y	Y	Y	
FBD-WF-WF			S		Y	S	S	S	Y	Y				Y	S	S	Y	Y		Y	Y	Y	Y	Y	

Table 5.3: Comparison of Sound-Proof against Google 2-Step Verification (Google 2SV), PhoneAuth [51], and FBD-WF-WF [191], using the framework of Bonneau et al. [28]. We use 'Y' to denote that the benefit is provided and 'S' to denote that the benefit is somewhat provided.

Part II

Web Server Authentication

Chapter 6
Introduction

In Part I of the thesis we looked at client authentication on the web, and in particular on how two-factor authentication aims at improving the poor security of password-based authentication. With Sound-Proof, we proposed the first two-factor authentication solution that is fully transparent to the user and is readily deployable with today's existing web technologies, with the intent of fostering adoption and thereby providing the security benefits of two-factor authentication at scale.

In this part of the thesis we shift focus from client to server authentication, yet still taking client authentication into account and investigating how these two properties interact with each other. As we described in Chapter 2, HTTP does not provide any server authentication per se. For HTTPS servers (i.e., HTTP over TLS) server authentication takes place at the TLS level, using server certificates. A server certificate binds the public key of a server to its domain (or domains) and it is issued by a Certificate Authority (CA) that is trusted by web browsers and other web clients.

The CA trust model, which server authentication relies upon, has been proven to be insufficient. This is due to a number of incidents involving the compromise of CAs (e.g., [2, 32, 46, 63, 188, 198]) that took place in recent years and in which unauthorized entities were able get access to valid certificates for domains that they did not own. An attacker that has access to a mis-issued certificate for a domain he does not legally own can mount a *TLS Man-In-The-Middle* (MITM) by suitably positioning himself on the network, intercepting the client-server communication and presenting the rogue, yet valid certificate to the user's browser during the TLS handshake phase. The browser, given that the certificate was issued by a trusted CA will readily accept it as valid, thereby allowing the attacker to impersonate the server to the user, who thinks he is connected to the legitimate server. Obviously, the TLS connection security warnings and indicators of the browser will not be of help in this case, since from the browse's perspective, this seems to be a legitimate connection.

Once the attacker has completed the first step of the attack, i.e., impersonate the server to the user, he then typically opens a TLS connection to the legitimate server and impersonates the user to the server. He achieves this by intercepting and stealing the user's credentials, such as username and password, and authentication cookies. Looking back at Part I of the thesis, most client authentication solutions, including Sound-Proof, are susceptible to this powerful attack, since they do not prevent the attacker from stealing the user's credentials off the network and using them in order to authenticate as the user to the server. This allows the attacker to get full access to the user's account and use it for harmful purposes, such as spying, money theft and so forth.

From the above, it is clear that the root cause of TLS MITM attacks is the fact that server authentication can be compromised, so it is reasonable to investigate and solve this issue. Indeed, the aforementioned weakness of the CA trust model, has ignited the interest of both the industry and academic research community in trying to find ways to enhance the robustness of the current trust model or even designing entirely new solutions. In Chapter 7 we review a range of techniques and proposals in this area.

Nevertheless, the careful readers may have noticed that another way to potentially prevent TLS MITM attacks would be to employ some form of client authentication that can resist theft. In this way, the attacker would not be able to steal the user's credentials and use them on a direct connection to the server in order to authenticate as the user and gain access to his account at the server.

As a matter of fact, techniques offering this level of strong client authentication do exist and are mainly based on public key cryptography. TLS client authentication, which we reviewed in Chapter 2 is such an example. The more recently proposed and significantly more usable compared to TLS client authentication) TLS Channel ID-based authentication [13, 51, 71] is another example of such strong client authentication.

After reviewing proposals that aim at preventing TLS MITM attacks by looking at the root cause of the problem, i.e., strengthen server authentication (Chapter 7), in Chapter 8 we consider proposals (such as those mentioned above) that attempt to resist TLS MITM attackers whose goal is user impersonation and account takeover, by using strong, theft resistant client authentication. We show that, in the context of web applications, such proposals and client authentication in general, no matter how strong it is, cannot alone resist such attackers. We devise an attack, called *Man-In-The-Middle-Script-In-The-Browser* (MITM-SITB), which leverages the web server's ability to ship and execute JavaScript in the user's browser, and which allows an attacker to completely bypass the security of aforementioned proposals. We then show how such strong client authentication techniques can be combined with the concept of *server invariance*, a weaker property than server authentication, in order to prevent such attacks. We present our concept, called *Server Invariance with Strong Client Authentica-*

tion (SISCA), and describe in detail how it can be implemented in practice and integrated into today's web infrastructure.

This part of our work highlights how web applications are unique and different from any other client-server protocol. Namely, while strong client authentication techniques are effective in preventing MITM attacks in other client-server protocols (e.g., SSH), this is not the case in web applications, with their unique ability of shipping and executing server-originating code to the client side.

Chapter 7
Related Work

In this Chapter we review proposals that try to enhance the CA trust model. By doing so, these solutions strive to enforce proper server authentication and thus defeat TLS MITM attacks at its root cause. In Chapter 8, after we present our work, we discuss more closely related work (Section 8.5).

7.1 CA Trust Model Enhancement

TLS MITM attacks are feasible mainly due to the fact that web browsers blindly trust hundreds of CAs to sign certificates for *any* domain. A way to improve the security of the CA trust model is therefore to reduce the level of trust placed in the CAs. In recent years various proposals have emerged that follow this idea and perform *enhanced certificate verification*. These proposals are mostly based on two techniques: pinning and multipath probing. We mention some of the existing proposals below. We refer the interested reader to [45] for a comprehensive survey.

7.1.1 Pinning

Pinning enables a web server to instruct browsers to accept only a specific set of certificates or public keys when establishing TLS connections to that server. It is considered more practical and flexible to pin keys rather than certificates. The pins can refer to leaf keys, i.e., the keys of the web server, or CA keys, which are found in the certificate chain. Pinning a CA key for a domain means that the browser will accept any certificate for this domain that was issued using the particular CA key. Pinning can be implemented in a variety of ways, which are described below.

Server-based Pins. The web server itself can instruct the browser which certificates or keys to pin and for how long. Example solutions include the Public Key Pinning Extension for HTTP (HPKP) [68], which as its name suggests operates on the HTTP protocol level, and Trust Assertions for Certificate Keys (TACK) [129], which operates at the TLS protocol level. Specifically in HPKP, the web server uses appropriate HTTP headers which instruct browsers to pin specific public keys. When establishing a TLS connection to that server, the browser accepts the connection, only if at least one of the pinned keys appears in the presented certificate chain during the TLS handshake. Similarly, TACK is a TLS extension which allows browsers to pin to a server-chosen signing key. The browser, subsequently, requires server certificates to be additionally signed by the server signing key.

Preloaded Pins. Server-based solutions such the above accommodate for dynamic updates of the pins. A more static approach can be followed, as well, with pins being pre-loaded within the browser or any other app that communicates with an HTTPS web server. Note that the pins can still be updated, via browser or app updates.

Other Approaches. DNS-Based Authentication of Named Entities (DANE) [94] allows domain owners to associate the public keys of their servers with the corresponding domain names using DNS Security Extensions (DNSSEC). These associations are placed in the DNSSEC-protected DNS records of each domain. A browser accepts a connection to a particular domain, only if the presented server certificate has been associated with the domain name. Sovereign Keys (SKs) [64] are server-chosen signing keys, which are used for cross-signing server certificates, a concept similar to TACK. However, unlike TACK, SKs are registered in public, cryptographically assured, append-only logs, called Timeline Servers (a concept similar to Certificate Transparency). The difference between these approaches and server-based ones, is that, the pins are not transmitted in-bound through the communication with web servers. Instead, they are delivered via out-of-band channels, namely DNS or public logs.

While pinning techniques can be effective at prevent TLS MITM attacks, they typically require significant maintenance effort, and pinning errors can result in rendering a website inaccessible. For this reason, Google Chrome, which currently supports both preloaded pins, as well as HPKP, is planning to deprecate pinning completely in a future release [159].

7.1.2 Multipath Probing

Multipath probing increases assurance about the legitimacy of the certificate by consulting (several) external sources. Like in the case of pinning, multipath probing can be realized in a variety of different ways.

Notaries. In Perspectives [234] and Convergence [128], the browser queries a set of trusted notaries for their view of the network (i.e, the server certificate that the notaries witness when attempting to connect to a particular server) and compares it with its own. Similarly, DoubleCheck [5] leverages nodes from the Tor network as notaries. The basic idea in these schemes is that a MITM attacker will only be able to mount his attack on just a part of the global network. Thus, notaries outside this part will see a different server certificate (the legitimate one), than what is presented to the target browser by the attacker, and the attack will be detected.

Public Logs. A promising idea of implementing multipath probing is to increase CA accountability through the use of publicly verifiable logs, which essentially act as trusted third parities. Certificate Transparency [118] implements this idea, by requiring every issued certificate to be published in cryptographically assured, append-only logs. This way, CAs and domain owners (acting as monitors) can monitor the logs for mis-issued certificates. At the same time, browsers (acting as auditors) can verify that a particular certificate has been added, or check for log misbehavior. Auditors and monitors communicate in a distributed fashion, in order compare their views of the logs and thus detect illegal log inconsistencies. Google Chrome plans to make Certificate Transparency a mandatory requirement for all publicly trusted web server certificates in 2018 [197].

Accountable Key Infrastructure (AKI) [111] extends the Certificate Transparency architecture by adding desired properties, which Certificate Transparency lacks. For example, it allows multiple CAs to sign a server certificate and allows the domain to specify in its certificate which CAs and logs are allowed to attest to the certificate's authenticity. Moreover it includes a key revocation architecture, thereby accommodating key loss and compromise. Attack Resilient Public-Key Infrastructure (ARPKI) [18, 19] is a redesign of AKI. It improves various aspects of AKI and offers stronger security guarantees which are formally verified. PoliCert [205] extends AKI and ARPKI by enabling domains to define policies which govern the usage of their own certificates.

7.1.3 Other Approaches

In recent years several other approaches, complimentary to pinning and multipath probing techniques, have been proposed. Here, we mention a few examples.

Physical Location of Server. SALVE (Server Authentication with Location VErification) [239] uses the server's geographic location as an additional factor of authenticity. SALVE is deployed as a TLS extenstion and is built on top of DNS (or DNSSEC) and the Location Service architecture (LCS) [1], which is a platform in the telecommunication infrastructure that disseminates location information of mobile devices. SALVE enables the browser to accept a TLS connection only if the server is present at a legitimate location, e.g., on the premises of a data center, thereby preventing remote adversaries with mis-issued certificates to impersonate the legitimate server.

Improved Certificate Revocation. Whenever a key is compromised any certificates issued for that key have to be revoked. A fast, efficient revocation process can help mitigate the impact of a key compromise.

Revocation is traditionally implemented by the issuing CA via certificate revocation lists (CRLs) [48], or via the online certificate status checking protocol (OCSP) [186]. Nevertheless, the drawbacks of existing techniques have led to the emergence of several proposals, which try to improve the revocation process. For example, short-lived certificates [177, 214] is a revocation concept, which requires certificates to have a very short lifetime, e.g., one day. In this case, the web server should update its certificates from the CA on a daily basis. Revocation Transparency [117] and PKI Safety Net (PKISN) [204], as well as AKI [111], ARPKI [18] and PoliCert [205] presented above, use publicly accessible logs, in the spirit of Certificate Transparency, to store not only certificates, but also revocations. Finally, RITM [203] leverages network middleboxes and content delivery networks (CDNs) to disseminate certificate revocation information.

7.2 User Impersonation Prevention

Some solutions assume that the attacker can successfully impersonate the server to the user and try to prevent TLS MITM attacks by ensuring that the attacker cannot impersonate the user to the server and compromise his online account. For example, this can be achieved by employing strong, theft-resilient client authentication, such as channel-bound credentials [51, 56]. As another example, some approaches prevent user impersonation by leveraging the characteristics of network round-trip latency as an additional authentication factor, which is hard to be forged by the attacker [109, 217].

In the rest of this Part, we focus on these proposals and show how they are not able to fully prevent such attacks by themselves. We therefore, refer the reader to the next Chapter for more details.

Chapter 8
SISCA: Server Invariance with Strong Client Authentication

Web applications increasingly employ the TLS protocol to secure HTTP communication (i.e., HTTP over TLS, or HTTPS) between a user's browser and the web server. TLS enables users to securely access and interact with their online accounts, and protects, among other things, common user authentication credentials, such as passwords and cookies. Such credentials are considered *weak*, as they are transmitted over the network and are susceptible to theft and abuse, unless protected by TLS.

Nevertheless, during TLS connection establishment, it is essential that the server's authenticity is verified. If an attacker successfully impersonates the server to the user, he is then able to steal the user's credentials and subsequently use them to impersonate the user to the legitimate server. This way, the attacker gains access to the user's account and data which can be abused for a variety of purposes, such as spying on the user [63, 188]. This attack is known as TLS Man-In-The-Middle (MITM).

TLS server authentication is commonly achieved through the use of X.509 server certificates. A server certificate binds a public key to the identity of a server, designating that this server holds the corresponding private key. The browser accepts a certificate if it bears the signature of any trusted Certificate Authority (CA). Browsers are typically configured to trust hundreds of CAs.

An attacker can thus successfully impersonate a legitimate server to the browser by presenting a valid certificate for that server, as long as he holds the corresponding private key. In previous years, quite a few incidents involving mis-issued certificates [2, 32, 46, 188, 198] were made public. Even in the case where the attacker simply presents an invalid (e.g., self-signed) certificate not accepted by the browser, he will still succeed in his attack if the user defies the browser's security warning.

In order to thwart such attacks, various proposals have emerged. Some proposals focus on enhancing the certificate authentication model. Their objective is to prevent an attacker possessing a mis-issued, yet valid certificate, from impersonating the server (e.g., [68, 111, 118, 234]).

Other proposals focus on strengthening client authentication. *Strong* client authentication prevents user credential theft or renders it useless, even if the attacker can successfully impersonate the server to the user.

One such prominent proposal is Channel ID-based client authentication, introduced in 2012. TLS Channel IDs [13] are experimentally supported in Google Chrome and are planned to be used in the second factor authentication standard U2F, proposed by the FIDO alliance [71].

In this work we show that Channel ID-based approaches, as well as web authentication solutions that focus solely on client authentication are vulnerable to an attack that we call *Man-In-The-Middle-Script-In-The-Browser (MITM-SITB)*, and is similar to *dynamic pharming* [107] (see Section 8.5). This attack bypasses Channel ID-based defenses by shipping malicious JavaScript to the user's browser within a TLS connection with the attacker, and using this JavaScript in direct connections with the legitimate server to attack the user's account.

Nevertheless, we show that TLS MITM attacks where the attacker's goal is user impersonation can still be prevented by strong client authentication, such as Channel ID-based authentication, provided that it is combined with the concept of *server invariance*, that is, the requirement that the client keeps communicating with the same entity (either the legitimate server, or the attacker) across multiple connections intended for the same server. Server invariance is a weaker requirement than server authentication, and thus, it is easier to achieve as no initial trust is necessary. Building on this observation, we propose a solution called *SISCA: Server Invariance with Strong Client Authentication*, that combines Channel ID-based client authentication and server invariance.

SISCA can resist TLS MITM attacks that are based on mis-issued valid certificates, as well as invalid certificates, requiring no user involvement in the detection of the attack (i.e., no by-passable security warnings when server invariance violation occurs). SISCA also thwarts attackers that hold the private key of the legitimate server.

Contributions. We analyze TLS MITM attacks whose goal is user impersonation and make the following contributions.

- We show, by launching a MITM-SITB attack, that Channel ID-based client authentication solutions do not fully prevent TLS MITM attacks.

- We further argue that effective prevention of MITM-based user impersonation attacks requires strong user authentication and (at least) server invariance.

- We propose a novel solution that prevents MITM-based user impersonation, based on the combination of strong client authentication and server invariance, which we call SISCA.
- We implement and evaluate a basic prototype of our solution.

The rest of this chapter is organized as follows. In Section 8.1 we present our system and threat model. Section 8.2 discusses TLS Channel ID-based authentication and shows how MITM attacks are possible using MITM-SITB. Section 8.3 mentions known solutions for addressing MITM attacks and presents SISCA. Section 8.5 discusses related work and Section 8.6 concludes Part II and discusses interesting research directions.

8.1 Model

8.1.1 System Model

We consider a typical web application setting. The user, or victim, has an account with a particular web server that the adversary is interested in. This could be a webmail application, a social networking site, an e-banking platform and so forth. We assume that the web server can be accessed exclusively over TLS. The victim uses a modern web browser in order to access the web server and login to his account.

8.1.2 Attacker Goal and Model

Attacker Goal. The attacker's goal in a MITM attack is typically to impersonate the user (victim) to the legitimate web server, in order to compromise the user's online account and data. This is indeed the case where the attacker wishes for example to spy on the user [63, 188], or abuse his account for nefarious purposes, e.g., perform fraudulent financial transactions. Alternatively, the attacker could aim to only impersonate the server to the user (and not the user to the server), such that he serves the user with fake content (e.g., fake news). In this part of the thesis, we take into account only the first scenario.

Attacker Model. We adopt the attacker model considered by Channel IDs [13]. The adversary is able to position himself suitably on the network and perform a TLS MITM attack between the user and the target web server. In other words, the attacker is able to successfully impersonate the server to the user. We distinguish between two types of MITM[1] attackers.

[1]We use the terms "TLS MITM" and "MITM" interchangeably.

The *MITM+certificate* attacker holds *(i)* a *valid* certificate for the domain of the target web server, binding the identity of the server to the public key, of which he holds the corresponding private key. The attacker, however, has no access to the private key of the target web server. This, for example, can happen if the attacker compromises a CA or is able to force a CA issue such a certificate. Such attacks have been reported in the recent years [2, 32, 46, 188]. Moreover, in this category we also consider a weaker attacker that only holds *(ii)* an *invalid* (e.g., self-signed) certificate. In this case, the attacker will still succeed in impersonating the server to the user if the latter ignores the security warnings of the browser, which is a common phenomenon [202].

The *MITM+key* attacker holds the *private key of the legitimate server*. While we are not aware of publicized incidents involving server key compromise, such attacks are feasible, as the Heartbleed vulnerability in OpenSSL has shown [211], and can be very stealthy, remaining unnoticed. Thus, they are well worth addressing [93, 95, 116].

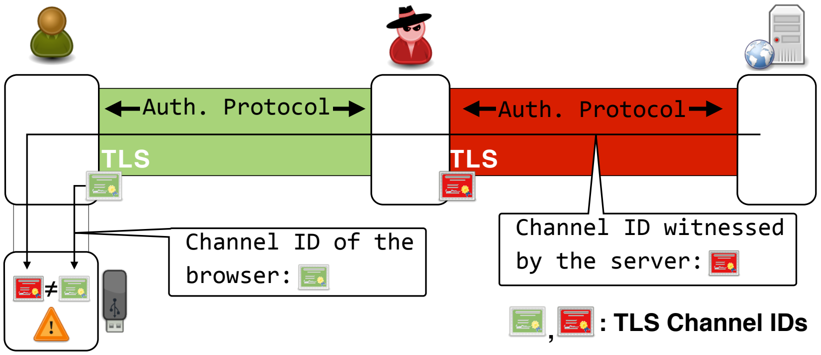
From the above it follows that the attacker is able to obtain the user's weak credentials, namely passwords and HTTP cookies. He is not, however, able to compromise and take control over the user's browser or his devices (e.g., mobile phones).

8.2 Channel ID-based Authentication and MITM Attacks

8.2.1 TLS Channel IDs

Channel IDs is a recent proposal for strengthening client authentication. It is a TLS extension, originally proposed in [56] as *Origin-Bound Certificates* (OBCs). A refined version has been submitted as an IETF Internet-Draft [13]. Currently, Channel IDs are experimentally supported by Google's Chrome browser and Google servers.

In brief, when the browser visits a TLS-enabled web server for the first time, it creates a new private/public key pair (on-the-fly and without any user interaction) and proves possession of the private key, during the TLS handshake. This TLS connection is subsequently identified by the corresponding public key, which is called the Channel ID. Upon subsequent TLS connections to the same web server, or more precisely, to the same web origin, the browser uses the same Channel ID. This enables the web server to identify the same browser across multiple TLS connections. We stress that, Channel IDs are not envisioned to be directly used by the web server



Auth. Protocol
TLS
Auth. Protocol
TLS
Channel ID of the browser:
Channel ID witnessed by the server:
≠
: TLS Channel IDs

Figure 8.1: PhoneAuth and FIDO U2F. Leveraging Channel IDs to secure the initial login against MITM attacks.

to authenticate the user or the browser. They are instead used by the web server to identify the same browser across multiple TLS connections, as the browser will be using the same Channel ID for these connections.

Channel ID-Based Authentication

By *Channel ID-based authentication* we refer to the use of Channel IDs *throughout* the user authentication process, designed to thwart both types of MITM attackers presented in Section 8.1 [13, §6], [51, §3].

Initial Login. When the user attempts to login to his online account for the first time from a particular browser, the web server requires that the user authenticates using a strong second factor authentication device, as in *PhoneAuth* [51] and *FIDO Universal 2nd Factor (U2F)* [71] protocols. These protocols leverage Channel IDs to secure the intial login process against MITM attacks. In brief, as part of the authentication protocol, the second factor device compares the Channel ID of the browser to the Channel ID of the TLS connection that the server witnesses. If they are equal, then the browser is directly connected to the web server through TLS (because they share the same view of the connection), and thus there is no MITM attack taking place. On the other hand, if the Channel IDs differ, then the server is not directly connected to the user's browser. Instead, as shown in Figure 8.1, there is an attacker in the middle, and the device aborts the authentication protocol, thereby stopping the attack.

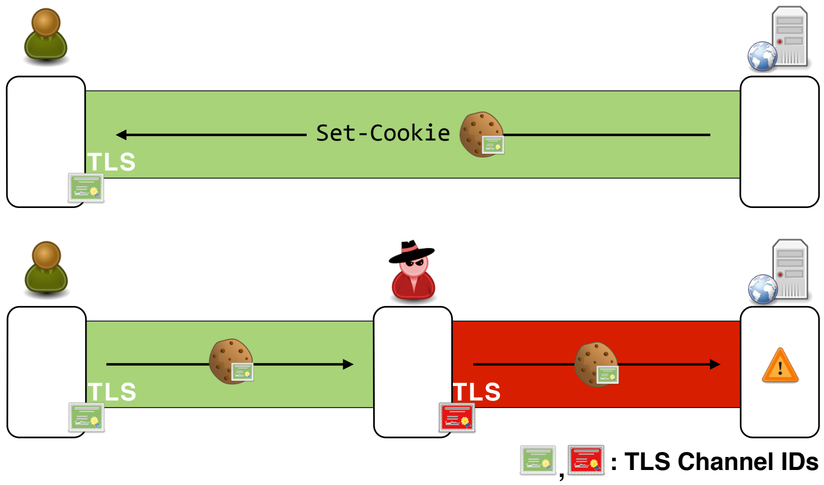


Figure 8.2: Binding authentication tokens (e.g., cookies) to the browser's Channel ID (green). A MITM attacker who steals such a cookie, cannot use it to impersonate the user, since the attacker has a different Channel ID (red).

Subsequent Logins. Upon successful initial authentication the server sets a cookie to the user's browser, and binds it to the Channel ID of the browser. As proposed in [56], a server may create a *channel-bound cookie* as follows: $\langle v,\ \mathrm{HMAC}(k, v|cid)\rangle$, where v is the original cookie value, cid is the browser Channel ID and k is a secret key only known to the server, used for computing a MAC over the concatenation of v and cid. The channel-bound cookie is considered valid only if it is presented over that particular Channel ID. Therefore, subsequent interaction with the server from that particular browser is protected by the channel-bound cookie. An attacker that manages to steal a channel-bound cookie, e.g., through a MITM attack, cannot use it to impersonate the user to the web server, since he does not know the private key of the correct Channel ID. Figure 8.2 illustrates this concept. Note that at this stage, the second factor device is not required for authenticating the user [50]. Whenever the channel-bound cookie is absent (e.g., it expired, the user deleted it, or the user tries to login from a new browser) or it is present but invalid (i.e., presented over an incorrect Channel ID), the server once again requires user authentication using the second factor device (as described above for the initial login process).

8.2.2 MITM Attack on Channel ID-Based Authentication

We show how Channel ID-based authentication still allows a MITM attacker to successfully impersonate the user. This is due to the way web applications are run and interact with the servers, which differs from other Internet client-server protocols (e.g., IMAP over TLS).

In particular, web servers are allowed to send scripting code to the browser, which the latter executes within the security context of the web application (according to the rules defined by the *same-origin policy* [17] – we refer the reader to Section 2.3 for more information). In fact, and as discussed in the introduction of this thesis, client-side scripting and especially JavaScript, is the foundation of dynamic, rich web applications that vastly improve user experience, and its presence is ubiquitous.

Moreover, a browser can establish multiple TLS connections with the same server. In addition, a typical web application loads resources, such as images and scripts, from multiple domains (*cross-origin* network access [17]). Assuming that all communication is TLS-protected, this means that the browser needs to establish TLS connections with multiple servers while loading a webpage.

Given the above, there is a conceptually simple attack that a MITM adversary can perform, which bypasses the security offered by Channel IDs (the attack can be realized by either MITM+certificate or MITM+key attackers). We assume that the user tries to access the target web server, say `www.example.com`. The attacker then proceeds as follows:

1. He intercepts a single TLS connection attempt made by the browser to `www.example.com`, and by presenting a valid certificate (or invalid with the user ignoring the browser's warning), he successfully impersonates the legitimate server to the browser.

2. Through the established connection, the browser makes an HTTP request to the server. The attacker replies with an HTTP response, which includes a malicious piece of JavaScript code. This script will execute within the origin of `www.example.com`.

3. The attacker closes the intercepted TLS connection. This forces the browser to initiate a new TLS connection in order to transmit subsequent requests, or use another existing one, if any (this behavior conforms with the HTTP specification [73]). At the same time, the attacker allows subsequent TLS connection attempts to pass through, without interfering with them. As a result, once the attacker closes that single intercepted connection, all other connections, existing and

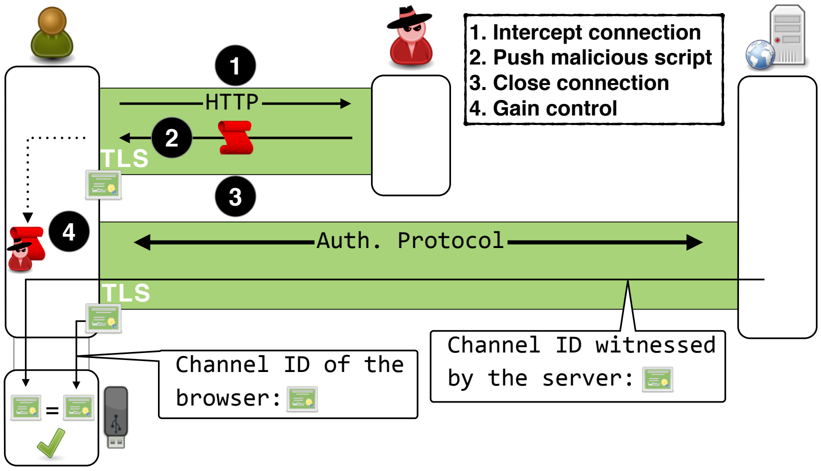


Figure 8.3: MITM-SITB attack on Channel ID-based PhoneAuth/U2F used for the initial login. The attacker's JavaScript code is executed within the origin of the target server (shown by the dotted arrow).

new, are directly established between the browser and the legitimate server.

4. The attacker gains full control over the user's session in that particular web application. His script has unrestricted access over the web documents belonging to `www.example.com` and can monitor all the client-side activity of the web application. Moreover, he can issue arbitrary malicious requests to the target server using the `XMLHttpRequest` object [228], in order to perform a desired action or extract sensitive user information. The malicious code can upload any extracted data to an attacker-controlled server. As another example, if the web application is AJAX-based, the attacker can perform *Prototype Hijacking* [161]. This allows him to eavesdrop and modify on-the-fly all the HTTP requests made through `XMLHttpRequest`.

In summary, the MITM attacker "transfers" himself (via the malicious script) within the user's browser, and continues his attack from there. We call this attack *Man-In-The-Middle-Script-In-The-Browser* (*MITM-SITB*).

Figure 8.3 illustrates the MITM-SITB attack in the case when the user is about to initially authenticate to `www.example.com` using PhoneAuth or U2F. The attacker intercepts a TLS connection, pushes his JavaScript code

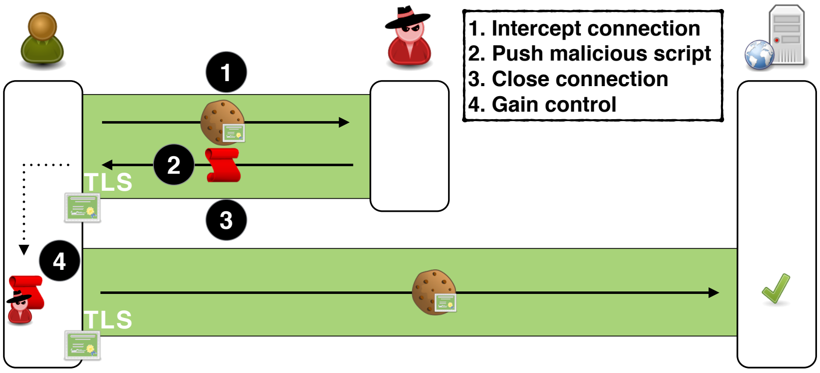


Figure 8.4: MITM-SITB attack on Channel ID-based authentication after the initial login, where requests are protected with a channel-bound cookie.

to the user's browser, and terminates the connection. The browser then establishes a new TLS connection for subsequent communication, only this time with the legitimate server; the attacker will not hijack it. This ensures that the user authentication is performed over a direct connection between the browser and the server, but with the attacker's code running in the browser. The view of the TLS channel will be the same for the browser and the server, and the Channel ID comparison made by the second factor device will pass.

