On Mitigation of Client-Side Attacks and Protection of Private Data

Dissertation

Tilman Frosch

29. 10. 2014

RUHR
UNIVERSITÄT
BOCHUM

RUB

Fakultät für
Elektrotechnik und
Informationstechnik

On Mitigation of Client-Side Attacks and Protection of Private Data

Dissertation

zur Erlangung des Grades eines Doktor-Ingenieurs
der Fakultät für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

vorgelegt von

Tilman Frosch

aus Würzburg

Bochum, 29. 10. 2014

Erstgutachter: Prof. Dr. Thorsten Holz
» Ruhr-Universität Bochum
Zweitgutachter: Prof. Dr. Jörg Schwenk
» Ruhr-Universität Bochum

RUHR
UNIVERSITÄT
BOCHUM

RUB

Fakultät für
Elektrotechnik und
Informationstechnik

On Mitigation of Client-Side Attacks and Protection of Private Data

Dissertation

zur Erlangung des Grades eines Doktor-Ingenieurs
der Fakultät für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

vorgelegt von

Tilman Frosch

aus Würzburg

Bochum, 29. 10. 2014

Tag der mündlichen Prüfung: 08. 01. 2015

Erstgutachter: Prof. Dr. Thorsten Holz
» Ruhr-Universität Bochum
Zweitgutachter: Prof. Dr. Jörg Schwenk
» Ruhr-Universität Bochum

Bonus vir semper tiro.

Marcus Valerius Martialis

Abstract

Computers and the Internet have facilitated almost every aspect of our life, except the ability to control on who can access our private data. While information committed to a sheet of paper or to old-fashioned, chemically developed film is relatively easy to control—the medium can be trusted to behave as expected—this is not necessarily the case for information stored and processed on a computer: Users have no meaningful assurance that the systems they use are as secure as claimed or behave as advertised. In recent years researchers have developed several approaches to improve on this front, however many of these approaches effectively mean abandoning existing architectures; the migration path from existing systems towards environments the allow for a tighter control is thus often rather long and in some cases nonexistent.

Instead of abandoning all existing architectures, the work at hand deals with the question how we can stay in control and improve the protection of private information that we store and process on today's *existing* computer systems. We argue that at least three preconditions should be fulfilled to gain the basic ability to retain control over data on a computer system: we need to proactively discover potential attack vectors, analyze existing systems to gain precise knowledge on their inner workings, and, based on both fields of research, must provide effective means of mitigations for existing and novel threads. Using the example of the seemingly harmless SVG format, we show how offensive research can help us to preemptively develop new mitigation mechanisms, before attacks can exploit flaws in existing standards. In the following we analyze the cryptographic protocols of TextSecure, a instant messenger that claims to preserve the confidentiality of messages and the authenticity of conversations. We aim at either verifying or falsifying the developers' claims by fully understanding the system at hand. We show unusual application of cryptographic primitives, attacks, and how these issues can be mitigated. We also follow up on the third precondition outlined above: the ability to detect and mitigate attacks. We present a system that allows for the detection of malicious code on websites directly within the browser. To this end we apply heuristics and machine learning techniques to identify the relevant malicious website elements that we then modify such that the intended attack is mitigated. The methods we present can be applied in many different contexts—from a technical point of view the nature of the data we intend to protect is not relevant. However, in the thesis at hand we aim at the protection of *private* data.

Up to this point we focussed on retrofitting existing systems with protective mechanisms. However, intricate knowledge about a system and attacks may indeed also mandate the decision to abandon the existing technology base and start over, a

situation we also face when developing solution for new fields of technology. This is especially relevant in technological fields that are core-elements of our daily life, such as mobile communication and transportation—fields where *not* using available technology can have a tremendous negative impact on our life. If we have the chance or the obligation to start over, security and the protection of private, personal and personally-identifiable information must not be treated as addons, but must be essential building blocks of every system architecture and implementation, without negatively impacting functionality. Using the example of electric vehicle charging—most likely a very important part of tomorrow's individual transport—we show how the authentic transmission and storage of billing-relevant data and be ensured, without compromising every customer's locational privacy.

Zusammenfassung

Im Gegensatz zu fast allen anderen Aspekten unseres täglichen Lebens ist die Kontrolle des Zugangs zu und der Verbreitung von privaten Daten durch den Einsatz von Computersystemen *nicht* einfacher geworden. Ist Information auf einem Blatt Papier oder einem klassischen, chemisch entwickelten Film noch relativ einfach zu kontrollieren – dem Medium kann vertraut werden, dass es sich exakt so verhält, wie es der Benutzer erwartet – gilt dies nicht uneingeschränkt für Daten, die auf heutigen Computersystemen gespeichert und verarbeitet werden: Der Benutzer hat keinerlei Garantie bezüglich Verhalten und Sicherheit des Systems. Ansätze dies zu ändern resultierten in der Forschung der vergangenen Jahre vielfach im vollständigen Überbordwerfen bestehender Architekturen; ein Migrationspfad von existierenden Systemen hin zu besser kontrollierbaren Umgebungen ist daher häufig lang und nicht in jedem Fall gegeben.

Angesichts dessen beschäftigt sich die vorliegende Arbeit mit der Frage, wie und unter welchen Bedingungen die Kontrolle über private Daten auf *existierenden* Computersystemen verbessert werden kann. Der Autor argumentiert, dass zumindest drei Vorbedingungen erfüllt sein müssen, damit grundsätzlich die Möglichkeit besteht, Kontrolle über die auf einem Computersystem gespeicherten und verarbeiteten Daten zu behalten: proaktive Erforschung potentieller Angriffsvektoren, Kenntnis der genauen Funktionsweise des zu schützenden Systems durch dessen Analyse und, auf basierend auf beidem, die Verfügbarkeit von Erkennungs- und, wenn möglich, Schutzmechanismen. Am Beispiel des offensiven Potential des scheinbar harmlosen SVG-Datenformats zeigt die vorliegende Arbeit, wie proaktive Schwachstellenforschung dazu geeignet ist, neuartige Schutzmaßnahmen zu entwickeln, bevor Angreifer sich global existierende Mängel in existierenden Standards und Software zu Nutze machen können. Die Arbeit beschäftigt sich im Weiteren mit der Analyse der kryptographischen Protokolle des Sofortnachrichtendiensts TextSecure, um auf Basis der vollständigen Kenntnis des Systems eine Aussage über die tatsächlich gebotene Sicherheit zu treffen. Auffälligkeiten in der Verwendung kryptographischer Primitive werden aufgezeigt, ebenso wie daraus resultierende Angriffe und wie diese zu verhindern sind. Im Folgenden behandelt die Arbeit die dritte oben genannte Bedingung für den Erhalt der Kontrolle über auf Computersystemen gespeicherte Daten: Die Fähigkeit Angriffe zu erkennen, ihren Effekt zu begrenzen und sie nach Möglich zu verhindern. Unter diesem Aspekt wird ein System vorgestellt, dass auf Basis heuristischer Methoden und maschinellem Lernen die Erkennung von Schadcode auf Webseiten direkt im Browser des Benutzers erlaubt. Relevante Elemente bösartiger Webseiten können hierbei so modifiziert werden, dass der Angriff nicht zur Ausführung kommt. Die vorgestellten Techniken haben dabei ein weiteres Anwendungsfeld – die Natur der zu schützenden Daten ist hier schlussendlich unter

technischen Aspekten nicht maßgeblich – die vorliegenden Arbeit legt den Fokus jedoch auf den Erhalt der Kontrolle über private Daten.

Bis zu diesem Punkt hat sich die vorliegende Arbeit auf die nachträgliche, systemerhaltende Integration von Schutzmaßnahmen fokussiert. Gerade jedoch in Technologiefeldern, die fester Bestandteil des täglichen Lebens sind, wie z.B. mobile Kommunikationsnetze oder individuelle Mobilität, ist es mit dem nachträglichen Erweitern existierender Systeme um Sicherheitskomponenten nicht getan. Ist die Verwendung einer Technologie für den Benutzer (nahezu) alternativlos, darf Sicherheit und der Schutz privater, persönlicher und personenbeziehbarer Daten kein Additiv, sondern muss integraler Bestand der Technologie sein, ohne in funktionalen Einschränkungen zu resultieren. Wie dieses Ziel zu erreichen ist, wird am Beispiel der im Aufbau befindlichen Ladeinfrastruktur für Elektrofahrzeuge verdeutlicht. Die vorliegende Arbeit zeigt, wie die authentische Übermittlung von abrechnungsrelevanten Daten sichergestellt und dennoch verhindert werden kann, dass ein detailliertes Bewegungsprofil jedes einzelnen Benutzers anfällt.

Contents

1 Introduction

I'm sure they'll listen to Reason.

—Neal Stephenson, *Snow Crash*

Throughout the history of mankind, humans have always created data. Most of it is lost, though, as the amount of data effectively stored was almost negligible compared to the amount of data created. Oral tradition—the transmission of cultural knowledge through vocal utterance—has turned out to be of rather ephemeral, imprecise nature; most, if not all of the data stored this way in prehistoric ages is lost or has been distorted beyond recognition.

One of the earliest forms of *non-ephemeral* data storage is *rock art*, man-made markings on natural stone, which date back to the Upper Palaeolithic period, *i. e.*, the Late Stone Age. The purpose of the data recorded here remains speculative, but its nature is *public*, necessarily accessible to everyone who happened upon it. While it makes sense to assume that also in these early periods of human existence there must have been secret or even private data, none of it has been stored on a *medium*.

The amount of stored data increases with the invention of writing systems. The moment of exit from prehistory, *i. e.*, the beginning of *recorded history*, differs vastly for various regions of the world: forms of proto-writing, *i. e.*, systems of ideographic or mnemonic symbols, have been dated to as early as the 6^{th} millennium BC. In contrast, some civilizations of northern Europe remained in proto-history[1] until the Middle Ages, while mediterranean societies of the classical antiquity are known to

[1] *i. e.*, their history is only documented in the writings of other civilizations

have had writing systems and stored both public and *secret* data. A well-known example of the latter kind can be found in Homer's Ilias: Bellerophon carried a dyptich—two (wax) tablets connected with a hinge that could be sealed—containing the written order for the intended receiver to kill him[2], a fact the carrier of the message was not aware of.

As literacy permeated wider parts of Greek and Roman societies, people did not only commit public or secret data to storage media, but also wrote *private* notes and letters [150, p. 804]. The word *private* itself has its origin in the Latin adjective *privatus*, meaning to be without office, non-public, personal. Put into perspective of the (pre)history of mankind, the concept of *storing private data* is thus rather recent.

Making a large step through time into our modern world, besides public, secret, or private data, we are also confronted with the concept of personal and personally identifiable information—data not necessarily *by*, but *about* a human being. Also, data storage has shifted from hand- or machine-written script to storing digital representations of data. As permanent digital storage was first developed, storing data was rather expensive, thus private data was seldom stored digitally, while secret data was. Private data only found its way on digital storage media with the advances of magnetic storage, that made digital storage space cheap.

Today, humans increasingly often store and process their own data digitally. For this purpose they use computing devices—be they mobile, like smartphones, or stationary, like personal computers—that still do not give them the same amount of control about their data as recording the data on a piece of paper by handwriting or recording it on a strip of old-fashioned, chemically developed film. One could say that we exchange control about our data against the many benefits modern computing and storage devices offer.

We also store more data than ever before, and what is more: The amount of data we willingly produce and store seems almost insignificantly small when compared to the amount of data that is stored *about* us. The places we go, the people we talk to, meet or potentially met, because they were at the same place at the same time, the products we buy or even only look at.

If a flat or house has been burglarized, the inhabitants often feel violated, due to the realization that their presumingly private space is not as safe and as private as they grew to expect it—after all a stranger has just walked in and ransacked the place. Users of compromised computers do not necessarily share these feelings—it's just the computer after all. However, given the large parts of our lives that we keep information on on our mobile and immobile computing devices, the emotional state after

[2] Ilias 6, 165-175

a compromise could probably be expected to be more similar. The effect of having a personal computer compromised is rather comparable to having the contents of ones deposit box and identity card stolen, all the private and sometimes embarrassing family pictures littered on a busy market place, and sometimes also losing access to all of the above. When it comes to their account data, even the sometimes vocal fraction of "I have nothing to hide"-proponents suddenly finds something worth hiding. Similarly, the public accessibility of tax data that is common in some Scandinavian countries is unimaginable to many people just south of the border, who consider this data private.

Extrapolating from this example we could conclude that one aspect of privacy is the ability to control the flows of (personal) information we emit, wittingly or unwittingly. This interest unites people from a large variety of demographics, from well-known actors like Mary E. Winstead, who expressed discontent with the fact that private photos of herself were accessible to non-intended audiences, stating "To those of you looking at photos I took with my husband years ago in the privacy of our home, hope you feel great about yourselves." [232], to the teenagers that danah boyd and Alice E. Marwick interviewed in the course of their research [44]. Their results contrast a 2008 survey by Harris Interactive of 2,089 teenagers that found that only 41% indicated that they were concerned about privacy and security issues when using their mobile [100]. Instead, boyd and Marwick find that all teens have a sense of privacy, although their definitions of privacy vary widely [44].

1.1 Motivation

Irrespective of our notion of privacy or private data, certain preconditions have to be met for us to be able to enforce it. Here we close the loop to computer security, which we opened up with the above example paralleling a burglarized flat and a compromised computer: Computer security is not an end in itself. Much of it aims at guaranteeing that data remains in the expected state: keeping confidential data confidential, private data private and generally ensuring the integrity and availability of existing data—be it stationary or in transfer—as well as, that computations are not manipulated in an undesirable fashion.

An ideal precondition for achieving these goals would be the ability to trust our hardware to behave only in expected ways, *i. e.*, that we must be aware and in control of its complete set of features. The same is required for all software that builds on top of the hardware base, from the microcode that defines a CPU's behavior to BIOS, bootloader, operating system kernel and components, drivers, third-party software.

However, while science has advanced many of these aspects—such as formal CPU verification [127], trusted computing base [231], verifiable micro-kernels and compilers [132] to name just a few—it takes time for these technologies to arrive in off-the-shelf software and systems, as some indeed represent a paradigm shift. In the meantime the prevalent approach is to retrofit security and privacy enhancing technologies or build on top of existing components to raise the stakes for an attacker. Code signing is an approach to increase trust insofar as users receive some assurance that the application they are going to install on their system has indeed been released by the issuer they expect—if none of the signing keys in the certificate chain has been compromised. Modern operation systems apply Address Space Layout Randomization (ASLR), support processors' NX bit and other techniques in an attempt to make it harder for the attacker to guess the right address to jump to after he was already able to inject code or to prevent the execution of undesired code. Methods to ensure the control flow integrity [8] of a program to prevent it from entering unexpected or undesired states have been retrofitted to proprietary systems [73], as has a trusted and capability-based Document Object Model (DOM) in the context of web browsers [104]. The state of network security is in parts similar, for instance, certificate pinning has been introduced as a hotfix for the fact that there are man-in-the-middle attacks and that those attacks are facilitated by the complexity and operational failures of the existing Certificate Authority (CA) system [212].

Assuming that our systems are imperfect, we thus must deviate from the pretense that we can fulfill the above ideal precondition, which demands a trustworthy system. Instead, we formulate a more modular set of preconditions that allow us to work with existing technology without needing to wait for a silver bullet. And while—with this work—we aim at better protecting private and personal information, these preconditions are of a more universal nature, *i. e.*, can serve to better secure any kind of data. We introduce these preconditions below.

Awareness of attack vectors. If we are completely unaware of a certain attack vector, we may be unable to defend ourselves. Thus, as a *precondition A* we need to strive for information that allows us to gain a more complete view of potential threats and attack vectors. This can be done by analyzing existing malicious code, a field that has seen many advances in recent years [133, 229, 230], preemptively pursuing offensive research to gain an advantage over potential attackers [70, 80, 110, 117, 123, 210], or extrapolating from existing vulnerabilities, flaws, and attack methods [124, 125, 234].

Review and analysis of technology. The existence of a solution that claims to protect someones confidential or private information is necessary, but not sufficient as the example of the Heartbleed Bug[3] in *OpenSSL* shows [27]. Source code needs to be read, its availability alone is not enough; software and technology that strives to fulfill a certain purpose needs to be reviewed and analyzed to determine, if it does indeed fulfill this purpose and does not willingly or unwillingly compromise it. In this context we must not limit ourselves to verifying a system's functionality, but also draw from the knowledge we gained on adversarial techniques following *precondition A*. In an attempt to avoid or at least reduce the need for this requirement we could posit a precondition that would mandate the exclusive use of fully formally verified code. While such code exists [132] and this may be a valid approach for future software engineering, it does neither help secure existing technology and software, nor does it prevent flawed protocols or standards. Thus, as a *precondition B*, trust in purportedly secure or privacy-enhancing technology should preferably only be given after the respective technology has received public scrutiny, where possible, and extensive review. Or, to use Claude Shannon's maxim: "The enemy knows the system." So users should better be (made) aware of its properties, too.

After the review of a system or technology we can either arrive at the conclusion that it fulfills its claims and does not contain flaws (to the best of our current knowledge), or that it does not fulfill its claims and is flawed. While some flaws can be fixed and the system amended, this is not possible if a technology or the design of a systems is fundamentally flawed. In this case we can take two different courses of action, which we will detail in the next two paragraphs: *Option 1* is to try and build protective measures on top of the flawed system, an approach made popular, for instance, by the Anti-Virus industry. *Option 2* is to start over and design a better system. A well-known example of the latter approach is Stanford's Clean Slate program[4], which aimed at redesigning the Internet with today's experience and knowledge.

Means of detection or mitigation of a compromise. If we store data that we consider private—like personal photos, tax or bank account details, or online gaming credentials—on a computer system and this system is compromised by an attacker, we are unable to control the (egress) flow of our data. Thus, using our awareness of existing flaws and attack vectors we need to build means of mitigation or at least detection of a compromise to maintain control over our information. We formulate this requirement as *precondition C*.

[3] http://heartbleed.com/
[4] http://cleanslate.stanford.edu/

Security and privacy by design. If we depend on technology in a way that *not* using it would incur severe penalties and significantly alter our lifestyle, as a *general design principle* we need to make sure that this technology empowers us—the user—to make an informed decision about who can access our personal information and does not treat security as an add-on but as a foundation. In the best case, this principle is applied when deciding on the initial design. However, if—after the analysis of a system—we arrive at the conclusion that it is fundamentally flawed and can or should not be amended, *security and privacy by design* is the paradigm that should guide architectural and development decisions during a redesign.

Moving out from the computer system perspective to larger systems, the Global System for Mobile Communications (GSM) is a well-known negative example: Many aspects of our lives revolve around the fact that we have access to mobile communication and in some situations, *e. g.*, when we have an accident, we depend on these technologies to get help in a timely manner. Yet, its cryptographic foundations are flawed and the use of this technology induces a constant penalty on our privacy, as we leave a rather detailed movement trail in our carriers' databases, readily there to be accessed—and out of our control. GSM and similar systems are prime examples for the fact that well-researched cryptographic primitives must be applied in a correct fashion from the beginning and effective means of privacy control need to be deep-rooted within a technology, as it is hard to impossible to impose these means on existing systems and infrastructure later on.

Neither the preconditions we formulated nor the principle of *security and privacy by design* are any guarantee for the ability to protect private—or to that end any—information on computer systems. However, they can be seen as a smallest common denominator that can allow us to achieve the goal of protecting private data on a computer.

1.2 Topics and Contributions

Chapter 2 contributes to *precondition A* above in such that we present an effective mitigation approach for new classes of attacks that resulted from offensive research. Web browsers present a large and almost ubiquitous attack surface—an attack surface that is even enlarged by independently maintained browser plugins and addons that also parse website elements and thus are as susceptible to malicious code, as the browser itself. Armed with a cornucopia of potential attack vectors, malicious actors have found the herd of unsuspecting users to be a veritable source of income from criminal actions. Many of the uses a criminal has for a compromised computer deeply violate the private space this computer constitutes for the affected

user. Or as Krebs puts it: Nearly every aspect of a hacked computer and a user's online life can be and has been commoditized [137]. We explore the effect of small, but in their outcome revolutionary changes that HTML5 introduced to the world wide web: Scalable Vector Graphics (SVGs), which formerly were embedded using `<embed>` or `<object>` tags and traditionally lacked browser support, were now part of the HTML standard and could also be included in novel ways, such as with `<img>` tags, CSS or inline. However, SVGs are more than just an image, although they are considered as such from a security point of view. Rather they should be considered as complete one-file web applications that can contain HTML, JavaScript, Flash, and other interactive content. We discuss several novel attack techniques targeted at major websites, various modern browsers, email clients, and comparable tools. We examine and present how current filtering techniques are circumventable by using SVG files and subsequently propose an approach to mitigate these risks. We also explore how the privacy of users, who rely on application layer web proxies to browse websites anonymously, is impacted by the fact that SVGs and various include methods are handled improperly by many of such proxies.

Chapter 3 contributes to the *precondition B* and approaches a privacy-enhancing technology from an analytical point of view. We analyze the cryptographic protocol of TextSecure, a popular mobile instant messaging application. TextSecure aims at providing confidential, authentic text communication to everyone, who is able to install an Android app from Google's Play store. TextSecure follows the paradigm posited by Bruce Schneier on 2014's TrustyCon that "one-click encryption is one click to many" [81] and aims to be a no-click solution to message encryption. To the best of our knowledge we are the first to analyze TextSecure's protocol, which curiously mixes key establishment and message exchange, but nevertheless can achieve stateful authenticated encryption given that appropriate means of mutual authentication of all parties involved in a message exchange are established. For our analysis we can rely on previous work on attacks against cryptographic protocols—offensive work that contributed to improve *precondition A* in this field.

Chapter 4 focuses *precondition C* from the perspective of web-based threats: We aim at a better detection of attacks against a user's computer and a better protection against a compromise. We discuss the dangers of the modern web, where the focus of attacks has shifted from server- to client-side applications, or, as Moshchuk et al. put it: In the span of just a few years, spyware has become the Internet's most "popular" download [168]. We analyze the attack surface and—taking known insufficiencies of browsers into account—present a novel approach for detecting and mitigating a diverse set of attacks against the DOM tree, protecting users of mobile devices and personal computers alike from the execution of unwanted code. To this end we apply light-weight instrumentation of JavaScript.

Chapter 5 applies the principle of *security privacy by design* to a concrete technological field. We focus on the protection of information that allows to infer habitual behavior about a person: the way we move in (public) space, as recorded by the data points we create when we make a purchase. We explore the field of *locational privacy* using the example of charging an electric vehicle using public charging stations, where it is highly desirable for the vendor *not* to have anonymous customers and thus be able to continue to use existing post-paid billing processes. Thus, we propose an architecture that allows to preserve customers' locational privacy in the absence of anonymous payments. At the time being, a clean slate approach can be recommended in this context, as e-mobility infrastructure is still in its early stages of evolution and many existing approaches primarily aim at functionality, leaving both security and privacy something to be desired.

1.3 Publications

In the course of his work the author contributed to several academic publications. The knowledge gained in the process forms the basis for this thesis, which as a consequence contains both published and unpublished material. This section outlines all publications in chronological order and identifies the author's respective contributions.

IceShield**: Detection and Mitigation of Malicious Websites with a Frozen DOM**
In this publication, we introduce a novel approach to analyze browser-based attacks by demonstrating how dynamic analysis of websites can be accomplished *directly in the browser*. We show how features of ECMA Script 5 can be used to *freeze* object properties, so they cannot be modified during runtime. We implemented a prototype version of IceShield and demonstrate that it detects malicious websites with a small overhead even on devices with limited computing power such as smartphones. Furthermore, IceShield can mitigate detected attacks by changing suspicious elements, so they do not cause harm anymore, thus actually protecting users from such attacks. The paper is joint work with Mario Heiderich and Thorsten Holz, published at the *14th International Symposium on Recent Advances in Intrusion Detection* (RAID), 2011 [106]. The authors thank Jörg Schwenk for supporting the idea to work on this topic and create this very paper. The author's contribution lies in the analysis of malicious websites and the development of heuristics distinguishing elements of code between benign sites and sites containing malicious code. He is also responsible for the implementation of the machine learning approach and the evaluation. Chapter 4 is based on this publication, which has been supported by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02).

The Bug that made me President After analyzing the demonstration website and source code of the Internet voting system Helios, we discovered several small flaws that can have a large security and privacy critical impact. An attacker is able to extract sensitive information, manipulate voting results, and modify the displayed information of Helios without any deep technical knowledge or laboratory-like prerequisites. The aim of this paper was, besides improving a widely-use voting system, to raise awareness in the electronic voting community that online voting applications should at least follow the latest vulnerability mitigation guidelines. E-Voting software driven by web browsers are likely to become an attractive target for attackers. Successful exploitation can have impact ranging from large scale personal information leakage, financial damage, calamitously intended information and state modification as well as severe real life impact in many regards. We published our findings at the *International Conference on E-Voting and Identity* (VoteID), 2011 [108]; the paper was joint work with Mario Heiderich, Marcus Niemietz, and Jörg Schwenk. The author contributed to the attack scenario and provided background and context on electronic and Internet voting systems.

Crouching Tiger—Hidden Payload: Security Risks of Scalable Vector Graphics In this paper, we introduce several novel attack techniques targeted at major websites, as well as modern browsers, email clients and other comparable tools. In particular, we illustrate that SVG images embedded via `<img>` tag and CSS can execute arbitrary JavaScript code. We examine and present how current filtering techniques are circumventable by using SVG files and subsequently propose an approach to mitigate these risks. The paper, published together with Mario Heiderich, Meiko Jensen, and Thorsten Holz at the *18th ACM Conference on Computer and Communication Security* (CCS), 2011 [107], showcases our research into the usage of SVG images as attack tools, and determines its impact on state-of-the-art web browsers such as Firefox 4, Internet Explorer 9, and Opera 11. The author's contribution lies in the analysis of the prevalence of language elements of SVG used in the wild, which allowed to develop an purification approach that has only a negligible negative impact on the representation of benign SVG files. He also contributed to the development of heuristics implemented in *SVGPurifier* and showed that the approach is applicable in a real-world setting through a large-scale study. Chapter 2 builds upon this research and extends it. While the author was not partial to finding the attack vectors in the first place, they are still illustrated in the chapter to underline the adversarial potential of SVG images as offensive tools and to allow for a better understanding of the mitigation approach. The initial research has been supported by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02).

Predentifier: Detecting Botnet C&C Domains from Passive DNS Data The Domain Name System (DNS) is mainly used for benign and legitimate Internet activities. Nevertheless, it also facilitates malicious intentions. Domain names have started to play an increasingly important role in the Command and Control (C&C) infrastructure of botnets. These domains can be added to blocklists or taken down, yet attackers can simply evade the countermeasures by creating hundreds of new domains every day. We propose a framework called Predentifier that combines a host's DNS configuration properties with secondary data to derive a set of distinctive features that can be used to describe the behavior of a host to detect C&C domains at an early stage. We employ methods of statistical learning to determine whether a domain belongs to a C&C server or if it is benign. We further show that it is possible to leverage passive DNS data to identify C&C domains without infringing on employment or customer rights, respecting their privacy. In this publication, together with Marc Kührer and Thorsten Holz, we summarize the author's Master's thesis as a chapter of *Advances in IT Early Warning* [89], a book published by Fraunhofer Verlag in 2013.

Improving Location Privacy for the Electric Vehicle Masses In this technical report, we propose a solution to preserve the locational privacy of users of electric vehicles, who recharge these vehicles using public charging infrastructure. Our approach is based on a group signature scheme that we adapt to the setting of next-generation cars. As every technology roll-out takes place in a certain legal framework, our approach aims at retrofitting privacy-enhancing technology to regulatory circumstances that may have been set with the best intentions for protecting customers, but ultimately threaten their privacy. The report is joint work with Sven Schäge, Martin Goll and Thorsten Holz and has been published as HGI-TR 2013-001 [91]. The author provided the initial idea, devised the approach and the adaption of the group signature scheme together with Sven Schäge and was responsible for the system design and architecture. Chapter 5 builds upon this work and generalizes it, while still using electric mobility as a working example. The work was supported by the German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil).

mXSS Attacks: Attacking well-secured Web-Applications using innerHTML Mutations This paper demonstrates a new class of XSS vectors, the class of *mutation-based XSS (mXSS)* vectors, which may occur in `innerHTML` and related properties. mXSS affects all three major browser families: IE, Firefox, and Chrome. We were able to place stored mXSS vectors in high-profile applications like Microsoft Hotmail, Yahoo! Mail, Rediff Mail, OpenExchange, Zimbra, Roundcube, and several

commercial products. mXSS vectors bypass currently deployed server-side XSS protection techniques (like HTML Purifier, kses, htmlLawed, Blueprint and Google Caja), client-side filters (XSS Auditor, IE XSS Filter), Web Application Firewall (WAF) systems, as well as Intrusion Detection and Intrusion Prevention Systems (IDS/IPS). We describe a scenario in which seemingly immune entities are being rendered prone to an attack based on the behavior of an involved party, in our case the browser. This paper is joint work with Mario Heiderich, Jörg Schwenk, Jonas Magazinius, and Edward Z. Young. It was published at the *20th ACM Conference on Computer and Communications Security* (CCS), 2013 [111]. The author's contribution lies in evaluating the attack surface, *i. e.*, the prevalence of `innerHTML` usage on the web and evaluating the developed light-weight client-side mitigation approach. He was able to demonstrate an exceptionally low overhead, both in terms of increased transfer volume and rendering speed in a controlled, as well as, a real-world scenario.

Communication Reduced Interaction Protocol Between Customer, Charging Station, and Charging Station Management System From a network operators perspective, charging stations are the one network element, where a customer can not only receive energy, but also deliver information into the network. This paper deals with the question how the customer's interaction with the network can be minimized, in an attempt to reduce the potential attack surface. The paper was published together with Karl-Heinz Krempels, Christoph Terwelp, Stefan Wüller, and Sevket Gökay at the *3rd International Conference on Smart Grids and Green IT Systems* (SMARTGREENS), 2014 [138]. The author redesigned the initial protocol from a cryptographic engineering perspective to ensure that customer authentication actually takes place. He also demonstrated how the protocol could be further modified to adequately protect customers' locational privacy. This work was supported by the German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil).

How Secure is TextSecure? This paper [90], which is under review at the time of writing, provides the first complete description of TextSecure's complex cryptographic protocol and the first thorough security analysis of TextSecure. Among other findings, we present an Unknown Key-Share Attack on the protocol, along with a mitigation strategy, which has been acknowledged by TextSecure's developers. Furthermore, we formally prove that—if our mitigation is applied—TextSecure's push messaging can indeed achieve the goals of authenticity and confidentiality. This is joint work with Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. The authors thank Tibor Jager for insightful com-

ments and Dominik Preikschat for his initial analysis and documentation of a previous version of TextSecure's source code in his Bachelor's thesis. The author's contribution lies in the reconstruction of TextSecure's protocol, together with Christian Mainka, and the further development of the initial attack vectors, together with Christian Mainka and Christoph Bader. He also pointed out and described two other issues and devised mitigation strategies for all flaws found in TextSecure. Chapter 3 builds upon this work, which is in submission at the time of writing. This work was supported by the German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil).

2

Security and Privacy Risks of Scalable Vectors Graphics

Fools make feaſts and wise men eat 'em.
—Benjamin Franklin, *Poor Richard's Almanack*

In the introduction to this work we phrased the active research into novel attack vectors and consequentially the development of better mitigation techniques as *precondition A*, necessary to protect private data on computer systems.

Classical image formats, like JPEG or PNG, were just that: images. While one could of course try to interpret such images as arbitrary data, web browsers generally did not, although image parsers themselves contained vulnerabilities in the past. With the inclusion of Scalable Vector Graphics (SVGs) into the HTML5 standard this paradigm changed. We show the severe implications on users' security and also discuss privacy aspects of the fact that a living, and thus constantly evolving, standard is not always fully understood by all developers. We discuss various novel attack vectors and show how they can be mitigated, thus enlarging the foundation of defensive work.

2.1 Introduction

One of the factors behind the huge success of the World Wide Web is its ability and capacity for viewing image files within a web browser. Compared to the text-only formats, an image can convey considerably more information. A typical browser supports many different image file formats, such as JPEG, PNG and GIF files, whilst

Figure 2.1: A classic GhostScript/SVG example

the vast majority of websites on the Web contain at least one graphic in either one form or another. Since image files are complex and need to be parsed and rendered before they can be displayed by a browser, it comes as no surprise that the images have security implications. To give an example, there were several cases in the past where the validation routine of image libraries contained security flaws leading to vulnerabilities [1, 2, 4]. For this reason, we need to consider the risk of images as the attack vectors.