Figure 8.4 shows how the attack works in the case when the user has already logged in on `www.example.com` in the past, and the server has set a channel-bound cookie in the user's browser. Like before, the attacker ships malicious JavaScript code to the browser by intercepting a TLS connection to `www.example.com`. He then terminates the intercepted connection. This forces the browser to establish a new TLS connection, which is not intercepted by the attacker. This ensures that any subsequent requests, either legitimate or malicious (issued by the attacker's script) are accepted by the legitimate server, since they will carry the channel-bound cookie, which authenticates the user, over the correct Channel ID.

From the above attack description there are various details that remain unclear. For example, which TLS connection the attacker should intercept, whether to "hit and run" or persist as much as possible, etc. Depending on the scenario, there are various alternatives, which are mostly implementation decisions. The attacker can for example choose the following strategy. He intercepts the *very first* TLS connection, i.e., the one that the browser initiates once it is directed to `www.example.com`. Depending on the situation, the attacker's HTTP response could contain the expected HTML document of the website's starting page, together with the appropriately injected malicious script, or it could only contain the malicious script, which will take care of loading the starting page in the browser. Then, as described before, the attacker closes this first connection and subsequent communication (malicious or not) takes place through a direct connection to the legitimate server.

The Cross-Origin Communication Case. Visiting a single webpage typically involves cross-origin communication with different domains in the background. For example a typical network optimization technique is to have the browser load the static resources of the website, such as images, style sheets and scripts, from so-called *cookieless domains* [99]. These domains, as their name suggests, do not set any cookies, so as to minimize network latency. As a matter of fact, on such domains, client authentication does not apply at all, as they are just used to serve static resources, which anyone, including the attacker, can access. Hence in those cases, the attacker can perform a conventional MITM attack against a cookieless domain, and inject his malicious code at the moment when the target web server requests a legitimate JavaScript file from that domain.

Figure 8.5 illustrates the attack. The attacker lets all communication to `www.example.com` (the main web server) pass through. Initially, the browser connects to `www.example.com` to load some page. The returned HTML document imports a JavaScript file from the cookieless domain `static.example.com`. Assuming that the script is not cached, the browser initiates a TLS connection to that domain, which is intercepted by the attacker. The attacker fetches the original script, injects his code and forwards the script to the browser, which is executed within the origin of `www.example.com`.

8.2.3 Proof of Concept Attack

We validate our attack against Channel IDs through a proof of concept implementation. We use two Apache TLS-enabled servers (one for the attacker, one for the legitimate server) and an interception proxy that can selectively forward TLS connections to either server. The legitimate server uses a patched OpenSSL version that supports Channel IDs and leverages them for creating channel-bound cookies. We use Google Chrome as the user's browser, since it supports Channel IDs, and ensure that it accepts the certificates of both servers. We are then able to inject JavaScript code to

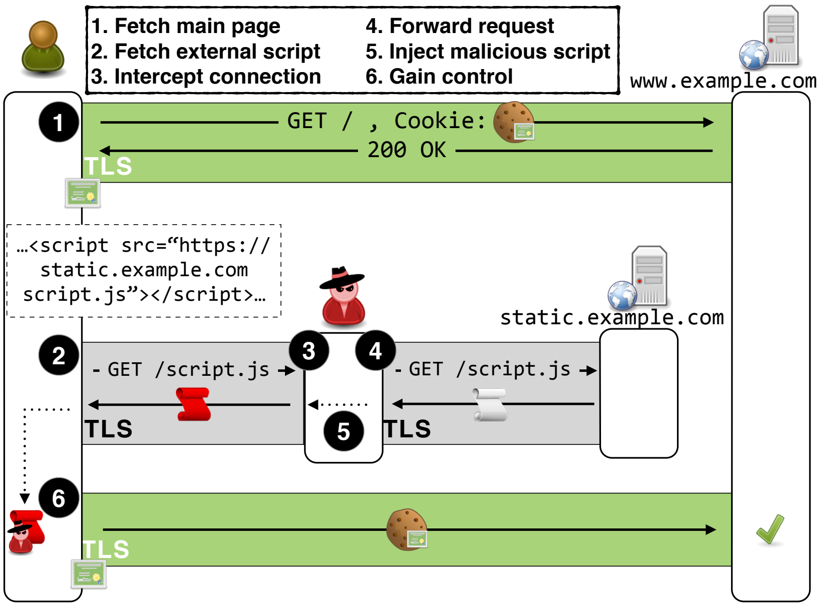


Figure 8.5: MITM-SITB attack on Channel ID-based authentication leveraging cross-origin communication. Channel IDs for `static.example.com` are of no use.

the user's browser from the attacker's server and issue HTTP requests that are accepted and processed by the legitimate server.

8.2.4 Scope and Implications of the Attack

The MITM-SITB attack presented in Section 8.2.2 is not specific to Channel ID-based client authentication protocols. In fact, it applies to *any* web client authentication method. This attack demonstrates that, in the context of web applications, it does not seem possible to prevent TLS MITM attacks via client authentication alone.

We provide the following informal reasoning for the above claim. Client authentication does not prevent an attacker from impersonating the legitimate server. This allows him to intercept a server-authenticated (i.e., TLS) connection and ship his JavaScript code to the user's browser. The browser, treating the attacker's code as trusted (as it came through a server-authenticated connection), executes it within the target server's origin. The attacker accesses the user's account through requests initiated by his code and transmitted over another, direct connection between the browser and the legitimate server.

As a result, schemes such as traditional TLS client authentication [55], TLS Session Aware User Authentication [153, 154], Browser-based Mutual Authentication [76], as well as solutions based on the characteristics of network latency [109, 217] are still susceptible to TLS MITM attacks, via MITM-SITB. The attacker succeeds in impersonating the user to the web server and compromising his account.

8.3 Addressing TLS MITM Attacks

As shown in Section 8.2, strong client authentication alone is not sufficient to prevent MITM attacks that lead to user impersonation in web applications. So, how can we effectively prevent such attacks? In this section we show that there are two *orthogonal* solutions; *(i)* the known solution of preventing the attacker from impersonating the legitimate server at all, i.e., ensuring correct server authentication; *(ii)* our novel approach of combining strong client authentication with server invariance, called SISCA.

8.3.1 Prevent Server Impersonation

The known and straightforward solution to the problem at hand is to prevent the attacker from impersonating the server in the first place. This way, the attacker can neither steal weak user credentials in order to mount a conventional MITM attack, nor ship malicious JavaScript in order to mount a MITM-SITB attack. Note that in this case, strong client authentication (e.g., Channel ID-based) is not necessary for preventing MITM attacks (it is, however, still useful for preventing other attacks, such as phishing and server password database compromise).

The solutions that try to prevent server impersonation address the issue of forged server certificates (and thus defeating MITM+certificate attacks), by performing *enhanced certificate verification*. Such solutions are based on *pinning* (e.g., [64, 68, 94, 129]), *multipath probing* (e.g., [18, 111, 118, 128, 234]) and other approaches (e.g., [109, 239]. We refer the reader to Chapter 7 where we review these solutions, as well as to [45] for a thorough survey.

8.4 Our Proposal: SISCA

8.4.1 Main Concept

The fact that strong client authentication alone cannot effectively prevent MITM attacks in web applications, raises the following question. Is there a way to somehow still benefit from strong client authentication with respect to addressing MITM attacks?

To answer, we make the following observation. In the context of web applications, a MITM attacker can perform user impersonation via two approaches:

1. The *conventional MITM* attack, in which the attacker compromises the user's credentials and uses them for impersonation. This attack can be effectively prevented by strong client authentication e.g., using Channel ID-based protocols (Figures 8.1, 8.2).

2. The *MITM-SITB* attack, presented in Section 8.2.2 (Figures 8.3, 8.4, 8.5). As discussed in Section 8.2.4, client authentication alone cannot prevent this attack.

For the MITM-SITB attack to be successful, the user's browser needs to communicate with two different entities, namely the attacker and the target web server. Communicating with the attacker is, of course, necessary for injecting the attacker's script to the browser through the intercepted TLS connection. In addition, communication with the target server is essential, so that the attacker accesses the user's account and data, through his script.

As a result, we can prevent MITM-SITB by making sure that the browser communicates only with *one entity*, either the legitimate server, or the attacker, but *not* with both, during a *browsing session* (a browsing session is terminated when the user closes the browser). In other words, we need to enforce server invariance. When combined with strong client authentication (e.g., Channel ID-based), which stops the conventional MITM approach, this technique manages to effectively thwart MITM attacks. Figure 8.6 illustrates the concept.

In the remaining section we present a novel solution, called *Server Invariance with Strong Client Authentication (SISCA)*, which stems from the above result. SISCA is able to resist MITM+certificate attacks, offering advantages compared to existing solutions that focus at preventing server impersonation (see Section 8.4.9), as well as MITM+key attacks under the assumption that the attacker does not persistently compromise the server (see Section 8.4.2). The details of our solution follow below.

8.4.2 Design Goals and Assumptions

In SISCA we seek to satisfy the following requirements: *(i)* incremental deployment, *(ii)* scalability, *(iii)* minimal overhead, *(iv)* account for cross-origin communication, assuming that the involved origins belong to, and

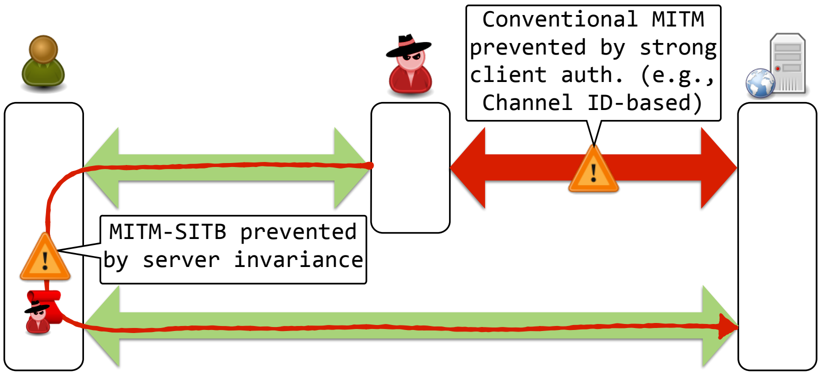

Conventional MITM prevented by strong client auth. (e.g., Channel ID-based)
MITM-SITB prevented by server invariance

Figure 8.6: TLS MITM attacks in web applications can be thwarted by combining strong client authentication with server invariance.

are administered by the *same entity*, *(v)* mitigation of MITM+key attacks (besides MITM+certificate attacks).

We make the following assumptions. First, strong client authentication, which prevents the conventional way of implementing MITM attacks (Figures 8.1, 8.2) is in place. Specifically, we assume that SISCA-enabled servers implement *Channel ID-based client authentication*. As mentioned before, Channel IDs are already experimentally supported in Google Chrome. Moreover, FIDO U2F leverages Channel IDs, as metiononed in Section 8.2.1, so it is likely that Channel ID-based authentication will become available in the foreseeable future.

Second, we assume that SISCA-enabled servers support *TLS with forward secrecy* by default [93, 95, 116]. As we discuss below, this is only required for preventing MITM+key attacks (not relevant for MITM+certificate attacks). Moreover, we assume that TLS is secure and cannot be broken by cryptographic attacks, such as those surveyed in [45].

We finally assume that the MITM+key attacker does not persistently compromise the target web server. As we discuss later, this enables SISCA to resist server key compromise (i.e., MITM+key attackers) through frequent rotation of the server secrets that are used in SISCA (see Section 8.4.8). We also note that if an attacker gained persistent control over the target server, he would probably not need to resort to MITM attacks to compromise the users' accounts, but at the same time he would increase the probability of being detected. In Part III of this thesis, we actually consider such attackers and try to provide data integrity guarantees, even under such powerful attacks.

8.4.3 Server Invariance Versus Authentication

As stated above, our goal is to combine strong client authentication with server invariance. Invariance is a *weaker* property than authentication, and thus, *easier* to achieve, as *no a priori trust* is necessary. In contrast, authentication requires some form of initial trust so that the client can correctly authenticate the server [60].

Consequently, we stress the following very important difference. Server authentication (and solutions that try to enforce it, like those mentioned in Section 8.3.1) implies that every single TLS connection should be established with the legitimate server. If the attacker attempts to intercept such a connection, he should be detected by the browser, i.e., no server impersonation should be possible.

In contrast, server invariance, embraces the fact that the attacker can successfully impersonate the server. As such, we distinguish two scenarios concerning the browser's *first connection* to a particular server: *(i)* The first connection is *not intercepted* by the attacker. Then, server invariance implies that the attacker is allowed to *intercept none* of the subsequent connections to that server. *(ii)* The first connection is *intercepted* by the attacker. Then, server invariance implies that the attacker has to *intercept all* subsequent connections to that server. In either scenario, if the attacker violates server invariance, he will be detected.

We consider server invariance as a *transient* property whose scope is one browsing session. Server invariance is *reset* whenever the browser restarts, i.e., the attacker is allowed again to choose whether to intercept or not the first connection to the server.

8.4.4 Towards Implementing Server Invariance

In order to implement server invariance, it is important to understand the implications of the fact that the attacker is allowed to impersonate the server. Namely, the attacker can intercept the first connection and influence the entire HTTP response, which clearly cannot be blindly trusted. Therefore, techniques that assume the attacker is able to influence only a part of the HTTP response, such as *Content Security Policy* (CSP) [221] for mitigating *Cross-Site-Scripting* (XSS) [156], as well as techniques that assume the first connection is trusted (i.e., not intercepted by the attacker), such as pinning, cannot be directly applied for implementing server invariance.

Instead, a server invariance protocol should consist of *two phases*, namely *invariance initialization* and *invariance verification* – initialization and verification for brevity. In the initialization phase, which is executed in the first connection to the server during a browsing session (and could be intercepted by the attacker), the browser establishes a *point of reference*. Then, in subsequent connections to the same server, the verification phase is executed, where the browser verifies that the point of reference remains unchanged, i.e., the browser keeps connecting to the same entity.

Server Public Keys. Assuming that we only consider MITM+certificate attackers, we can leverage the servers' public keys as the point of reference. Even if the attacker intercepts the first connection, he will not be able to let any subsequent connections reach the legitimate server, because the server's public key will be different from the attacker's. Nevertheless, servers of the same domain may use different public keys and also, cross-origin interacting domains will have different keys. To solve this issue, we need to "tie" all the involved public keys together, to reflect the fact that they belong to the same entity and thus server invariance should hold across all these domains and keys. We sketch the following technique for implementing server invariance.

During initialization (first connection), the server sends a list of all the involved domains and all their public keys to the browser, and the latter uses the witnessed key as well as the list as the point of reference. Then, in subsequent connections, the browser verifies *(i)* that the public key which the server presents is contained in the list which was received during initialization, and *(ii)* that the server agrees on the legitimacy of the public key that was originally witnessed by the browser during initialization. Notice how this differs from pinning, which operates under the assumption that the initial connection is trusted, and thus does not seek to verify the legitimacy of the initial connection, and consequently of the received pins, upon subsequent connections.

The above technique is useful when considering MITM+certificate attacks and can be used to implement the server invariance protocol in SISCA. Nevertheless, in the following sections we present an alternative approach that does not leverage server public keys, and aims to mitigate MITM+key attacks, as well. We note that the security analysis as well as most of the design patterns that are discussed in the approach that follows (e.g., how to prevent downgrade attacks and allow for partial support and exceptions, how to secure resource caching, etc) would similarly apply to the previously sketched technique, as well.

Our High-Level Approach. In SISCA we choose to implement server invariance as a simple challenge/response protocol. In the initialization phase (first connection) the browser sets up a fresh challenge/response pair (which acts as the point of reference) with the server. Then, in the verification phase (subsequent connections) the browser challenges the server to verify server invariance, i.e., that it is the same entity with which the browser executed the initialization.

SISCA has to be executed before any HTTP traffic influenced by the attacker is processed by the browser or the server. We choose to implement the protocol at the application layer, over established TLS sessions via an HTTP header, named `X-Server-Inv`, and transmitted together with the *first* HTTP request/response pair over a particular TLS connection. For the protocol to be secure, on the client side this header is controlled solely by the browser. It cannot be created or accessed programmatically via scripts (similar to cookie-related headers [228]).

Alternatively, we could implement the server invariance protocol in SISCA as a TLS extension, i.e., at the transport layer. We deem the application layer more appropriate, since server invariance encompasses semantics that are naturally offered by the application layer, such as cross-origin interaction and content inclusion.

Figure 8.7 depicts a simple example of how a protocol based on our approach can look like. In this example protocol, during the initialization phase the browser and server generate random numbers r_b and r_s, which they both store (the server also stores the browser's Channel ID cid_b). The browser subsequently uses r_b as a challenge during the verification phase, expecting the response r_s by the server. The latter looks up r_s by using r_b and cid_b. For the shake of brevity, we do not analyze this example, but we make the following important remarks.

First, this example requires the server to store per-client state. This may be undesirable and it also makes it harder for multiple servers belonging to the same entity to share the common state which is needed in order to be able to correctly execute the protocol. For this reason, SISCA uses symmetric cryptography (MAC), in order to securely offload the state to the clients.

Second, during the verification phase, the server should process the incoming HTTP request, only if the lookup succeeds. If it fails, it means that the attacker intercepted the first connection (initialization phase) and that the incoming request may be malicious. We explain this concept further in the analysis of the SISCA protocol. We note that due to this fact, SISCA uses a second MAC tag in order to enable the server perform this check.

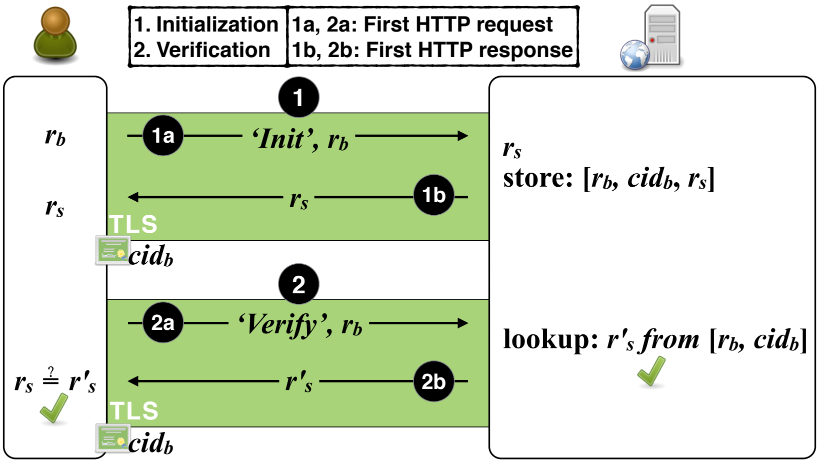

1. Initialization
2. Verification
1a, 2a: First HTTP request
1b, 2b: First HTTP response
1
1a
'Init', r_b
r_b
r_s
1b
TLS
cid_b
r_s
store: [r_b, cid_b, r_s]
2
2a
'Verify', r_b
r'_s
2b
$r_s \overset{?}{=} r'_s$
lookup: r'_s from [r_b, cid_b]

Figure 8.7: An example challenge/response-based server invariance protocol requiring per-client server state.

8.4.5 Basic Protocol

We now describe the server invariance protocol of SISCA in detail. We follow a structural approach, meaning that we start with a basic version of our protocol, described in this section. Then, in subsequent sections, we incrementally add features.

Figure 8.8 illustrates the protocol, assuming no attack. Prior to the protocol execution, the server, `www.example.com`, generates two keys k_{s1} and k_{s2}, called *SISCA keys*. The same SISCA keys are used for all protocol executions (i.e., not for a specific client) and are never disclosed to other parties. Moreover, recall that the server and client deploy Channel ID-based authentication. Each TLS connection will therefore have a Channel ID cid_b, that is created by the user's browser. As already mentioned, the protocol consists of two phases.

Initialization. The initilization phase occurs once the browser establishes a TLS connection to `www.example.com`, for the *first time* in a browsing session (upper connection in Figure 8.8). The browser picks a random number r_b. It then sends $\langle$'$Init$', $r_b\rangle$ to the server ('$Init$' is a string constant), within the *first* HTTP request[2] over that connection. Upon receiving

[2]Note that this is a request that browser would anyway submit, i.e., required for loading the webpage. It is not an extra request.

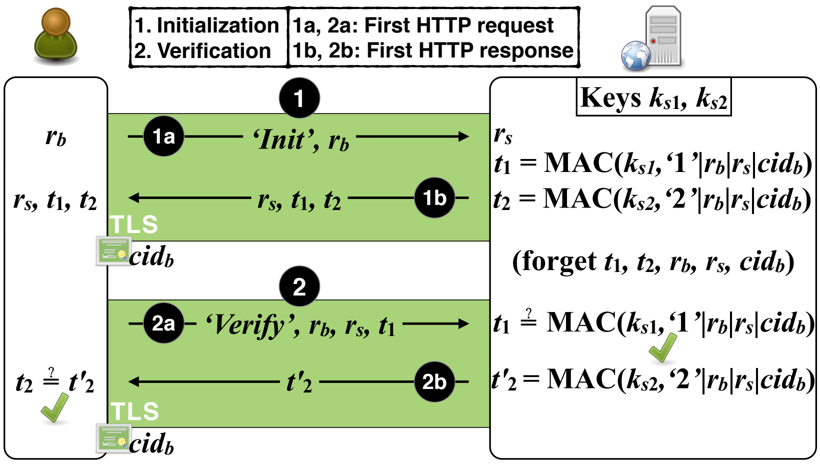

1. Initialization
2. Verification
1a, 2a: First HTTP request
1b, 2b: First HTTP response
Keys k_{s1}, k_{s2}
1
1a
'Init', r_b
r_b
r_s
t_1 = MAC(k_{s1}, '1'|r_b|r_s|cid_b)
t_2 = MAC(k_{s2}, '2'|r_b|r_s|cid_b)
1b
r_s, t_1, t_2
r_s, t_1, t_2
TLS
cid_b
(forget t_1, t_2, r_b, r_s, cid_b)
2
2a
'Verify', r_b, r_s, t_1
$t_1 \overset{?}{=}$ MAC(k_{s1}, '1'|r_b|r_s|cid_b)
t'_2 = MAC(k_{s2}, '2'|r_b|r_s|cid_b)
2b
t'_2
$t_2 \overset{?}{=} t'_2$
TLS
cid_b

Figure 8.8: Basic SISCA protocol.

this message, the server chooses a random number r_s and computes the following message authentication tags:

$$t_1 = \text{MAC}(k_{s1}, \text{`1'}|r_b|r_s|cid_b) \tag{8.1}$$

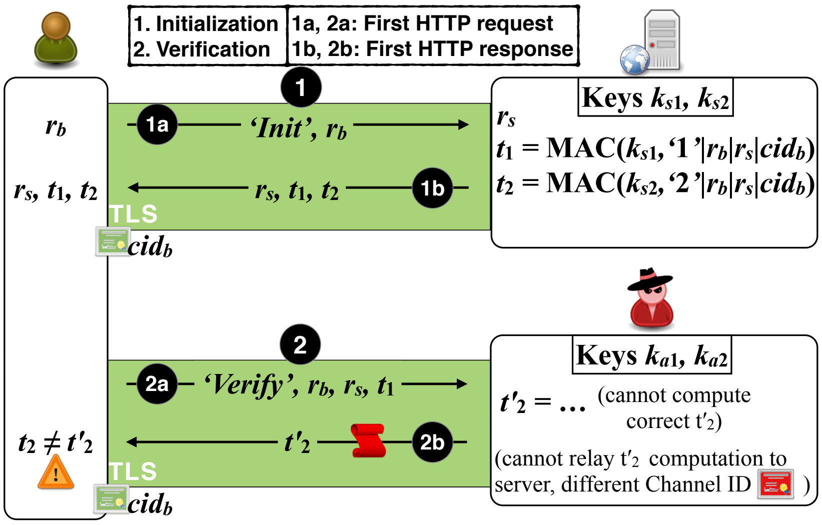
$$t_2 = \text{MAC}(k_{s2}, \text{`2'}|r_b|r_s|cid_b) \tag{8.2}$$

where '1' and '2' are strings constants. Notice that the server binds the computed tags to the browser's Channel ID cid_b. r_b, r_s and the MAC tags will be used in subsequent TLS connections to verify server invariance.

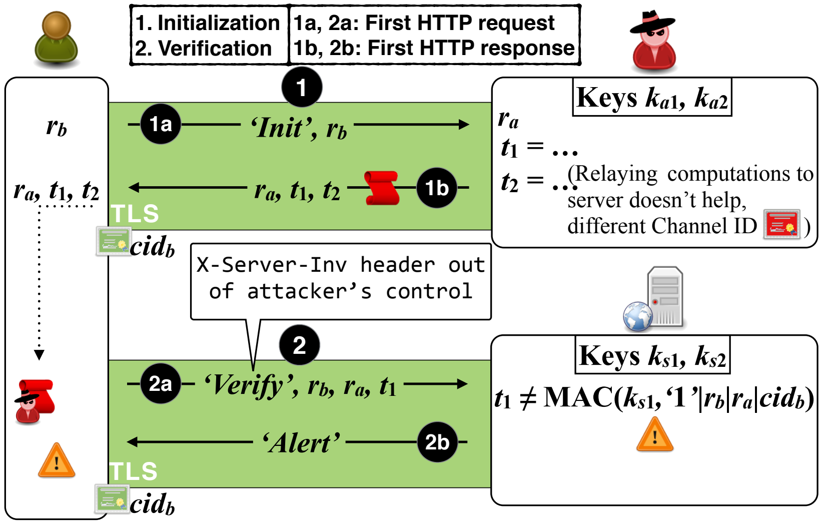
Finally, the server sends $\langle r_s, t_1, t_2\rangle$ to the browser within its first HTTP response. The browser stores $\langle r_b, r_s, t_1, t_2\rangle$, while the server does not store any client-specific information. At this point, the initialization phase is complete. Subsequent HTTP requests and responses over that particular TLS connection do *not* include an `X-Server-Inv` header.

Verification. The verification phase takes place upon every subsequent TLS connection to `www.example.com`, which occurs within the same browsing session (lower connection in Figure 8.8). Like in the first phase, the protocol messages are exchanged within the *first* HTTP request/response pair. The browser sends $\langle \text{`}Verify\text{'}, r_b, r_s, t_1\rangle$ to the server, as part of the first request. After receiving the request, and *before* processing it, the server first checks if

$$t_1 \overset{?}{=} \text{MAC}(k_{s1}, \text{`1'}|r_b|r_s|cid_b). \tag{8.3}$$


1. Initialization
2. Verification
1a, 2a: First HTTP request
1b, 2b: First HTTP response
Keys k_{s1}, k_{s2}
r_b
r_s, t_1, t_2
'Init', r_b
r_s, t_1, t_2
r_s
$t_1 = MAC(k_{s1}, `1'|r_b|r_s|cid_b)$
$t_2 = MAC(k_{s2}, `2'|r_b|r_s|cid_b)$
TLS
cid_b
Keys k_{a1}, k_{a2}
'Verify', r_b, r_s, t_1
t'_2
$t_2 \neq t'_2$
$t'_2 = \ldots$ (cannot compute correct t'_2)
(cannot relay t'_2 computation to server, different Channel ID)

(a) Attacker intercepts the verification phase.


1. Initialization
2. Verification
1a, 2a: First HTTP request
1b, 2b: First HTTP response
Keys k_{a1}, k_{a2}
r_b
r_a, t_1, t_2
'Init', r_b
r_a, t_1, t_2
r_a
$t_1 = \ldots$
$t_2 = \ldots$ (Relaying computations to server doesn't help, different Channel ID)
TLS
cid_b
X-Server-Inv header out of attacker's control
Keys k_{s1}, k_{s2}
'Verify', r_b, r_a, t_1
'Alert'
$t_1 \neq MAC(k_{s1}, `1'|r_b|r_a|cid_b)$

(b) Attacker intercepts the initialization phase.

Figure 8.9: Resilience of SISCA to MITM-SITB (conventional MITM is prevented by Channel-ID based authentication).

Here, cid_b corresponds to the Channel ID of the TLS session within which the protocol is currently being executed, which, if under attack, might differ from the Channel ID that was used in the initialization phase. If the check passes, the server computes

$$t'_2 = \text{MAC}(k_{s2}, \text{‘2’}|r_b|r_s|cid_b), \tag{8.4}$$

processes the received request, and passes $\langle t'_2 \rangle$ within the HTTP response to the browser. Finally, the browser checks if $t'_2 \overset{?}{=} t_2$ and if it succeeds, it means that server invariance holds for this TLS connection.

Analysis When Under Attack. Figure 8.9 illustrates how the protocol prevents MITM attacks. Recall that, due to the usage of Channel ID-based authentication, the attacker cannot perform the conventional attack (Figures 8.1, 8.2) – the attacker's TLS sessions will have a different Channel ID than the client's and will thus be rejected. Instead, he has to execute a MITM-SITB attack.

In Figure 8.9 we illustrate two possible attack scenarios (based on the previous discussion in this Section) and we show why the attacker fails in both. In Figure 8.9(a) the attacker intercepts the verification phase of SISCA. Since the attacker didn't participate in the initialization phase of the protocol, he does not know the correct MAC response t_2 to the client's challenge. Moreover, since he does not have access to k_{s2}, he cannot calculate the correct t_2 either (Eq. (8.4)). As a result, the user's browser rejects the attacker's response and terminates the session, notifying the user (no user decision is required). Even if the attacker pushes a malicious script in his response, it will not get a chance of being executed.

In the second scenario, depicted in Figure 8.9(b), the attacker intercepts the first TLS connection to `www.example.com`. He thus executes the initialization phase with the browser and injects his script, which is executed within the web origin of `www.example.com`. To successfully complete his attack, the attacker needs to let a subsequent TLS connection reach the legitimate server, and access the user's account via that connection.

After the browser establishes a connection with the legitimate server, the two of them execute the verification phase, as part of the first HTTP request/response pair. The server, before processing the HTTP request (which might as well be malicious), checks whether Condition (8.3) is true. Since the attacker does not have access to key k_{s1}, he could not have computed the correct t_1 (Eq. (8.1)). Thus, during the initialization phase, he sends a t_1 value to the browser that is not the correct one. Consequently, Condition (8.3) will not be satisfied. In this case the server does not process the request, and instead notifies the browser by sending an empty HTTP response containing $\langle$‘$Alert$’$\rangle$ in the `X-Server-Inv` header. This indicates violation of the server invariance and the browser aborts the session.

We remark that in the second scenario, it is the legitimate server that checks server invariance, detects the ongoing MITM attack and notifies the browser. This is important in order to prevent even a single malicious request from being accepted and processed by the server.

We conclude our analysis, with a few remarks that are relevant for both of the scenarios described above. First, note that the attacker cannot relay any of the necessary MAC computations to the legitimate server. In other words, he cannot manipulate the server to compute for him the values needed for cheating in the protocol. This is because the server binds all its computations to the channel ID of the client with whom it communicates (the attacker's channel ID will be different from the user's).

Second, note that the protocol is secure so long as the attacker cannot "open" already established TLS connections between the browser and the legitimate server (i.e., connections that he chose not to intercept). If he could do that, he would be able to extract the correct values of both t_1 and t_2 and successfully cheat. Recall that, the MITM+key attacker holds the private key of the legitimate server. Therefore, in order to prevent such an attacker from eavesdropping on already established TLS connections, it is essential that these connections have TLS forward secrecy enabled.

Third, when considering MITM+key attacks it is reasonable to assume that the attacker can also extract the SISCA keys, similar to the private key of the server. As stated in the assumptions (Section 8.4.2) and explained in Section 8.4.8 the SISCA keys, unlike the private key, can be frequently rotated. SISCA can thus resist MITM+key attacks, assuming no persistent server compromise.

Finally, the attacker can choose not to reply at all, when executing SISCA with the user. This essentially leads to a Denial of Service (DoS) attack. However, such attacks can already be achieved even by attackers less powerful that those considered here. That is, attackers that cannot perform TLS MITM attacks, but can block network traffic between the browser and the server.

Different Origins. The SISCA protocol execution is guided by the same-origin policy [17]. In particular, SISCA is executed independently, i.e., different *protocol instances*, when loading webpages and documents that belong to different origins. For example, assume that the browser navigates to `www.example.com` for the first time in the current browsing session. Then, a new instance of SISCA will be created for this origin and its initialization phase will be executed on the first TLS connection. If the browser further navigates to pages belonging to `www.example.com`, and this triggers the creation of new TLS connections by the browser, then for those connections the browser will execute the verification phase of the previously created SISCA instance corresponding to `www.example.com` (same origin). When the browser navigates to another website (different origin), say `www.another.com`, then a new instance of SISCA will be created and used for the loading of documents from that origin (assuming that this is the first visit to `www.another.com` in that browsing session). Also any HTTP redirections during navigation that lead to different origins will cause the corresponding SISCA instances for those origins to be created and used.

8.4.6 Cross-Origin Communication

In the previous section we assumed that accessing the webpages of the website at `www.example.com` involves communication only with that domain, i.e., web origin. However, this is not a realistic scenario in today's web applications. Many websites perform cross-origin requests, e.g., to load resources. SISCA can accommodate for such scenarios so long as all the involved domains belong to, and are administered by the *same entity*, such that the required SISCA keys, k_{s1} and k_{s2}, can be shared across all relevant servers.

Therefore, for cross-origin communication the browser uses the SISCA instance corresponding to the initiating origin. For example, assume that a page loaded from `www.example.com` performs a cross-origin request to `static.example.com`. The browser will create a TLS connection to `static.example.com` and will execute the verification phase of the SISCA instance that corresponds to `www.example.com`. Any potential HTTP redirections will also use the SISCA instance of the initiating origin, `www.example.com`.

Different Channel IDs. The basic protocol we described in Section 8.4.5 also works in the cross-origin communication scenario, provided that the Channel ID used by the browser is the *same*. The Channel ID specification draft already recommends using the same Channel ID for a domain and its subdomains [13, 56] (to account for cookies that have the `Domain` attribute set). For example, the browser should use the same Channel ID for `www.example.com` and `static.example.com`. Nevertheless, for

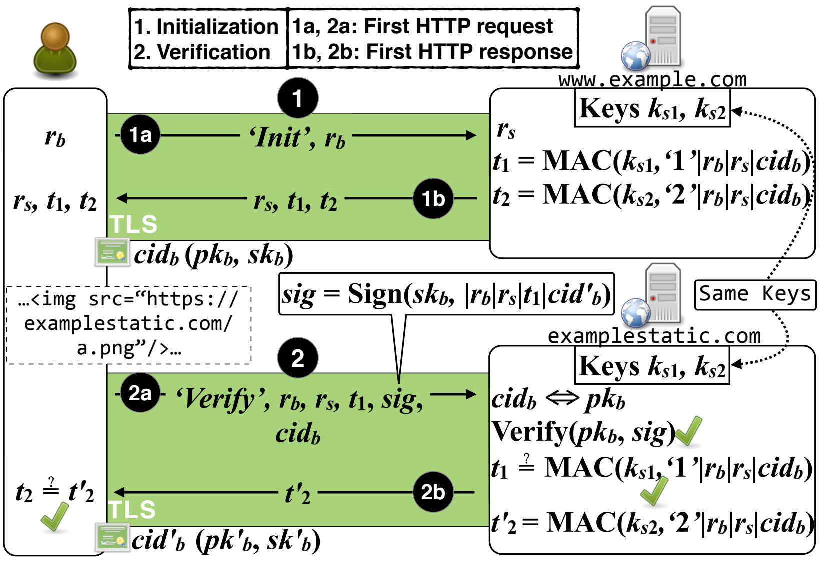

1. Initialization
2. Verification
1a, 2a: First HTTP request
1b, 2b: First HTTP response
www.example.com
Keys k_{s1}, k_{s2}
1
1a
'Init', r_b
r_b
r_s
t_1 = MAC(k_{s1}, '1'|r_b|r_s|cid_b)
t_2 = MAC(k_{s2}, '2'|r_b|r_s|cid_b)
r_s, t_1, t_2
1b
TLS
cid_b (pk_b, sk_b)
…<img src="https://examplestatic.com/a.png"/>…
sig = Sign(sk_b, |r_b|r_s|t_1|cid'_b)
Same Keys
examplestatic.com
Keys k_{s1}, k_{s2}
2
2a
'Verify', r_b, r_s, t_1, sig, cid_b
$cid_b \Leftrightarrow pk_b$
Verify(pk_b, sig)
$t_1 \overset{?}{=}$ MAC(k_{s1}, '1'|r_b|r_s|cid_b)
t'_2 = MAC(k_{s2}, '2'|r_b|r_s|cid_b)
$t_2 \overset{?}{=} t'_2$
t'_2
2b
TLS
cid'_b (pk'_b, sk'_b)

Figure 8.10: SISCA adapted for cross-origin communication (the origins share the same SISCA keys), when the browser uses a different Channel ID for each origin. Here, `www.example.com` performs a cross-origin request to `examplestatic.com`.

privacy reasons, the specification recommends using different Channel IDs for unrelated domains. In such cases, SISCA has to account for using different Channel IDs across domains, when cross-origin communication takes place.

Figure 8.10 depicts how the protocol works in such a scenario. The browser navigates to `www.example.com`, and starts a new SISCA instance for that origin. The browser uses Channel ID cid_b (with public key pk_b, and private key sk_b). At some later point in time, the page loaded from `www.example.com` performs a cross-origin request to `examplestatic.com`, which is controlled by the same entity. Nevertheless, since it corresponds to a different domain (i.e., not a subdomain), the browser uses a different Channel ID, say cid'_b (with pk'_b, sk'_b being the corresponding public/private key pair). In this case, although the initialization phase of SISCA was executed using cid_b, the verification phase will have to be executed over a TLS connection with Channel ID cid'_b.

The browser needs to tell the server (`examplestatic.com`) to use cid_b instead of cid'_b, but do so in a secure way. To achieve this, the browser endorses cid'_b, by signing it with sk_b, and thus proving to the server that it owns the private keys of both Channel IDs cid_b and cid'_b. The browser extends the '$Verify$' message by appending cid_b and a signature over cid'_b (i.e., the Channel ID of that TLS connection) and the rest of the message parameters using sk_b. The server, before processing the request, verifies the signature on cid'_b using the supplied cid_b (i.e., pk_b). If it passes, then the server uses cid_b for the subsequent steps of the verification phase, which remain unchanged.

Overlapping Cross-Origin Access. Browsers typically send multiple HTTP requests over the same network connection (persistent connections [73]). Due to the existence of cross-origin communication, a TLS connection to a particular domain, say `static.example.com`, can be used by the browser to transmit cross-origin requests to `static.example.com` made by different initiating origins. For example, the browser uses the same TLS connection to `static.example.com`, to transmit, first, a request originating from a document belonging to `www.example.com` and then, a request originating from a document belonging to `shop.example.com` (we still assume that all three domains belong to the same entity). In this case, the TLS connection to `static.example.com` has to be verified using SISCA for both initiating domains, independently.

In the above scenario, the browser executes the verification phase with the SISCA instance of `www.example.com`, upon establishing the TLS connection to `static.example.com` and sending the first HTTP request, originating from `www.example.com`. Subsequently, when the browser wants to reuse the same connection to send a cross-origin request from `shop.example.com` to `static.example.com`, it once again executes the verification phase, only this time with the SISCA instance of the domain `shop.example.com`. This takes place upon transmitting the first HTTP request, which originates from `shop.example.com`.

Origin Change. A webpage is allowed to change its own origin (effective origin) to a suffix of its domain, by programmatically setting the value of `document.domain` [142]. This allows two pages belonging to different subdomains, but presumably to the same entity, to set their origin to a common value and enable interaction between them (we note that, both pages have to explicitly set `document.domain`, even if the origin of one page is already equal to the desired domain suffix). For example, a page from `www.example.com` and a page from `shop.example.com` can both set their origin to `example.com`. In such a case, the attacker can attack the user account at `shop.example.com`, by intercepting the first connection to `www.example.com` (or any other `example.com` subdomain), or vice versa.

To prevent such an attack, the browser has to verify that server invariance holds across each pair of origins that change their effective origin to a common value, before allowing any interaction between them. Each origin has its own SISCA instance established, and we must ensure that both SISCA instances were initialized with the same remote entity. This can be achieved by running the verification phase of both instances over the same TLS connection (established to either origin). The browser can reuse an already established and verified connection with one origin, and just verify the connection with the SISCA instance of the other origin. If no such connection exists at that time, then the browser can create a new one to either origin and execute the verification phase of both SISCA instances. If there is no actual HTTP request to be sent at that time, the browser can make use of an HTTP `OPTIONS` request.

Partial Support and Downgrade Attacks. SISCA must be incrementally deployable, which means that it must maintain compatibility with legacy web servers, without compromising the security of the SISCA-enabled servers. Moreover, websites must be able to opt for partial support. As an example, a domain implements SISCA but still needs to perform cross-origin requests to another domain, called *incompatible*, that either does not support SISCA, or supports it but belongs to a 3rd party, i.e., it has different SISCA keys (we discuss on the security of such design choices at the end of this section).

The above can be achieved by allowing *exceptions*. If a particular domain does not support SISCA (including legacy servers that are not aware of SISCA at all), then it can simply ignore the `X-Server-Inv` header, sent during the initialization phase, and reply without including any SISCA-related information. This will be received by the browser as an *exception claim*. Moreover, if a domain supports SISCA but performs cross-origin communication with one or more incompatible domains, then it can append an *exception list* in its response, during the initialization phase, designating the incompatible domains.

However, we note that if the attacker intercepts the initialization phase of the protocol, then he could perform a *protocol downgrade attack*, by providing false exception claims or exception lists in his response.

To prevent downgrade attacks, the browser should verify any exception that was received during the initialization phase, upon every subsequent

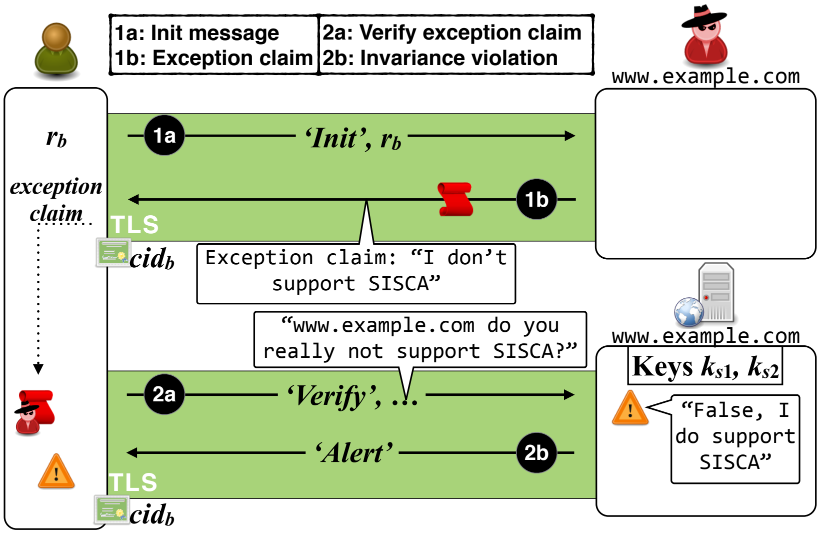


Figure 8.11: Preventing downgrade attacks (same-origin case).

connection. If the attacker intercepted the initialization phase and replied with fake exception claims, then if any of the subsequent connections reaches the legitimate server, the browser, with the help of the legitimate server, would detect the attack. This scenario is illustrated in Figure 8.11.

Regarding cross-origin communication, in order to help SISCA-enabled legitimate servers detect fake exception lists previously received by the browser, SISCA protocol messages should include (in the `X-Server-Inv` header) the origin associated with the SISCA instance. Suppose for example, that the browser executes the initialization phase with `www.example.com` which supports SISCA (executes the protocol normally), but also includes an exception list stating that it performs cross-origin requests to domain `shop.example.com` which does not support SISCA. Whenever the browser connects to `shop.example.com` to perform a cross-origin request from `www.example.com`, the browser includes the origin of the SISCA instance (`www.example.com`) and asks `shop.example.com` whether it indeed does not support SISCA with respect to that origin. Assuming that the connection was not intercepted, `shop.example.com` can leverage the supplied origin information to decide whether the exception reported by the browser is valid. If not, then it should abort processing the request and notify the browser of the detected attack. Note that the above assumes that each

SISCA-enabled server is aware of all the domains that is compatible to execute SISCA with (i.e., domains with which it shares the same SISCA keys), which is not difficult to implement.

Third-party Content Inclusion. As mentioned above, a domain implementing SISCA, say `www.example.com`, can still perform cross-origin requests to incompatible 3rd party domains as long as it designates those domains as exceptions for the protocol. This of course means that TLS connections to those domains will not be protected by SISCA, and could be MITM-ed by the attacker to perform a user impersonation attack on `www.example.com`.

This can be indeed the case if `www.example.com` includes active content [141] (in particular, JavaScript and CSS) from those domains. While JavaScript from 3rd party sites is generally not recommended, and usually there are ways of avoiding it [149], Subresource Integrity (SRI) [145] can be used to include active content in a secure manner. SRI enables a website to ensure that the included files have not been manipulated through the use of cryptographic hashes that are checked by the browser. Alternatively, depending on the use case, it may be possible to use *iframes* to isolate active 3rd party content, instead of directly embedding it within the target origin (the `sandbox` attribute can help even further).

The embedding of passive content only, such as images, does not give the attacker the ability to execute his code within the target origin. Hence, with respect to preventing user impersonation, such embeddings are safe and do not undermine the security offered by SISCA.

8.4.7 Resource Caching

Caching of static resources, such as scripts and images, helps reduce webpage loading times as well as server resource consumption. However, the way caching is currently implemented [73, 82] can give a MITM attacker the opportunity to subvert SISCA.

Resource caching attacks are described in detail in [102]. For the case of SISCA the attacks works briefly as follows. During one browsing session, the attacker intercepts all TLS connections and ensures that a legitimate, yet maliciously modified script that is required by the target web server is cached by the browser. Then, during a second browsing session, the attacker lets all connections pass through. When the legitimate webpage asks for the inclusion of the aforementioned script, the browser will load it from cache, essentially enabling the execution of the attacker's malicious code. The attacker will thus be able to access the target web server.

To prevent the above attack, we need to change the way caching is performed for active content that would enable this attack (JavaScript and CSS files). We need to make sure that the browser *always* communicates with the server in order to verify that the cached version is the most recent and also the correct one (i.e., not maliciously modified). This means that no long-term caching is allowed, rather the browser has to perform the check every time, before using the cached version. Thus, caching of such files should be performed only using Entity Tags (ETags) [73], but in a more rigorous way than specified in the current HTTP specification. In particular, if a web server wishes to instruct a browser to cache a JavaScript or CSS file, the server should use an `ETag` header which always contains a cryptographic hash of the file. The browser, before using, and caching the file should *verify* that the supplied hash is correct. Subsequently, before the browser uses the cached version of the file, it first verifies that the local version matches the version of the server (using the `If-None-Match` header, as currently done).

Note that, if the attacker intercepts the connection and lies to the browser about the validity of the cached resource, SISCA will not allow any other connection during the same browsing session to reach the legitimate server, so the attacker cannot mount his attack through the maliciously cached file.

8.4.8 Key Rotation

In SISCA, the server has a pair of secret keys, k_{s1} and k_{s2}. To resist key compromise (i.e., MITM+key attackers), these keys, unlike the server's private key, can be easily rotated. This is because the SISCA keys need not undergo any certification process, and can thus be rotated frequently, e.g., weekly, daily, or even hourly. The more frequent the rotation the smaller the attacker's window of opportunity to successfully mount MITM attacks.

The key transition, of course, has to be performed such that it does not break the execution of active browser SISCA instances that rely on the previous keys. At a high level, one way of achieving this, is to have the server keep previous keys for a certain period of time (i.e., allow partial overlap of keys). This can enable browsers with active SISCA instances that rely on the previous keys to securely transition to new protocol parameters, i.e., t_1 and t_2, computed using the new server SISCA keys.

For domains served by a single machine, this is only a matter of implementing the corresponding functionality in the web server software (e.g., Apache). For multiple domains controlled by the same entity and served by multiple machines, located in the same data center or even in different data centers across the world, arguably more effort is required in order to distribute the ever-changing keys and keep the machines in sync. Nevertheless, a similar mechanism is needed for enabling TLS forward secrecy while supporting TLS session tickets [115]. According to Twitter's official blog [95], Twitter engineers have implemented such a key distribution mechanism.

8.4.9 SISCA Benefits and Drawbacks

SISCA offers the following advantages regarding MITM+certificate attack prevention. Compared to multipath probing solutions, SISCA does not rely on any third party infrastructure, trusted or not. Since SISCA is built on top of Channel ID-based authentication, it has to assume that no MITM attack takes place during user enrollment. Nevertheless, after this step, no "blind" trust is required when the user uses a new or clean browser, contrary to pinning solutions (except preloaded pins), as discussed in Section 8.4.4. Moreover, in SISCA no user decision is necessary whenever server invariance violation is detected. This can occur either due to an attack, or due to an internal server fault, thus the browser can abort (possibly after retrying) the session. SISCA is scalable since it can be deployed incrementally by web providers (assuming browser support). Finally, SISCA resists MITM+key attacks, assuming that the attacker does not persistently compromise the server.

The main disadvantage of SISCA is that it only protects against MITM attackers whose goal is to impersonate the user to the server. This is arguably the most common and impactful attacker goal. SISCA does not protect against attackers whose objective is to provide fake content to the user. In such cases the attacker can simply intercept *all* connections and interact with the user by serving his own, fake content. In contrast, the techniques that focus on ensuring the correctness of server authentication (Section 8.3.1) can protect against such attacks (MITM+certificate attackers). As a result, a recommended strategy would be to use SISCA in conjunction with any of these techniques. Finally, SISCA requires coordination between an entity's different domains, in the sense they must have access to the same SISCA keys. This is needed for securing cross-origin communication and, depending on the scale of the entity, can be challenging from an engineering perspective to set up.

8.4.10 Interaction With Other Web Technologies

SPDY. SPDY [22] multiplexes concurrent HTTP requests over the same TLS connection to improve network performance. In order for SISCA to be compatible with the general SPDY functionality, the browser must ensure that before the SISCA protocol is completed successfully (i.e., the first request/response pair is exchanged), no further requests are sent through the SPDY connection.

Furthermore, SPDY IP Pooling allows, under certain circumstances, HTTP sessions from the same browser to different domains (web origins) to be multiplexed over the same connection. Version 3 of SPDY is compatible with Channel IDs (recall that different Channel IDs may need to be used for different origins, but now there is only one TLS connection). SISCA is compatible with IP Pooling, as long as the browser manages the multiplexed HTTP sessions independently, with respect to the execution of the SISCA protocol.

WebSocket. SISCA is compatible with the WebSocket protocol [70], when the latter is executed over TLS. This, of course assumes that *(i)* Channel IDs are used for the WebSocket TLS connections, *(ii)* the SISCA protocol is executed during the WebSocket handshake (i.e., first request/response pair), and *(iii)* JavaScript is not be able to manipulate the `X-Server-Inv` header.

Web Storage. Web Storage [227] is an HTML5 feature that allows a web application to store data locally in the browser. SISCA can protect `code.sessionStorage` (temporary storage), but does not prevent a MITM attacker from accessing information stored in `window.localStorage` (permanent storage), so no sensitive information should be stored there.

Offline Web Applications. HTML5 offers Offline Web Applications [222] which allow a website to create an offline version, stored locally in the browser. As with regular file caching (see Section 8.4.7), this feature can be leveraged by the attacker to bypass SISCA. Making this feature secure requires the introduction of design concepts similar to what we proposed for regular caching.

Other Client-Side Technologies. The attacker might attempt to leverage various active client-side technologies besides JavaScript, such as Flash, Java and Silverlight. Such technologies allow the attacker to create direct TLS connections to the legitimate server. Some of the APIs offered by those technologies also allow the attacker to forge and arbitrarily manipulate HTTP headers, including cookie-related headers or the `X-Server-Inv` header. However, provided that Channel IDs and SISCA are not integrated with these technologies (i.e., a TLS connection created by such an API will have to manually create and use its own Channel IDs, and that the browser will not automatically execute SISCA over those connections), the attacker will not be able to impersonate the user and compromise his account on the legitimate server.

8.4.11 Prototype SISCA Implementation

We created a proof of concept implementation of the basic SISCA protocol, with additional support for cross-origin communication, provided that the same Channel ID is used. On the server side we use Apache 2.4.7 with OpenSSL 1.0.1f [152] , patched for Channel ID support. SISCA is implemented as an Apache module and consists of 313 lines of C code. On the client side we implement SISCA by modifying the source code of Chromium 35.0.1849.0 (252194) and the WebKit (Blink) engine. We make a total of 319 line modifications (insertions/deletions) in existing files and we add 6 new files consisting of 418 lines of C++ code.

We use Base64 encoding for binary data transmission. When using 128-bit random values (r_b and r_s) and HMAC-SHA256 (i.e., 256-bit tags, t_1 and t_2), the client's lengthiest message is 114 bytes long, plus the origin of the SISCA instance that has to be sent as well. The server's lengthiest message is 132 bytes long.

We finally verified that our implementation successfully blocks our proof of concept MITM-SITB attack.

Performance Evaluation. To assess the performance overhead imposed by SISCA (the server invariance part, not the overhead due to Channel IDs), we measured the latency of HTTP request/response roundtrips, with SISCA enabled and disabled. For the measurements we used a 4KB HTML page, as well as an 84KB jQuery compressed file, retrieved over a domain that we set up as being "cookieless". Chromium ran on a Macbook Pro laptop (2.3GHz CPU, 8GB RAM) and Apache ran on a typical server machine (six core Intel Xeon 2.53GHz, 12GB RAM), connected through the campus network.

We found that the overhead of the basic SISCA protocol is negligible, as no increase in latency was measured (averaged over 300 repetitions). Moreover, the HTTP request to the cookieless domain was able to fit in a single outgoing packet (a typically desired objective).

Regarding cross-origin communication over different Channel IDs (see Section 8.4.6), approximately 180 bytes are further added to the request (one ECDSA public key and signature in Base64 encoding), which can still fit in a single packet (for cookieless domain requests). Furthermore, the server has to perform one ECDSA signature verification. This overhead could be minimized, if the browser used the same Channel ID, not only for subdomains of the same domain, but also for domains belonging to the same entity. Although we do not elaborate on this idea here, this could be heuristically determined by the browser, based on which domains are involved in the execution of the same SISCA instance.

Finally, recall that a SISCA instance is executed only once per TLS connection and not on every HTTP request/response.

8.5 Related Work

In chapter 7 we reviewed existing techniques that try to enforce proper server authentication by enhancing the current CA trust model. Those solutions focus on addressing the issue of forged server certificates (and thus defeating MITM+certificate attackers), essentially not relying on client authentication at all. In the following we review work more directly related to our attack, MITM-SITB, as well as our proposed solution, SISCA.

The use of server impersonation for the compromise of the user's account by serving the attacker's script to the victim's browser was first introduced in [107]. In this attack, called *dynamic pharming*, the attacker exploits DNS rebinding vulnerabilities in browsers, by dynamically manipulating DNS records for the target server, in order to force the user's browser to connect either to the attacker (to inject his script) or to the legitimate server.

MITM-SITB is therefore very similar to dynamic pharming in that it leverages server impersonation to serve the script to the victim's browser. Dynamic pharming focuses on the attacker's ability to control the client's network traffic via DNS attacks, while in this paper we do not make such assumptions. Instead, MITM-SITB can leverage any form of MITM where the attacker controls the communication to the client (e.g., an attacker sitting on a backbone) and relies only on the behavior of the browser to re-establish a connection (with the legitimate server) once the attacker closes the connection within which he injected his script to the browser. Dynamic pharming can equally be used to successfully attack Channel ID-based solutions. Recently, the act of leveraging script injection via server impersonation against TLS client authentication was also discussed in [164].

We note that MITM-SITB (as well as dynamic pharming) differs from *Man-In-the-Browser* (MITB) [158]. The latter implies that the attacker is able to take full control of the browser by exploiting some vulnerability, or installing a malicious browser plugin. In MITM-SITB, the attacker runs normal JavaScript code within the target web origin and only within the boundaries established by the JavaScript execution environment. Therefore, no browser exploitation is required. Similarly, MITM-SITB is different from XSS [156]. In XSS the attacker is able to influence only parts of the served document by exploiting a script injection vulnerability, while in MITM-SITB he is able to impersonate the server and thus influence the entire HTTP response sent to the browser. Nevertheless, the end result of arbitrary client-side code execution within the target web origin is the same. SISCA does not prevent MITB or XSS and addressing these attacks, e.g., as in [220], is orthogonal to our work.

To prevent dynamic pharming, the locked same-origin policy (SOP) was proposed [107]. Weak locked SOP considers attackers with invalid certificates, while strong locked SOP also defends against attackers with valid, mis-issued certificates. Strong locked SOP refines the concept of origin by including the public key of the server and can also accommodate for multiple server keys. It is a form of key pinning, with the particularity that instead of rejecting TLS connections with not endorsed server public keys, strong locked SOP isolates web objects coming from connections with not endorsed server public keys in a separate security context (i.e., different origin). Strong locked SOP per se does not prevent a MITM attacker from mounting a conventional MITM attack in order to impersonate the user. A strong client authentication solution should be used in conjunction, as with SISCA.

Locked SOP does not resist MITM+key attacks, as SISCA does. Moreover, locked SOP is not able to secure cross-origin active content inclusion. The risks involved when a webpage imports active content, such as JavaScript, that can be intercepted and modified by an attacker are discussed in [100]. SISCA can secure cross-origin inclusions as long as the involved domains belong to the same entity and thus share the same SISCA keys.

The current Channel ID specification [13] was recently found to be vulnerable to *triple handshake attacks* [24], which affect TLS client authentication in general. A MITM attacker can exploit a protocol flaw during TLS session resumption in order to trick the legitimate server into believing that the attacker holds the private key that corresponds to the user browser's Channel ID. This allows the attacker to mount a conventional MITM attack in order to impersonate the user to the server. The mitigation proposed in [24] has already been implemented in the version of Chromium that we used in this work. SISCA assumes that Channel IDs work as expected, so eliminating triple handshake attacks is essential for its security. However, we note that addressing triple handshake attacks does not prevent MITM-SITB attacks.