One image format that had received limited scrutiny and little attention from the web development community is *Scalable Vector Graphics* (SVG [5]). This family of file formats comprises several specifications and specification drafts for composition and rendering of the vector based images and graphics. SVG is based on XML and was first published by the W3C in 1999. SVG images have not gained much traction from the web developers, as the support provided by major browsers was not consistent and only a small subset of SVG features had been known to work reliably on a sufficiently large base of the web browsers. Browsers, like Firefox 1.5, already supported a decent subset of SVG features in November 2005, showcasing SVGs such as the famous *GhostScript Tiger* shown in Figure 2.1. This particular image is often used to illustrate the abilities of the vector based graphics to display complex structures. Other browsers, namely Internet Explorer, did not support SVG, unless a user installed an external plug-in.

All this has significantly changed with the appearance of HTML5: the W3C and WHATWG draft specifications for HTML5 *require* modern web browsers to support SVG images' embedding in a multitude of ways [114]. SVG images can now for example be engrafted in a given document either in the classical way via specific tags such as `<embed>` or `<object>` tags, or in the novel ways such as with `<img>` tags or inline in any HTML5 document. Internet Explorer 9 currently supports a large subset of SVG features as tests with the tech previews and available beta versions

show. Furthermore, both Firefox and Webkit-based browsers, such as Chrome and Safari, as well as Opera, provide thorough SVG support.

Hitherto, SVG is mainly used in the context of screen and print design, as well as in the cartography and medical imagery, which all can be attributed to the lack of proper browser support [163]. This is however expected to change, owing to the SVG support being implemented in all modern web browsers, consequently lifting SVG files from being a niche format for W3C-compliant and plug-in-equipped browsers only, to a widely used toolkit for enhancing images, diagrams and rich-text documents across the board. Depending on the rendering client's capabilities, an SVG file can contain interactive and animated elements. Processing events and raster images, embedding videos, and rich-text are also feasible. Contrary to popular belief, SVG files should thus not be considered to be plain images or animations, and it is necessary to treat them as fully functional, one-file web applications capable of potentially containing HTML, JavaScript, Flash and other interactive code structures.

2.1.1 Contribution

In this chapter, we elaborate on the security risks of improper SVG handling. Before we discuss ways to mitigate these risks, we introduce several novel attack techniques of using SVG images to target modern, real life web applications (such as MediaWiki installations like Wikipedia, DeviantArt, and other high profile websites), as well as their unsuspecting users. Specifically, in Section 2.3 we present an *Active Image Injection* (AII) attack, in which arbitrary JavaScript code can be delivered via SVG files. Several other attacks such as SVG-based cross site scripting attacks or SVG chameleons (*i. e.*, files that are interpreted differently depending on how they are opened) are also delivered. AII attacks are particularly important, since they are caused by faulty SVG implementations in modern browsers, thus affect all websites allowing users to embed external images. Our initial research thereby significantly extends and partly falsifies the information available in the Browser Security Handbook, so far only covering risks connected to browser-deployed SVG in plugin containers. Furthermore, we discuss how current state-of-the-art filtering techniques are deceivable via using SVG files.

The basic idea behind all of our attacks is the fact that SVG files can accommodate active content, whereby browsers actually interpret this content, as it is standard-compliant. This idea is related to similar attacks that take advantage of code embedded in document formats [18, 60, 153] and we show that SVG images can be turned into an attack vector. In addition, we demonstrate the damaging potential of SVG files embedding arbitrary data formats and show how this property can be used to carry out attacks using Adobe Reader, Java Runtime Engine and Flash

Player vulnerabilities. We discuss the impact of our attack on modern browsers such, showing that especially inline SVG grants new possibilities for bypassing website- and browser based XSS filters.

To mitigate the risks introduced by SVG-based attacks, we debate and evaluate several defense strategies. We showcase a filtering solution that is capable of removing potential malicious content from a given SVG file. Our approach does not break the functionality of the core features of fully interactive and descriptive SVG images. Instead, we extend an existing and widespread filtering software to support filtering of illegitimate and malicious content from SVG files, without damaging the benign file structure and contents. We have formerly implemented a prototype of the system and tested it with 105,509 SVG images obtained from Wikipedia. We found that we can filter 98.5% of the files without causing any difference in the visual appearance of the image, and for the remaining 1.5% we determined the visual deviation to be negligible in more than half of the cases.

2.1.2 Outline

The remainder of this chapter is structured as follows: in Section 2.2, we first provide technical background on the SVG file format and discuss various deployment techniques, as well as, related work. In Section 2.3 we present several novel attack vectors against websites and browsers, and discuss their impact. In the following Section 2.4 we show how SVG-based attacks can be effectively mitigated and present a large-scale evaluation of our approach in Section 2.5. We end the chapter with a discussion and a summary.

2.2 Technical Background

This section provides a brief overview of the SVG file format and discusses the attack surface enabled by this image format, *i. e.*, we illustrate the different ways available for an attacker to send arbitrary SVG files to a victim.

2.2.1 Overview and Benefits

The *Scalable Vector Graphics* (SVG) file format was introduced in 1999 when it was published by the W3C in an attempt to combine the best of both the specification drafts for *Precision Graphics Markup Language* (PGML) developed and published by Adobe, and the *Vector Markup Language* (VML) developed and published by

Microsoft, Autodesk, Hewlett Packard, and others, all in 1998. SVG is a vector graphics format, *i. e.*, it uses geometrical primitives such as points, lines and curves to describe an image, while it supports both static and dynamic content. The above mentioned static content and dynamic behavior are described in an XML-based format, which implies that SVG files are in fact text files.

The impact of SVG on the Internet can be described with the following characteristics:

- **Scalability**: SVG files are, as the name indicates, *scalable* based on their nature as vector graphics. This means that graphical output devices of any size can render SVG images without significant information loss or facing deficiencies in display quality. In times of websites' inhomogeneous output devices such as browsers, feed- and screen readers, smartphones and *Wireless Markup Language* (WML) compatible cellphones, this enables web developers to publish rich online documents without having to worry about the screen dimension of the device requesting the document.

- **Openness**: Unlike classical, raster-based image formats, SVG files are neither stored in a binary format nor is there a compression scheme rendering the actual content of the file unreadable for the human eye. SVG images can be enriched with metadata and comments, so that a (handicapped) human as well as a program (*e. g.*, a search engine or comparable parser) can effecively extract relevant information from the file in question. In this case accessibility is ensured by storing more descriptive information in a given file, compared to the rather limited possibilities of image comments provided by GIF and PNG files, or the embedded *Exchangeable Image File Format* (EXIF) data in RAW and JPEG images. Note that gzipped SVG files (also know as SVGZ) create an exception, for they use lossless compression as means to reduce the file size.

- **Accessibility**: Related to the aforementioned openness, an SVG image can be enriched with sufficient metadata and information to be *Web Accessibility Initiative* (WAI) compliant, *i. e.*, visually impaired users can extract relevant information by having their tools parse and read the metadata embedded in the SVG. Furthermore, screen readers can (theoretically) parse SVG data and describe the shapes and visuals used by the image, allowing a broader range of users to benefit from its contents. In contrast, raster-based images are void of this kind of support.

- **XML compliance**: SVG images are supposed to consist of valid XML. This commissions browsers to pre-validate the structure and well-formedness of an SVG image before rendering its contents. These validation possibilities ensure consistent quality of SVG images, while at the same time preventing malformed

SVG files from being shown and causing damage to the processing device. Yet the validity aspect and well-formedness constraint do not apply in case of the inline SVG as a part of HTML5 documents, which we inspect in greater detail in Section 2.3.4.

The SVG family consists of several members and we use the following three file types as examples in later sections:

- **SVG Full 1.1**: SVG Full describes the full SVG feature set including 81 different SVG elements and tags. The specification is designed without a special focus on the devices parsing the SVG data.

- **SVG Basic 1.1**: SVG Basic is supposed to deliver a subset of the SVG Full specification to ease the implementation for developers of browsers for PDAs and handheld devices. SVG Basic only provides 70 of the 81 SVG elements specified in SVG Full 1.1. Contrary to SVG Tiny 1.2, the SVG Basic 1.1 features also include support for SVG fonts.

- **SVG Tiny 1.2**: SVG Tiny is specifically designed for smartphones and similar mobile devices with limited computing, rendering, and display capabilities. The subset of allowed SVG elements and tags has been reduced to 47 elements. SVG Tiny also ships several exclusive possibilities for event binding and external resource loading which we discuss in Section 2.3.

Additionally, the SVG specification provides interface descriptions for an *SVG Document Object Model* (DOM), which implies that SVG files also offer some dynamic capabilities. Users can create SVG files capable of providing event handling, effects, time-based changes and animations, as well as zoom effects and other helpful display enhancements. A large set of filters can be applied to the elements of SVG files to even more greatly increase the possibilities for image transformation and animation.

The ability to combine SVG with the *XML Linking Language* (XLink) features allows SVG files to link elements to other elements in the same image file, other image files or arbitrary objects referenced via *Uniform Request Identifiers* (URIs). Furthermore, these image files support the implementation of *International Color Consortium* (ICC) and *Standard Red-Green-Blue* (sRGB) color profiles, allowing the embedment of arbitrary content such as Flash, PDFs, Java and HTML via the `<foreignObject>` element.

2.2.2 HTML, SVG, and XML

Being historically an XML-based language, processing of SVG documents has been quite different from the way browsers process classic HTML websites. For instance, a slight violation of the XML syntax, such as missing closing tags or attribute value quotations, typically cause SVG processors to exit with an error. However, with the integration of SVG capabilities into modern browsers, this strict parsing approach got amalgamated with their more tolerant way of processing HTML, CSS, JavaScript and the like.

This mixture is causing browsers to process SVG through using two different processing modes: an HTML processor engine for CSS and JavaScript contents, and an additional SVG parser supporting XML-specific features like XML transformations (*e. g.*, XSLT), XML Entity resolution, and tracking of XML Namespace bindings. Depending on the particular website's style of using SVG, the browser switches between the two processing modes on the go. For instance, encountering an inline `<svg>` tag within an HTML5 document causes the browser to switch from HTML mode to XML/SVG mode. Vice versa, if the browser encounters an HTML-specific tag (*e. g.*, `<p>`) within an SVG mode context, it automatically closes all open SVG elements, switches to HTML mode, and renders the given tag.

As we show in the following sections, this approach is error-prone and may cause a lot of SVG-related vulnerabilities in most state-of-the-art browser engines.

2.2.3 Deployment Techniques

The capabilities, in terms of scripting and content inclusion of SVG files, strongly depend on how they are embedded in a website or loaded by the browser attempting to display them. In this section, we focus on five diverse manners of SVG files being deployed by a webserver or web application. In addition, we outline the attack surface we have discovered in connection to the five deployment techniques. The specific attack vectors we use are discussed in detail in Section 2.3, followed by Section 2.4 focusing on how to mitigate and defend against those attacks.

1. **SVG deployed via uploaded files**: A large number of tested web applications (*e. g.*, MediaWiki and Wikipedia, OpenStreetMaps, DeviantArt, OpenClipArt, and several other free image hosting services) consider SVG files to be equivalent to raster images such as PNG, JPEG, and GIF files in terms of security implications. MediaWiki and Wikipedia claim to block the upload of SVG files containing script code, but we did manage to easily bypass this restriction. As we show in the next section, SVG files should be displayed and executed with a heavily limited set of features to prevent *universal XSS*

attacks, since these files might contain scripts, embed arbitrary content and process events. In addition, we discovered alternative course of action for outmaneuvering the capability limitations of current browsers, which are used to protect sensitive DOM properties such as a website's cookies. Those are discussed in detail in Section 2.3.2.

2. **SVG deployed via CSS backgrounds and *img* tags**: This way of deploying malicious SVG files can be considered as the most dangerous and effective attack, granted that the majority of the web applications judge `<img>` tags as part of user generated HTML to be harmless: `<img src="evil.svg">`.
   Filter software, such as the HTMLPurifier [235], OWASP AntiSamy [68], and similar tools whitelist image tags and a large number of web applications allowing user-generated HTML are prone to be sensitive to a novel class of attacks we term *active image injections*. Again, we underline that SVG files should be displayed and executed with a heavily limited set of features to prevent universal XSS attacks. In Section 2.3.2 and 2.3.5, we particularize on the attack vectors we discovered through using this presumably harmless way of deployment, and outline an innovative method of attacking browsers and high traffic web applications.

3. **SVG deployed via inline SVG**: The HTML5 specification draft suggests the web browsers to support websites providing *inline SVG*. This means a developer and an attacker are equally able to inject arbitrary SVG content right into the markup tree of a HTML document. The browser will then switch its parsing mode, use an intermediary layer to parse the (possibly non-well-formed) SVG content, clean it up, pass it on to the internal XML parser and layout engine, and then commence parsing and rendering the remaining optional HTML content. The last step likely includes even more inline SVG elements [69] that are capable of interacting with the already parsed content. In Section 2.3.4, we illustrate how this facilitates XSS filter bypasses of existing websites, filter libraries, and most importantly browsers and comparable user agents.

4. **SVG deployed as font file (*SVG Fonts*)**: The SVG standard specifies several possibilities to create font files completely consisting of SVG data [3]. Modern browsers allow their inclusion via CSS and the `@font` directive. In case when the browser supports SVG fonts, for every character with an SVG font assigned, the parser checks whether the character has a representation as an SVG path/glyph data and applies this to the view port if possible. SVG fonts provide a prominent range of features for detailed and complex font formatting, Unicode support, alternative glyphs, default behavior for missing glyphs, and

more. We detected attack vectors allowing the deployment of arbitrary plug-in content via SVG fonts working on a variety of desktop and mobile user agents.

5. **SVG deployed via *iframe*, *embed*, or *object* tags**: The attack surface is comparably large to the one for the classic XSS and does not differ much from the regular `<iframe>` and `<script>` injections. It will thus not be discussed in more depth in this chapter.

2.2.4 Related Work

Not surprisingly, being one of the most common problems in the area of web security, the Cross Site Scripting (XSS) problem has received a lot of attention during the last decade. [37, 40, 98, 126, 129, 131, 151, 176, 194, 220, 226]. On the offensive side, several different kinds of attacks were studied [40, 129, 151]. Approaches to prevent XSS attacks include information flow and taint tracking [98, 176, 194, 220], and analysis on the client- or server-side [37, 131, 220]. John's dissertation elaborated on the attack and defense techniques in detail [126], while Phung et al. presented specific defense techniques against client-side and JavaScript-based attacks [193]. Nevertheless, none of these works dealt with the threat of malicious image files in the JavaScript execution context. In this chapter, we introduce a new way to mitigate innovative attacks that in turn highlight the fact that in the era of HTML5, even a previously unsuspicious `<img>` tag may introduce security vulnerabilities due to the tight integration of SVG images into the modern browsers.

One exception is the work by Barth et al., who discussed attacks and mitigations around faulty and jaunty content sniffing [24]. Deprecated browsers such as Opera 9 and Internet Explorer 6 allowed to execute JavaScript by combining image tags with JavaScript URIs, but none of the tested modern browsers supports this kind of render behavior anymore, as this particular attack vector has been recently fixed. In contrast, the risks of Cross Site Scripting and related attacks against browsers induced by SVG images have not yet been investigated.

SVG as a subject itself surfaced rarely in the scientific security community. One notable exception is given by Damiani et al. [71], who dealt with access control requirements of parts of SVG files. Their assumption was that SVG files containing sensitive personal information should be rendered differently for different viewers, hence requiring some parts of an SVG document to be deleted (or kept encrypted). However, they did not manage to cope with the threat of misusing SVG files as attack vectors. In the same line of work, Mohammed et al. [161, 162, 163] investigated the use of SVG images in medical contexts, where certain security guarantees have

to be granted for sensitive information contained in an SVG image. Again, their publication did not resolve the offensive use of SVG files.

One area of research closely related to the results presented in this chapter deals with the problem of code embedded in document formats. For example, Backes et al. showed that maliciously prepared PostScript files can be used as an attacker vector [18], and Checkoway et al. discussed malicious TEX files that can, among other consequences, lead to an arbitrary code execution and data exfiltration based on TEX's Turing-complete macro language [60]. Even pure text files might contain shellcode as shown by Mason et al. [153]. We continue this line of scientific enquiry and present attacks related to SVG images and show how they can be mitigated.

An orthogonal area of research are alternative browser designs [26, 65, 96, 223]. These browsers explore how the security of state-of-the-art browsers can be improved, for example by creating separate protection domains. The results presented in this chapter need to be taken into account when designing more secure browser and especially the fact that `<img>` tags might lead to suspicious content have to be considered, as we detail below.

2.3 Attack Vectors Using SVG Files

Based on the prerequisites discussed in the previous section, we now introduce several different attacks based on SVG files and discuss their security impact.

2.3.1 Responsible Disclosure and Ethical Aspects

We describe several novel attacks related to SVG files and their security impact, ranging from universal XSS attacks to triggering vulnerabilities based on SVG images. Presenting such attacks is obviously an ethically sensitive area and one question that arises is if it is acceptable and justifiable to publish the attack details. In the following, we describe most attack vectors from a high-level point of view and do not present all implementation details. Furthermore, we have contacted all major browser vendors and informed them about these problems; several reported problems have already been fixed. As a result, an attacker cannot easily take advantage of these identified attack vectors.

While these attack vectors are interesting, our findings are not an end in themselves. They allow us to introduce an approach to mitigate the attacks presented in this chapter based on removing suspicious content from SVG files at the server side in Section 2.4. Thus, we effectively extend the set of tools available to defend against attackers.

2.3.2 Local JavaScript Execution and SVG Chameleons

One of the least sophisticated attack techniques (that is still rather likely to work in real-world scenarios) is tricking the victim into saving an SVG image from a website and opening it later on for repeated viewing pleasure. There are only a few ways for technically less affine users to tell a classic raster-based image (PNG/JPEG/GIF) apart from an SVG image. Once saved locally and double-clicked, the browser will open the file—since most users do not have a dedicated software installed that changes the application to handle the SVG MIME type. The SVG file is consequently opened from a *file* URI and in case it contains JavaScript, this code will be executed in the same context. Depending on the web browser the victim is using, the JavaScript can then attempt to read other files from the hard-disk or neighboring directories, and cause a data leakage incident. A thrifty adversary can cause the locally running JavaScript to load an applet from an arbitrary domain, thus even bypassing many of the security restrictions modern browsers apply for local script execution.

Similar attacks could be performed with *SVG Chameleons*, *i. e.*, files containing both SVG and HTML content. Using in-line XML transformation (XSLT), we managed to craft an SVG file that acts like an image if embedded via `<img>`, CSS or similar ways, but unfolds to a full stack HTML file containing no SVG elements anymore as soon as opened directly [61]. This attack works with Gecko-based browsers, since it appears to be the only layout engine supporting in-line XSLT in SVG files. The attack would involve uploading an SVG Chameleon to a website such as Wikipedia, and trick the victim into right-clicking the image shown embedded and choosing to view the original. As soon as that happens, the XSLT will transform the SVG into an HTML file and execute embedded script code or worse[103].

Interestingly, some browsers such as Firefox do not allow cookie access in case the SVG file is being opened directly. This is especially important in cases where an attacker can upload SVG files to the same domain a targeted user is being logged into. The reason for that limitation is a different handling of the *SVGDOM* compared to the regular website's DOM. The SVGDOM does not know properties such as `document.cookie` or even `document.body`. We discovered ways to get around this limitation, though, by having the SVG create a `<foreignObject>` tag containing an Iframe loading the affected website. After the `onload` event of the Iframe, we injected JavaScript into its scope capable of extracting sensitive data such as the cookies and bypassing the more or less unconscious security restrictions.

2.3.3 Facilitating Cross Site Scripting Exploits

SVG images provide many options for rather uncommon ways to execute JavaScript. Typical web developers are not necessarily aware of these ways and filter software that aims at protecting websites agains XSS attacks does consequentially not cover these vectors. As Heiderich [104] detailed in his PhD thesis, SVG Tiny, for instance, allows to execute JavaScript using a `handler` element with an `event` attribute. Listing 2.1 illustrates this example. If the event that is assigned to the handler is specified as `load`, the handler element's text content will be executed as JavaScript without user interaction. Filters following a blacklist approach are typically unaware of these unusual ways of code execution and are thus unable to detect and prevent this kind of attacks.

Listing 2.1: Example for uncommon SVG-based JavaScript execution via <handler> tag

```
<svg xmlns="http://www.w3.org/2000/svg">
 <handler
  xmlns:ev="http://www.w3.org/2001/xml-events"
  ev:event="load">
    alert(1)
 </handler>
</svg>
```

Heiderich also demonstrated another, equally uncommon way of embedding malicious JavaScript in SVG files, which is shown in Listing 2.2. By utilizing SVG's `<set>`, we can dynamically equip an `<feImage>` tag with an `xlink:href`, which points to a `data:` URI. While this kind of image element is intended to apply overlay effects for SVG elements using external resources, the URI in the example contains another SVG image that in turn contains malicious JavaScript. The script is executed immediately on loading the `<feImage>` tag.

Listing 2.2: Example for uncommon SVG-based JavaScript execution via <set> tag

```
<svg
 xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <feImage>
  <set
   attributeName="xlink:href"
   to="data:image/svg+xml;charset=utf-8;
   base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53
   My5vcmcvMjAwMC9zdmciPjxzY3JpcHQ%2BYWxl
   cnQoMSk8L3NjcmlwdD48L3N2Zz4NCg%3D%3D"/>
 </feImage>
</svg>
```

Among others, these ways of executing JavaScript from within an SVG file were used to bypass the filter used by the software MediaWiki, which is the most commonly used open source wiki software and, among many others, the platform used by Wikipedia.

2.3.4 Facilitating Filter Bypasses

One discriminating feature between the rendering behavior of HTML on the one hand and XHTML- and XML-based websites and documents on the other is the handling of entities in plain text tags. These tags are HTML elements that typically contain plaintext information, such as `<script>`, `<style>`, or `<noscript>`, `<noframes>` and `<nostyle>` tags. In HTML documents, entities such as `&#x61;` are treated as such, while in XHTML and XML documents these entities will be treated as their canonical representation (in our example: `a`). This leads to the situation that in XHTML/XML contexts the code `<script>&#x61;lert(1)</script>` will lead to the execution of the `alert` method, which, in contrast, would simply lead to an error indication by the script engine in in the context of a HTML document.

As SVG files are regular XML documents, they are interpreted in the former way. From a web security point of view it is worth noting that, for most browsers, this behavior also applies to inline SVG. In consequence, the handling of entities as their canonical representation can be transfered to regular HTML documents, under the condition that somewhere in their markup tree they contain an opening `<svg>` tag. While the `<script>` example above will not execute in an HTML document, `<svg><script>&#61;lert(1)<p>` will.

For reasons of usability, browsers' parsers are traditionally rather tolerant about well-formedness. This also extends to inline SVG, which in consequence needs neither attribute delimiters, nor requires balanced tags or closing tags. The `<p>` element in the last example is enough for the parser to conclude that the inline SVG ended and the following should again be interpreted as HTML. In an attempt to correctly interpret our malformed code, the browser thus closes `<svg>` and `<script>` by itself, which necessarily triggers the execution of the `alert` method. When we initially published these findings this technique in combination with an injection, allowed for bypassing most of the common XSS filters.

2.3.5 Active Image Injection

Several ways of abusing SVG files to execute JavaScript in situations where no script execution should happen at all have been fixed after we reported them back

to browser vendors. Some vendors even completely restricted access to the DOM from an SVG context. This makes it very hard to execute same-domain JavaScript from within SVG files delivered via CSS, image tags, CSS fonts, or other ways in which browsers deliver images. Even if a web browser can be tricked into executing JavaScript via SVG deployment methods unintended for this, the script will run in the context of `about:blank` and cannot get access to the deploying website's DOM. This effectively disables XSS attacks, since they require their payload to execute on the targeted domain and not on a bogus fully qualified domain name (FQDN) such as `about:blank`.

A yet unsolved problem for several state-of-the-art web browsers is plugin content. The Opera browser, for instance, allows to use SVG files in order to deploy plugin content such as Flash, Java, and PDF files, depending on which plugins have been installed and registered on the user's system. This does not only hold for SVG files embedded via tags such as `<embed>`, `<object>`, or `<iframe>`, but also for `<img>` tags and CSS. This means that an attacker can execute arbitrary plugin code on a victim's machine *without* any user interaction by just having the victim browse a website containing an image tag. The image can be delivered from any arbitrary domain, thus most high-traffic websites and web applications allowing user generated image content are affected by this problem.

The Opera security team has been informed about this issue in summer 2010, but so far this vendor did not immediately address the issue with a sufficient fix. This left applications such as Facebook, Google Mail, Yahoo! Mail, and many other websites prone to this kind of *Active Image Injection* attack—in case their users visited the page with Opera version 9 to 11. An example showcase had been set up to demonstrate the severity of the vulnerability, also trying to enforce an urgent fix from the Opera team [102].

Early versions of the Firefox 4 beta browser were prone to AII attacks as well, but these bugs have been spotted and fixed with Firefox 4 Beta 9, and did not surface in the final release version. Nevertheless, all browser vendors should monitor the security boundaries of SVG deployment closely, since small changes can cause vulnerabilities affecting a majority of web applications at once.

2.3.6 Browser Vulnerabilities

So far we covered attacks utilizing malicious SVG files to be instrumented in attacks against websites. We demonstrated how SVG files can be used to facilitate XSS attacks that bypass existing and well-configured filter mechanisms and security best practices, such as encoding user-generated data into HTML entities. SVG files can

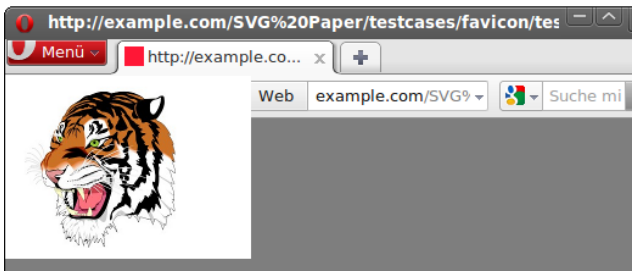


Figure 2.2: An SVG containing plugin content delivered via favicon

have other purposes for attackers, though, and be used to leverage attacks against the browser itself, or even against the underlying operating system. During our tests, it became evident that especially complex SVGs or SVG chameleons containing executable plugin code have the potential of easily crashing web browsers. We observed and reported several cases of memory corruption occurring in state-of-the-art browsers, which were caused by faulty or incomplete implementation of SVG features, or interference effects between the browser components delivering the SVG data and other components delivering embedded plugin code and Iframes.

One significant example showing the dangers of SVG files when used as attack tools against browsers is a specific bug in Opera version 11.50 24581. This version marked a turning point in the Opera browser history, since it was the first officially released version supporting inline SVG so far. We investigated an attack scenario where an attacker creates a website providing an SVG image as *favicon*. This SVG image deployed malicious content in form of a Flash file and a Java applet, as well as, an embedded PDF file. When opening the malicious website, the Opera browser attempted to load the *favicon* to decorate the loaded page's tab and address bar, as shown in Figure 2.2. Although this context should never execute plugin content or JavaScript at all, the browser started to play the Flash video we used for testing, and delivered the applet and PDF file—within the address bar as can be seen in Figure 2.2. The code was executed in the browser context, thus an exploit like this could easily haven been escalated from a proof of concept to a full attack, demonstrating how arbitrary vulnerabilities in browser plugins can be triggered via image files.

Furthermore, SVG-based attack vectors should be considered relevant for another category of software as well: mail clients such as Thunderbird and Opera Mail make use of the same (or only slightly modified) rendering and layout engines as their respective browser counterparts. We tested the latest versions of Opera Mail and Thunderbird 3.3, and discovered that both products allow usage of inline SVG inside

HTML mails. One attack vector caused Thunderbird 3.3 alpha 3 to automatically store an SVG file in the temporary folder, open it, and execute JavaScript in the *file://* context. The only required user interaction is a click on an arbitrary part of the displayed mail body. Attacks like this can be used to place malicious software or to steal sensitive information from the */tmp* folder or other directories, depending on the location of the SVG file and the post-exploitation techniques used by the attacker.

2.3.7 Compromising User Privacy

In the previous sections we discussed attacks against websites and how SVGs facilitate the exploitation of browser vulnerabilities. In the following we focus on a privacy-enabling technology that relies on the ability to completely recognize and correctly parse all elements of all web standards the various browsers interpret: application layer web proxies. Users turn to web proxies for a variety of reasons, prominently among them the desire or need to cloak their identity against the website they are visiting. Application layer web proxies—sometimes still referred to as CGI proxies, although few are still based on the actual Common Gateway Interface—are a widespread technology that is based on the rewriting or wrapping of HTML elements. These proxy servers are web applications that present a web form to users, where they can enter a URL of the website they want to browse. The proxy then retrieves the requested URL and presents it to the user. If the target webpage contains hyperreferences (`href`) the proxy rewrites these elements, such that the `href` again points at the proxy, which will retrieve the target when the user's browser requests it. Popular examples of such proxies are HideMyAss[5], KProxy[6], webproxy.net, or MetaGer[7]. While these are popular services backed by commercial entities, a hidden champion among application layer proxies is Glype[8], a free PHP-based proxy that has more than 4000 publicly accessible instances on the Internet at the time of writing. Among these instances is also `webproxy.stealthy.co`, a platform originally developed to facilitate the dissemination of information during the Arab Spring. Glype is published in version 1.4.6 at the time of writing.

The promise all these services implicitly or explicitly make to the user, is that the operator of the target website will be unable to identify the user browsing the website through the proxy, as each and every request to any target website will be redirected through the proxy. Thus, the use of such a proxy promises a form of privacy to the user, *i. e.*, anonymity towards the target website. As HTML is a living standard,

[5] https://www.hidemyass.com/
[6] https://www.kproxy.com/
[7] https://metager.de/
[8] https://www.glype.com/

fulfilling this promise is a constant struggle to keep up with each and every new aspect not only of HTML, but also of CSS, SVG, and all other languages and file formats parsed by the user's browser.

Listing 2.3: Example for proxy circumvention using SVG and data URI

```
<img src="data:image/svg+xml,&lt;svg%20xmlns='%68ttp:%2f/www.w3.org
    /2000/svg'%20xmlns:xlink='%68ttp:%2f/www.w3.org/1999/xlink'&gt;&lt
    ;image%20xlink:hr%65f='%68ttp:%2f/leakme.com/svg-via-data'&gt;&lt
    ;/image&gt;&lt;/svg&gt;">
```

Anonymous users are undesirable for various groups of content providers on the Web. For the operator of a website, who wants to identify users that pass through a proxy, or an advertising company, whose revenue depends on the fact that it can serve the right ads to the right target audience, the incomplete understanding of SVG's capabilities by proxy developers and operators offer a way to compromise users' privacy. Listing 2.3 and Listing 2.4 represent two examples that are not correctly rewritten by any application layer web proxy we evaluated. The latter example is rather surprising, as it is a very straight-forward use of SVG syntax and was only slightly varied from the example found in Mozilla's Developer Network documentation[9]. While HideMyAss or KProxy attract high traffic volumes, the bypass of Glype is worrisome due to its high proliferation on the Internet and use in contexts that aim at high profile targets, like `webproxy.stealthy.co`, that initially aimed at protecting dissidents against repressive governments. The examples listed here only represent a current subset of potential occurrences, where developers are unable to keep up with a standards evolution at all times, and are easily mitigated by amending the rewriting engine. However, they illustrate that continuously representing a moving target completely and correctly within a static approach is continuously prone to failure.

Listing 2.4: Example for proxy circumvention using SVG fill and mask attributes

```
<svg>
  <defs>
   ...
    <mask id="Mask">
      <rect x="0" y="0" width="200" height="200" fill="url(https://
          leakme.com/svg-fill)"  />
    </mask>
  </defs>
  <rect x="0" y="0" width="200" height="200" fill="green" />
  <rect x="0" y="0" width="200" height="200" fill="red" mask="url(
      https://leakme.com/svg-mask)" />
</svg>
```

[9] `https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Clipping_and_masking#Masking`

2.4 Mitigation Techniques

In the following section, we cover mitigation techniques to fend the attacks presented in the previous part. We start with a discussion of the common XML filtering and sanitization techniques, point out which problems occur when SVG is being used in modern web browsers, and list arguments as to why the classic and formerly approved approaches cannot be applied to SVGs used on the Internet. Based on these insights, we introduce and discuss *SVGPurifier*, a PHP-based, server-side SVG filter software we have developed to mitigate the identified attacks. What is more, we outline a set of recommendations for browser vendors addressing non-standard behavior that causes security problems with malicious SVG images and inline SVG parsing. Some of the listed issues have already been adopted by browser vendors during the preparatory phase of the paper we published on these issues [107].

2.4.1 XML Sanitization

The most common approach for verifying an XML document's validity is the use of *Document Type Definitions* (DTDs), *XML Schema*, or *RelaxNG* descriptions. All of these languages authorize a precise specification of which XML elements, attributes, and other tokens may be used within a specific XML document. For instance, the SVG Tiny specification provides a RelaxNG description of all XML elements and attributes that may be used in "Tiny"-compliant SVG documents. We have investigated these descriptions for practical usage in terms of removing malicious contents from SVG files. Unfortunately, we determined that their capabilities of restricting SVG contents are more focused on the XML documents' *structure* rather than restricting the content values of the SVG elements and attributes. Though both XML Schema and RelaxNG provision the means to restrict an SVG attribute value's data type to integer, string, URI, or other data types, we concluded this is insufficient for effectively filtering malicious SVG contents, such as those exemplified above.