Recent work has proposed leveraging Channel ID-based authentication to strengthen federated login [57] and Cloud authorization credentials [26], against MITM attacks and credential theft in general. However, such proposals fail to address MITM attacks, as they are susceptible to MITM-SITB, unless augmented with server invariance, as we propose in this paper with SISCA.

The concept of server invariance is based on *sender invariance* which was formally defined in [60]. SISCA is inspired by this notion, assuming that the server's authenticity cannot be established via server certificate verification and instead trying to enforce the weaker property of invariance, i.e., the browser always communicates with the same entity during the a browsing session.

8.6 Summary and Future Work

In this Part of the thesis we discussed the requirements to effectively preventing TLS MITM attacks in the context of web applications, when the attacker's goal is to impersonate the user to the legitimate server and gain access to the user's account and data. Striving to defeat this type of attack is important, especially given the recent revelations about government agencies (e.g., the NSA) mounting such attacks in order to perform mass surveillance against users of major internet services [63, 188].

We showed that strong client authentication alone, such as the recently proposed Channel ID-based authentication, cannot prevent such attacks. Instead, strong client authentication needs to be complemented with the concept of server invariance, which is a weaker and easier to enforce property than server authentication. Our solution, SISCA, shows that server invariance can be implemented with minimal additional cost on top of the proposed Channel ID-based approaches, and can be deployed incrementally, thus making it a scalable solution. Given its security benefits, we believe that SISCA can act as an additional, strong protection layer in conjunction with existing proposals that focus on amending today's server authentication issues, towards the effective prevention of TLS MITM attacks.

8.6.1 Future Work

Given the incidents involving CA compromises and issuance of rogue certificates, it is clear that robust and resilient server authentication remains an open problem, and this is why it has been an active area of research in the recent years and will likely continue to be so.

With SISCA, we showed how combining strong client authentication with server invariance, can prevent TLS MITM attackers from compromising users' online accounts. Nevertheless, as we described earlier, server invariance can be difficult to apply in some settings. In particular, in websites that are served by multiple servers and in websites that include content from multiple origins, especially when these origins are not owned by the same administrative domain. We thus argue that SISCA could benefit from further research in finding ways on how to more effectively apply the server invariance property in the aforementioned web application deployments.

Part III

Data Authenticity and Integrity in Web Applications

Chapter 9
Introduction

In Part II of the thesis we have focused on web server authentication, and in particular on how to prevent TLS Man-In-The-Middle (MITM) attacks, where the attacker's goal is to compromise the user's online account. To achieve this, the attacker first impersonates the server to the user, by intercepting the user's browser connection to the legitimate server and presenting an illegally obtained, yet valid TLS certificate for the domain of the legitimate server. After this step, the attacker is able to steal the user's credentials, e.g., his username and password, or the cookie that authenticates an existing session between the user and the server, and use them in order to impersonate the user to the server, and hijack the user's account.

We have shown how the above attack can be prevented by combining strong client authentication, e.g., based on TLS channel IDs, with the concept of server invariance in our proposal called SISCA. As we described in Chapter 8, SISCA still allows the attacker to successfully impersonate the server to the client, but prevents the attacker from compromising the user's account. Given this, it follows that SISCA does not prevent the attacker from presenting false information to the user. The attacker can actually attempt to precisely mimic the website's look and feel, in order to give the impression to the user that he is viewing the legitimate website and thus make the presented false information appear as legitimate, coming from the original server.

Moreover, even when the attacker is not able to mount a TLS MITM, i.e., server authentication is working as expected, he may still try to compromise the server itself. As we discussed in the introduction of this thesis, web applications typically have a large attack surface and are compromised frequently [4, 114, 219] via a variety of attack vectors and vulnerabilities (e.g., [212]). If a server compromise occurs then server authentication, no matter how strong and robust it is, it is of no help, as the attacker is in control of the server and can perform arbitrary actions. Besides being able to access all the data and server-side state of the application, he is, like in the case of SISCA, able to present false information to the user, by manipulating the contents of the webpage that is displayed in the user's browser. In such a case, the attacker is trying to cause harm, not by compromising the user's account, but rather pushing fake information to the user.

In this part of the thesis we focus on the ability of the attacker to affect the integrity of the data of a web application, once he manages to compromise the server side. We argue that the authenticity and integrity of the web application data is of equal, and sometimes even greater importance, than the confidentiality of it. The reason for this is, data accessed by web applications are often used in critical decision making processes, in a variety of sectors, such as healthcare, government, military, financial services and large corporate environments. Maliciously manipulated data can lead to wrong decisions being made, which in turn can lead to serious harm which, depending on the use case, might even prove to be fatal.

As an example, let us consider a medical web application on which physicians can access diagnostic data of their patients, who are being monitored either physically or remotely. If a physician is presented with manipulated information about the health status of a patient, he might incorrectly diagnose his condition which would lead to incorrect treatment with potentially harmful or even fatal effect on the patient. In order for this to be avoided, it must be ensured that the data the physician is accessing is authentic and integrity protected, even if the server is compromised.

The research community has long been aware of the importance on ensuring the integrity of data, even under server compromise. The fairly large body of work in the area of authenticated data structures (ADS) [49, 80, 122, 123, 130, 160, 241] is an indicator of this fact. An ADS is a data structure that provides integrity protection guarantees for the data that it stores. Most of this work pertains to data stored in databases or file systems. Nevertheless, as we discuss later on in this Part and to the best of our knowledge, none of the existing proposals for providing data authenticity and integrity is directly applicable to one of the most popular ways through which people access data online today, i.e., web applications. As part of this thesis we investigate this topic and attempt to overcome the challenges in order to provide data integrity protection for web applications.

After providing background information on Authenticated Data Structures in Chapter 10, we propose Verena in Chapter 11. To the best of our knowledge, Verena is the first web application platform that provides data authenticity and freshness, even when the attacker is in complete control of the server. Verena gives the developer the necessary notions and API in order to express the integrity policy of the web application, and then enforces the specified policy, end-to-end. Verena thus prevents a malicious server and potential colluding users from manipulating the data and query computation results that honest users view in their browsers, without being detected.

In order to evaluate our research in a realistic setting, we contacted one medical implant manufacturing company and obtained access to the web interface of their remote patient monitoring system. This, together with discussions with a cardiologist, allowed us to better understand the access control requirements and integrity protection policy of this application. We evaluate Verena on a basic implementation of the aforementioned web application and show that Verena can enforce the application's integrity policy with modest overhead.

Chapter 10
Background

In this chapter we provide short background information on authenticated data structures (ADS). Verena, our proposed web framework for providing end-to-end integrity guarantees in web applications, leverages authenticated data structures as its underlying integrity protection building block. In fact, as we discuss in the next Chapter of this thesis, Verena does not rely on a specific ADS, rather it can be easily adapted to support the functionality offered by the chosen underlying ADS. In particular, in our implementation of Verena we make use of one-dimension red-black binary Merkle hash trees with the ability to support projection queries, as well as aggregation queries based on the tree-based technique of [123]. Here we summarize how such trees work and refer the reader to the literature (e.g., [49, 80, 122, 123, 130, 135, 160, 206, 241, 242]) for a detailed analysis.

Authenticated data structures are data structures, whose operations can be offloaded by the owner of the data structure and the data its, to an untrusted entity, called the prover. The prover produces compact proofs for the operations it executes on behalf of the owner, which allows the owner, as well as other verifiers, to verify the results of these operations for their authenticity. In other words, ADSes allow their owner to outsource their maintenance and operation to an untrusted server, with added integrity protection guarantees. A typical example use case of ADSes is outsourcing a database to an untrusted cloud, in which the owner executes insert, update and delete operations, while other clients retrieve data using database queries in an authentic way.

Depending on the nature of the authenticated data structure, different types of read and write operations are supported, with varying performance guarantees. Tree-based data structures, are commonly used as ADSes. Since database indexes are based on trees, tree-based ADSes can be used to support authenticated database queries(e.g., [123, 241].

Merkle hash trees [21, 133] constitute the most basic example of a tree-based ADS. Briefly, in a Merkle hash tree data is stored in the leaf node while each internal node stores the cryptographic hash of the content (data, or hash value) stored in its children nodes. The hash at the root node of the tree is called the root hash and, in essence, it constitutes the cryptographically-secure fingerprint of the entire tree and its stored data.

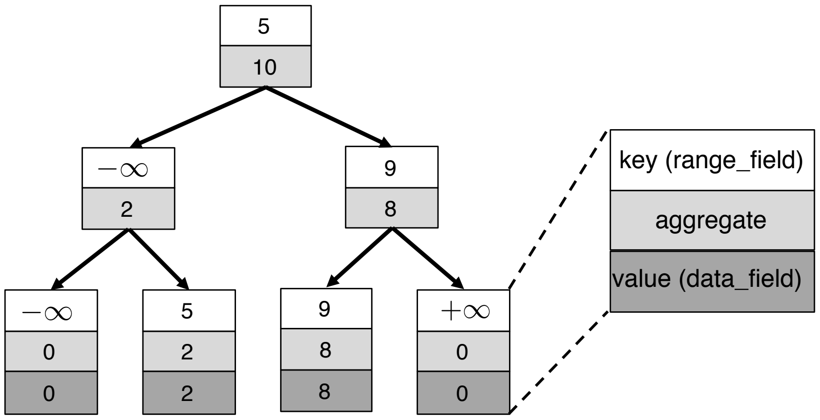


Figure 10.1: Example tree-based ADS supporting aggregations. Node hashes realizing the Merkle hash tree, as well as node color, realizing the red-black tree balancing mechanism used in our implementation, are not shown.

Consider a database table consisting of two fields (i.e., columns), namely a range field and an aggregation field, i.e., a field in which stored data is used in aggregation queries. In SQL notation, a user runs queries of the form “SELECT sum(*aggr. field*) FROM table WHERE $x \leq$ *range field* $\leq y$”.

Figure 10.1 shows an example of such a tree, sorted (i.e., keyed) by the range field. Note that the red-black property is only used for keeping the tree balanced, and hence Figure 10.1 omits the color of the nodes. Also not shown in Figure 10.1 are the hashes of each node, which constitute the Merkle hash tree. The client, who is the owner of the data stored in the tree, keeps the root hash of the Merkle hash tree, and the untrusted server keeps the entire tree.

Data is stored on leaf nodes. Each node stores a key and an aggregate value, while leaf nodes further store the data values. Given our aforementioned example, the keys are the range field values and data stored on leaf nodes is the data of the aggregation field. For leaf nodes, the aggregate value is equal to the data value itself. For each internal node, the aggregate value is the aggregation over the aggregate values of its children. The tree in Figure 10.1 features sum as the aggregation operation.

Assume that the client issues the query “SELECT sum(*aggr. field*) FROM table WHERE $2 \leq$ *range field* ≤ 5”. The server responds with the sum of interest, 2 in this case, together with a proof that the sum is correct. We explain briefly what the proof consists of and refer the reader to [123] for more details. The proof consists of two parts. The first part is, for each edge of the interval, the server provides two nodes in the tree whose range fields include the edge of the interval, together with their Merkle hash paths to the root. The client checks these Merkle paths against the Merkle hash root it stores and ensures that the edge of the interval is inside this interval. Note that this is always possible because the keys $\pm\infty$ (containing dummy data values) are also in the tree. The second part of the proof is a minimal covering set for the range $[2,5]$ together with a Merkle hash path up to the root. In our example, this minimal covering set consists of the internal node with key $-\infty$ and aggregate value 2. In general, the covering set is a logarithmic number of nodes.

The client then checks that these nodes cover the range of interest entirely and verifies their hashes and Markle hash paths against the root hash that the client stores. By the properties of Merkle hash trees, if the verification is successful, the server provided the correct aggregate value. Overall, the client performs $O(\log n)$ work per value returned where n is the number of nodes in the tree. A similar computation happens when inserting, updating and deleting data, with some additional details.

Note that the server does not have precomputed the aggregate value for each range. The ADS tree has one data entry (leaf node) per range field value and there is a quadratic number of possible ranges. Clients can query arbitrary ranges, and these ranges could contain a large number of nodes. The server transforms these ranges into a set of subranges, and the client then aggregates the aggregate values for each range. The maximum number of subranges is logarithmic in the number of nodes in the tree. Hence, the client does little aggregation work because it aggregates only a logarithmic number of values.

Chapter 11
Verena: End-to-End Integrity Protection for Web Applications

Web applications store a wide range of data including sensitive personal, medical and financial information, as well as system control and operational data. Users and companies rely on these servers to protect the integrity of their data and to answer queries correctly. Unfortunately, web application servers are compromised frequently [4, 114, 219] via a variety of attack vectors and vulnerabilities (e.g., [212]), thereby enabling an attacker to tamper with data or computation results displayed in a webpage, and consequently violating their integrity.

The integrity of webpage content is especially important in applications in which displayed data affects decision making. This is well exemplified by medical web platforms where patient diagnostic data is stored on web servers and remotely accessed by physicians. Modification of this data might result in miss-diagnosis, lead to incorrect treatment and even death. A recent study estimates that millions of people are miss-diagnosed every year in the US with a half of these cases potentially causing severe harm [195]. Another study estimates that miss-diagnoses causes 40,000 deaths annually [235]. Some of the main reasons for miss-diagnoses were related to failure by the patients to provide accurate medical history, and errors made by a physician in interpreting test results [194]. If web applications with patient and test result data are corrupted, treatment decisions will therefore be made based on incorrect data, likely resulting in substantial harm. In Section 11.3.1, we discuss a concrete medical web application used to monitor patients with implanted cardiac devices, where a web server compromise can lead to serious patient harm.

In addition to physical safety, webpage integrity is important for basic security properties such as confidentiality against active attackers, for example, by providing integrity protection to data structures defining access control.

A web server which is not compromised protects end-to-end integrity in a few ways. Many web applications involve multiple users and therefore enforce *access control* policies (e.g., a particular patient's data may be manipulated only by his physician). Furthermore, the web server ensures that data requests and queries submitted by clients are executed *correctly* on data that is *complete* and *up-to-date* (i.e., fresh). An attacker who compromises the web server could therefore violate some or all of these properties.

In this work, we propose Verena, the first web framework that provides end-to-end integrity for web applications, enforcing the properties above. Using Verena, the application developer specifies an integrity policy and the user's browser checks that a webpage received from a web server satisfies this policy, even when the server is fully compromised by an attacker. Verena checks the integrity of code, data, and query computation results within a webpage by ensuring that these results are complete, correct, and up-to-date.

Verifying query results efficiently in the web setting is challenging. While much progress has been made in generic tools for verifiable computation [23, 162, 229], using these tools for database queries and web server execution remains far too slow. Instead, work on *authenticated data structures* (ADS) [49, 80, 122, 123, 130, 160, 241] provides better performance by targeting a more specific, yet still wide class of functionality. These tools enable efficient verification without downloading data on the client and re-executing the computation. However, such tools are far from providing a sufficient system for web applications, as work on ADS assumes that a single client owns all the data and this client has persistent state to store some hashes. Web applications are inherently *multi-user* and *stateless*. Different users can change different portions of data and a query computation can span data modified by multiple users.

The first challenge for Verena is determining an API for developers that captures the desired query integrity properties, such as correctness, completeness and freshness, at the same time with multi-user access control. To address this issue, within Verena's API, we introduce the notion of query *trust contexts* (TC) coupled with *integrity query prototypes* (IQPs). A trust context refers to the group of users who are allowed to affect some query result, e.g., by inserting, modifying or deleting data used in a query. An IQP is a declared query pattern associated with one or more trust contexts. Each query runs within a specified trust context. Verena prevents a malicious server or a user outside of the trust context from affecting the results of this query. Queries may also span a set of trust contexts not known a priori. A mechanism called *the completeness chain* ensures that the returned result is complete, i.e., *all* the results of *all* the relevant trust contexts were included. The integrity policy is hence associated with queries and not with the data. Nevertheless, the policy implicitly carries over to data because data is accessed through queries.

The second challenge is verifying query results in a multi-user setting. To address this challenge, Verena builds on ADS work [49, 80, 122, 123, 130, 160, 241], and maintains a forest of ADS trees, by automatically mapping trust contexts to ADSes. To ensure completeness on queries spanning multiple trust contexts, as specified by completeness chains, Verena logically nests trees within other trees. Currently, Verena implements an ADS that can verify range and equality queries as well as aggregations, such as sum, count, and average. It is simply a matter of substituting the underlying ADS, in order to extend Verena to support a wider range of queries.

The third challenge, also brought by the multi-user and web setting, is a known impossibility result. Namely, when there are no assumptions on the trustworthiness of the server and the connectivity of clients, one cannot prevent fork attacks [124, 131] and hence cannot guarantee freshness. To provide freshness, one must use some trust on the server side. Verena manages to use a small trusted base, in particular a hash server that runs less than 650 lines of code. The hash server may also be compromised, as long as it does not collude with the main server. The hash server stores a small amount of information (mostly hashes and version numbers), based on which Verena constructs freshness for the entire database in an efficient way. The hash server also addresses the problem of web clients being stateless and not always online.

We implemented Verena on top of the Meteor framework [134] and evaluated it on a remote patient monitoring application, as well as two other existing applications. Our evaluation results show that Verena incurs a modest overhead in terms of latency and throughput. Our measurements also demonstrate the simplicity of the hash server, compared to the main server. In particular, the hash server achieves significantly higher throughput than the main server.

Contributions. We focus on the problem of providing end-to-end data authenticity and integrity protection in web applications, under full server compromise, and make the following contributions.

- We propose Verena, the first framework that provides end-to-end integrity guarantees for web applications, by enforcing the application's integrity policy and ensuring that data requests and queries submitted by the clients are executed correctly on data that is complete and fresh.

- As part of Verena, we introduce an API which developers can use in order to express the integrity policy of the web application, which will be enforced by Verena.
- We implement a prototype of Verena and use it to provide data integrity on an example medical web application. We then evaluate Verena on this application and demonstrate its practicality by showing that it incurs modest overhead.

The rest of this chapter is organized as follows. Section 11.1 details our system and threat model. We introduce Verena and present its high-level architecture in Section 11.2. In Section 11.3 we introduce our running example of a medical web application and discuss Verena's basic concepts and API. In Section 11.4 we discuss the mechanisms that Verena uses in order to enforce the integrity policy that the developer specified. The hash server and the way it operates is described in Section 11.5. In Section 11.6 we describe the communication protocol and query processing steps in Verena, while in Section 11.7 we provide an informal argument on the security of our system. We discuss limitations and extensions in Section 11.8. In Section 11.9 we present our prototype implementation of Verena, based on which we evaluate our proposal in Section 11.10. Section 11.11 reviews related work. Finally, Section 11.12 concludes Part III and sets future research directions.

11.1 Model

11.1.1 System Model

We consider a typical web application scenario, where clients access a web server through web browsers. The *clients* could be browsers, operated by human users, or any device capable of communicating with the web server over the network. The *main server*, sometimes simply referred to as *server*, is a typical web server consisting of a web application front-end and a database server.

Our setup further consists of the following parties. A *hash server*, an *identity provider* (IDP), and the *developer* who creates and maintains the web application code. The hash server and IDP can be co-located on the same machine. They each have a public-key pair and their public key is hardcoded in Verena applications. In Section 11.2 we describe the role of the different parties in the Verena architecture.

Moreover, for describing Verena's API in Section 11.3, we use a No-SQL API, which is typical in modern web applications. For consistency, we use a syntax and terminology similar to MongoDB [137], which we simplify for brevity. This is also compatible with the Meteor framework [134], which we use to implement our Verena prototype (described in Section 11.9). Nevertheless, Verena's API could be easily cast in a variety of other database syntaxes (both SQL and No-SQL).

The web application's database consists of collections (equivalent of tables in SQL), each having a set of documents (equivalents of rows in SQL), and each document has a set of fields (which are similar to columns in SQL). A developer can issue queries to this database from the web application: "insert" (to insert documents), "update" (to update documents), "remove" (to delete documents), "find" (to read document data) and "aggregate" (to compute sum, average and other aggregate functions). The find and aggregate operations can read data based on filters, also called selectors, on certain fields using range or equality. Queries are defined using a JavaScript-like syntax. For example, "patients.find({patientID:2})" fetches all documents from the collection "patients" whose "patientID" field equals 2.

11.1.2 Threat Model

Verena considers a strong attacker who can corrupt the main server arbitrarily. This means that an attacker can modify the data in the database and modify query or computation results returned by the server. There are numerous ways in which an attacker could modify query results. For example, a malicious server can return *partial* results to a range query, it can return old data items, it can compute aggregates incorrectly or on partial or old data. Furthermore, the server can create fake user accounts or collude with certain users.

This strong threat model addresses powerful attackers in the following use cases. A web application server runs in a cloud and a malicious cloud employee attempts to manipulate unauthorized information. Alternatively, an attacker hacks into the web application server through vulnerability exploitation and even obtains root access to the web and database servers, so he can change the server's behavior.

An attacker can also corrupt the hash server, but importantly, we assume that an attacker can corrupt *at most one* of the main and hash server. In other words, we assume that at least one of the hash server and main server behave correctly. For example, these two servers could be hosted on different clouds such that the employee of one cloud does not have access to the second cloud. Alternatively, the hash server, which we show to be very lightweight compared to the main server, could be hosted in-house,

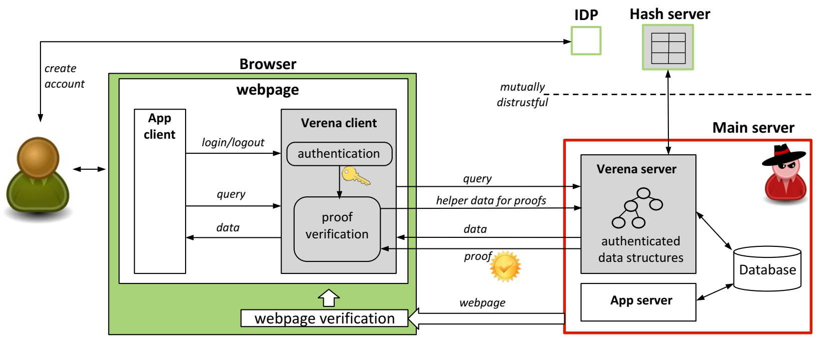


Figure 11.1: Verena system overview. Grey-shaded components are modules introduced by Verena on top of a basic client-side architecture setup.

while the resource-intensive main server could be outsourced to a cloud provider. We also stress that, given that the hash server runs a very small code base, and answers to a very narrow interface, it will be significantly less likely to be compromised by a remote attacker.

The same threat model applies to the IDP server. We use the IDP for the task of certifying each pair of username and public key. Verena requires that only one of the IDP and main server to behave correctly. Hence, the IDP and the hash server can be co-located, as depicted in Figure 11.1, where the mutually distrustful servers are separated by a dashed line.

Clients are also not fully trusted. They may attempt to bypass their write permissions by modifying data which they are not allowed to change. They might even collude with either the main server or the hash server (but not with both of them at the same time). Nevertheless, clients are allowed to arbitrarily manipulate the data they legitimately have access to. If the main server colludes with a client, the server cannot affect the integrity of data owned by other clients which was not shared with the corrupted client.

Finally, we assume that the developer wrote the web application faithfully and followed Verena's API to specify an integrity policy. In contrast, the service provider and server operator are not trusted at all (and these fall in the main server trust model above).

11.2 Architecture

Figure 11.1 illustrates Verena's architecture, which we describe throughout this section.

11.2.1 Basic Setup

To lay out the foundations for Verena's security mechanisms, Verena starts with the following base setup. First, Verena is a client-side web platform, a popular trend in recent years [27, 134]. These platforms provide advantages not only for functionality and ease of development, but also for security.

The dynamic webpage (with personalized content) is assembled on the client side from static code and data coming from the server. Previous work [169] has shown how to check in a client's browser whether the webpage code (such as HTML, JavaScript and CSS) has not been tampered with by an attacker at the server. Since the code is static, such a check essentially verifies the code against a signature from the developer. This check runs in a browser extension. Verena incorporates this mechanism and the browser extension as well. From now on, we consider that the webpage code passed this integrity check and we refer the reader to [169] for more details.

Second, a standard requirement in multi-user systems providing cryptographic guarantees is an *identity provider* (IDP), i.e., an entity that certifies the public key of each user. For example, it can be similar to OpenID or Keybase [108], or could be hosted at the same place as the hash server. Without such an IDP, an attacker at the server may serve an incorrect public key to a user. For example, if user A wants to grant access to user B, user A requests the public key of user B from the server who replies with the attacker's public key such that the attacker obtains access. The IDP is involved minimally, when a user creates an account. At that point, it signs a pair of username and public key for each user creating an account. Although we do not discuss key revocation in this work, enabling revocation in Verena would require the involvement of the IDP as well.

11.2.2 Verena Components

Now that we laid out the basic setup, we describe the mechanisms that Verena provides to prevent the server from corrupting data and query results. At a high level, the application developer, using the *Verena API*, specifies the integrity protection requirements (integrity policy) of the application. This allows Verena to derive the access rights of each user for each data item or query. Based on this API, the server accompanies any integrity-protected query operation with a proof that it follows this policy and the client can verify this result. Also, whenever the client sends a query to the server, the client accompanies the query with helper data for constructing the proofs, if needed.

More concretely, as shown in Figure 11.1, on the client side, a webpage consists of two parts, namely the application's code written by the developer on top of the Verena framework, and the *Verena client*. When a user logs in, the Verena client performs authentication to derive the user's key from his password. If passwords are deemed unsafe, one can use other available secret derivation mechanisms [28].

In typical client-side web frameworks, the app client issues database queries. These queries are sent to the web server, which sends them to the database. In Verena, all such queries pass through the Verena client. Verena then determines if it is a query that must be integrity protected. If so, the Verena client provides helper data (such as challenges) to the server to be used in proofs. When the server returns the results, the server also provides a proof of correctness for these results that the Verena client checks before returning to the app client.

The main server consists of the *Verena server* and the regular app server. The app server, also written by the developer, performs operations that are not integrity-sensitive and do not require verification. All server-side operations that require verification pass through the Verena server.

The Verena server carries the difficult task of constructing proofs of correctness for query results that are efficiently verifiable by the Verena client. Verena builds upon work on authenticated data structures (ADS) and in particular tree-based ADSes [49, 80, 122, 123, 130, 160, 241], which we briefly introduced in Chapter 10. ADSes enable efficient verification without downloading data on the client and re-executing the computation. Verena enables these ADSes to be used in a multi-user and stateless setting. In this manner, Verena can verify a wide range of common queries, but not any general query. Table 11.2 lists the read queries that are currently supported by Verena. Moreover, Section 11.8 discusses how Verena can be extended to support a broader range of query types.

Since applications have different access policies, Verena needs to translate these policies into ADSes. Our new API, based on the notion of query *trust contexts* (TC), *integrity query prototypes* (IQP) and *completeness chains*, presented in Section 11.3, captures an application's policy.

Moreover, due to the multi-user setting in Verena, different users are allowed to modify different portions of the data stored at the server. Thus, the Verena server has to maintain different ADSes for chunks of data that are modifiable by different sets of users. Respectively, Verena clients must ensure that, for each integrity-protected database operation, the server presents proofs for all relevant ADSes that data was modified only by legitimate users. To address this, Verena maintains a forest of ADS trees, by automatically mapping the developer's Verena API calls into the appropriate ADSes.

Since multiple users may be able to change the same data item, users do not know what was the hash of the last change. Additionally, in web applications, not all users are online at the same time and cannot notify each other of their changes. To make the problem worse, since the web setting is stateless, whenever a user logs off, any state he stored in his browser is typically lost. Moreover, the user should be allowed to login from a new browser where there is no state. This means that, even though ADSes help ensure integrity of some snapshot of the data, the server can still provide stale data. In fact, Mazières and Sasha [131] prove that, without any trust at the server or connectivity assumptions between the users, one cannot guarantee data freshness.

To address this problem, Verena uses a *hash server*. The hash server a simple server whose main task is to serve the hash, version and last modifier for a given entry. As long as the hash server does not collude with the main server, Verena's integrity guarantees hold. To check the correctness and freshness of query results, a tempting approach is to store the entire ADS trees at the hash server. We show in Section 11.5 that we can avoid this approach, and maintain the task of the hash server simple, namely the hash server stores one entry per tree, corresponding to the root of the tree. As a result, the hash server is easier to secure, since it runs a small code base, answers to a narrow interface, and is lightly utilized. The hash server could be collocated with the IDP server because Verena assumes the same trust model for these two servers.

11.3 Integrity Policy API

In this section, we describe the main concepts behind Verena's API for expressing an integrity policy. In Verena we are concerned only with write access control. As discussed in Section 11.8, systems like Mylar [169] can be used for expressing and enforcing cryptographically read access control.

In order to illustrate Verena's concepts and API, we use consistently the following running example of a medical web application.

11.3.1 Running Example: Remote Monitoring Medical Application

Our running example is a remote patient monitoring system used to connect cardiac device patients with their physicians. Such systems are deployed by a number of medical implant manufacturers such as [25, 29, 35, 132].

In order to evaluate Verena on such an application, we contacted one of the implant manufacturing companies and obtained access to the web interface of their remote monitoring system. This provided us with a better understanding of the type of webpages that these systems expose, access control that they implement and the type of data that they expose to the physicians. We then discussed with a cardiologist to gain a better understanding of the integrity policy of this application.

Modern implantable cardiac devices, such as cardioverter defibrillators (ICDs), cardiac monitors and pacemakers, monitor the patient's cardiac activity and take certain actions. In particular, implants measure data such as therapy delivered, heart rate, EKG (electrocardiogram) data, and status of implant and leads. To facilitate access to this data, implants communicate remotely with their clinics. This is supported by wireless telemetry devices which, when in the proximity of the patient, query the implant and then communicate the data further to the clinic server.