For once, there is no feasible way to restrict the `xlink:href` attribute of `URI` data type to point to same-domain locations only, which would have been essential for resisting certain XSS attacks. The only viable way to perform such a verification while using XML Schema or RelaxNG capabilities comes down to setting the attribute's data type to a restricted string value that must conform to a given regular expression pattern. Expressing a value restriction as stated above would thus require a regular expression to be crafted for each type of restriction necessary for fending off the attacks. Although this might be (somehow and somewhat) feasible

for same-domain URIs, it is easy to imagine this approach to fail for complete CSS declarations, Base64-encoded contents, and the like.

Listing 2.5: XML Entity Resolution leading to element injection

```
<!doctype html><svg><style>
&lt;img src=null onerror=alert(1)&gt;<p>
```

Another example that demonstrates the impossibility of XML Schema to remove suspicious content from a given file is shown in Listing 2.5. When processing this code fragment, Firefox 4 resolves the `<` entity automatically, resulting in the alert being triggered. The crux is that the XML entity introduces an additional HTML element "on the fly" during parsing. An XML Schema validator would seen a `<style>` element with odd contents, but nothing to be alerted about (note that the missing closing tags and attribute value quotations are added automatically by the parser engine.). However, the HTML renderer resolves the entities, hence introducing the additional `<img>` tag, and triggering the `onerror` event due to the missing `null` file.

To summarize, we established that common XML validator techniques, like XML Schema or RelaxNG validation are not capable of fending the specific SVG attack vectors described above. Especially in the case of inline SVG, where HTML, CSS, JavaScript, and SVG elements are mixed arbitrarily, the approach of XML Schema validation must be revoked as completely ineffective in practice.

2.4.2 SVG Purification

Due to the limitations discussed above, we require another way to prevent the attacks introduced in Section 2.3. The basic idea of our methodology is to *purify* SVG images, *i. e.*, remove all suspicious content from a given file and preserve as much content as possible. As a result of this transformative process, the visual impact is minimized and the suspicious content is removed. Next, we discuss the overall design and present some implementation details.

The open source community provides a lot of tools claiming to possess the skills to filter user generated input for web applications in order to eliminate active markup and script code. Their main purpose is usually XSS mitigation and markup sanitization, as well as restructuring for validity and well-formedness' sake. For PHP-based web applications, several filtering solutions are available and those most commonly used include:

- *kses* [99], that has been incorporated into a highly customized version by the popular WordPress software.
- *htmlLawed* [190], which claims to be the fastest and most compact, yet complete solution.
- *HTMLPurifier* [235], which not only sanitizes data from possibly malicious code fragments, but also generates valid and well-formed XHTML output.

We analyzed all three XSS filters and have managed to bypass each of them, demonstrating that even the most sophisticated filtering software can never be able to fully protect against malicious markup. Some of the bypasses worked only for injections into SVG files, some even in a HTML/XHTML context.

Despite these drawbacks of the server-side filtering approaches, we have decided to choose *HTMLPurifier* as the foundation for our SVG attack mitigation tool. One major reason for this determination was the fact that *HTMLPurifier* is very well maintained, receives frequent updates and security fixes. Another reason is the quality of filtering: we have only identified a few bypasses for this tool and every single one of them was fixed very quickly upon having contacted the developers. Still, most importantly, *HTMLPurfier*'s internal API allows to filter arbitrary XML data and is not limited to HTML by design, unlike the other tested tools. This knowledge allowed us to create an *SVGPurifier* branch, a software that is using the *HTMLPurifier* API, but is not touching the core components. Our *SVGPurifier* has been supplied with a large array of data based on the SVG specifications defining which tags and attributes should be allowed in the user-generated SVG files. We explicitly *whitelist* tags and attributes, as well as the tag-attribute combinations and specific value ranges for attributes. Uncommon sources for cross-site scripting attacks, such as the `<set>` tag on older Webkit-based browsers, are limited by *SVGPurifier*.

The `<set>` tag can be used in SVG files, similarly to the timing driven equivalent `<animate>`, although the `attributeName` value can only consist of a limited number of values. Specifically, we only allow those values which cannot be used to initiate or overwrite event handlers, change `xlink:href` values on the fly, or reposition elements and element groups on a website. Our tests showed that the `<set>` and `<animate>` tags can be used to assign `javascript:` and `data:` URIs to existing elements. This can either enable attacks by initiating malicious remote inclusions, or apply malicious URL schemes to the existing elements.

Attacks like the one just portrayed, as well as those shown in Listing 2.6, can be no longer carried out. Take account of the fact that the `<set>`/`<animate>` functionality is not removed completely, only the remote includes and the assignment of

malicious URL handlers have been blocked. Furthermore, the resulys of our evaluation show that only two files out of more than 100,000 tested instances from the Wikipedia servers made use of the `<set>`/`<animate>` feature at all (see Section 2.5 for details).

Listing 2.6: Initiating JavaScript execution via set/animate elements

```
<svg xmlns="http://www.w3.org/2000/svg">
  <set
    attributeName="onmouseover"
    to="alert(1)"/>
  <animate
    attributeName="onunload"
    to="alert(1)"/>
</svg>
```

SVGPurifier completely forbids and removes `<script>` as well as `<foreignObject>` tags and event handler usage. Later versions of our purification approach might prove to add a supplementary scripting layer to allow basic JavaScript execution, but hinder scripts from reading, and overwriting sensitive data, or conduct other activities capable of leaking sensitive data or deploying malicious code [115]. As further elaborated on in Section 2.5, our tests showed that none of the analyzed and purified SVG used actual `<script>` tags. Surprisingly, a large percentage of the test files were making use of `<foreignObject>` tags, which can be used to include for example base64-encoded binary data. The reason behind it is that the software *Adobe Illustrator* uses this tag to hide proprietary metainfo in the SVG images generated by the tool [9]. Removing these tags does not affect the visual information provided by the SVG file.

Similar problems can be caused by maliciously crafted *SVG Cascading Stylesheets* (SVGCSS). SVG styles support more properties than classic CSS for (X)HTML documents and specifically extend the feature set with font formatting, typographic features, extended pointer event behavior, and the possibility to reference to other SVG elements containing definitions and visual effects. Arbitrary SVG elements can constitute reference to other SVG elements or even let external SVG files to borrow visual information or functionality including event handling. Those references can be defined via *FunctionIRI* or the fully qualified paths via protocol schemes such as data, HTTP and others. *SVGPurifier* guarantees that no external references can be loaded by elements allowing script execution. The `<use>` tag on modern Opera browser versions is conversely problematic. This tag can be utilized to include external resources executing JavaScript or providing links with potentially malicious URL handlers.

Currently, *SVGPurifier* scans style elements and attributes of the purified SVG for potentially malicious patterns and neutralizes them by overwriting certain parts of the payload. This includes replacing strings indicating the use of CSS expressions, Opera link and link target properties, as well as, data binding approaches with the placeholder `INVALID`. Listing 2.7 demonstrates an example for a purification result. Be assured that we do not forbid dangerous tags such as `<set>`, but analyze the attribute values and remove them in case an attack could be initiated by their contents.

Listing 2.7: A malicious SVG before and after purification

```
// before
<svg xmlns="http://www.w3.org/2000/svg">
  <circle r="50" fill="red" cx="30" cy="30">
    <set attributeName="onclick"
      to="javascript:alert(1)//">
    <set attributeName="fill" to="green">
  </circle>
</svg>

// after
<svg xmlns="http://www.w3.org/2000/svg">
<circle r="50" fill="red" cx="30" cy="30">
<set to="INVALID"></set><set attributeName
="fill" to="green"></set></circle></svg>
```

Section 2.5 will further build upon the results of this purification process and provide insight into how far our (for XML data unconventional) approach affects the visual information provided by the SVG test set.

SVGPurifier itself has undergone substantial testing from the security community during a public demonstration over a time-frame of several months [105]. The results helped us to refine the filtering mechanism and spot all the less obvious and difficult to find browser behaviors requiring dedicated fixes to deliver effective filtering and keep the security promise that the tool poses. During the testing phase we logged about 500 attempts targeted to break the filter functionality of the *SVGPurifier* and inject malicious content. Of those, about 15 were successful and resulted in the refinement of our algorithms. The *SVGPurifier* performance scales with the number of SVG tags and elements to sanitize, but can be considered uncritical since the main use case for the tool is on the server-side and, depending on the respective application, only needs to be executed once per SVG image (*i. e.*, each SVG image is transformed on upload to remove suspicious content).

A server-side solution has the advantage that a website owner can performantly protect its users from attacks using SVGs and not requiring all users to upgrade

their client-software. Our evaluation showed that the *SVGPurifier* was capable of removing malicious code in all of the discussed test cases. We examined possibilities to craft a purely client-side SVG filter combined with the possibility of limiting DOM elements and their capabilities. Our initial research showed that this is feasible and considered as future work.

2.4.3 Unexpected Browser Behavior

We also found several cases of unusual and (depending on the execution context) often problematic browser behaviors that forced us to adapt *SVGPurifier* to address them:

- To the best of our knowledge, the Opera AII attacks mentioned in Section 2.3.5 have not been fixed by the vendor despite several bug reports from our side. This problem complicated the implementation of *SVGPurifier* since basically any external image resource loaded by an SVG file could contain suspicious plugin code and initiate an attack.

- Most browsers support the SVG `<use>` tag, but so far only Opera allows to include external SVG resources containing script code to execute, or links to show and point to possibly malicious URIs via URL handlers such as `javascript` and `data`. Most browsers tested permit utilizing the `<set>` as well as the `<animate>` tag to transform `xlink:href` attributes and set them with *JavaScript* and *data* URIs, too. This should be restricted by browsers for the sake of avoiding injection attacks via `<use>`, `<set>`, and `<animate>`. However, none of the over 100,000 SVG files we have tested during our evaluation actually used this feature.

- Plain text tags inside SVG images such as `<script>`, `<style>`, `<noscript>`, and similar tags allow to use HTML entities, giving them an equivalent syntactical meaning as their canonical forms. This was relevant for several of the XSS filter bypasses we described in Section 2.3.4. Especially the automated decoding of entities such as `&lt;` and `&gt;` could be used to bypass XSS filters and common protection mechanisms. Browsers therefore need to be more selective in determining which entities get automatically decoded and which do not. For example, Google Chrome went as far as to completely disable the automated decoding.

2.5 Evaluation

We have implemented a prototype of *SVGPurifier*, consisting of 5,663 lines of PHP code. To evaluate the tool, we compiled a test set of SVG images obtained from Wikipedia. We chose this platform for several reasons: SVG images are widely used within Wikipedia, the content of the platform consists of the contributions from a large community, and among the contributors, the employment of a diverse set of tools to create the images can be observed. As a result, we have a heterogeneous test set that enables us to study the robustness and versatility of *SVGPurifier*.

To download the files, we used *wikix*, a tool that uses a snapshot of Wikipedia (exported as XML) to generate the URLs of the SVG files hosted at `upload.wikimedia.org`. The latest snapshot of the English Wikipedia at the time of writing the initial paper referenced a total of 112,646 SVG images. 105,509 were actually available for download at that time and we used those files for our evaluation.

2.5.1 Evaluation Setup

In order to minimize the impact of *SVGPurifier* on usability, the tool should not alter the visual appearance of an image since this would decrease the user experience. Our implementation only removes elements from an SVG file, thus (by construction) no new image element will appear in a purified image. However, the cleaning process might be too aggressive, *i. e.*, cases where we remove elements that have a visual impact on the resulting image might occur. To evaluate this effect, we compare the original image with the purified one and determine if the file was altered during the process. Since the size of our test set is too large for a manual evaluation, we developed an approach to compare SVG files in an automated way.

Comparing two SVG images for similarity is difficult, resulting from the fact that there are countless ways of achieving the same visual appearance. Consequently, contrasting only the XML markup does not enable us to determine if a pair of images has the same visual appearance. Therefore, we decided to convert the SVG files to *Portable Network Graphics* (PNG) format and then perform the comparative step. PNG is a raster graphics format providing lossless data compression. Each pixel is defined in an 24 bit RGB color-space with an optional 8 bit alpha channel (32 bit RGBA). Comparing two PNG files for similarity can thus be achieved by matching the value of each channel for each pair of pixels, which in turn results in a numerical value representing the absolute error a. A value of $a = 0$ indicates that no difference between the two images was found, while $a > 0$ denotes some discrepancy. Note that this evaluation measures the visual impact of our tool and approximates the deviation caused by the transformation process.

We tested the following four tools regarding their capability to convert SVG images to PNG format:

- Apache *batik* (`http://xml.apache.org/batik/`)
- *rSVG* (`http://librsvg.sourceforge.net/`)
- *GIMP* (`http://www.gimp.org/`)
- *Inkscape* (`http://inkscape.org/`)

Based on our test set, we have empirically found that the Apache *batik* toolkit was able to convert the largest number of files: only 23 of the 105,509 files could not be converted by the tool, prompting us to remove them from the test set. All files from the resulting evaluation set were converted to a PNG image with a fixed width, assuring the aspect ratio's preservation. As *batik* does not fully support declarative animations, we create static PNG images from the SVG files. In this manner, if an image was animated beforehand, we will only consider the visual state it is in before the animation begins. There are five different elements within a SVG file to accomplish animation: `<set>`, `<animate>`, `<animateMotion>`, `<animateColor>`, and `<animateTransform>`. A close analysis showed that only two SVG files in our test set actually contained one of these elements, which indicates that this feature is not (yet) widely used. We therefore consider comparison of static images exclusively to be only a minor limitation of our evaluation.

After converting images to PNGs, we compare the original and the purified image using the *ImageMagick* toolkit, which provides methods for a pixel-wise comparison of raster images, as well as, for gathering statistics on the amount of aberration. Furthermore, the tool is capable of creating *difference images* that visually indicate what regions of an image contain an error, which eventually enables us to manually examine cases in which actual changes occur. Other than determining the absolute number of pixels that differ between the two images, we calculate the normalized mean absolute error for each pair of images as metrics. We consider the normalized mean absolute error to be more relevant in our scenario than the root mean square error, as the former weights every aberration equally. Moreover, we log both the size of the original file, and that of the purified one, factually determining the compression ratio resulting from the purification process.

2.5.2 Evaluation Results

Based on the procedure outlined above, we analyzed all 105,486 files belonging to the evaluation set. For 98.5% of the samples, no visual difference exists in the appearance of the original versus the purified image (*i. e.*, absolute error a is 0 and, therefore,

all other error metrics are 0 as well). The 99th percentile of the normalized mean absolute error is 0.00000474877, where 0 represents no error at all and 1 indicates completely dissimilar images.

When investigating the error cases and the corresponding different images, we often found that although the image contained some kind of aberration that can be expressed numerically, a visual difference cannot be easily found by a human observer. Specifically, we manually analyzed 1,000 test cases in which the absolute error was larger than zero. We tried to determine in how many cases a human observer would notice an aberration. While this is not believed to be an approach that is valid overall (*i. e.*, specific circumstances like medical applications require precise conversion), a visual impact in the context of a website exists only if a user can actually spot it.

During the manual inspection process, we spent about 10 seconds on each pair of the images to compare them (aided by the *difference image* to support the user). The results of this manual examination indicate that only 46.3% of the erroneous samples were perceived as different from their original. Since user experience may differ, we provided a website at the time of the initial publication, where we presented all defective images complete with their original and the *difference images*, inviting others to freely inspect these cases [6].

2.6 Discussion

Several side effects were observed during the purification process and the evaluation of this process' results. An additional positive side effect we found was that due to the removal of elements not contributing to the visual appearance of the image, *SVGPurifier* actually compresses files with an average compression ratio of 2.6. Some files had a slightly larger file size after the purification process, which was caused by the transformation the tool has performed on broken files. In most cases, the increase in file size was based on the addition of missing closing tags by *SVGPurifier*.

1.59% of the files in our test set contained one or more instances of the `<foreignObject>` element. In the majority of the 1,686 cases, this element was used as what had appeared to be an artifact of *Adobe Illustrator* to store a base64-encoded representation of its proprietary AI file format within the SVG file. *SVGPurifier* deleted these elements and no visual impact resulted from this removal. However, the image size was reduced significantly.