The server then exposes a web interface to physicians, through which they can access patient profiles, status of implants and measurements. Besides viewing this data, physicians ask the web server for certain *aggregate computation* such as average heart rate, number of heart beats per day (e.g., observed over a three-day period) and number of sinus pauses (i.e., skipped heart beats/asystoles). Physicians can change a patient's therapy by reprogramming the implant in the clinic, using short-range inductive coil telemetry.

The information a physician receives from the web server influences the decisions the physician makes for a patient and is thus integrity critical. Although practices among physicians can vary and there might be other inputs that influence the therapy decisions, we were told that incorrect modifications of these values or aggregates will likely lead to a change in the delivered therapy and can cause serious patient harm. Moreover, the status of the implant and leads connecting the implant to the heart is integrity critical. If these are thought of malfunctioning, this might trigger their replacement which requires surgery.

The main subjects in the system that we had access to, include the administrators of the clinic, physicians and the medical implants. Each implant can be seen as a user with write access to the corresponding patient's implant status and measurement data. Main objects are patient related information which are entered by physicians, as well as measurement and implant status data which are entered directly into the system by the implants.

collection	**fields in a document**
patients	(groupID, patientID, patient_name, profile)
patient_groups	(groupID, group_name)
patient_measurements	(recordID, patientID, heart_rate, timestamp)

Table 11.1: Database collections used in the running example of a remote monitoring medical application.

Instantiation. To illustrate Verena's API, we give simplified examples of this application. Table 11.1 shows the database collections that are relevant for our example. Patients are organized into four groups based on their cardiac disease. Each patient is present in only one such group. These groups also represent the unit of write access control. Physicians are granted write access to one or more of these groups, and they can modify only patient profiles in those groups.

The collection patient_measurements contains measurement data originating from a patient's medical device and can be modified only by the patient's device.

11.3.2 Trust Contexts

Trust contexts are the units of write-access control in Verena. A trust context, identified by a *unique name*, consists of a set of users, called *members*. We also refer to this set of users as the trust context membership list or access control list (ACL).

Each query whose results are integrity critical runs in a particular trust context. This means that only the members of that trust context could have affected the result of the query.

The user who creates a trust context is the *owner*. The owner of a trust context can add other members to the trust context ACL or remove them from it. Currently, only the owner of the trust context can manage its members, but delegating this to other users is straightforward. We discuss how Verena maintains and verifies the membership of trust contexts in Section 11.4.4.

Returning to our running example and its protection requirements, the developer should define one trust context per disease group (whose name can be "groupID") containing the physicians allowed to modify the corresponding patient profiles. The contents of patient_groups can be changed by members of an "admins" group so the developer declares a trust context for "admins", as well. Furthermore, the data in collection patient_measurements as well as the query results on this data can be modified only by the patient's device. Hence, we also have a trust context per patientID.

11.3.3 Integrity Query Prototypes

In Verena, the developer specifies the desired integrity policy via a set of integrity query prototypes (IQPs) with associated trust contexts. The IQPs are query patterns which specify that a certain set of read queries run in a certain trust context. Only members of the trust context may affect the result of those queries. The integrity specification is therefore associated with read queries and not with data. Nevertheless, the policy implicitly carries over to data, because data is accessed through queries. Moreover, the IQPs tell Verena what computation will run on the data so that Verena prepares data structures for verifying such computation. We now show the syntax of an IQP and explain each element in it:

iqp = collection.**IQP** ({
  trustContext: unique_name or tc_field,
  eq-range: [rf_1, rf_2, ...],
  ops: {o_1: [f_1, ...], o_2: [f_1', ...], ...}})

- "iqp" is an IQP handle.
- "collection" is the collection on which a query with this pattern runs.
- "trustContext" specifies the trust context. The trust context can be a name, such as "admins" or can be the name of a field in this collection, such as groupID in the patients collection. In the first case, there is one fixed trust context for all documents in this collection.

In the latter case, there can be different trust contexts for different documents. For example, if “patients” contains documents (groupID: “A”, patientID: “10”, ...) and (groupID: “B”, patientID: “11”, ...), and an IQP specifies the “trustContext: groupID”, the trust context for the first document is “A” and for the second document is “B”.

Each trust context must have a unique name. For example, if both patientID and groupID are trust contexts for some IQPs and can both have a value of 2, the developer should choose trust context names of the form “patient 2” and “group 2”, in order to differentiate them. If it is desirable for patientID to remain an integer, the developer could include another field in the document, which will serve as the trust context field, e.g., “group_tc”. In the rest of this chapter, we assume that the trust contexts are the IDs and they are unique.

- “eq-range” specifies that the queries corresponding to this query pattern filter documents by range on the tuple $(rf_1, rf_2, \ldots)$. A set of filter possibilities fit in this pattern. For example, if the IQP for “patient_measurements” contains “eq-range: (patientID, timestamp)”, a query could have an equality match by patientID and a range match on timestamp, or there can be equality on both fields. Our current implementation supports only one range filter, namely the last declared field in the tuple $(rf_1, rf_2, \ldots)$, with the rest of the fields being used as equality filters. However, Verena can be extended to support more complex filters (e.g., multidimensional range queries and text search queries) by simply using ADSes that support such operations [160].
- “ops” indicates the projections and aggregations performed and on what fields. The operations supported are listed in Table 11.2.

Verena will protect the integrity of all fields specified in an IQP, namely the fields projected, aggregated, in eq-range, or in trustContext – these fields can be modified only by members of the corresponding trust context. We call these fields the *protected fields* of an IQP.

Let us walk through an example. In the medical application, a physician can fetch the recorded heart rates of a patient over a period of time to visualize how the heart rate fluctuates in that time period. Additionally, a physician can view the average heart rate over a time period.

The first read operation is a projection on the heart_rate field, and the second is an average computation on the same field. The trust context in both operations is designated by the patientID field, i.e., only the patient’s implant is allowed to provide these measurements. In the medical application, this entity is represented as a user with patientID. Moreover, the

Operation	Explanation
project: $[f_1, \dots]$	projects the fields f_1, …from the document
count	returns the number of documents
sum: f	returns the sum of the values in the field f
min/max: f	returns the minimum/maximum value over the data in field f
avg: f	returns the average of the values in the field f
sum_F: $[f_1, \dots]$	a more generic aggregate: returns the sum of a general function F whose inputs are $[f_1, \dots]$

Table 11.2: Operations supported in read queries.

operations use the timestamp field as a range selector. Consequently, to integrity protect these operations we can define the following IQP:

```
iqp_measurements = patient_measurements.IQP ({
   trustContext: patientID,
   eq-range: timestamp,
   ops: {project: [recordID, heart_rate], avg: [heart_rate]}})
```

11.3.4 Queries API

Once the developer specifies the necessary IQPs, which reflect the application's integrity specification, he can express and issue queries *in the same way* as in a system without Verena, by invoking "find" and "aggregate" on the corresponding IQP handlers This minimizes the amount of effort needed by the developer to enable Verena in existing web applications.

An example query for listing the average heart rate of patientID 121 over a period of one month is:

Write Queries	Explanation
insert(d)	inserts a document d
update(id, d)	updates the document with identifier id with data from the document d
remove(id)	deletes the document with identifier id

Table 11.3: Write operations in Verena.

```
iqp_measurements.find ({
   patientID: 121,
   timestamp: {“$gte": new Date(“2016-03-01"),
                “$lte": new Date(“2016-04-01")}})
```

A read query must be a subset of the queries described by the corresponding IQP. Moreover, if the trustContext of this IQP is a field, the query must specify its concrete value (for example, “patientID: 121”).

The developer does not have to specify IQPs for write queries, and simply invokes “insert”, “remove”, or “update” operations on the desired collection. The access control for write queries is derived from read queries. For write queries, Verena checks against *all* declared IQPs whether the current user is allowed to perform them. We elaborate on these checks in Section 11.4.3.

Supported Functionality. The read queries supported by Verena are those that can be expressed using an IQP. The write queries supported are in Table 11.3: insert, delete, update. Verena currently supports update and delete queries only by id, but extending to eq-range style filters is straightforward.

11.3.5 Querying Across Trust Contexts

So far, each read query specifies one trust context in which it runs. We now discuss how Verena supports queries spanning multiple trust contexts.

In the medical application example, recall that patient profiles are categorized in groups according to their disease and physicians may only modify profiles within certain groups. The following IQP enables fetching the *complete list* of patients within a group:

```
iqp = patients.IQP ({trustContext: groupID,
                    ops: {project:[all]}})
```

and the following IQP enables fetching all groupIDs from “patient_groups” using the “admins” trust context:

```
iqp_groups = patient_groups.IQP ({ trustContext: “admins”,
                                   ops: {project: [all]}})
```

In our running example, a physician may also view the list of all patient profiles from *all* patient groups. Clearly, this query spans multiple trust contexts. In a non-Verena system, the developer can simply run a read query fetching all entries in “patients” collection. However, if the server is compromised, the list of patients returned can be incomplete. To ensure completeness using Verena, the developer would need to do more work. He should fetch all the groups using “iqp_groups”, loop over the groups returned, fetch all patients in each group using “iqp” and merge the results. This results in a complete set of patients, but requires more work from the developer.

To make the work of the developer easier, we extend slightly Verena’s API with a mechanism called *completeness chain*. This mechanism essentially does the above work automatically for the developer. The completeness chain retrieves the involved trust contexts of a query by querying a different IQP and trust context, called the *root trust context*, which *endorses* the relevant trust contexts. In other words, the root trust context protects the list of trust context names of the query we want to execute, and thus we leverage it to establish completeness for that query. The developer simply runs the query:

```
iqp.find ({ groupID: iqp_groups.find({},{groupID:1}) },
          { ... })
```

The inner query projects the “groupID” fields from all documents in “patient_groups”. Section 11.4.2 and Figure 11.3 describe how Verena implements the completeness chain mechanism.

Alternative. It is worth mentioning an interesting alternative to completeness chains, which demonstrates the expressivity of Verena’s trust contexts. The developer can specify a new trust context “all_physicians”, which contains the set of all physicians, and an IQP “iqp_all_patients” that fetches all patient profiles (across all patient groups) in the trust context of “all_physicians”. Then, the developer can directly fetch all patient profiles by running “find” on “iqp_all_patients”. In this way, any non-physician cannot affect the completeness or integrity of the patient list. However, a physician who is not authorized to modify a certain group can now affect the completeness and integrity of patient profiles in that group. As a result, the integrity guarantee provided by the above IQPs is weaker than with completeness chains, and not sufficient for this application.

However, for applications with different access control requirements or different threat models, a developer might find this alternative sufficient. In this case, verification of aggregates is faster than with the completeness chain. Due to the layout of Verena’s data structures described in Section 11.4, the Verena client checks one aggregate value overall instead of one aggregate value per trust context with the completeness chain.

11.3.6 Deriving Trust Contexts From User Input

In some applications, the trust context for running certain queries is derived from user input. This requires special care from the developer and the user. Such a situation arises in applications where *anyone* can create units of data and give write access to others. For example, in a chat application, anyone can create a room and invite certain users to those rooms. Only the invited users may modify the contents of a chat room. This situation does not occur in the medical application because access control is rooted in a fixed entity, namely the “admins” trust context, which endorses and manages access to the trust contexts of patient groups.

In the chat application, a natural trust context for the messages in each room is the room name. A user, say Alice, reads the list of room names and clicks on the room she wants to visit. She expects the messages in the room to come from authorized users, and makes decisions based on them. However, an attacker can also create a room with the same name or a syntactically similar name (“business” vs. “busines”) tricking Alice into clicking on the attacker’s room. The contents of the attacker’s room are certified by the attacker, so Verena does not trigger an integrity violation.

Hence, in such cases, the developer must display *unambiguous names* to users. In order to do this, the developer can choose human-friendly names for trust contexts (e.g., the name of a room, as defined by a user) and then, display directly the trust context names in a prominent way to the user. Our hash server prevents two trust contexts from having the same name, and one can also expand this protection to prevent two trust contexts from having syntactically similar names. Moreover, the developer can display the owner of a trust context. Depending on the use case, the developer can display both the trust context name and its owner, or either of the two, in order to help the user verify the authenticity of the displayed data. The user needs to perform this check, for example in order to verify he is entering the intended room. This requirement is similar to phishing prevention where the user needs to check the URL he is visiting.

11.3.7 Integrity Guarantees

The guarantee Verena gives to a developer, given the assumptions in the threat model (Section 11.1), is, informally:

> If Verena does not detect a corruption, the result of a read query (find or aggregate) that corresponds to an IQP with a trust context `tc` reflects a correct computation on the complete and up-to-date data (according to linearizability semantics), as long as all clients running on behalf of the members of `tc` (or all involved trust contexts in the case of a completeness chain) follow Verena's protocol.

In particular, the query result could not have been changed by a malicious server or any user outside of the relevant trust contexts. Moreover, a data item is "up-to-date", or fresh, if it reflects the contents of the latest committed write as in linearizability semantics. In particular, the server cannot perform fork attacks [124, 131] because every client can always get the latest committed write of any protected data.

The resulting guarantee to the user is:

> The webpage consists of: (1) the authentic developer's code, (2) correct and "up-to-date" information (data or query computation results) generated only by authorized users.

Verena does not guarantee availability of the server. In other words, an attacker that manages to compromise a Verena-protected server can choose to not respond to incoming client requests and thus perform a denial of service attack.

11.4 Integrity Protection Mechanism

We now describe how Verena enforces the integrity policy that was specified by the developer, through Verena's API.

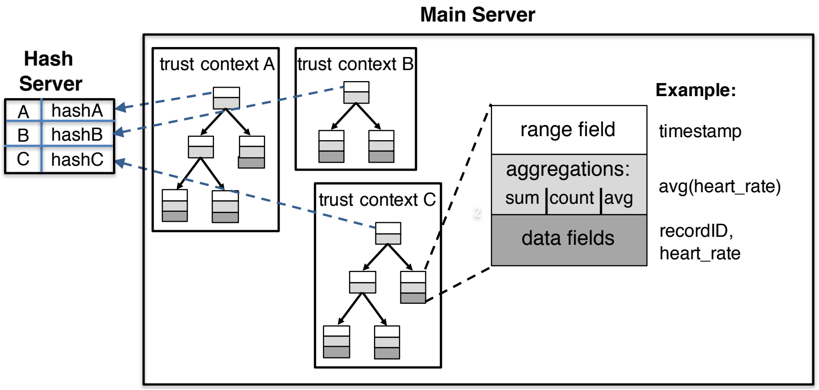


Figure 11.2: Forest of ADS trees. Verena maintains a forest of ADS trees, in order to protect the data associated with different trust contexts. The trees are stored on the main server while the hash server stores the root hash of each tree.

11.4.1 ADS Forest

Verena leverages authenticated data structures (ADS) [49, 80, 122, 123, 130, 160, 241] as its underlying integrity protection building block. The ADS we use [123] consists of a search tree sorted by the eq-range field(s) and combined with Merkle hashing. We refer the reader to Chapter 10, where we provide needed background on ADSes.

Based on the IQPs declared by the developer and the write operations that are issued throughout the application's lifetime, Verena creates and maintains a forest of ADSes, as illustrated in Figure 11.2. For each IQP, Verena creates one ADS per trust context that is used in queries of that IQP.

For example, consider the IQP we discussed before:

```
iqp_measurements = patient_measurements.IQP ({
   trustContext: patientID,
   eq-range: timestamp,
   ops: {project: [recordID, heart_rate], avg: [heart_rate]}})
```

Based on this IQP, every patientID constitutes a trust context, and Verena will maintain one different ADS for every value of patientID, in order to protect the data and aggregation operations specified by the IQP (in this case, the projection of recordID and heart_rate, and the average calculation on heart_rate). Chapter 10 explains on a high level how the ADS organizes and stores the protected data.

As shown in Figure 11.2, the forest of trees is stored at the main server. The hash sever stores only the Merkle hash roots (one entry per tree, containing the root hash and additional necessary information, as described in Section 11.5).

11.4.2 Completeness Chain Implementation

ADS trees can be logically nested within other trees as shown in Figure 11.3. The completeness chain mechanism, which we introduced in Section 11.3.5, logically nests ADSes within another ADS. In this example, a trusted entity, such as the administrator of the medical application, uses a static, i.e., predefined, trust context, named “admins”, owned by the system administrator, to manage the patient groups. One of the protected fields is used to store the trust context name of each group. This field can be used as a reference to identify all the correct trust contexts that correspond to the patient groups, which in turn protect the patient profile data. Thus, we can use the “admins” trust context as a root trust context to establish completeness, for the query that reads patient profile data from all (unspecified) groups.

11.4.3 IQP Analyzer

The IQP analyzer checks whether a user can run a certain query based on the IQPs defined and the trust contexts to which this user belongs.

For each read query, the Verena client ensures that the query matches the IQP handle it was invoked on. A query matches an IQP if all of the following conditions hold:

- the query filters on the same list of fields as in eq-range of the IQP or on a prefix of these fields,
- the query performs a subset of operations and aggregates from “ops” of the IQP, and,
- if the trust context of this IQP is a field instead of a fixed trust context, the query specifies the value for this field (e.g., “patientID: 121” in the query in Section 11.3.4).

When inserting or deleting a document, the Verena client and server check that the user who inserts this document is a member of *all* the trust contexts defined by any IQP on this document. When updating a field f,

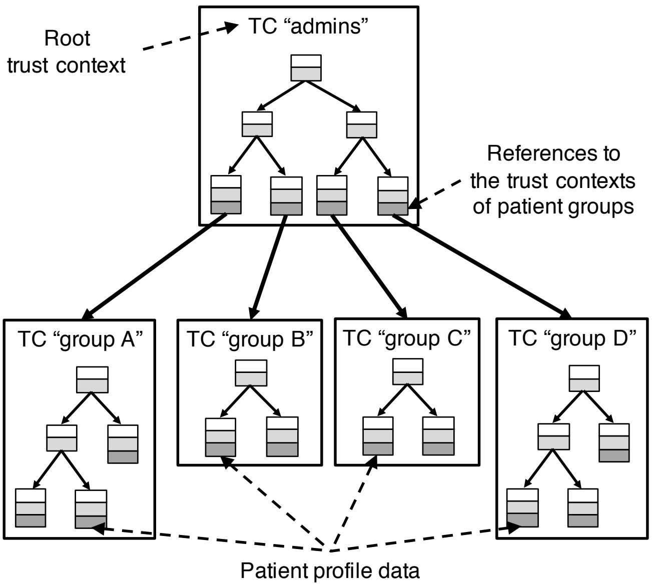


Figure 11.3: Illustration of the completeness chain mechanism. The root trust context "admins" can be used to guarantee completeness for the query that fetches patient profiles from all available patient groups.

the Verena client and server check that the user is a member of each trust context defined by an IQP on this document that has f as a protected field. When updating a field that is a trust context for an IQP, the user performing the update must belong to both the old and new trust contexts.

Of course, if both the Verena client and server performing these checks are compromised and collude, they will not perform these checks and will allow unauthorized actions. However, as we discuss in Section 11.6, the Verena clients running on behalf of honest users will detect and flag this issue. The unauthorized client is still not able to sign ADS updates with an authorized public key.

11.4.4 Trust Context Membership Operations

As described in Section 11.3.2 and Section 11.3.3, write access control is expressed by associating trust contexts with protected data, through IQPs. Only the members of a trust context are allowed to affect the results of a query associated with that trust context. The owner (creator) of a trust context is responsible for managing the membership, or in other words the

Function	Explanation
declareIDP(*url*, *pubkey*)	Specifies the *url* and *pubkey* of the IDP.
createAccount(*uname*, *passw*, [*creator*])	Creates an account for user *uname*. Must be called by the user when his account is created or, if a creator user is specified, by the creator.
lookupUser(*uname*, [*creator*])	Looks up the user *uname* as created by *creator*. If the creator is not specified, Verena considers the default which is the IDP.
login(*uname*, *passw*)	Logs in user *uname* with the specified password.
logout()	Logs out the currently logged-in user.
createTC(name)	Creates a new trust context *tc* with name *name* owned by the current user.
isMember(*tc*, *user*)	Returns whether *user* is member of the trust context named *tc*.
addMember(*tc*, *user*)	Adds *user* to *tc* only if the current user is the owner of the trust context *tc*.
removeMember(*tc*, *user*)	Removes *user* from *tc* only if the current user is the owner of the trust context *tc*.

Table 11.4: User and trust context API in Verena. Each function runs in the user's browser and *current user* denotes the currently logged in user who performed this action.

access control list (ACL), list of the trust context, by adding and removing users. We note that one can create additional groups for further levels of nesting and have a trust context consist of a list of users and groups of users. For simplicity, we do not describe groups beyond trust contexts in this work. Also recall that, each trust context is identified by a unique name. Table 11.4 shows the API for adding or removing trust context members.

The ACL of a trust context is a piece of information that needs to be integrity protected, just like other sensitive data in the web application. Verena internally maintains a collection for storing the trust context ACLs and protects it by declaring an appropriate IQP. Consequently, both ACL modification operations, as well as read operations for verifying whether a user belongs to a trust context ACL, are integrity protected. The corresponding entries on the hash server, i.e., those that store the latest root hashes of the ADSes protecting the ACLs of trust contexts are created in a special way (see Section 11.5), to make sure that only the owner of a trust context can update the root hash of its ACL, and thus manipulate the ACL.

We note that an extra step has to be performed when removing a user from a trust context. As we describe in Section 11.5, each hash server entry stores the public key PK of the last user that modified the entry. When the owner removes a user u from a trust context `tc`, the owner must update the entries at the hash server that correspond to ADS trees for `tc` (these are the ADSes that protect data associated with `tc`) that were last modified by u. In order to achieve this, the owner makes a no-op modification so these entries now appear modified by the owner, which is valid, because he is a member himself. This update is necessary because, without it, clients verifying if the last modification to the ADS tree was permitted will notice that this modification was performed by a user who is not in the trust context anymore.

Membership Verification. As part of verifying the result of a query (described in detail in Section 11.6), the Verena client needs to check that the user who last modified the relevant protected data was authorized to do so. In other words, the Verena client needs to verify that a user u with public key PK is a member of a given trust context `tc`. To construct such a proof, the server provides to the client the binding of a username u to PK along with the signature from the IDP for this binding. Based on this signature, the client can verify that user u is indeed the owner of PK. Subsequently, the server has to prove that u is a member of the trust context `tc`. For this goal, the server fetches the entry for `tc` from the hash server and produces a proof from the ADS that protects the ACL of `tc`, in a process similar to any integrity protected read query.

11.5 Hash Server

The hash server has a simple functionality, similar to a key-value store. Its task is to store the most recent root hash for each ADS that exists in a Verena application, together with information about which user made the latest update. The hash server provides this information signed for authenticity to Verena clients. The clients use it to verify that the data they read is fresh and complete.

The hash server stores a map, in which the key is an ID and the value is an entry E = (hash h, version v, public key PK, flag `fixedPK`). The version *v* helps serialize concurrent operations to each entry. Depending on the value of `fixedPK`, we distinguish two types of entries, which we describe below.

Trust Context ACL Entry. Entries of this type store the root hash h of the ADS that protects the membership list (ACL) of a trust context. The ID of such an entry is uniquely derived from the trust context to which it corresponds. The version v indicates the number of modifications made so far to this entry. The public key PK belongs to the user who created the trust context, i.e., the owner, and `fixedPK` is true to indicate that only the creator of this entry is permitted to modify the entry. This reflects the fact that only the owner of the trust context is allowed to manipulate the trust context ACL.

ADS Entry. Entries of this type store the root hash h of an ADS that protects application data associated with some trust context. The ID of such an entry is uniquely derived from the IQP and trust context to which they correspond. The version v of an entry E indicates the number of modifications made so far to this entry, and PK is the public key of the user who last modified the hash of this entry. `fixedPK` is false indicating that anyone is allowed to modify this entry. The hash server does not check if the client modifying this entry was allowed to modify it. Instead, since all hash server requests go through the main server, the main server must check that the client is authorized. If the server misbehaves and allows unauthorized modifications, honest Verena clients will later detect this misbehavior by checking if the PK of the latest modification was allowed to perform this modification.

The hash server does not need to understand how each entry is used for integrity enforcement. It only implements the following simple interface consisting of two functions, `HS_GET` and `HS_PUT`:

```
HS_GET(ID):
  1: return map[ID]
HS_PUT(ID, $E_{old}$= (h, v, PK), $E_{new}$= (h', v', PK', fixedPK'), sig(ID, $E_{old}$, $E_{new}$)):
  1: Verify sig on (ID, $E_{old}$, $E_{new}$) using PK'
  2: if ID not in map and v' = 1 then
```

```
3:     map[ID] = (h', 1, PK', fixedPK'); **return** true
4: **end if**
5: **if** ID not in map **then return** false
6: E = map[ID]
7: **if** E.fixedPK and PK' $\neq$ PK **then return** false
8: **if** E.v = $E_{old}$.v and v' = v+1 and E.h = $E_{old}$.h and E.PK = PK **then**
9:     map[ID] = (h', v', PK', E.fixedPK); **return** true
10: **end if**
11: **return** false
```

As shown in Figure 11.4, when a Verena client makes a request to the hash server, the client attaches a random nonce. The hash server assembles the response as above and then signs it together with the request and the nonce. The signature and nonce prevent an attacker from replacing the response of the hash server with an invalid or an old response.

The hash server can receive batched requests of the same type from the same client. The hash server signs all the responses into one signature, for increased performance. When the client sends multiple HS_PUT requests, the hash server executes them atomically, i.e., it executes them only if all of them return true.

11.6 Communication Protocol and Query Processing

We now describe the protocol that governs the interaction between the client, the main server, and the hash server, as well as the operations that are executed during the processing of read and write queries.

Figure 11.4 shows the communication protocol in Verena. The sequence of operations in this protocol is the same no matter what the query from the client is, i.e., regardless of whether this query is reading some data, performing an aggregate, writing some data, or adding a member to a trust context. Only the details of the operations differ.

When issuing a query, the Verena client adds a randomly generated nonce to the query, to be included in the hash server's signed response. Based on the query, the main server derives a set of hash server requests whose results will help in assembling a proof of correctness for the query's results. The server submits them together as part of one big request to the hash server. The hash server executes the request atomically, as explained

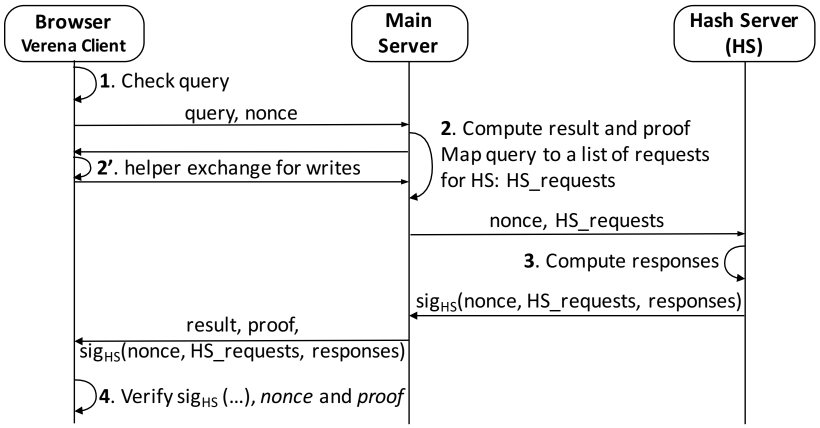

Browser
Verena Client
Main
Server
Hash Server
(HS)
1. Check query
query, nonce
2. Compute result and proof
Map query to a list of requests
for HS: HS_requests
2'. helper exchange for writes
nonce, HS_requests
3. Compute responses
sig$_{HS}$(nonce, HS_requests, responses)
result, proof,
sig$_{HS}$(nonce, HS_requests, responses)
4. Verify sig$_{HS}$ (...), *nonce* and *proof*

Figure 11.4: Communication protocol in Verena. Step 2' consists of an additional roundtrip that happens only for write operations.

in Section 11.5. Then, it signs its response together with the request and the client's nonce, and provides the signed response to the main server. The server computes the result of the query based on the server's state and uses the hash server's signed response to prove correctness of the query result to the client. The main server often needs to add extra information to show that some data hashes to the hashes provided by the hash server. The nonce prevents the main server from performing replay attacks on the hash server and serving old data to the client.

In Figure 11.4, Step 1 is explained in Section 11.4.3. Step 3 is explained in Section 11.5. We next describe Steps 2 and 4 for both read and write operations. For each query, if the server is honest, it still checks the regular read and write access control of an application, as coded by the developer, and rejects a query if the issuing client is not authorized to execute it. If the server is malicious, the server might skip this step, but write access control specified by IQPs will still be enforced by Verena. Clients will detect that the server violated the integrity access control specified in the IQPs.

11.6.1 Read Query

Read queries can be projection or aggregation queries. During Step 2, the server executes:

1. Create an empty list of requests for the hash server, called `HS_requests`, and an empty list for proofs to be given to the client, called `proof_list`.

2. Execute the query on the database and produce a result `result`.
3. Identify the relevant ADS instance for the trust context of this query or ADS instances for a query using a completeness chain. Assemble a proof of correctness of the query result based on these ADSes and their roots and add the proof to `proof_list`. Add requests for the hashes of the roots to `HS_requests`.
4. Identify the last client who modified each ADS. Assemble a proof that this client is in the trust context for the relevant IQP (per Section 11.4.4), add the corresponding hash requests to `HS_requests` and the proof to `proof_list`.