2.7 Summary

With this chapter we advanced *precondition A* posited in Section 1.1, *i. e.*, we presented a mitigation approach for a novel attack vector that resulted from offensive research. To this end we provide an overview of Scalable Vector Graphics (SVG) and their security impact on the World Wide Web as based on the new HTML5 specification drafts. We show that this image format (which exists for more than a decade), significantly changes the browser and web security landscape. We introduce several novel attacks against modern browsers and show that this phenomenon can have major impact on web applications that allow their users to post images. In particular, we illustrate that SVG images embedded via `<img>` tag and CSS can execute arbitrary JavaScript code and similar attacks. Subsequently, the discussed XSS filter bypasses, which work against several browsers, can have a similarly high impact on a targeted attack scenario.

To mitigate the attacks presented, we proposed *SVGPurifier* as a first practical solution available and capable of removing potentially malicious code from SVG files. We have empirically shown that the software is usable for real-world scenarios such as a purification of the SVG files stored by Wikipedia. Furthermore, many of the identified attacks were already fixed by major browser vendors at the time of the initial publication [107].

3

Analysis of a Cryptographic Instant Messaging Protocol

> Love your Enemies, for they tell you your Faults.
>
> —Benjamin Franklin, *Poor Richard's Almanack*

In Chapter 1 we introduced the review and analysis of technology as *precondition B*: We posited that the mere existence of a technical solution that claims to achieve a certain goal does not necessarily warrant any trust that it does indeed achieve this goal. There are many examples that support this idea both from a functional, security, and privacy point of view. A rather notorious example of the latter two aspects is Cryptocat[10], a browser-based chat application that claimed to ensure confidentiality of conversations. Independent reviews showed that the implementation of the key generation function was flawed [216], among other critical bugs [109]. At that point Cryptocat already had a significant user base that relied on its—then unfounded—promises of guaranteeing secrecy of the written word.

In this chapter we contribute to the same goal as Cryptocat's auditors—building a justified foundation of trust in a system, based on an unbiased assessment of what it can and cannot achieve. While the notions of privacy people have can differ widely and people may not explicitly state these notions, their actions let us conclude what these persons deem worth protecting: users that choose to use an encrypted chat solution, for instance, can be expected to have a desire to *not* have their conversations overheard, which also implies that they need to make sure the respective communication partner is indeed the person they claim to be. In this chapter we venture into the context of mobile chat applications that claim to preserve the confidentiality of messages, as well as, ensure the authenticity of message and entity.

[10] https://crypto.cat/

3.1 Introduction

Since more than a decade, *Instant Messaging* (IM) has attracted a lot of attention by users for both private and business communication. IM has several advantages over classical email communication, especially due to the chat-like user interfaces provided by popular tools. However, compared to the security mechanisms available for email such as PGP [48] and s/MIME [202], text messages were sent unprotected in terms of authenticity and confidentiality on the Internet by the corresponding IM tools: in the early days, many popular IM solutions like MSN Messenger and Yahoo Messenger did not provide any security mechanisms at all. AOL only added a protection mechanism similar to s/MIME to their IM service later on and Trillian's SecureIM messenger encrypted the data without providing any kind of authentication. Nowadays, most clients provide at least client-to-server encryption via TLS. Mechanisms like *Off the Record* (OTR) communication [42] are available that provide among other security properties end-to-end confidentiality.

As the popularity of smartphones grows, the Internet is accessible almost everywhere and mobile communication services gained a lot of attraction. IM is one of the most popular services for mobile devices and apps like WhatsApp and Skype are among the top downloaded apps in the popular app stores. Unfortunately, both applications are closed-source and it is unknown which security mechanisms—beside a proprietary or TLS-based client-to-server encryption—are implemented. As such, it is hard to assess which kind of security properties are provided by these apps and especially end-to-end encryption is missing. In the light of the recent revelations of mass surveillance actions performed by intelligence services such as NSA and GCHQ, several secure IM solutions that are not prone to surveillance and offer a certain level of security were implemented.

One of the most popular apps for *secure* IM is TextSecure, an app developed by *Open WhisperSystems* that claims to support end-to-end encryption of text messages. While previously focussing on encrypted short message service (SMS) communication, *Open WhisperSystems* introduced data channel-based push messaging in February 2014. Thus, the app offers both an iMessage- and WhatsApp-like communication mode, providing SMS+data channel or data channel-only communications [185]. Following Facebook's acquisition of WhatsApp, TextSecure gained a lot of popularity among the group of privacy-concious users and has currently more than 500,000 installations via *Google Play*. Its encrypted messaging protocol has also been integrated into the OS-level SMS-provider of CyanogenMod [184], a popular open source aftermarket Android firmware that has been installed on about 10 million Android devices [66].

Despite this popularity, the messaging protocol behind TextSecure has not been rigorously reviewed so far. While the developers behind TextSecure have a long history of research in computer security [144, 145, 146, 147, 148, 149] and TextSecure has received praise by whistleblower Edward Snowden [7], a security assessment is needed to carefully review the approach.

3.1.1 Contribution

In this chapter, we perform a thorough security analysis of TextSecure's protocol. To this end, we first review the actual security protocol implemented in the app and provide a precise mathematical description of the included security primitives. Based on this protocol description, we perform a security analysis of the protocol and reveal an *Unknown Key-Share attack*, an attack vector first introduced by Diffie et. al. [77]. To the best of our knowledge, we are the first to discuss an actual attack against TextSecure. We also reveal several other (minor) security problems in the current version of TextSecure. Based on these findings, we propose a mitigation strategy that prevents the UKS attack. Furthermore, we also formally prove that TextSecure with our mitigation strategy in place is secure and achieves *one-time authenticated encryption*.

In summary, we make the following contributions:

- We are the first to completely and precisely document and analyze TextSecure's secure push messaging protocol.
- We found an Unknown Key-Share attack against the protocol. We have documented the attack and show how it can be mitigated. The attack has been communicated to the developers of TextSecure. We show that our proposed method of mitigation actually solves the issue.
- We show that if long-term public keys are authentic, so are the message keys, and that the encryption block of TextSecure is actually one-time authenticated encryption. Thus, we prove that TextSecure's push messaging—with our amendments—can indeed achieve the goals of authenticity and confidentiality.

3.1.2 Outline

In the remainder of this chapter we first provide technical background on TextSecure's protocol in Section 3.2, where we also discuss related work. In Section 3.3 we discuss issues with the protocol and show how these issues can be mitigated. In

the following Section 3.4 we show that TextSecure can indeed achieve a high level of security, if our mitigations are applied. In Section 3.5 we discuss the implications of our work and future fields of analysis.

3.2 Technical Background

We start with a precise description of the protocol implemented by TextSecure. We obtained this information by analyzing the source code of the Android app and recovering the individual building blocks of the protocol. TextSecure builds upon a set of cryptographic primitives. For ECDH operations, these are Curve25519 [36] as implemented in Google's Android Native Library. For symmetric encryption, TextSecure relies on AES in both counter mode without padding and cipher block chaining mode with PKCS5 padding. For authenticity and integrity, HMACSHA256 is used. Security considerations of the cryptographic primitives are not within the scope of this work.

For push messaging via data channel, TextSecure relies on a central server[11] ($\mathcal{TS}$) to relay messages to the intended recipient. Parties communicate with $\mathcal{TS}$ via a REST-API using HTTPS. $\mathcal{TS}$'s certificate is self-signed, the certificate of the signing CA is hard-coded in the TextSecure app. Actual message delivery is performed via Google Cloud Messaging (GCM), which basically acts as a router for messages.

3.2.1 TextSecure Protocol Flow

TextSecure's protocol consists of several phases. We distinguish (i) *registration*, (ii) *sending/receiving a first message*, (iii) *sending a follow-up message*, and (iv) *sending a reply*.

Before a client is able to communicate, it needs to generate key material and register with $\mathcal{TS}$. When a party $\mathcal{P}_a$ first decides to use TextSecure's data channel communication, it chooses an asymmetric long-term key pair (a, g^a), referred to as *identity key* by TextSecure's developers. It also uses SHA1PRNG as provided by the Android Native Library to choose a password (pw), a registration ID (regID_a), and two keys $k_{enc,signal,a}$, $k_{mac,signal,a}$, each of 128 bit length. Additionally, the client chooses 100 asymmetric ephemeral key pairs, so-called *prekeys*, and one asymmetric *last resort key* (k_{lr}). When a party calculates a message authentication code (MAC), it uses HMACSHA256 as implemented by Android's respective Native Library.

[11] `textsecure-service.whispersystems.org`

For a party $\mathcal{P}_a$ to send a message to a party $\mathcal{P}_b$, $\mathcal{P}_a$ requests one of $\mathcal{P}_b$'s public prekeys from $\mathcal{TS}$, uses it to derive a shared secret, forms a message, whereof parts are encrypted and/or protected by a MAC, authenticates with $\mathcal{TS}$, and transmits the message to $\mathcal{TS}$. $\mathcal{TS}$ shares a symmetric long-term key $(k_{enc,signal,b}, k_{mac,signal,b})$ with $\mathcal{P}_b$, which it uses to encrypt all parts of $\mathcal{P}_a$'s message that are to be transmitted to $\mathcal{P}_b$. $\mathcal{TS}$ then hands off this encrypted message to GCM for delivery to $\mathcal{P}_b$. If $\mathcal{P}_a$ wants to send a follow-up message to $\mathcal{P}_b$, it derives a new key using a function f that is seeded with existing key material. When a party does not merely send a follow-up message, but a reply within a conversation, it also introduces new entropy into the seed of f and transmits a new ephemeral public key.

3.2.2 Detailed Description of Messages

In the following we give a detailed description of messages sent and processed in the different phases, as well as the key derivation.

3.2.2.1 Registration

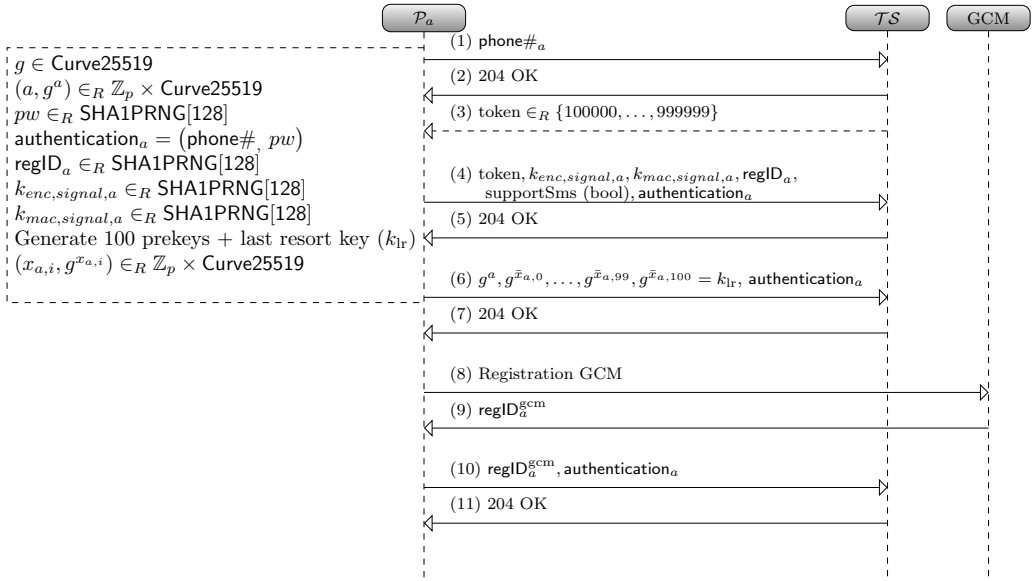


Figure 3.1: TextSecure registration.

The registration process is depicted in detail in Figure 3.1. To register with TextSecure, a party $\mathcal{P}_a$ requests a verification token by transmitting its phone number

($\mathsf{phone\#}_a$) and its preferred form of transport to $\mathcal{TS}$ (Step 1), which $\mathcal{TS}$ confirms with a HTTP status 204 (Step 2). Depending on the transport $\mathcal{P}_a$ chose, $\mathcal{TS}$ then dispatches either a short message or a voice call containing a random token (Step 3) to the number transmitted in Step 1. $\mathcal{P}_a$ performs the actual registration in Step 4, where it shows ownership of $\mathsf{phone\#}_a$ by including the token, registers its credentials with the server via HTTP basic authentication [86], and sets its *signaling keys*. In this step, the client also states whether it wishes to communicate only via data channel push message or also accepts short messages. The server accepts if the token corresponds to the one supplied in Step 3 and the phone number has not been registered yet.

In Step 6, $\mathcal{P}_a$ supplies its 100 *prekeys* and k_{lr} to $\mathcal{TS}$. Prekeys are not transmitted individually, but within a JSON structure consisting of a *keyID* z, a *prekey* $g^{x_{a,i}}$, and the long-term key g^a. The *last resort key* is transmitted in the same way and identified by *keyID* 0xFFFFFF. The server accepts, if the message is well-formed and HTTP basic authentication is successful. $\mathcal{P}_a$ then registers with GCM (Step 8) and receives its $\mathsf{regID}_a^{\mathrm{gcm}}$ (Step 9), which it transmits to $\mathcal{TS}$ in Step 10 after authenticating again.

3.2.2.2 Sending an Initial Message

We define the period in which $\mathcal{P}_a$ employs one *prekey* to communicate with $\mathcal{P}_b$ as a *session*. When a new session is created to exchange messages, three main cryptographic building blocks are applied: a) a key exchange protocol with implicit authentication to exchange a secret, b) a key update and management protocol (the so-called axolotl ratchet [191]), which updates the encryption and MAC keys for every outgoing message, and c) an authenticated encryption scheme. The process is depicted in detail in Figure 3.2.

Intuitively, the key exchange is a triple Diffie-Hellman (DH) key exchange using long-term and ephemeral secret keys. This is the only step in the protocol flow that uses the long-term keys.

According to the developers [191], the key management protocol provides both forward secrecy (which roughly means that *past* sessions remain secure even if the long-term key of a party is corrupted) and future secrecy (which translates to the idea that even after leakage of a currently used shared key *future* keys will remain secure).

The result of the key exchange and key management is input to the authenticated encryption scheme [189]. The state of the encryption scheme is provided by the key management system and handed over from every call of the encryption and

decryption algorithm, respectively, to the next for the whole session. Every new message is encrypted under a fresh key. The scheme guarantees confidentiality and authenticity of the exchanged messages, which we discuss in detail in Section 3.4.

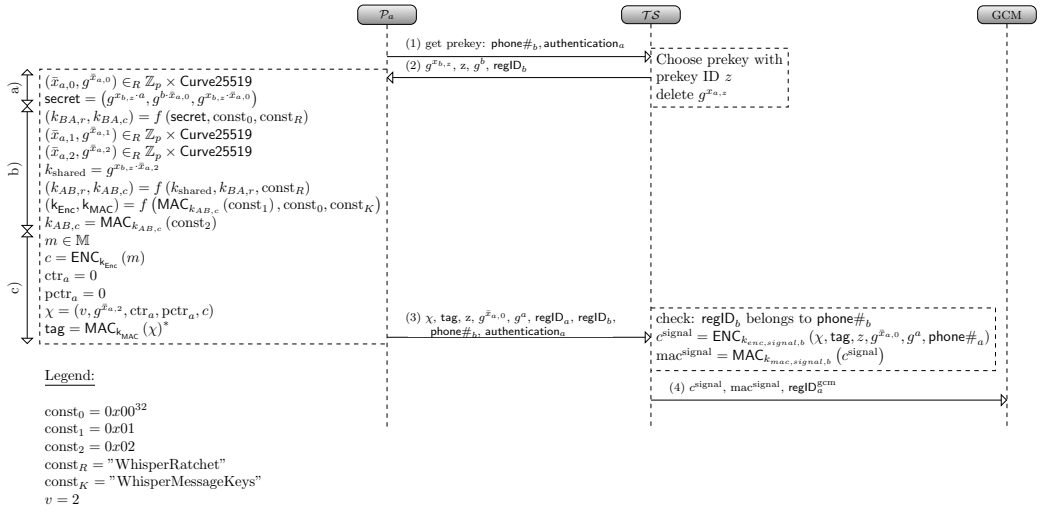


Figure 3.2: Sending an initial TextSecure message.

Key Exchange In the first step, $\mathcal{P}_a$ requests a *prekey* for $\mathcal{P}_b$ and receives a JSON structure consisting of prekeyID z, a prekey $g^{x_{b,z}}$, and $\mathcal{P}_b$'s long-term key g^b. $\mathcal{P}_a$ also receives regID_b from $\mathcal{TS}$ an then chooses a new ephemeral key to calculate a secret as the concatenation of three DH operations, combining $\mathcal{P}_b$'s prekey, $\mathcal{P}_a$'s long-term key, $\mathcal{P}_b$'s long-term key, and $\mathcal{P}_a$'s freshly chosen ephermeral key.

Key Management (axolotl ratchet) After $\mathcal{P}_a$ has completed the initial key exchange, it derives two symmetric keys $(k_{BA,r}, k_{BA,c})$ for receiving messages using f (cf. Algorithm 1), an implementation of HKDF [136]. f is here seeded with secret. For all respective parameters of f see Figure 3.2. $\mathcal{P}_a$ then chooses a new ephemeral keypair $(\bar{x}_{a,1}, g^{\bar{x}_{a,1}})$, which is never used and just exists because of reuse. It then chooses another ephemeral keypair $(\bar{x}_{a,2}, g^{\bar{x}_{a,2}})$, which it uses to calculate k_{shared} as the output of a DH operation that takes $\mathcal{P}_b$'s prekey $g^{x_{b,z}}$ and $\bar{x}_{a,2}$ as input. $\mathcal{P}_a$ then derives two symmetric keys $(k_{AB,r}, k_{AB,c})$ for sending messages. Here f is seeded with k_{shared} and $k_{BA,r}$. Finally, $\mathcal{P}_a$ uses f, seeded with $k_{AB,c}$, to derive the message keys $(\mathsf{k_{Enc}}, \mathsf{k_{MAC}})$ and in the end derives a new $k_{AB,c}$ as $\mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_2)$, where $\mathsf{const}_2 = 0x02$.

Algorithm 1 $f\,(input, key, string)$

$k_{pr} \leftarrow \mathsf{MAC}_{key}\,(input)$
$k_0 \leftarrow \mathsf{MAC}_{k_{pr}}\,(string, 0x00)$
$k_1 \leftarrow \mathsf{MAC}_{k_{pr}}\,(k_0, string, 0x01)$
return (k_0, k_1)

Authenticated Encryption A message $m \in \mathbb{M}$ is encrypted using AES in counter mode without padding as $c = \mathsf{ENC}_{\mathsf{k}_{\mathsf{Enc}}}(m)$. $\mathcal{P}_a$ then forms message (3.) and thus calculates $\mathsf{tag} = \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi)$, where $\chi = (v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, c)$. v represents the protocol version and is set to $0x02$. For ordering messages within a conversation ctr and pctr are used. Both are initially set to 0. ctr is incremented with every message a party sends, while pctr is set to the value ctr carried in the message a party is replying to.

Upon receiving message (3.), $\mathcal{TS}$ checks if regID_b corresponds to $\mathsf{phone\#}_b$. It then encrypts the parts of message (3.) intended for $\mathcal{P}_b$ with $\mathcal{P}_b$'s signaling key, using AES in CBC mode with PKCS5 padding. $\mathcal{TS}$ additionally calculates a MAC over the result, which we denote as $\mathrm{mac}^{\mathrm{signal}}$. $\mathcal{TS}$ sends both, encrypted message data c^{signal} and $\mathrm{mac}^{\mathrm{signal}}$, to the GCM server, together with $\mathsf{regID}_b^{\mathrm{gcm}}$ as the receipient. The result of this additional encryption layer is that Google's Cloud Messaging servers will only be able to see the receipient but not the sender of the message.

The receiving process is depicted in Figure 3.3. $\mathcal{P}_b$ receives the message in Step (5.). First, $\mathcal{P}_b$ verifies $\mathrm{mac}^{\mathrm{signal}}$ and, if successful, decrypts c^{signal}. It looks up its private key that corresponds to prekeyID z and calculates secret.

$\mathcal{P}_b$ then derives two symmetric keys $(k_{BA,r}, k_{BA,c})$ for sending messages by seeding f with secret. Afterwards, $\mathcal{P}_b$ calculates k_{shared} as the output of a DH operation that takes $\mathcal{P}_a$'s latest ephemeral key $g^{\bar{x}_{a,2}}$ and $\mathcal{P}_b$'s private prekey $x_{b,z}$ as input. In the next step $\mathcal{P}_b$ uses $f\,(k_{\mathrm{shared}}, k_{BA,r}, \mathsf{const}_R)$ to derive two symmetric keys $(k_{AB,r}, k_{AB,c})$ for receiving messages and derives the message keys $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$.

$\mathcal{P}_b$ now verifies the MAC and, if successful, decrypts the message. In the end, $\mathcal{P}_b$ also derives a new $k_{AB,c} = \mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_2)$.

3.2.2.3 Follow-up Message

If $\mathcal{P}_a$ follows up with a message before $\mathcal{P}_b$ replies, $\mathcal{P}_a$ derives a new pair $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}}) = f\left(\mathsf{MAC}_{k_{AB,c}}(\mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K\right)$, which it then uses as detailed above.

3.2.2.4 Reply Message

If $\mathcal{P}_b$ wants to reply to a message within an existing session with $\mathcal{P}_a$, it first chooses a new ephemeral keypair $(\bar{x}_{b,0}, g^{\bar{x}_{b,0}})$ and calculates k_{shared} as the output of a DH operation that takes $\mathcal{P}_a$'s latest ephemeral public key $g^{\bar{x}_{a,2}}$ and its own freshly chosen ephemeral private key $\bar{x}_{b,0}$ as input. $\mathcal{P}_b$ then derives $(k_{BA,r}, k_{BA,c})$ by seeding f with k_{shared} and $k_{AB,r}$.

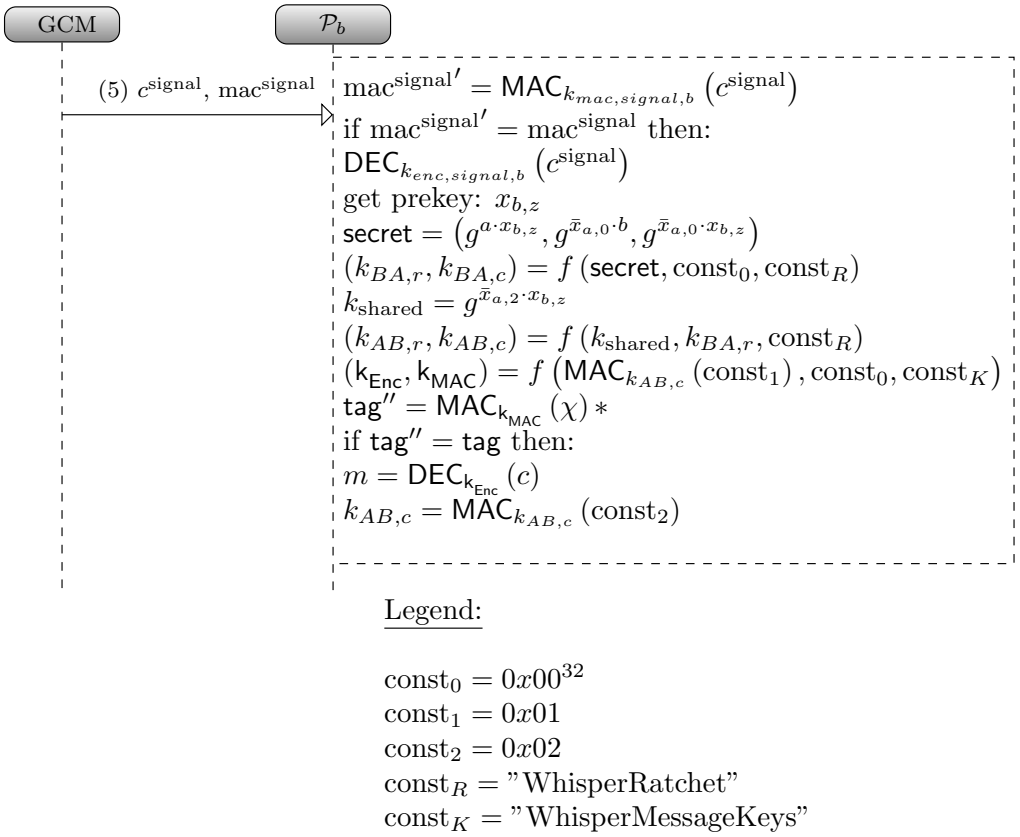


Figure 3.3: Receiving an initial TextSecure message.

3.2.3 Key Comparison

In an attempt to establish that a given public key indeed belongs to a certain party, TextSecure offers the possibility to display the fingerprint of a user's long-term public key. Two parties can then compare fingerprints using an out-of-band channel,

for example, a phone call or an in-person meeting. If two parties meet in person, TextSecure also offers to conveniently render the fingerprint of one's own long-term public key as a QR code, using a third-party application on Android, which the other party can then scan using the same application on their mobile device. TextSecure then compares the fingerprint it just received to the party's fingerprint it received as part of a conversation. Figure 3.4 pictures these fingerprints.

3.2.4 Related Work

The body of work that explicitly aims at providing or analyzing secure instant messaging protocols is comparably small to the prevalence of instant messaging applications in our daily life: At the 2004 Workshop on Privacy in the Electronic Society (WPES), Borisov et. al. [42] presented a protocol for "Off the Record" (OTR) communication. The OTR protocol was designed to provide authenticated and confidential instant messaging communication with strong perfect forward secrecy and *deniability*: no party can cryptographically prove the authorship of a message. The deniability property of OTR has been discussed by Kopf and Brehm [134]. The work of Di Raimondo et. al. [201], who analyzed the security of OTR, is in its nature closely related to our work. The authors point out several issues with OTR's authentication mechanism and also describe a UKS attack on OTR, as well as, a replay attack along with fixes. We note, however, that the authentication mechanisms of OTR and TextSecure have little in common: Though it aims to provide deniability, OTR explicitly uses signatures for authentication while TextSecure does not. The UKS attack on OTR described by Raimondo et al. [201] directly targets the key exchange mechanism of the protocol, whereas the attacks presented in this chapter are rather subtle and exploit the protocol structure and key derivation of TextSecure.

Besides OTR, which has been widely adopted, there exist protocols for secure instant messaging like IMKE [143], which aims at being verifiable in a BAN-like logic, but has never found a wider adoption. SILC [204] also has received a certain adoption and some discussion, but is rarely used today, as is FiSH [219], a once popular plugin for IRC clients that used Blowfish with pre-shared keys to encrypt messages.

Thomas [216] analyzed the browser-based instant messager Cryptocat and found that, due to an implementation error, Cryptocat used private keys of unsufficient length when establishing a group chat session. Green [94] recently discussed Cryptocat's group chat approach from a protocol perspective and points out several issues.

Further protocols exist that aim at securing instant messaging communication but have, to the best of our knowledge, not received public scrutiny. Among these are Threema [217], Surespot [215], and Silent Circle's SCIMP [166].

3.3 Issues and Mitigation

Based on the recovered protocol description, we can analyze its security properties. In the following, we discuss our findings.

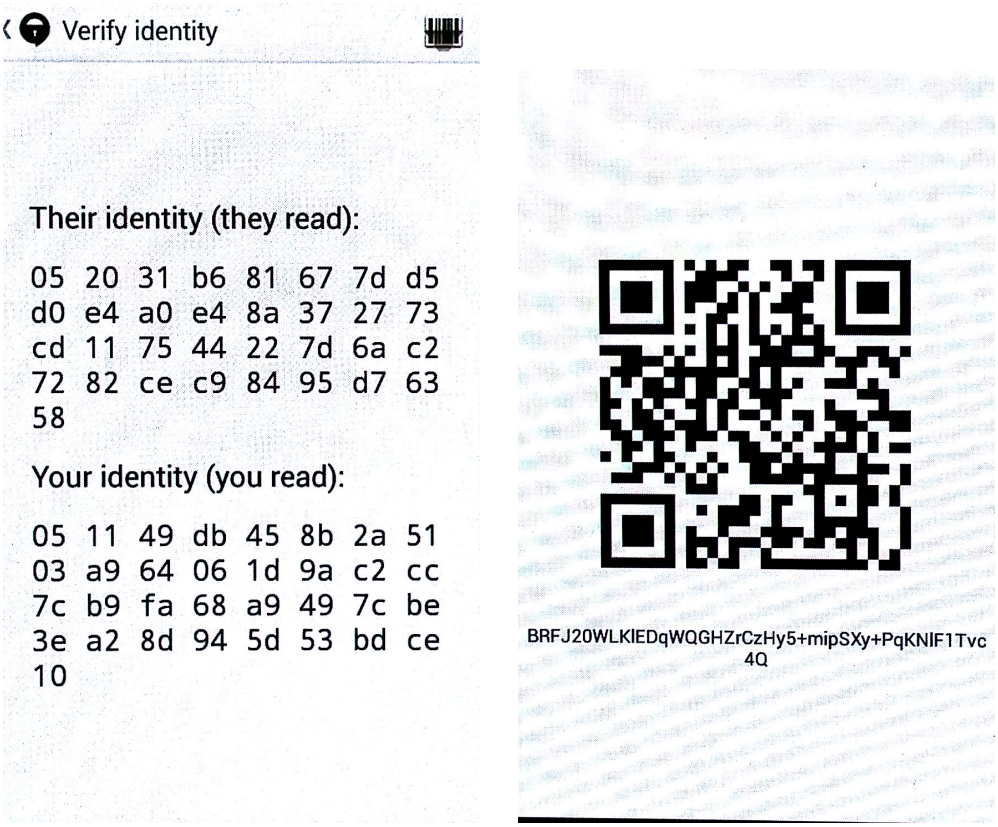
3.3.1 MAC Image Space Only Partially Used

In Section 3.2, we stated that TextSecure uses HMACSHA256 to calculate MACs. Surprisingly, TextSecure does not transmit the complete output of HMACSHA256. The message `tag` in Figures 3.2 and 3.3 does only represent the first 64 bit of the 256 bit MAC. Upon request, TextSecure's developers stated that this just happens to reduce message size. However according to NIST [22, chapter 5.5], a MAC length of 80 bit is recommended when a MAC is used for *key confirmation*, which is the case in TextSecure (at least) in the first message of a new session.

3.3.2 Unknown Key-Share Attack

An *Unknown Key-Share Attack* (UKS) is an attack vector first described by Diffie et al. [77]. Informally speaking, if such an attack is mounted against $\mathcal{P}_a$, then $\mathcal{P}_a$ believes to share a key with $\mathcal{P}_b$, whereas in fact $\mathcal{P}_a$ shares a key with $\mathcal{P}_e \neq \mathcal{P}_b$.

For a better understanding how this can be related to TextSecure, suppose the following example: Bart ($\mathcal{P}_b$) wants to trick his friend Milhouse ($\mathcal{P}_a$). Bart knows that Milhouse will invite him to his birthday party using TextSecure (*e. g.*, because Lisa already told him). He starts the UKS attack by replacing his own public key with Nelsons ($\mathcal{P}_e$) public key and lets Milhouse verify the fingerprint of his new public key. This can be justified, for instance, by claiming to have a new device and having simply re-registered, as that requires less effort than restoring an encrypted backup of the existing key material. Now, as explained in more detail below, if Milhouse invites Bart to his birthday party, then Bart may just forward this message to Nelson who will believe that this message was actually sent by Milhouse. Thus, Milhouse ($\mathcal{P}_a$) believes that he invited Bart ($\mathcal{P}_b$) to his birthday party, where in fact, he invited Nelson ($\mathcal{P}_e$).



(a) Conversation fingerprints (b) Own fingerprint QR code

Figure 3.4: TextSecure fingerprint verification.

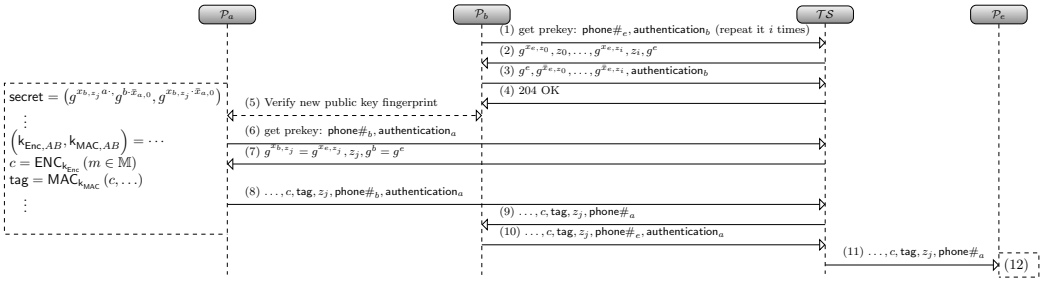


Figure 3.5: UKS attack on TextSecure: $\mathcal{P}_a$ believes to share a key with $\mathcal{P}_b$ but shares one with $\mathcal{P}_e$.