In Step 4 of Figure 11.4, the client verifies the proof from the server. For each ADS involved in the query, the client verifies (1) the ADS proof, (2) that the root hash of this ADS corresponds to the one from hash server's signature, (3) the hash server's signature, (4) the server's proof that the last client who changed the root hash (as specified by the public key in hash server response) was authorized, as explained in Section 11.4.4.

11.6.2 Write Query

A write query can be an insert, update, or remove. Verena provides linearizability guarantees. In particular, there is a total order between all read and write queries, and each read will see the latest committed write. A client considers a write query committed when the protocol in Figure 11.4 completes successfully.

To prevent the server from cheating during serialization, a natural solution is to have the hash server serialize requests. However, this strategy will increase the complexity at the hash server and our goal is to keep the hash server simple. Instead, the main server will do the serialization work in a verifiable way. The only job of the hash server is to ensure that each `HS_PUT` to an entry increments the version of the entry. Based on the version number, clients can verify that the server did not serve an old hash or attempted a fork attack [124].

A write query (e.g. insert) can cause modification of multiple ADSes. The server serializes the changes to these ADSes by locking access to each ADS involved. Parallel write queries affecting different ADS structures can still proceed concurrently.

Clients can issue a delete or update query to a certain document `ID`. If the developer wants to delete or update documents selected by a filter, the developer should first fetch those documents using a read operation and then update them by `ID`. Verena's design can be extended to enable such queries directly by employing verification as for read queries, but in interest of simplicity, we do not describe these changes here.

We now describe the steps involved in an insert, while delete is similar. Update proceeds as a delete and insert that happen at the same time (so there is no need to run the communication protocol twice).

During an insert operation, the client must help the server update ADS trees. For each ADS tree, the client first checks that the ADS tree at the server is correct. Namely, its root hash matches the corresponding hash at the hash server and was changed by an authorized client. The client does not have to download the entire ADS tree to perform this check. Only the relevant path in the tree is required. Then, the client inserts the new value and recomputes the new root hash. It signs this hash and provides it to the main server, to be included in a hash server `HS_PUT` request.

To avoid the need for an additional round trip between the main and hash servers, the main server maintains a copy of the hash server map. Thus, the client obtains the hash root $\texttt{hash}_{\text{old}}$ from the main server instead of the hash server. Of course, the main server could provide an incorrect value, but both the hash server and the client detect this behavior as follows. When the client performs the update, the client provides a new hash along with $\texttt{hash}_{\text{old}}$ in a signature sent to the hash server as part of `HS_PUT`. The hash server checks that $\texttt{hash}_{\text{old}}$ matches the value at the hash server as discussed in Section 11.5.

Due to updates, some data content may cycle back to an old hash. The version numbers prevent a malicious server from replaying previous updates on this repeated hash value.

During Step 2 and 2', the server runs:

1. Check, using regular access control, if the client is allowed to write. If not, return.
2. Identify the relevant ADSes $A_1 \ldots A_n$ that need to be updated. Acquire a lock for each one of these.
3. Send a message to the client containing a proof for each ADS A_i as discussed above. Instead of contacting the hash server for the tuple E = (hash h, v, PK), send this information from the server's storage. Also, send a proof that PK belongs to a user who is allowed to make the change as in the read operation.

In Step 2', the client runs:

1. Verify all the proofs as in a read operation.
2. If verification passes, for each ADS involved, provide nonce and $\texttt{sig}_{user}$(ID, E, E_{new}), where ID is the id of the corresponding entry at the hash server, E_{new}= (h', v' = v+1, PK'), where h' is the new hash after the change.

The server runs:

1. Check the client's signature, h', v', and PK'. If everything verifies, update the database, the ADS trees, and send E_{new} and $\texttt{sig}_{user}$(ID, E, E_{new}) to the hash server.
2. After receiving the response from the hash server, forward it to the client.
3. Release the locks.

Finally, the client verifies the hash server's response. The signature from the hash server should verify with the nonce and this indicates that the hash server accepted the change. If so, the write completed. Otherwise, the main server misbehaved. The main server also has a timeout during which it keeps locks on behalf of the client. To provide liveness, if the client takes too long to answer in Step 2', the server aborts this request and releases the lock.

11.7 Informal Security Argument

In §11.3.7, we describe the guarantees provided by Verena. Here, we present a high-level argument of why these guarantees hold. To argue that Verena's read queries return results satisfying the guarantees in §11.3.7, we show the following two properties. Given a read query q, let ADS be the ADS corresponding to q and tc be its trust context.

1. The hashes of the roots of ADS and tc at the hash server correspond to the *latest* modification by a user in tc.
2. Given the root hash of ADS and tc that satisfy the property above, a client can detect if the query's result does not satisfy the guarantees in §11.3.7

The second property follows from the properties of ADS trees (recall that tc is also implemented as an ADS tree). Let us explain why the first property holds. Assuming a trusted hash server, the hash server will return to a read query the latest hash from a write query. Moreover, the Verena client checks for each read query that the latest write was created by an authorized user. At the same time, due to the nonce used by clients when receiving responses from the hash server, a client who committed a write knows that the hash server persisted its update. For queries that span multiple trust contexts, given a root trust context and the properties of ADS trees, the completeness chain mechanism can guarantee the completeness and correctness of the results.

11.8 Discussion

Limitations and Extensions. Verena does not support all possible query types, although it supports a common class of queries. Section 11.3.4 describes the queries our current system supports. Nevertheless, the overall Verena architecture is mostly agnostic to the underlying ADS. The literature provides ADSes for other types of queries, such as multidimensional range queries or text search queries [160]. Adding them to Verena should be straightforward.

Moreover, Verena does not support triggers. With a trigger, a database server notices when a certain condition on the data is satisfied and contacts the relevant users. If a server is compromised, it can choose not to contact the users. A mitigation to support triggers is to have the client check the triggers after performing an update or periodically.

Hash Server Trust. The design so far assumed that the hash server is trusted. Verena can survive compromise of the hash server as long as an attacker does not compromise both the hash server and the main server. This is simple to achieve, by having the main server check the answers provided by the hash server. This requires minimal change to the design so far because (1) the main server stores a copy of the entries at the hash server anyways and (2) all hash server responses pass through the server. The main server can detect misbehavior of the hash server and warn of a potential compromise.

User Signature Verification. The signature verification during `HS_PUT` (Section 11.5) of the user who performs the update can be removed from the hash server and instead performed in the clients. However, we decided to perform this verification on the hash server because it improves client latency and it is a simple operation. It avoids the need for clients to check this signature every time they check a proof involving it.

Data Confidentiality. Verena can be combined with a web framework such as Mylar [169], which protects data confidentiality against server compromise. This results in a solution offering both confidentiality and integrity protection against an active server attacker.

Using Verena Correctly. Verena provides protection only if the developer specifies the integrity policy correctly, which is not always easy. For example, the developer should not make write access control decisions based on data from the server that is not integrity protected. As part of our future work, we are interested in designing a tool that assists the developer and helps him make less mistakes.

11.9 Implementation

We developed a prototype implementation of Verena in order to evaluate our proposal and demonstrate its feasibility.

Web Platform. We implemented Verena on top of Meteor version 1.1.0.2. Meteor [134] is a JavaScript web application framework that uses Node.js [150] on the server side and MongoDB [137] as the database backend.

We chose Meteor as it offers some desirable features that make it attractive for our implementation. Meteor employs client-side webpage rendering based on HTML templates that are populated by data retrieved from the server. This means that there is a clear separation between application code and data. The code, which consists of the JavaScript code, the HTML templates and the CSS files, is signed by the developer and its integrity is verified by a browser extension upon loading, as in [169]. The integrity of data, which is the dynamic part of the web application, is enforced by Verena, according to the policy specified by the developer.

Moreover, Meteor features a uniform data model between the client and the server. In other words, clients are aware of how the data is organized in the MongoDB backend. This uniformity is beneficial to Verena because it allows both the client and server to understand the integrity policy and the database queries that will be executed. Thus, the server can identify which proofs to accompany the reply with, and the client can identify which proofs to expect from the server.

Meteor uses a publish/subscribe mechanism in which the web server automatically propagates data changes to clients who have subscribed for the results of a certain query. This mechanism is not compatible with the freshness guarantees Verena aims for, since a malicious server might not propagate changes to the interested clients. Hence, Verena follows the conventional approach of explicitly requesting the data of interest through the use of RPC requests. One can transform the publish/subscribe mechanism into a pull-based approach, in which the client polls the server periodically, thus providing time-bounded freshness guarantees.

Main Server and Client. We implemented the Verena server and client as a set of Meteor packages. The main server's implementation is 5100 lines of code. The main component consists of approximately 3100 lines of code. The storage and manipulation of the authenticated data structures, as well as the production of the necessary proofs is implemented as a separate service, which runs as a Node.js process and consists of about 2000 lines of code. We note that in this prototype implementation, the ADSes are stored in-memory, and not persisted on disk. For a production-quality implementation of Verena, the system should implement the ADSes within the database itself for better performance. The ADS manipulation logic is also replicated to the client, so that the client can verify the proofs presented by the server. We use 224-bit ECDSA for public-key operations, and SHA-256 as a cryptographic hash function. We perform most cryptographic operations in JavaScript using the SJCL library [200]. Nevertheless, to improve client performance, we implement ECDSA signature operations as a Google Chrome Native Client module [81].

Hash Server. The hash server is implemented as a Go HTTP server, backed by a RocksDB [179] persistent key-value store. The cryptographic operations (ECDSA signing and verification) are delegated to a separate process, written in C, which uses OpenSSL [152] (version 1.0.2d). The reason is that the native Go ECDSA implementation is currently significantly slower than the OpenSSL one. The hash server consists of 630 lines of code in total (497 for the Go component and 133 for the C component), not counting standard libraries such as OpenSSL. By contrast, an application server's total codebase consists of our Verena server's implementation plus the server-side code of the actual application. The actual application can easily have thousands to tens of thousands of lines of code.

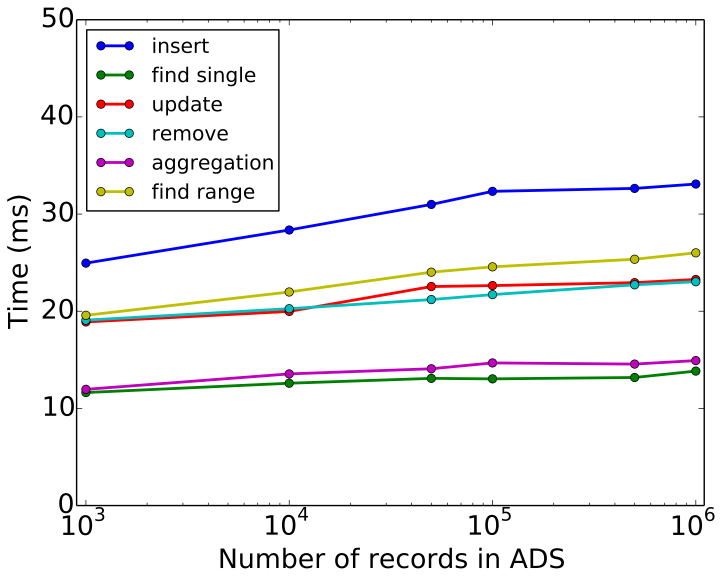


Figure 11.5: End-to-end latency of various read and write operations in Verena.

11.10 Evaluation

We used our prototype implementation to evaluate the performance of the various components of Verena. The evaluation setup is as follows. Verena's main server ran on a Macbook Pro "Mid 2012" (iCore7 2.3 GHz), while the hash server ran on an Intel Xeon 2.1 GHz processor with fast SSD storage, with a recent version of Ubuntu Linux installed. To perform our end-to-end latency measurements (Section 11.10.1, Section 11.10.3) we used a client using the Chrome browser, version 49, on a second Macbook Pro "Mid 2012" laptop. To measure throughput (Section 11.10.2), we employed multiple machines running many concurrent client instances using the headless browser PhantomJS [166], in order to saturate the system under test. All the machines that we used for the evaluation were connected to the university network.

11.10.1 End-to-End Latency

Fig 11.5 shows the end-to-end latency of the basic operations that are performed by Verena, i.e., write and read operations on a single ADS (or in

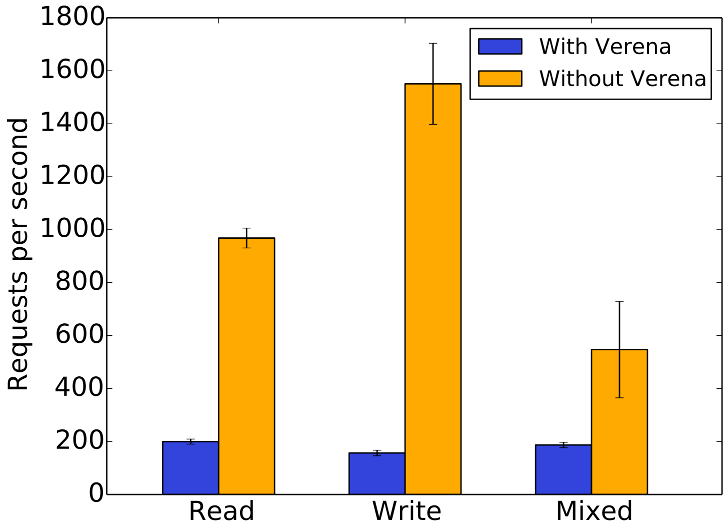


Figure 11.6: Throughput evaluation on the main server when running with Verena enabled, as well as without Verena.

other words a particular trust context). More concretely we tested insertion, update and removal of records as well as read operations, namely, fetching a single record ("find single"), fetching a range of 20 records ("find range") and computing an aggregate value (sum) on a particular field over a range of half of the currently inserted records. Each inserted record had a size of 1 KB.

We measured and averaged the latency of these operations, over 1000 iterations, for different numbers of inserted records in the ADS. We notice that latency slightly increases (for all operations) as the size of the ADS becomes larger. Even for an ADS size of 10^6 records, all operations, take less 30ms on average, except insert which takes slightly over 30ms for large ADS sizes.

11.10.2 Throughput

Main Server. We measured the throughput of Verena for read and write operations issued by multiple concurrent clients, on ADSes containing 10^4 records. When clients perform only read operations (in specific, fetch a range of 20 records) the average throughput is 200 (± 8) requests/sec. When performing only write operations (specifically, inserting records to

Operation type	Requests/second	Std.Dev.
GET	8420	673
PUT	2100	92
Mixed	5890	548

Table 11.5: Hash server throughput.

different trust contexts so that they can be processed in parallel), the average throughput is 156 (±10) requests/sec. Finally, when clients perform a mix of the above read and write operations (4 reads for 1 write) the average throughput becomes 187 (±10) requests/second.

Figure 11.6 displays the above throughput measurements and contrasts them with the throughput of the same server, but without Verena. As expected, the throughput is higher in all cases when Verena is not activated. We note that performing operations to different ADS trees can be run in parallel and independent of each other, which can increase the overall throughput when using additional server machines.

Hash Server. We also measured the throughput of the hash server, and the results are displayed in table 11.5. When the hash server receives only `GET` requests, the average throughput is 8420 (±673) requests/seconds. When it receives only `PUT` requests the average throughput is 2100 (±92) requests/second. This result is as expected because `PUT` requests perform an additional signature verification. When the hash server receives a mix of requests (4 `GET` requests for 1 `PUT`) the average throughput is 5890 (±548) requests/second.

It is important to note the significant difference in performance between the main server and the hash server. The hash server, which provides a very simple functionality, compared to the main sever, achieves an order of magnitude higher throughput than the main server.

11.10.3 Evaluation on the Example Medical Application

As introduced in Section 11.3.1, our running example is a remote patient monitoring system, that is used to connect cardiac implant (e.g., pacemaker) patients with their clinics and physicians. After receiving access to the provider (clinic/physician) web interface of a remote monitoring system and after discussions with a cardiologist, we implemented an example application of this system in Meteor and used Verena to secure its most relevant functions.

We specified three types of trust contexts as discussed in Section 11.3.2. In our measurements, we create 1000 patients per group, totaling 4000 patients. We evaluated the average latency (over 1000 iterations) of some representative example views that are displayed by the application to the physicians (summarized in table 11.6). For these particular views the integrity policy can be captured with 4 IQPs, three of which were discussed (simplified) in Section 11.3.3 and Section 11.3.5. We describe these views below.

Patient List. This view shows a list of patients across all groups, limited to 20 patients per page. This is one of the most complex views. The application has to first perform a query on the “admins” trust context in order to retrieve all the patient groups and corresponding trust contexts. Then, for each group it needs to perform a range query over that particular trust context to fetch the patients of each group, and then merge the results together. In other words, 5 read operations are required, assuming 4 patient groups. The overall latency for loading this view is 66ms (±7ms), the individual read operation latency being 13ms (±2ms). In other words the total latency for loading the page is approximately the sum of the individual read operations. We note that, through the use of a completeness chain (as described in Section 11.3.5), the developer can express the view using a single read query, and then Verena automatically takes care of performing all five needed queries.

Patients for Review. This view displays all the patients that are flagged for review. This view performs similar operations with the previous one. Assuming there are 50 patients from each group that are flagged for review, the overall latency for loading this view is 82ms (±7ms).

Patient Profile. This view displays the basic profile information of a single patient. This view requires a single read operation to fetch the particular profile. The overall latency for view is 14ms (±2ms).

EKG. This view displays a 30sec EKG recording of a particular holter episode. The recording contains double precision values of the measured heart electrical activity, sampled at 200 Hz, thus having a size of approximately 50 KB. This view requires a single read operation and the overall latency is 23ms (±4ms).

Application view	Load time (ms)	Std.Dev.
Patient list	66	7
Patient for review	82	7
Patient profile	14	2
Episode EKG	23	4
Avg. heart rate	13	3

Table 11.6: Latency for loading views in our example medical application whose data integrity is protected by Verena.

Average Heart Rate. This view displays the average heart rate of a patient as measured by his monitoring device, over a period of a few months. This view requires a read aggregation operation on the average over a set of samples. The latency for loading this view is 13ms (±3ms).

Summary. We summarize the end-to-end latency of the above views in table 11.6. We can see that Verena introduces acceptable latencies even for the most complex views that are implemented in this web application. We argue that Verena can be used to protect the integrity of an application such as this medical application, without disrupting the browsing experience of its users.

11.10.4 More Applications

To further evaluate the expressivity of Verena's integrity API, we considered two other applications, namely a chat and a class application. Both of these applications are written in Meteor, existed before Verena, were written by other developers, and have a multi-user setting that benefits from Verena's integrity guarantees.

We investigated whether Verena's API can express the write access control policy of these applications and how many IQPs need to be declared for this purpose. As we elaborate below, we found that Verena can express these applications' policy. In a few cases, we found that Verena provides a time-bounded freshness property as opposed to strict freshness. This happened for queries ran via Meteor's publish-subscribe mechanism. As discussed in Section 11.9, these rely on the server to notify the client of changes to a query result. Since the server cannot be trusted, clients must instead poll and run this query periodically. Thus, a freshness violation is confined to the period's duration.

Chat Application. We examined a chat application, called kChat. In kChat users can create rooms, the creator of a room can invite users to the chat room, and users within the room exchange messages. Each user is allowed to write only in a chat room to which he was invited. In terms of queries, users fetch all messages in a room, perform range queries to select the latest messages, count the messages in the room, fetch the list of people who are online, and so forth.

We found that Verena's IQPs can capture the write-access policy of this application. This means that Verena brings freshness, completeness and correctness for kChat.

Interestingly, there are multiple natural integrity policies for this application providing different integrity guarantees. A common query in this application is fetching all the messages in a room. If the developer trusts the users in the room and wants to protect against users outside of the room, the developer can specify the trust context to be all users in this room. In this case, Verena won't prevent a user in this room who colludes with the server from changing the messages of another user in the same room. The resulting integrity policy is short and consists of 3 IQPs. If the developer wants stricter integrity, namely, to prevent a user in the room from changing the message of another user in the same room, the developer declares two IQPs for this query. The first IQP is for fetching the trust context names of users in the room. The trust context for this IQP is owned by the creator of each room. The second IQP is for fetching the messages of a user in the room with a trust context of that user. The second IQP has should be chained to the first, so that the Verena's completeness chain mechanism can be employed, when performing a read query to fetch messages of a particular room. In this case, the kChat integrity policy can be captured with 4 IQPs.

Homework Submission Application. We also examined a web application used at MIT for managing student assignments, homework and grades for a computer science class. Students are allowed to submit their homework, as well as review and grade the homework of other students. The staff (i.e., the professor who teaches the course and the course assistants) are responsible for managing the student accounts, the homework assignments, the allocation of peer reviews for submitted homework of each student, as well as the final feedback and grade for each submitted homework. Verena can capture the integrity policy of this app with a total of 7 IQPs.

11.10.5 Storage and Memory Overhead

Main Server. We evaluated the overhead imposed by Verena in terms of storage and memory and found it to be modest. The memory footprint of adding Verena to the remote patient monitoring application (Section 11.10.3), is roughly 1.2× that of running the application without Verena. This overhead is mostly due to the memory required by the ADS storage service, as well as additional data structures that are maintained by the server for implementing the functionality of Verena.

Regarding the storage overhead, the main contributing factor is the storage of the ADSes. The space required to store an ADS (red-black Merkle binary tree, in our implementation) depends on the cardinality of its nodes, which depends linearly on the number of records that are protected by the ADS. An ADS that contains 10^4 records needs ~1.64 MB (~1.95 MB if the ADS also computes one aggregate value on the record). The relative overhead in this case depends on the size of the records under protection. Assuming an average record size of 0.5 KB, like the user messages in a chat application (Section 11.10.4), the overhead of storing the ADS that protects their integrity is approximately 1.4× the storage required for simply storing the messages themselves. For protecting larger records, as in the medical application for example, the storage overhead of protecting the 30sec EKG recordings (each recording amounts to 50 KB of data, as described in Section 11.10.3) is only ~1.003× the storage needed for storing just the EKGs. Finally, Verena requires approximately 1 KB per user for storing his wrapped private key, public key, and IDP certificate.

Hash Server. The hash server storage requirements are minimal compared to the rest of the system. The hash server stores only one entry for each trust context and ADS that exists in the system. Our hash server prototype stores the users' ECDSA public keys and the SHA-256 digests of the ADS roots as hexadecimal strings, and each entry needs less than 200 bytes of storage. This can be further reduced by using base64 or binary encoding for storing keys and digests, and by reducing the redundant storage of copies of public keys that may be stored in multiple entries. As an example, for the medical application (which we described and evaluated in terms of performance in Section 11.10.3), which contains thousands of users, trust contexts and ADSes, the storage requirements are less than 5 MB.

11.11 Related Work

To the best of our knowledge, Verena is the first web framework that provides integrity and freshness of the data and query results contained in a webpage, in the presence of a fully compromised web server.

11.11.1 Systems Providing Integrity

File Systems and File Storage. A few file systems, such as SUNDR [124], Sirius [78], SNAD [136], CRUST [77], Plutus [106], SAFIUS [199], Tresorium [113], CloudProof [167], Athos [79], and Caelus [110] aim to provide integrity in the face of a corrupted server. However, these are constructed for the simpler setting of a file server, so they do not verify query computation results (range queries or aggregations, as well as completeness and freshness for these computations), and do not consider the web setting which is stateless. Some of these systems (e.g., SUNDR) make no trust assumption on the server, but as a result, they either support only one client or do not provide freshness (e.g., SUNDR provides fork consistency which allows a server to present different views to different users). Caelus [110] provides time-bounded freshness by assuming a trusted always-online attestor per client.

Generic Services. Some works, such as [37], propose protocols through which mutually trusting clients can verifiably outsource computation to a generic untrusted service, and check the consistency of the server responses. In Verena, not all clients are mutually trusting, thus the Verena API gives the ability to express the desired integrity policy. Moreover, such works do not consider the web setting, in which clients may have zero state (not even some hashes). In Verena, the hash server solves this problem, as well as provides the ability to offer full linearizability, instead of the weaker consistency properties, such as fork-linearizability.

Trusted Hardware. Trusted hardware systems such as Haven [20] promise confidentiality and integrity against a compromised server. Unlike Verena, Haven relies on trusted hardware, and places the entire application code in the trusted code base. The only server assumption in Verena is that the hash and main servers are do not collude. Moreover, in Verena, the server application code is not in the trusted code base: in fact, if the application is buggy or exploitable, and thus corrupts integrity protected query results, clients will be able detect it.

11.11.2 Work Related to Verena's Building Blocks

In recent years, there has been much progress in tools for generic verifiable computation [23, 162, 229]. Nevertheless, for the web setting considered here, such tools remain impractical. Instead, work on authenticated data structures (ADS) [49, 80, 122, 123, 130, 160, 241] provides better performance and Verena uses these as building blocks. This line of work targets a more specific class of computation, such as aggregations on range queries, thus being more efficient. As previously discussed, these tools alone are not sufficient for addressing all the challenges of providing integrity protection for web applications.

Verena's hash server is related to the trinket component of TrInc [121]. The trinket is a piece of trusted hardware that stores and increments a counter; it can sign the counter along with a supplied string (e.g., a hash), and ensures that each counter is signed only once. TrInc can be used to provide freshness in SUNDR. Our hash server additionally stores the hash, public key and the `fixedPK` flag. These extra values enable useful properties in Verena, while the hash server still has high performance and small code base (see Section 11.10). Providing freshness in SUNDR+TrInc requires clients to download and verify a chunk of the history of changes to an item and to treat each "get" operation as a "put", which results in significantly lower performance.

11.11.3 Complementary Systems

A few systems can be used in complement to Verena, to provide a wider range of security guarantees.

Language Approaches/Information Flow Control. A few systems aim to help a developer not make programing mistakes that can lead to integrity or confidentiality violations. Using information flow control and/or language-based techniques, systems such as SIF [44], [178], Urflow [43], and Resin [238] ensure that an application obeys a security policy. However, if an attacker takes control of the server in these systems, the attacker can run any code of his/her desire, bypassing these tools completely, and violating integrity. In contrast, Verena protects against this situation. Nevertheless, these tools can be used in conjunction with Verena to ensure that a developer does not inadvertently leak data, as well as prevent against various client-side attacks.

Confidentiality. Mylar [169], CryptDB [168], and ShadowCrypt [92] aim to provide confidentiality against a corrupted web server, but do not address most integrity properties, such as freshness, completeness or query computation correctness. Mylar, also implemented on top of Meteor, is easy to integrate with Verena.

11.12 Summary and Future Work

In this part of the thesis we highlight the importance of providing integrity protection guarantees in web applications whose functionality is part of critical decision making processes, such as in the healthcare, government, military and financial services sectors. While the integrity protection of data accessed over a web application is typically guaranteed by the server itself, this does not hold once the web server gets compromised by an attacker.

Verena is the first web application platform that provides end-to-end integrity guarantees for data and query results in a webpage against attackers that have compromised the web server. In Verena, the user's browser can verify the integrity of a webpage, by verifying the results of the database queries which are used to populate the page content. Our evaluation results show that Verena can support real applications with modest overhead. Verena attempts to close the gap between the research efforts of protecting the integrity of database systems, and the application of this research in web applications, which consist one of the most popular use cases of databases.

11.12.1 Future Work

Simplify Development Effort. As we discussed in Section 11.8, Verena relies on the developer to correctly specify the integrity policy. This requires that the developer comprehends all the important notions of Verena, such as trust contexts, IQPs and completeness chains, as well as is familiar enough with the respective APIs, such that he can use them correctly in order to be able express the integrity policy (which Verena is responsible of enforcing). While we strived to make the Verena concepts and API intuitive, we acknowledge that this requirement on behalf of the developer is not always easy to satisfy, and the whole process of implementing Verena on top of a web application adds an extra burden to developers, especially for large and complex web applications.

Given the above limitation, an interesting research direction that comes naturally is to focus on simplifying the development effort and making it easier to integrate Verena in web applications. One way to achieve this would be to design a tool that can assist the developer in expressing the integrity policy using the Verena API by checking the developer's code, identifying potential pitfalls and making suggestions on how to make correct use of the API.

Another, likely better way of simplifying the development effort would be to give the ability to express the integrity policy in an easier, more intuitive manner. Drawing inspiration from model driven security and modeling languages, such as SecureUML [125], a promising direction would be to create a framework that takes as input the integrity policy of a web application expressed in a high-level descriptive language and automatically outputs the necessary code for enabling Verena. This code would then need to be integrated in the application's codebase with minimal effort on behalf of the developer.

Formalization of Verena Protocols. In Section 11.7 we provided an informal argument about the security of Verena. As part of future work, it is important to formalize the protocols and algorithms used in Verena and formally prove that they are correct and satisfy Verena's integrity guarantees, as presented in Section 11.3.7.

Chapter 12
Closing Remarks

Web applications are one of the most common ways of providing online services to users, customers and employees. People around the world user their desktop or mobile browsers daily to interact with many different websites for a variety of services, such as news, social media, email, cloud storage, health care, financial services and many other more. This implies that sensitive information stored and accessed through web applications. With more and more of user data being stored online and more and more services being offered online, it would not be an exaggeration to say that people's entire lives are digitalized and accessed, one way or another, via the web. As a result, web application security is a very important aspect of today's web.

In this thesis we focused on authentication and integrity protection on the web which is arguably one of the most important aspects of web application security. In particular, we looked into client (i.e., user) authentication, server authentication, as well as data authenticity and integrity protection, thus essentially covering all different sides of authentication around web applications. Our research was driven by our desire propose solutions that improve authentication security, as well as are practical and applicable in the real world. This means that we aimed, whenever possible, for solutions which are efficient, usable and deployable with existing web technologies and infrastructure.

In the first Part of this thesis we looked into the issue of poor security of authenticating users using passwords in web applications, as well as the low adoption rates of currently available two-factor authentication technologies. We argued that poor usability of existing proposals is one of the main reasons for their unpopularity. We, therefore, proposed Sound-Proof which is the first two-factor authentication approach that is both transparent to the user, i.e., no user interaction is required, and deployable with current browser technologies, i.e., without needing extra software to be installed on the user's computer. In Sound-Proof, the second authentication factor is the proximity of the user's phone to the computer from which the web login is taking placing. Proximity is verified by comparing the ambient sound which the two devices simultaneously record at the time of login. Only if the two recordings match is the login deemed successful.

We evaluated Sound-Proof in a variety of environments from train stations and cafeterias to office and home settings. Our evaluation showed that Sound-Proof works across environments, without the user having to interact with his phone, even if his phone is in his pocket or bag. We conducted a small user study in order to evaluate the usability of Sound-Proof in comparison with Google 2-Step Verification. We found that users found Sound-Proof to be more usable and appreciated the fact that is significantly faster than the code-based approach. More importantly, most users said that they would be willing to use Sound-Proof in scenarios where two-factor authentication is optional. This indicates that improving the usability of two-factor authentication can foster user adoption, such that more and more online accounts can benefit from the increased security that two-factor authentication offers.

In the second part of this thesis, we looked at web server authentication and the issues which the current CA trust model is facing. In particular we examined a recent proposal, based on TLS Channel IDs, which aims at thwarting TLS MITM attacks where the attacker's goal is user credential theft in order to impersonate the user to the server and compromise the user's account. We demonstrated how the proposal fails to prevent such attacks, due to the nature of web applications, which allow the server to ship and execute JavaScript code to the user's browser. Our attack, called Man-In-The-Middle-Script-In-The-Browser, takes advantage of this feature, which enables the attacker to bypass the security offered by TLS Channel ID-based authentication against user credential theft by simply mounting his attack via JavaScript shipped to the user's browser. We showed that client authentication alone, no matter how strong it is, cannot prevent such attacks.

We then went a step further and showed how combining strong client authentication, e.g., based on TLS Channel IDs, together with the concept of server invariance, can indeed prevent the aforementioned TLS MITM attacks. We described the property of server invariance and how it is easier to achieve compared to server authentication, since no initial trust is required. In our solution, called SISCA (Server Invariance with Strong Client Authentication), we discussed ways of implementing server invariance and showed how it can be integrated in today's web infrastructure.

In the third part of the thesis we focused on the authenticity of the data itself which is of high importance in cases where it influences decision making. We investigated the scenario where the attacker is able to fully compromise a web server. In such cases server authentication becomes irrelevant for the security of the data which the web application stores. Building upon work on data structure authenticity and integrity protection, we proposed Verena, the first framework that can provide end-to-end data authenticity and integrity guarantees in web applications even under full server compromise. Verena provides an API that can be used by the web application developer to express the application's data integrity protection policy. Verena enforces the expressed policy by making sure the code, data, and query computation results contained in the webpage and displayed in the user's browser are authentic (manipulated only by authorized users), complete and fresh.

We implemented a prototype of Verena on top of the Meteor framework and evaluated its performance on a medical web application which we builded based on a real such application, which we were given access to. Our evaluation showed that Verena incurs overhead which is small enough as to not hamper the user browsing experience. We further showed how Verena's API can support a variety of access control policies to for protecting data integrity.

We conclude this thesis by noting two messages that were highlighted throughout this thesis. First, when designing security solutions, usability and deployability are equally as important goals as security itself is and thus should not be overlooked. Even if this leads to a trade-off, where security is slightly decreased in favor of usability, it can still be beneficial as it may lead to higher adoption and thus increased security overall. Finally, data authenticity and integrity protection is at least as, or in cases, more important than data confidentiality, making it a worthwhile property to focus security research on.

Appendices

A System Usability Scale

This section lists the items of the System Usability Scale [33] used to evaluate Sound-Proof. All items were answered with a 5-point Likert-scale from *Strongly Disagree* to *Strongly Agree.*

Q1 I think that I would like to use this system frequently.
Q2 I found the system unnecessarily complex.
Q3 I thought the system was easy to use.
Q4 I think that I would need the support of a technical person to be able to use this system.
Q5 I found the various functions in this system were well integrated.
Q6 I thought there was too much inconsistency in this system.
Q7 I would imagine that most people would learn to use this system very quickly.
Q8 I found the system very cumbersome to use.
Q9 I felt very confident using the system.
Q10 I needed to learn a lot of things before I could get going with this system.

B Post-test Questionnaire

This section lists the items of the post-test questionnaire used to evaluate Sound-Proof. All items were answered with a 5-point Likert-scale from *Strongly Disagree* to *Strongly Agree.*

Q1 I thought the audio-based method was quick.
Q2 I thought the code-based method was quick.
Q3 If Two-Factor Authentication were mandatory, I would use the audio-based method to log in.
Q4 If Two-Factor Authentication were mandatory, I would use the code-based method to log in.
Q5 If Two-Factor Authentication were optional, I would use the audio-based method to log in.
Q6 If Two-Factor Authentication were optional, I would use the code-based method to log in.
Q7 I would feel comfortable using the audio-based method at home.
Q8 I would feel comfortable using the audio-based method at my workplace.

Q9	I would feel comfortable using the audio-based method in a cafe.
Q10	I would feel comfortable using the audio-based method in a library.
Q11	I would feel comfortable using the code-based method at home.
Q12	I would feel comfortable using the code-based method at my workplace.
Q13	I would feel comfortable using the code-based method in a cafe.
Q14	I would feel comfortable using the code-based method in a library.

C User Comments

This section lists some of the comments that participants added to their post-test questionnaire when evaluating Sound-Proof.

> "Sound-Proof is faster and automatic. Increased security without having to do more things"

> "I would use Sound-Proof, because it is less complicated and faster. I do not need to unlock the phone and open the application. In a public place it would feel a bit awkward unless it becomes widespread. Anyway, I am already logged in most websites that I use."

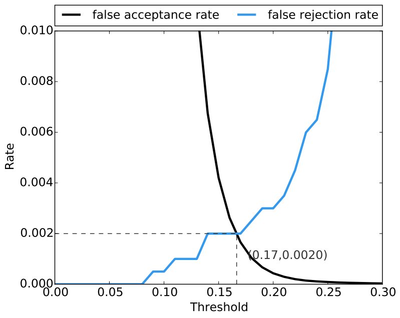
> "I like the audio idea, because what I hate the most about two-factor authentication is to have to take my phone out or find it around."

> "Sound-Proof is much easier. I am security-conscious and already use 2FA. I would be willing to switch to the audio-based method."

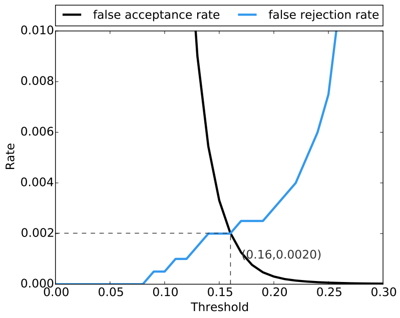
> "I already use Google 2SV and prefer it because I think it's more secure. However, Sound-Proof is seamless."

D Other Frequency Bands

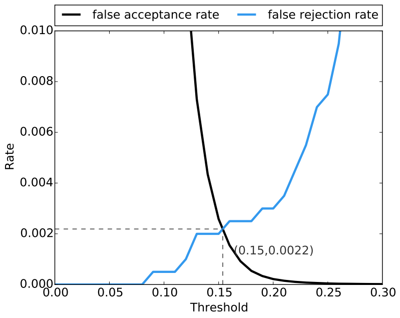
In this section, we present all the plots for the considered band ranges we analyzed during our experimental campaign setting the EER to be 0.0020 as discussed in Chapter 5.



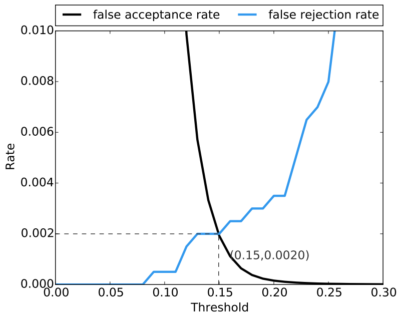
(a) $B = [50\text{Hz} - 630\text{Hz}]$



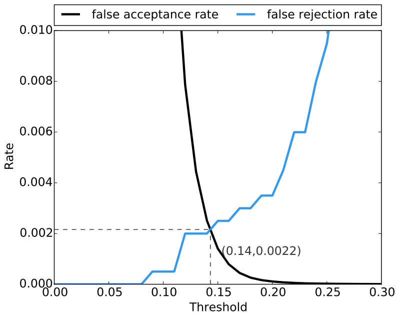
(b) $B = [50\text{Hz} - 800\text{Hz}]$



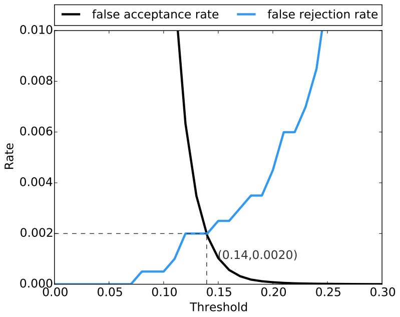
(c) $B = [50\text{Hz} - 1000\text{Hz}]$



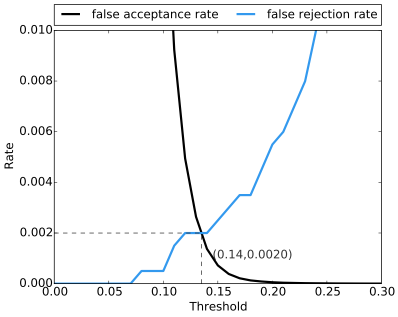
(d) $B = [50\text{Hz} - 1250\text{Hz}]$



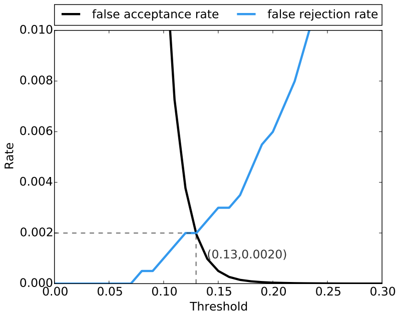
(e) $B = [50\text{Hz} - 1600\text{Hz}]$



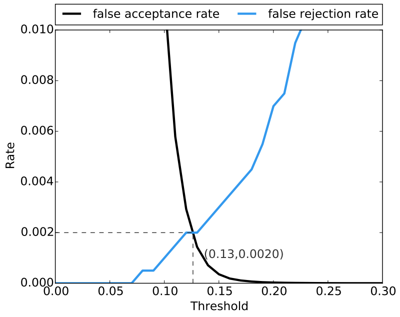
(f) $B = [50\text{Hz} - 2000\text{Hz}]$



(g) $B = [50\text{Hz} - 2500\text{Hz}]$

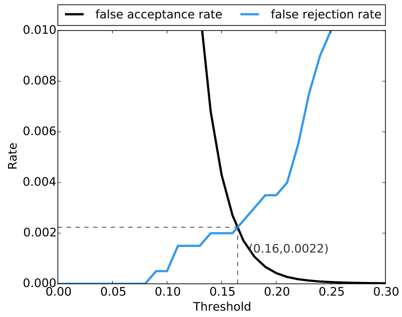


(h) $B = [50\text{Hz} - 3150\text{Hz}]$

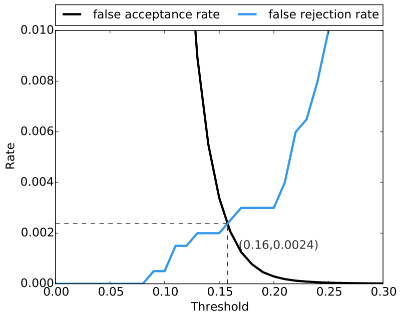


(i) $B = [50\text{Hz} - 4000\text{Hz}]$

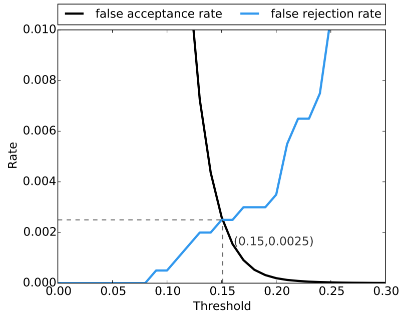
Figure D.1: False Rejection Rate and False Acceptance Rate as a function of the threshold τ_C for bands in the $B = [50\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range



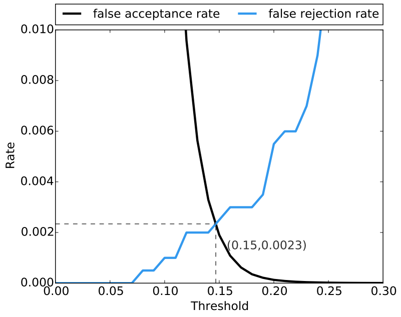
(a) $B = [63\text{Hz} - 630\text{Hz}]$



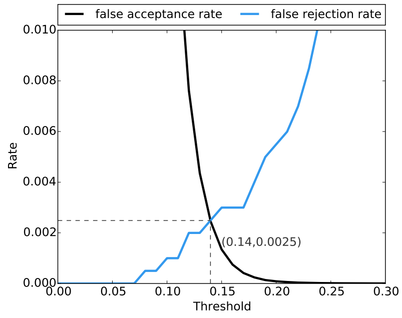
(b) $B = [63\text{Hz} - 800\text{Hz}]$



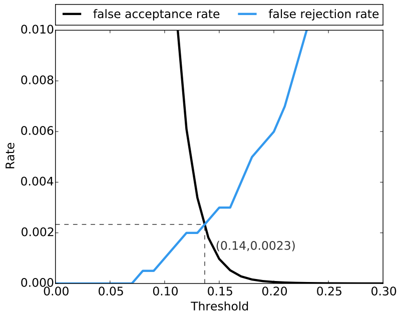
(c) $B = [63\text{Hz} - 1000\text{Hz}]$



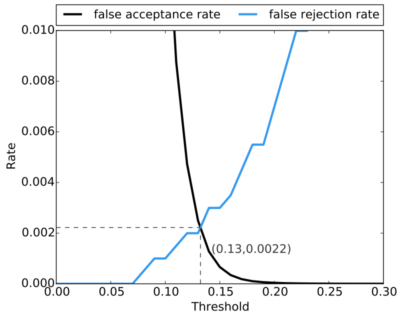
(d) $B = [63\text{Hz} - 1250\text{Hz}]$



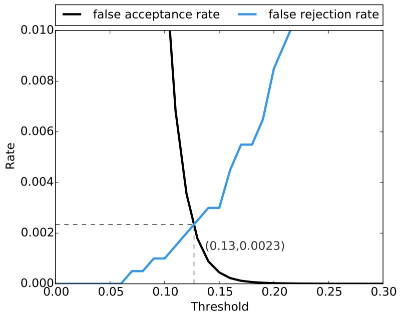
(e) $B = [63\text{Hz} - 1600\text{Hz}]$



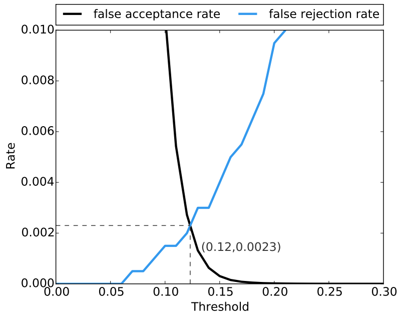
(f) $B = [63\text{Hz} - 2000\text{Hz}]$



(g) $B = [63\text{Hz} - 2500\text{Hz}]$

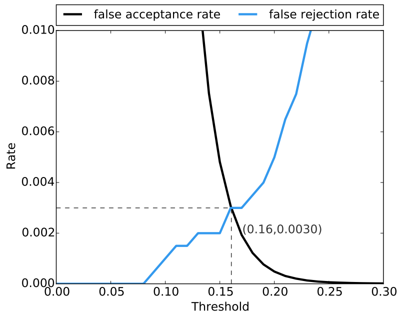


(h) $B = [63\text{Hz} - 3150\text{Hz}]$

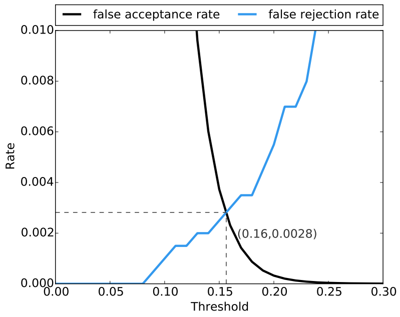


(i) $B = [63\text{Hz} - 4000\text{Hz}]$

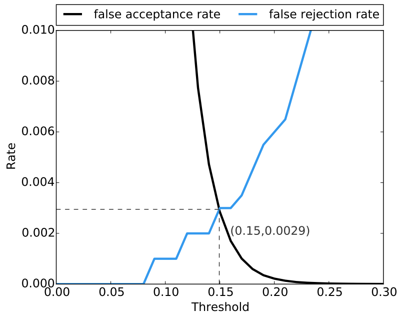
Figure D.2: False Rejection Rate and False Acceptance Rate as a function of the threshold τ_C for bands in the $B = [63\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range



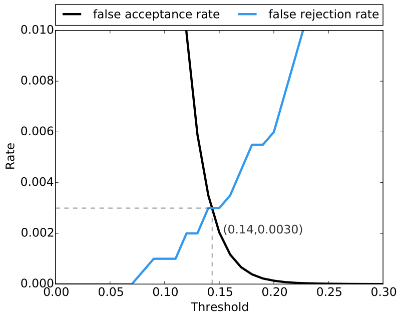
(a) $B = [80\text{Hz} - 630\text{Hz}]$



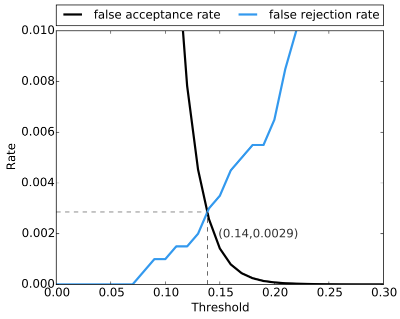
(b) $B = [80\text{Hz} - 800\text{Hz}]$



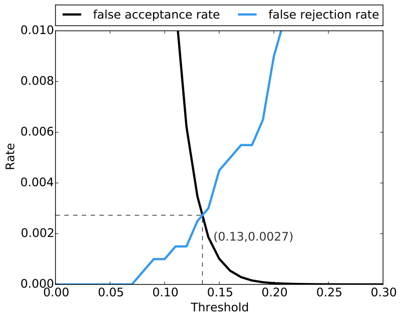
(c) $B = [80\text{Hz} - 1000\text{Hz}]$



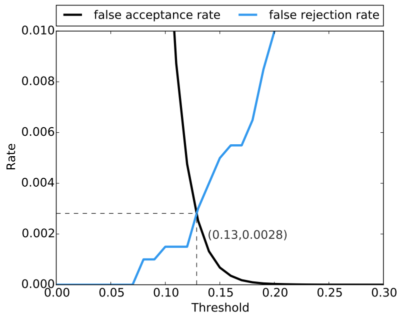
(d) $B = [80\text{Hz} - 1250\text{Hz}]$



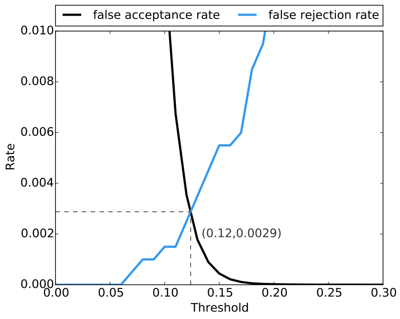
(e) $B = [80\text{Hz} - 1600\text{Hz}]$



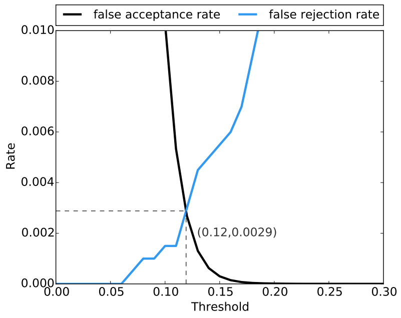
(f) $B = [80\text{Hz} - 2000\text{Hz}]$



(g) $B = [80\text{Hz} - 2500\text{Hz}]$

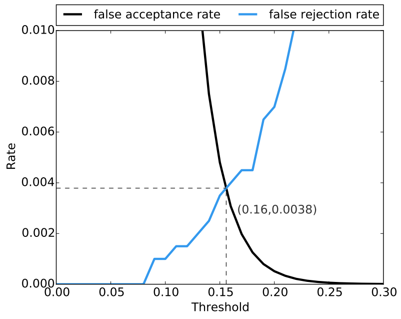


(h) $B = [80\text{Hz} - 3150\text{Hz}]$

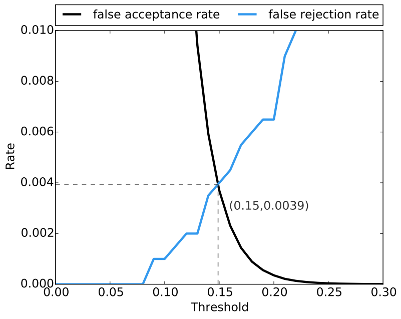


(i) $B = [80\text{Hz} - 4000\text{Hz}]$

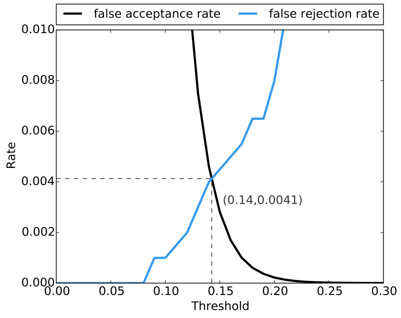
Figure D.3: False Rejection Rate and False Acceptance Rate as a function of the threshold τ_C for bands in the $B = [80\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range



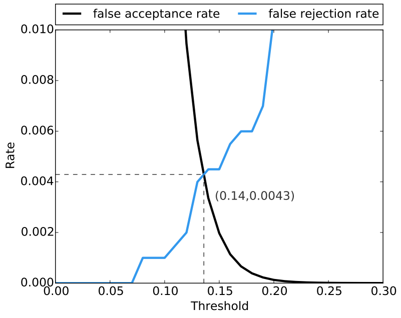
(a) $B = [100\text{Hz} - 630\text{Hz}]$



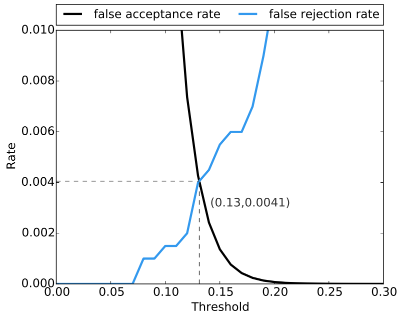
(b) $B = [100\text{Hz} - 800\text{Hz}]$



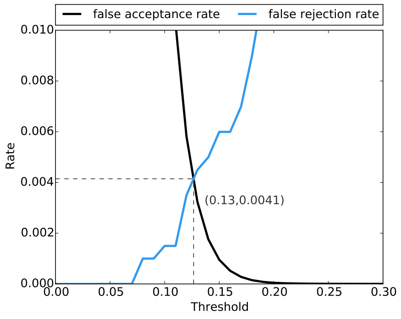
(c) $B = [100\text{Hz} - 1000\text{Hz}]$



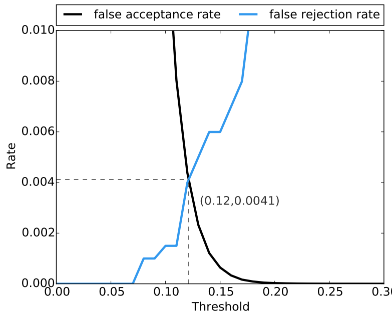
(d) $B = [100\text{Hz} - 1250\text{Hz}]$



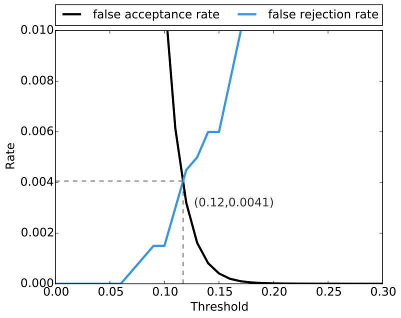
(e) $B = [100\text{Hz} - 1600\text{Hz}]$



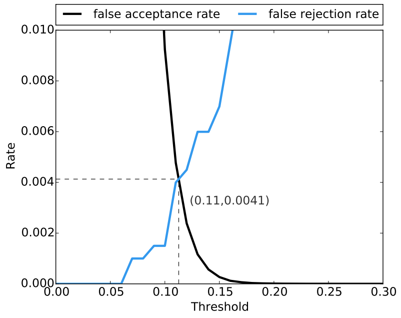
(f) $B = [100\text{Hz} - 2000\text{Hz}]$



(g) $B = [100\text{Hz} - 2500\text{Hz}]$



(h) $B = [100\text{Hz} - 3150\text{Hz}]$



(i) $B = [100\text{Hz} - 4000\text{Hz}]$

Figure D.4: False Rejection Rate and False Acceptance Rate as a function of the threshold τ_C for bands in the $B = [100\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range

Bibliography

[1] 3GPP. TS 23.271 - Functional stage 2 description of Location Services (LCS).

[2] H. Adkins. An update on attempted man-in-the-middle attacks, last access 2017. `https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html`.

[3] D. Akhawe and A. P. Felt. Alice in Warningland: A Large-scale Field Study of Browser Security Warning Effectiveness. In *USENIX Security Symposium*, USENIX'13, 2013.

[4] Alert Logic, Inc. Web Application Attacks - Crushing the Competition for Security Breaches, last access 2017. `https://www.alertlogic.com/blog/web-application-attacks-crushing-the-competition-for-breaches/`.

[5] M. Alicherry and A. D. Keromytis. DoubleCheck: Multi-path verification against man-in-the-middle attacks. In *IEEE Symposium on Computers and Communications*, ISCC'09, 2009.

[6] Apple Inc. Accelerate framework, last access 2017. `https://developer.apple.com/documentation/accelerate`.

[7] Apple Inc. Notifications for Developers, last access 2017. `https://developer.apple.com/notifications/`.

[8] W. A. Arentz and U. Bandara. Near ultrasonic directional data transfer for modern smartphones. In *International Conference on Pervasive and Ubiquitous Computing*, UbiComp'11, 2011.

[9] ARM Ltd. ARM NEON, last access 2017. `https://developer.arm.com/technologies/neon`.

[10] Authy Inc. Authy, last access 2017. `https://www.authy.com`.

[11] M. Backes, T. Chen, M. Dürmuth, H. P. A. Lensch, and M. Welk. Tempest in a Teapot: Compromising Reflections Revisited. In *IEEE Symposium on Security and Privacy*, SP'09, 2009.

[12] M. Backes, M. Dürmuth, and D. Unruh. Compromising Reflections-or-How to Read LCD Monitors around the Corner. In *IEEE Symposium on Security and Privacy*, SP'08, 2008.

[13] D. Balfanz and R. Hamilton. Transport Layer Security (TLS) Channel IDs, v01 (IETF Internet-Draf), 2013. `http://tools.ietf.org/html/draft-balfanz-tls-channelid-01`.

[14] S. Baluja and M. Covell. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition*, 41(11), 2008.

[15] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6), 2008.

[16] A. Barth. HTTP State Management Mechanism (RFC 6265), 2011. `https://tools.ietf.org/html/rfc6265`.

[17] A. Barth. The web origin concept (RFC 6454), 2011. `http://tools.ietf.org/html/rfc6454`.

[18] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. ARPKI: Attack Resilient Public-Key Infrastructure. In *ACM Conference on Computer and Communications Security*, CCS'14, 2014.

[19] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Oct. 2016.

[20] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *Symposium on Operating Systems Design and Implementation*, OSDI'14, 2014.

[21] G. Becker. Merkle signature schemes, Merkle trees and their cryptanalysis. 2008.

[22] M. Belshe and R. Peon. SPDY protocol (IETF Internet-Draf), 2012. `https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00`.

[23] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *International Cryptology Conference*, CRYPTO'13, 2013.

[24] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *IEEE Symposium on Security and Privacy*, SP'14, 2014.

[25] Biotronik SE & Co. KG. Home monitoring, last access 2017. `https://www.biotronik.com/en-us/products/services/home-monitoring`.

[26] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the Cloud. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[27] M. Bodnarchuk. Why should you use client-side MVC framework?, last access 2017. `http://jster.net/blog/why-should-you-use-client-side-mvc-framework`.

[28] 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 Symposium on Security and Privacy*, SP'12, 2012.

[29] Boston Scientific. Remote patient monitoring, last access 2017. `http://www.bostonscientific.com/en-US/products/remote-patient-monitoring.html`.

[30] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor authentication: Somebody you know. In *ACM Conference on Computer and Communications Security*, CCS'06, 2006.

[31] Brian Krebs. Dropbox Smeared in Week of Megabreaches, last accaircableess 2017. `https://krebsonsecurity.com/2016/06/dropbox-smeared-in-week-of-megabreaches/`.

[32] P. Bright. Independent Iranian Hacker Claims Responsibility for Comodo Hack, last access 2017. `http://www.wired.com/2011/03/comodo_hack/`.

[33] J. Brooke. SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194), 1996.

[34] BrowserAuth.net. TLS Client Authentication, last access 2017. `http://www.browserauth.net/tls-client-authentication`.

[35] H. Burri and D. Senouf. Remote monitoring and follow-up of pacemakers and implantable cardioverter defibrillators. *Europace*, 11(6), 2009.

[36] BusinessWire. Impermium Study Unearths Consumer Attitudes Toward Internet Security, last access 2017. `http://www.businesswire.com/news/home/20130627005473/en/Impermium-Study-Unearths-Consumer-Attitudes-Internet-Security`.

[37] C. Cachin and O. Ohrimenko. Verifying the consistency of remote untrusted services with commutative operations. In *Principles of Distributed Systems*, OPODIS'14, 2014.

[38] N. Carlini, A. P. Felt, and D. Wagner. An Evaluation of the Google Chrome Extension Security Architecture. In *USENIX Security Symposium*, USENIX'12, 2012.

[39] V. Chandrasekhar, M. Sharifi, and D. A. Ross. Survey and Evaluation of Audio Fingerprinting Schemes for Mobile Query-by-Example Applications. In *International Society for Music Information Retrieval Conference*, ISMIR'11, 2011.

[40] Chaos Computer Club. How to fake fingerprints?, last access 2015. `http://dasalte.ccc.de/biometrie/fingerabdruck_kopieren.en`.

[41] Chaos Computer Club. Chaos Computer Club breaks Apple TouchID, last access 2017. `http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid`.

[42] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-Resistant Password Vaults Using Natural Language Encoders. In *IEEE Symposium on Security and Privacy*, SP'15, 2015.

[43] A. Chlipala. Static checking of dynamically-varying security policies in database-backed applications. In *Symposium on Operating Systems Design and Implementation*, OSDI'10, 2010.

[44] S. Chong, K. Vikram, and A. C. Myers. SIF: Enforcing confidentiality and integrity in web applications. In *USENIX Security Symposium*, USENIX'07, 2007.

[45] J. Clark and P. C. van Oorschot. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE Symposium on Security and Privacy*, SP'13, 2013.

[46] M. Coates. Revoking trust in two TurkTrust certificates, last access 2017. `https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/`.

[47] Code School. Single-page Applications, last access 2017. `https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications`.

[48] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 5280), 2008. `https://tools.ietf.org/html/rfc5280`.

[49] S. A. Crosby and D. S. Wallach. Authenticated dictionaries: Real-world costs and trade-offs. *ACM Transactions on Information and System Security,*, 14(2), 2011.

[50] A. Czeskis and D. Balfanz. Protected login. In *Workshop on Usable Security*, USEC'12, 2012.

[51] A. Czeskis, M. Dietz, T. Kohno, D. S. Wallach, and D. Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *ACM Conference on Computer and Communications Security*, CCS'12, 2012.

[52] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The Tangled Web of Password Reuse. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[53] X. de Carné de Carnavalet and M. Mannan. From Very Weak to Very Strong: Analyzing Password-Strength Meters. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[54] T. P. Diakos, J. A. Briffa, T. W. C. Brown, and S. Wesemeyer. Eavesdropping near-field contactless payments: A quantitative analysis. *The Journal of Engineering*, 2013.

[55] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol, version 1.2 (RFC 5246), 2008. `http://tools.ietf.org/html/rfc5246`.

[56] 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 Symposium*, USENIX'12, 2012.

[57] M. Dietz and D. S. Wallach. Hardening Persona – Improving federated web login. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[58] Django, last access 2017. `https://www.djangoproject.com/`.

[59] Doug Gross. 50 million compromised in Evernote hack, last access 2017. `http://edition.cnn.com/2013/03/04/tech/web/evernote-hacked/`.

[60] P. H. Drielsma, S. Mödersheim, L. Viganò, and D. Basin. Formalizing and analyzing sender invariance. In *Formal Aspects in Security and Trust*, FAST'06, 2006.

[61] Duo Security, Inc. Duo Push, last access 2017. `https://duo.com/product/trusted-users/two-factor-authentication/duo-mobile`.

[62] S. Eberz, K. B. Rasmussen, V. Lenders, and I. Martinovic. Preventing lunchtime attacks: Fighting insider threats with eye movement biometrics. In *Annual Network and Distributed System Security Symposium*, NDSS'15. The Internet Society, 2015.

[63] P. Eckersley. A Syrian MITM attack against Facebook, last access 2017. `https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook`.

[64] P. Eckersley. The Sovereign Keys project, last access 2017. `https://www.eff.org/sovereign-keys`.

[65] Encap Security. Encap Security, last access 2017. `https://www.encapsecurity.com/`.

[66] Engadget. Latest Adobe Flash vulnerability allowed hackers to plant malware, last access 2017. `https://www.engadget.com/2017/10/16/adobe-flash-vulnerability-hackers-plant-malware/`.

[67] Entersekt. Transakt, last access 2017. `https://www.entersekt.com/Technology-Transakt`.

[68] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP (IETF Internet-Draf), 2013. https://tools.ietf.org/html/draft-ietf-websec-key-pinning-09.

[69] eyeLock Inc. Advanced iris authentication and recognition solutions, last access 2017. http://www.eyelock.com.

[70] I. Fette and A. Melnikov. The WebSocket protocol (RFC 6455), 2011. http://tools.ietf.org/html/rfc6455.

[71] FIDO alliance. Universal 2nd Factor (U2F) overview, Version 1.0 (review draft). http://fidoalliance.org/specs/fido-u2f-overview-v1.0-rd-20140209.pdf.

[72] FIDO Alliance. Specifications Overview, last access 2017. https://fidoalliance.org/specifications/.

[73] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), 1999. http://tools.ietf.org/html/rfc2616.

[74] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *Information Forensics and Security, IEEE Transactions on*, 8(1), Jan 2013.

[75] Fujitsu. PalmSecure, last access 2017. http://www.fujitsu.com/us/Images/palmsecure_datasheet.pdf.

[76] S. Gajek, M. Manulis, A.-R. Sadeghi, and J. Schwenk. Provably secure browser-based user-aware mutual authentication over TLS. In *ACM on Asia Conference on Computer and Communications Security*, ASIACCS'08, 2008.

[77] E. Geron and A. Wool. CRUST: cryptographic remote untrusted storage without public keys. *International Journal of Information Security*, 8(5), Oct 2009.

[78] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. In *Annual Network and Distributed System Security Symposium*, NDSS'03, 2003.

[79] M. T. Goodrich, C. Papamanthou, R. Tamassia, and N. Triandopoulos. Athos: Efficient authentication of outsourced file systems. In *Proceedings of the 11th International Conference on Information Security*, 2008.

[80] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *DARPA Information Survivability Conference and Exposition II (DISCEX II)*, 2001.

[81] Google Chrome. Native client, last access 2017. `https://developer.chrome.com/native-client`.

[82] Google Developers. Leverage Browser Caching, last access 2017. `https://developers.google.com/speed/docs/insights/LeverageBrowserCaching`.

[83] Google Inc. Google 2-Step Verification, last access 2017. `http://www.google.com/landing/2step/`.

[84] Google Inc. Google Cloud Messaging: Overview, last access 2017. `https://developers.google.com/cloud-messaging/gcm`.

[85] Google Inc. WebRTC, last access 2017. `http://www.webrtc.org/`.

[86] N. Gunson, D. Marshall, H. Morton, and M. A. Jack. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. *Computers & Security*, 30(4), 2011.

[87] J. Haitsma, T. Kalker, and J. Oostveen. An efficient database search strategy for audio fingerprinting. In *Workshop on Multimedia Signal Processing*, MMSP'02, 2002.

[88] J. A. Halderman, B. Waters, and E. W. Felten. A Convenient Method for Securely Managing Passwords. In *International Conference on World Wide Web*, WWW'05, 2005.

[89] T. Halevi, D. Ma, N. Saxena, and T. Xiang. Secure Proximity Detection for NFC Devices Based on Ambient Sensor Data. In *17th European Symposium on Research in Computer Security*, ESORICS '12, 2012.

[90] E. Haselsteiner and K. Breitfuss. Security in near field communication (NFC). In *RFIDSec'06*, 2006.

[91] M. Hazas and A. Ward. A Novel Broadband Ultrasonic Location System. In *International Conference on Pervasive and Ubiquitous Computing*, UbiComp'02, 2002.

[92] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song. Shadowcrypt: Encrypted web applications for everyone. In *ACM Conference on Computer and Communications Security*, CCS'14, 2014.

[93] P. Higgins. Pushing for perfect forward secrecy, an important web privacy protection, last access 2017. https://www.eff.org/deeplinks/2013/08/pushing-perfect-forward-secrecy-important-web-privacy-protection.

[94] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) protocol: TLSA (RFC 6698), 2012. http://tools.ietf.org/html/rfc6698.

[95] J. Hoffman-Andrews. Forward secrecy at Twitter, last access 2017. https://blog.twitter.com/engineering/en_us/a/2013/forward-secrecy-at-twitter.html.

[96] How-To Geek. Uninstall or Disable Plugins to Make Your Browser More Secure, last access 2017. https://www.howtogeek.com/209156/uninstall-or-disable-your-browser-plug-ins-to-make-your-browser-more-secure/.

[97] HTML Living Standard, last access 2017. https://html.spec.whatwg.org/multipage/.

[98] A. Inc. 1Password, last access 2017. https://agilebits.com/onepassword.

[99] IT Support Guides. WordPress - How to serve static content from a cookieless domain, last access 2017. https://www.itsupportguides.com/knowledge-base/wordpress/wordpress-how-to-serve-static-content-from-a-cookieless-domain/.

[100] C. Jackson and A. Barth. Beware of finer-grained origins. In *Web 2.0 Security and Privacy,*, W2SP'08, 2008.

[101] Jaikumar Vijayan. Hackers crack more than 60% of breached LinkedIn passwords, last access 2017. http:

//www.computerworld.com/article/2504078/cybercrime-hacking/hackers-crack-more-than-60--of-breached-linkedin-passwords.html.

[102] Y. Jia, Y. Chen, X. Dong, P. Saxena, J. Mao, and Z. Liang. Man-in-the-browser-cache: Persisting HTTPS attacks via browser cache poisoning. *Computers & Security*, 55(C), Nov. 2015.

[103] Z. Jorgensen and T. Yu. On mouse dynamics as a behavioral biometric for authentication. In *ACM on Asia Conference on Computer and Communications Security*, ASIACCS'11, 2011.

[104] A. Juels and R. L. Rivest. Honeywords: making password-cracking detectable. In *ACM Conference on Computer and Communications Security*, CCS'13, 2013.

[105] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0 (RFC 2898), 2008. https://tools.ietf.org/html/rfc2898#section-5.2.

[106] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *USENIX conference on File and Storage Technologies*, FAST'03, 2003.

[107] C. Karlof, U. Shankar, J. D. Tygar, and D. Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *ACM Conference on Computer and Communications Security*, CCS'07, 2007.

[108] Keybase. Welcome to Keybase, last access 2017. https://keybase.io/.

[109] I. Khalil, Z. Dou, and A. Khreishah. Your Credentials Are Compromised, Do Not Panic: You Can Be Well Protected. In *ACM on Asia Conference on Computer and Communications Security*, ASIACCS'16, 2016.

[110] B. H. Kim and D. Lie. Caelus: Verifying the consistency of cloud services with battery-powered devices. In *IEEE Symposium on Security and Privacy*, SP'15, 2015.

[111] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor. Accountable Key Infrastructure: A proposal for a public-key validation infrastructure. In *International Conference on World Wide Web*, WWW'13, 2013.

[112] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *3rd Symposium on Usable Privacy and Security*, SOUPS'07, 2007.

[113] I. Lam, S. Szebeni, and L. Buttyan. Tresorium: Cryptographic file system for dynamic groups over untrusted cloud storage. In *International Conference on Parallel Processing Workshops*, 2012.

[114] S. Lamb. Vulnerability statistics and trends in 2015, last access 2017. `https://www.contextis.com/resources/blog/vulnerability-statistics-trends-2015/`.

[115] A. Langley. How to botch TLS forward secrecy, last access 2017. `https://www.imperialviolet.org/2013/06/27/botchingpfs.html`.

[116] A. Langley. Protecting data for the long term with forward secrecy, last access 2017. `http://googleonlinesecurity.blogspot.ch/2011/11/protecting-data-for-long-term-with.html`.

[117] B. Laurie and E. Kasper. Revocation Transparency, last access 2017. `https://www.links.org/files/RevocationTransparency.pdf`.

[118] B. Laurie, A. Langley, and E. Kasper. Certificate transparency (RFC 6992), 2013. `http://tools.ietf.org/html/rfc6962`.

[119] Lee Munson. Yahoo prompts password reset after mass attack on email service, last access 2017. `https://nakedsecurity.sophos.com/2014/01/31/yahoo-prompts-password-reset-after-mass-attack-on-email-service/`.

[120] Let's Encrypt, last access 2017. `https://letsencrypt.org/`.

[121] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Symposium on Networked Systems Design and Implementation*, NSDI'09, 2009.

[122] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *ACM SIGMOD International Conference on Management of Data*, SIGMOD'06, 2006.

[123] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM Transactions on Information and System Security*, 13(4), 2010.

[124] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Symposium on Operating Systems Design and Implementation*, OSDI'04, 2004.

[125] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *International Conference on The Unified Modeling Language*, UML'02, 2002.

[126] M2SYS Technology. M2-PalmVein - Secure Palm Vein Scanner, last access 2017. `http://www.m2sys.com/palm-vein-reader/`.

[127] MariaDB, last access 2017. `https://mariadb.org/`.

[128] M. Marlinspike. SSL And The Future Of Authenticity, last access 2017. `https://moxie.org/blog/ssl-and-the-future-of-authenticity/`.

[129] M. Marlinspike and T. Perrin. Trust Assertions for Certificate Keys (TACK) (IETF Internet-Draft), 2013. `http://tack.io/draft.html`.

[130] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39, 2001.

[131] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *Symposium on Principles of Distributed Computing*, PODC'02, 2002.

[132] Medtronic. The CareLink network, last access 2017. `http://www.medtronic.com/us-en/healthcare-professionals/products/cardiac-rhythm/managing-patients/information-systems/carelink-network.html`.

[133] R. Merkle. Secrecy, authentication and public key systems / A certified digital signature. 1979.

[134] Meteor, Inc. The fastest way to build JavaScript apps, last access 2017. `https://www.meteor.com`.

[135] A. Miller, M. Hicks, J. Katz, and E. Shi. Authenticated Data Structures, Generically. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'14, 2014.

[136] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. C. Reed. Strong Security for Network-attached Storage. In *USENIX Conference on File and Storage Technologies*, FAST'02, 2002.

[137] MongoDB, last access 2017. `https://www.mongob.org/`.

[138] Mozilla Developer Network. HTTP authentication, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication`.

[139] Mozilla Developer Network. <iframe>, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe`.

[140] Mozilla Developer Network. Introduction to the DOM, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction`.

[141] Mozilla Developer Network. Mixed content, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content`.

[142] Mozilla Developer Network. Same-origin policy, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Same_origin_policy_for_JavaScript`.

[143] Mozilla Developer Network. Set-Cookie, last access 2017. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie`.

[144] Mozilla Developer Network. Web technology for developers, last access 2017. `https://developer.mozilla.org/bm/docs/Web`.

[145] Mozilla Developer Network. Subresource Integrity, last access 2018. `https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity`.

[146] Mozzilla. Location-Aware Browsing, last access 2017. `https://www.mozilla.org/en-US/firefox/geolocation/`.

[147] Narayan Prusty. Bypass Same Origin Policy, last access 2017. http://qnimate.com/same-origin-policy-in-nutshell/.

[148] Network Time Foundation. NTP: The Network Time Protocol, last access 2017. http://www.ntp.org/.

[149] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large-scale evaluation of remote Javascript inclusions. In *ACM Conference on Computer and Communications Security*, CCS'12, 2012.

[150] Node.js, last access 2017. https://nodejs.org/.

[151] Nuance Communications Inc. Multi-modal biometric authentication solutions, last access 2017. https://www.nuance.com/omni-channel-customer-engagement/security/multi-modal-biometrics.html.

[152] OpenSSL, last access 2017. https://www.openssl.org/.

[153] R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12), 2006.

[154] R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication revisited. *Computers & Security*, 27(3-4), 2008.

[155] OWASP. Cross-Site Request Forgery (CSRF), last access 2017. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[156] OWASP. Cross-site Scripting (XSS), last access 2017. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[157] OWASP. HttpOnly, last access 2017. https://www.owasp.org/index.php/HttpOnly.

[158] OWASP. Man-in-the-browser attack, last access 2017. https://www.owasp.org/index.php/Man-in-the-browser_attack.

[159] C. Palmer. Intent To Deprecate And Remove: Public Key Pinning, last access 2017. https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/he9tr7p3rZ8/eNMwKPmUBAAJ.

[160] H. Pang and T. Kian-Lee. *Query answer authentication*. Morgan and Claypool Publishers, 2012.

[161] S. D. Paola and G. Fedon. Subverting Ajax. 23rd Chaos Communication Congress, 2006.

[162] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, SP'13, 2013.

[163] B. Parno, C. Kuo, and A. Perrig. Phoolproof Phishing Prevention. In *10th International Conference on Financial Cryptography and Data Security*, FC'06, 2006.

[164] A. Parsovs. Practical issues with TLS client certificate authentication. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[165] T. Petsas, G. Tsirantonakis, E. Athanasopoulos, and S. Ioannidis. Two-factor authentication: Is the world ready?: Quantifying 2FA Adoption. In *8th European Workshop on System Security*, EuroSec'15, 2015.

[166] PhantomJS, last access 2017. `http://phantomjs.org/`.

[167] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling security in cloud storage SLAs with CloudProof. In *Proceedings of the 2011 USENIX Annual Technical Conference*, 2011.

[168] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles*, SOSP'11, 2011.

[169] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan. Building Web Applications on Top of Encrypted Data using Mylar. In *Symposium on Networked Systems Design and Implementation*, NSDI'14, 2014.

[170] N. Provos and D. Mazières. A future-adaptive password scheme. In *USENIX ATC*, ATC'99, 1999.

[171] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J. Frahm. iSpy: Automatic reconstruction of typed input from compromising reflections. In *ACM Conference on Computer and Communications Security*, CCS'11, 2011.

[172] K. B. Rasmussen and S. Capkun. Realization of RF distance bounding. In *USENIX Security Symposium*, USENIX'10, 2010.

[173] K. B. Rasmussen, M. Roeschlin, I. Martinovic, and G. Tsudik. Authentication using pulse-response biometrics. In *Annual Network and Distributed System Security Symposium*, NDSS'14. The Internet Society, 2014.

[174] N. Ratha, R. Bolle, V. Pandit, and V. Vaish. Robust fingerprint authentication using local structural similarity. In *IEEE Workshop on Applications of Computer Vision*, 2000.

[175] C. Reis, A. Barth, and C. Pizano. Browser Security: Lessons from Google Chrome. *ACM Queue*, 7(5), 2009.

[176] J. Reschke. The 'Basic' HTTP Authentication Scheme (RFC 7617), 2015. `https://tools.ietf.org/html/rfc7617`.

[177] R. L. Rivest. Can We Eliminate Certificate Revocations Lists? In *International Conference on Financial Cryptography*, FC'98, 1998.

[178] W. Robertson and G. Vigna. Static enforcement of web application integrity through strong typing. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[179] RocksDB, last access 2017. `http://rocksdb.org/`.

[180] A. Rodríguez Valiente, A. Trinidad, J. R. García Berrocal, C. Górriz, and R. Ramírez Camacho. Extended high-frequency (9 - 20 kHz) audiometry reference thresholds in 645 healthy subjects. *International Journal of Audiology*, 53(8):531–545, 2014.

[181] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *USENIX Security Symposium*, USENIX'05, 2005.

[182] RSA Security LLC. RSA SecurID Hardware Tokens, last access 2017. `https://www.rsa.com/en-us/products/rsa-securid-suite/rsa-securid-access/securid-hardware-tokens.html`.

[183] Ruby on Rails, last access 2017. `http://rubyonrails.org/`.

[184] D. A. Russell, J. P. Titlow, and Y.-J. Bemmen. Acoustic monopoles, dipoles, and quadrupoles: An experiment revisited. *American Journal of Physics*, 67(8), 1999.

[185] A. Sabzevar and A. Stavrou. Universal Multi-Factor Authentication Using Graphical Passwords. In *IEEE International Conference on Signal Image Technology and Internet Based Systems*, SITIS'08, 2008.

[186] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP (RFC 6960), 2013. `https://tools.ietf.org/html/rfc6960`.

[187] S. Schneegass, Y. Oualil, and A. Bulling. SkullConduct: Biometric User Identification on Eyewear Computers Using Bone Conduction Through the Skull. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI'16, 2016.

[188] B. Schneier. New NSA leak shows MITM attacks against major Internet services, last access 2017. `https://www.schneier.com/blog/archives/2013/09/new_nsa_leak_sh.html`.

[189] D. Schürmann and S. Sigg. Secure Communication Based on Ambient Audio. *IEEE Trans. Mob. Comput.*, 12(2), 2013.

[190] S. Shepherd. Continuous authentication by analysis of keyboard typing characteristics. In *European Convention on Security and Detection*, 1995.

[191] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan. Two-Factor Authentication Resilient to Server Compromise Using Mix-Bandwidth Devices. In *The Network and Distributed System Security Symposium*, NDSS'14, 2014.

[192] B. Shrestha, N. Saxena, H. Truong, and N. Asokan. Drone to the rescue: Relay-resilient authentication using ambient multi-sensing. In *Financial Cryptography and Data Security*, FC'14, 2014.

[193] B. Shrestha, M. Shirvanian, P. Shrestha, and N. Saxena. The Sounds of the Phones: Dangers of Zero-Effort Second Factor Login Based on Ambient Audio. In *ACM Conference on Computer and Communications Security*, CCS'16, 2016.

[194] H. Singh, T. Giardina, A. Meyer, S. Forjuoh, M. Reis, and E. Thomas. Types and origins of diagnostic errors in primary care settings. *JAMA Internal Medicine*, 173(6), 2013.

[195] H. Singh, A. Meyer, and E. Thomas. The frequency of diagnostic errors in outpatient care: estimations from three large observational studies involving US adult populations. *BMJ Quality and Safety*, 2014.

[196] S. Sivakorn, J. Polakis, and A. D. Keromytis. The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information. In *IEEE Symposium on Security and Privacy*, SP'16, 2016.

[197] R. Sleevi. Certificate Transparency in Chrome - Change to Enforcement Date, last access 2017. `https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/sz_3W_xKBNY/6jq2ghJXBAAJ`.

[198] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. In *Financial Cryptography and Data Security*, FC'11, 2011.

[199] V. Sriram, G. Narayan, and K. Gopinath. SAFIUS - A secure and accountable filesystem over untrusted storage. In *Proceedings of the 4thInternational IEEE Security in Storage Workshop*, 2007.

[200] Stanford JavaScript crypto library (SJCL), last access 2017. `https://crypto.stanford.edu/sjcl/`.

[201] StatCounter. StatCounter global stats, last access 2017. `http://gs.statcounter.com/`.

[202] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX Security Symposium*, USENIX'09, 2009.

[203] P. Szalachowski, L. Chuat, T. Lee, and A. Perrig. RITM: Revocation in the Middle. In *IEEE International Conference on Distributed Computing Systems*, ICDCS'16, 2016.

[204] P. Szalachowski, L. Chuat, and A. Perrig. PKI Safety Net (PKISN): Addressing the Too-Big-to-Be-Revoked Problem of the TLS Ecosystem. In *IEEE European Symposium on Security and Privacy*, EuroSP'16, 2016.

[205] P. Szalachowski, S. Matsumoto, and A. Perrig. PoliCert: Secure and Flexible TLS Certificate Management. In *ACM Conference on Computer and Communications Security*, CCS '14, 2014.

[206] R. Tamassia. Authenticated Data Structures. In *European Symposium on Algorithms*, ESA'03, 2003.

[207] TechCrunch. SlickLogin Aims To Kill The Password By Singing A Silent Song To Your Smartphone, last access 2017. `https://techcrunch.com/2013/09/09/slicklogin-wants-to-kill-the-password-by-singing-a-silent-song-to-your-smartphone/`.

[208] The American National Standards Institute. ANSI S1.11-2004 - Specification for Octave-Band and Fractional-Octave-Band Analog and Digital Filters, 2004.

[209] The CherryPy team. CherryPy, last access 2017. `http://www.cherrypy.org/`.

[210] The Guardian. Mobile web browsing overtakes desktop for the first time, last access 2017. `https://www.theguardian.com/technology/2016/nov/02/mobile-web-browsing-desktop-smartphones-tablets`.

[211] The Heartbleed Bug, last access 2017. `http://heartbleed.com/`.

[212] The OWASP Foundation. OWASP Top 10 2017 - The Ten Most Critical Web Application Security Risks, last access 2017. `https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf`.

[213] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. Data breaches, phishing, or malware? Understanding the risks of stolen credentials. In *ACM Conference on Computer and Communications Security*, CCS'17, 2017.

[214] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh. Towards Short-Lived Certificates. In *Web 2.0 Security and Privacy*, W2SP'12, 2012.

[215] Troy Hunt. ';–have i been pwned?, last access 2017. `https://haveibeenpwned.com/`.

[216] H. T. T. Truong, X. Gao, B. Shrestha, N. Saxena, N. Asokan, and P. Nurmi. Comparing and fusing different sensor modalities for relay attack resistance in Zero-Interaction Authentication. In *International Conference on Pervasive Computing and Communications*, PerCom'14, 2014.

[217] V. A. Vallivaara, M. Sailio, and K. Halunen. Detecting Man-in-the-middle Attacks on Non-mobile Systems. In *ACM Conference on Data and Application Security and Privacy*, CODASPY'14, 2014.

[218] I. Vér and L. Beranek. *Noise and Vibration Control Engineering*. Wiley, 2005.

[219] Verizon Communications, Inc. 2016 Data Breach Investigations Report, last access 2017. `http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf`.

[220] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Krügel, and G. Vigna. Cross Site Scripting prevention with dynamic data tainting and static analysis. In *Annual Network and Distributed System Security Symposium*, NDSS'07, 2007.

[221] W3C. Content Security Policy Level 3 (W3C Working Draft). `http://www.w3.org/TR/CSP/`.

[222] W3C. Offline web applications (W3C Working Group Note). `https://www.w3.org/TR/offline-webapps/`.

[223] W3C. Cross-Origin Resource Sharing (W3C Recommendation), last access 2017. `https://www.w3.org/TR/cors/`.

[224] W3C. HTML5 (W3C Recommendation), last access 2017. `https://www.w3.org/TR/html5/`.

[225] W3C. Indexed Database API 2.0 (W3C Recommendation), last access 2017. `https://www.w3.org/TR/IndexedDB-2/`.

[226] W3C. Media Capture and Streams (W3C Candidate Recommendation), last access 2017. `https://www.w3.org/TR/mediacapture-streams/`.

[227] W3C. Web storage, last access 2017. `http://www.w3.org/TR/webstorage/`.

[228] W3C. XMLHttpRequest, last access 2017. `https://www.w3.org/TR/XMLHttpRequest/`.

[229] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them: From theoretical possibility to near-practicality. In *Communications of the ACM*, 2015.

[230] A. Wang. The Shazam music recognition service. *Commun. ACM*, 49(8), 2006.

[231] Web Bluetooth Community Group. Web Bluetooth (Draft Community Group Report), last access 2017. `https://webbluetoothcg.github.io/web-bluetooth/`.

[232] C. S. Weir, G. Douglas, M. Carruthers, and M. A. Jack. User perceptions of security, convenience and usability for ebanking authentication tokens. *Computers & Security*, 28(1-2), 2009.

[233] C. S. Weir, G. Douglas, T. Richardson, and M. A. Jack. Usable Security: User Preferences for Authentication Methods in eBanking and the Effects of Experience. *Interacting with Computers*, 22(3), 2010.

[234] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX ATC*, ATC '08, 2008.

[235] B. Winters, J. Custer, S. M. Galvagno, E. Colantuoni, S. G. Kapoor, H. Lee, V. Goode, K. Robinson, A. Nakhasi, P. Pronovost, and D. Newman-Toker. Diagnostic errors in the intensive care unit: A systematic review of autopsy studies. *BMJ Quality and Safety*, 2012.

[236] Wireless Cables Inc. AIRCable, last access 2017. `https://www.aircable.net/extend.php`.

[237] K.-P. Yee and K. Sitaker. Passpet: Convenient Password Management and Phishing Protection. In *Symposium on Usable Privacy and Security*, SOUPS'06, 2006.

[238] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek. Improving Application Security with Data Flow Assertions. In *ACM Symposium on Operating Systems Principles*, SOSP'09, 2009.

[239] D.-Y. Yu, A. Ranganathan, R. J. Masti, C. Soriente, and S. Capkun. SALVE: Server Authentication with Location Verification. In *International Conference on Mobile Computing and Networking*, MobiCom '16, 2016.

[240] Yubico. Yubikey hardware, last access 2017. `https://www.yubico.com/`.

[241] Y. Zhang, J. Katz, and C. Papamanthou. IntegriDB: Verifiable SQL for outsourced databases. In *ACM Conference on Computer and Communications Security*, CCS'15, 2015.

[242] Q. Zheng, S. Xu, and G. Ateniese. Efficient Query Integrity for Outsourced Dynamic Databases. In *ACM Workshop on Cloud Computing Security Workshop*, CCSW'12, 2012.