In detail, the attacker (Bart, $\mathcal{P}_b$) has to perform the steps shown in Figure 3.5 for this attack (only the important protocol parameters and steps are mentioned):

(1-2) $\mathcal{P}_b$ requests $g^{x_{e,z_0}}, \ldots, g^{x_{e,z_i}}$ from $\mathcal{TS}$ using $\mathsf{phone\#}_e$.

(3-4) $\mathcal{P}_b$ commits $g^{x_{e,z_0}}, \ldots, g^{x_{e,z_i}}$ to $\mathcal{TS}$ as his own prekeys plus g^e as its own long-term public key.

(5) $\mathcal{P}_b$ lets $\mathcal{P}_a$ verify the fingerprint of its new public key g^e. Note that this step uses QR-codes and is thus offline.

(6-7) Once $\mathcal{P}_a$ wants to send the message to $\mathcal{P}_b$, $\mathcal{P}_a$ requests a prekey for $\mathcal{P}_b$ by using $\mathsf{phone\#}_b$. $\mathcal{TS}$ returns $g^{x_{b,z_j}} = g^{x_{e,z_j}}$ and the long-term key $g^b = g^e$.

(8-9) $\mathcal{P}_a$ computes the secret using $g^{x_{b,z_j}}$ and g^b from which $\left(\mathsf{k}_{\mathsf{Enc},AB}, \mathsf{k}_{\mathsf{MAC},AB}\right)$ are going to be derived. For computing those keys, he uses in fact $\mathcal{P}_e$'s prekey and identity key although he believes to use $\mathcal{P}_b$'s ones. He then encrypts message $m \in \mathbb{M}$, computes the respective MAC tag, and sends it to $\mathcal{P}_b$ (GCM omitted).

(10-11) $\mathcal{P}_b$ is neither able to verify the tag nor to decrypt the message c. He sends the ciphertext and message tag to $\mathcal{P}_e$.

(12) $\mathcal{P}_e$ processes the incoming message as usual. He computes the same secret as $\mathcal{P}_a$, because $g^{x_{b,z_j}} = g^{x_{e,z_j}}$ and $g^b = g^e$. The secret is then used to compute $\left(\mathsf{k}_{\mathsf{Enc},AE} = \mathsf{k}_{\mathsf{Enc},AB}, \mathsf{k}_{\mathsf{MAC},AE} = \mathsf{k}_{\mathsf{MAC},AB}\right)$ so that $\mathcal{P}_e$ is able to read and verify the message.

In Step 10, $\mathcal{P}_b$ has to forward the message to $\mathcal{P}_e$, such that it appears to be sent by $\mathcal{P}_a$. Therefore, he needs to include $\mathsf{authentication}_a$ for $\mathcal{TS}$ to include $\mathsf{phone\#}_a$ in Step 11, so that $\mathcal{P}_e$ will receive $\mathsf{phone\#}_a$ with the forwarded message. This can be achieved in several ways:

- $\mathcal{TS}$ is corrupted. In this case, it is a trivial task to get or circumvent $\mathsf{authentication}_a$.
- If $\mathcal{TS}$ is benign, an attacker might be able to eavesdrop $\mathsf{authentication}_a$. Although TLS is used for all connections between clients and server, future or existing issues with TLS implementations [12, 13, 128, 156, 157] can not be ruled out and would allow for a compromise of $\mathsf{authentication}_a$. Another possibility to obtain $\mathsf{authentication}_a$ could be a governmental agency (legally) enforcing access to the TLS keys.
- In contrast to a party's other key material, the password is stored unencrypted and is not protected by TextSecure's master password. Thus, the easiest possibility to realize this attack might be for an attacker to recover the password for $\mathsf{authentication}_a$ from TextSecure's preferences[12].

[12] File: *shared-prefs/org.thoughtcrime.securesms_preferences.xml*

Physical access to a mobile device is the most straight-forward way for an attacker to recover the password and can often be achieved trivially: Devices are left unattended on tables in bars and clubs, users can be compelled to hand devices over in a traffic control, stop-and-frisk operations, immigration and customs, or when passing through airport security. The widely used Android Unlock Pattern has been shown to be a weaker protection than a three-digit PIN [218] and can often also easily be recovered from the smudges a user leaves on the screen during the unlock process [16].

3.3.3 Unknown Key-Share Attack Variant

In this variant, there is no need for the attacker to know $\mathsf{authentication}_a$. Instead, he must only be able to stop/intercept one message, for example, by controlling one active network element in the path between $\mathcal{P}_a$ and $\mathcal{TS}$, like a WiFi accesspoint.

The attack from the previous section makes $\mathcal{P}_a$ (the sender of a message) believe he shares a key with $\mathcal{P}_b$ (intended receiver of that message) while he in fact shares a key with $\mathcal{P}_e$ (actual receiver of the message). To this end, $\mathcal{P}_b$ had to replace his own public key by the public key of the intended receiver.

An attack on $\mathcal{P}_e$ is also possible, using similar techniques: Suppose the attacker replaces his own public key with the public key of the sender. Then any message that is sent from $\mathcal{P}_a$ to $\mathcal{P}_e$ may also originate from $\mathcal{P}_b$. This makes $\mathcal{P}_e$ (the receiver) believe in that he shares a key with $\mathcal{P}_b$ (claimed sender) while he actually shares a key with $\mathcal{P}_a$ (actual sender).

This is a practical issue in competitions where, for instance, the first to send the solution to $\mathcal{P}_e$ wins a prize.

To mount such an attack using TextSecure, $\mathcal{P}_b$ replaces his own public key with the public key of $\mathcal{P}_a$ and lets $\mathcal{P}_e$ verify it. We stress again that replacing the public key and letting $\mathcal{P}_e$ verify it is not an issue for $\mathcal{P}_b$ in practice. Now, when $\mathcal{P}_a$ starts a new session with $\mathcal{P}_e$, $\mathcal{P}_b$ can mount the attack by intercepting the message sent from $\mathcal{P}_a$ to $\mathcal{P}_e$. He relays the message to $\mathcal{P}_e$, but uses his own $\mathsf{authentication}_b$.

3.3.4 Mitigation of Unknown Key-Share Attack

Let us consider the message that is sent in Step 8 of Figure 3.5:

$$\chi, \mathsf{tag}, g^{\bar{x}_{a,0}}, g^{a}, \mathsf{regID}_a, \mathsf{regID}_e, \mathsf{phone\#}_e, \mathsf{authentication}_a,$$

where $\chi = (v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, c)$ and $\mathsf{tag} = \mathsf{MAC}_{\mathsf{k}_{\mathsf{MAC}}}(\chi)$. Intuitively, if both $\mathcal{P}_a$'s and $\mathcal{P}_e$'s identity were protected by the tag, then the attacks above do not longer work. As identities we propose to use the respective parties' phone numbers, as they represent a unique identifier within the system. χ would thus be formed as

$$(v, g^{\bar{x}_{a,2}}, \mathrm{ctr}_a, \mathrm{pctr}_a, \\ \mathsf{phone\#}_a, \mathsf{phone\#}_e, c).$$

If $\mathsf{k}_{\mathsf{MAC}}$ is secret (*i. e.*, only shared among $\mathcal{P}_a$ and $\mathcal{P}_e$) and if MAC is secure, the inclusion of both identities in the tag provides a proof of $\mathcal{P}_a$ towards $\mathcal{P}_e$ that $\mathcal{P}_a$ is aware of $\mathcal{P}_e$ as its peer, *i. e.*, that the message is indeed intended for $\mathcal{P}_e$. Moreover, $\mathcal{P}_e$ is convinced that $\mathcal{P}_a$ actually sent the message. Thus, $\mathcal{P}_b$ will not be able to mount the above attacks.

Remark 1. *Our mitigation resembles the concept of* strong entity authentication *[43]. However, this concept is not directly applicable here, since we consider asynchronous message exchange.*

3.3.5 No Cryptographic Authentication

While the *Unknown Key-Share Attacks* are mitigated if the message in Step 8 is modified as we propose in Section 3.3.4, the underlying problem is not resolved. It results from a party's erroneous assumption that a communication partner's long-term identity key is authentic, if they have compared key fingerprints and these fingerprints matched their assumptions. However, this is not necessarily the case. Given the attack scenario in Section 3.3.2, a malicious party would always be able to present a third party's long-term public key as their own, as only fingerprints are compared—a party is not required to show their knowledge of the corresponding secret key.

3.3.6 Mitigation of Authentication Issue

In the following, we present two means of authentication. The first one provides a mutual cryptographic authentication with the help of digital signatures, while the second one achieves this goal requiring less effort in terms of implementation and also integrates seamlessly into TextSecure's existing user experience.

3.3.6.1 Signature-based Cryptographic Authentication

$\mathcal{P}_a$ and $\mathcal{P}_b$ can establish the authenticity of their respective long-term keys as follows

1. $\mathcal{P}_a$ choses a token $r_A \in_R \mathsf{SHA1PRNG}[128]$ and presents a QR code containing r_A.
2. $\mathcal{P}_b$ scans this QR code, creates a signature σ over r_A using his long-term private key, chooses a token $r_B \in_R \mathsf{SHA1PRNG}[128]$ and creates a QR code containing both the signature over r_A and his own token.
3. $\mathcal{P}_a$ scans the QR code presented by $\mathcal{P}_b$ and verifies σ with respect to r_A using $\mathcal{P}_b$'s long-term public key. $\mathcal{P}_a$ then creates a signature σ' over (r_B, r_A) using his long-term private key and creates a QR code containing σ'.
4. $\mathcal{P}_b$ scans the QR code presented by $\mathcal{P}_a$ and verifies σ' with respect to (r_B, r_A) using $\mathcal{P}_a$'s long-term public key.

If the verification in Step 3 is successful, $\mathcal{P}_a$ is assured that $\mathcal{P}_b$'s long-term public key is authentic and $\mathcal{P}_b$ does know the corresponding private key. Likewise, if the verification in Step 4 is successful, $\mathcal{P}_b$ is assured that $\mathcal{P}_a$'s long-term public key is authentic and $\mathcal{P}_a$ does know the corresponding private key. In comparison to simply reading out or scanning two fingerprints, a mutual cryptographic authentication that also requires to demonstrate knowledge of the respective private key requires the creation of one additional QR code and thus one additional scanning process. Though we believe that the above solution works it requires some overhead since signatures are not included in TextSecure, yet. Therefore we propose another mitigation in the next section (and analyze it in section 3.4) that blends well with the cryptographic primitives that are already implemented in TextSecure.

3.3.6.2 Alternative Authentication Method

The challenge response process detailed below can achieve cryptographic authentication with the primitives already used in TextSecure. The process is as follows:

1. $\mathcal{P}_b$ chooses a keypair $(r, g^r) \in_R \mathbb{Z}_p \times \mathsf{Curve25519}$. It creates a QR code containing $\mathsf{chall} = g^r$.
2. $\mathcal{P}_a$ scans the QR code, derives $\mathsf{resp} = \mathsf{chall}^a$ $(= g^{r \cdot a})$ and creates a QR code containing resp.
3. $\mathcal{P}_b$ scans the QR code presented by $\mathcal{P}_a$ and checks if $g^{a \cdot r} =^? \mathsf{resp}$.

If resp matches $g^{a \cdot r}$, $\mathcal{P}_b$ is assured that $\mathcal{P}_a$ knows the private key corresponding to the long-term public key that $\mathcal{P}_b$ expected to belong to $\mathcal{P}_a$. If $\mathsf{resp} \neq g^{a \cdot r}$, the authentication fails. Here, we call $\mathcal{P}_a$ prover and $\mathcal{P}_b$ verifier. The process can then be repeated with reversed roles to achieve mutual authentication of $\mathcal{P}_b$ and $\mathcal{P}_a$. We discuss the security of this approach in detail in the following section.

3.4 Security of TextSecure Key Exchange and Messaging

As mentioned in Section 3.3, the underlying problem that allows for our attacks is that the shared key between two parties is not cryptographically authentic. In the same section we explain how to face this problem. In this section we first show that the method proposed in Section 3.3.6.2 actually solves this issue (under some reasonable restrictions). Next, we show that if public keys are authentic then so is $k = (\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$. Moreover we show k to be uniformly distributed. Once we have established this, we finally show that the encryption block of TextSecure is actually one-time authenticated encryption (a primitive which needs uniformly distributed and authenticated keys to provide security).

3.4.1 Offline Verification gives authenticity

In this section, we prove the proposed protocol of Section 3.3.6.2 to be secure. Ideally, we could prove the possession of the secret key through a *zero knowledge proof of knowledge*. However, we do not know if the protocol satisfies this strong property. Rather we prove that if an "honest" verifier accepts, then with high probability the prover's public key is unique. Since the public key is later on used in the means of authentication, this gives authenticity. Note that we do not consider collusion of parties that deliberately agree to use the same long-term key pair here. In this case, attacks as above are always possible. We assume in the following that no two parties collude.

Consider the security game that is depicted in Figure 3.6. We say that an attacker (t, ϵ)-wins the security game if it runs in time t and $\mathcal{C}$ outputs 1 with probability at least ϵ. We argue that this security game actually models malicious behaviour of an adversary convincing an "honest" verifier to have public key g^a. To this end, suppose that $\mathcal{A}$ publishes the public key g^a that is already registered as the public key of $\mathcal{P}_a$ as its own public key. If $\mathcal{C}$ returns 1, then $\mathcal{A}$ is obviously successful in convincing a party that follows the protocol honestly that g^a is authentic with respect to $\mathcal{A}$ (which it is not). However, if the probability that $\mathcal{A}$ succeeds in the security game is *small* then this is not very likely to happen which means that g^a is registered only once

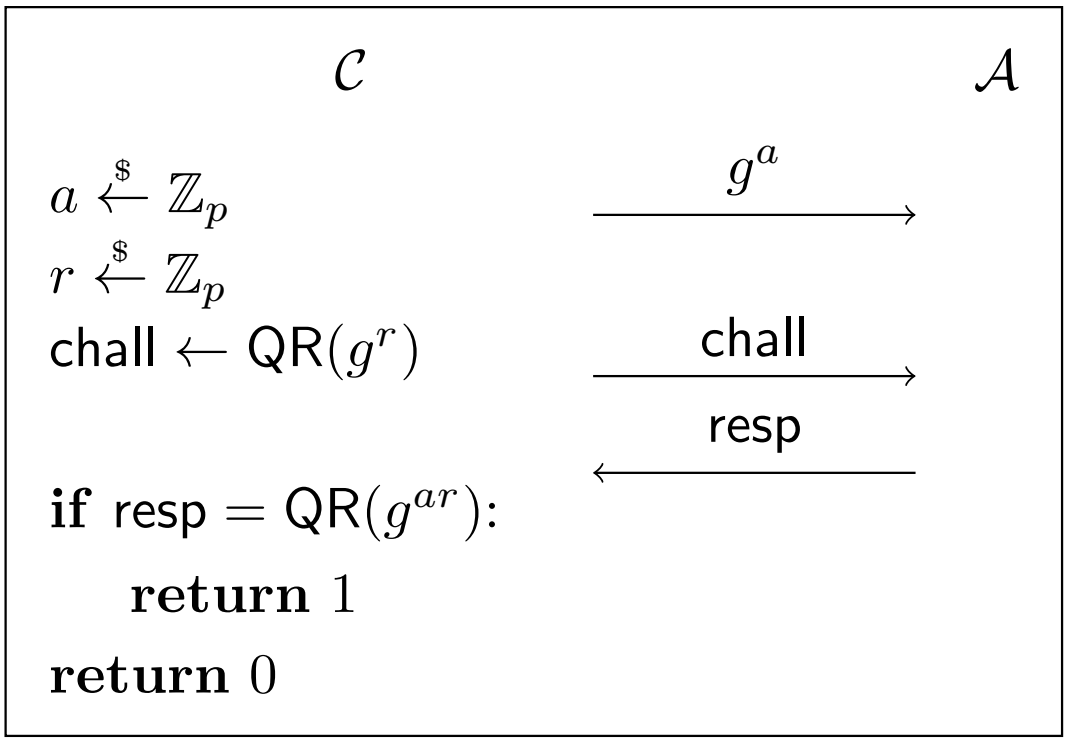


Figure 3.6: Challenge and Response security game.

with overwhelming probability[13]. We note that we do not allow the adversary to challenge back the challenger. This somewhat artificial restriction is due to the fact that we cannot prove the scheme to be secure without this requirement (see below). We remark, however, that this protocol is carried out when prover and verifier meet face-to-face and, moreover, that no data is sent through the data channel. Thus, the adversary is not a network attacker and in particular has not the capability to read, replay, delay, alter, or drop any data that is exchanged throughout a protocol run between two honest parties. A proving party will always be able to confirm who the verifier actually is and may refuse to prove something to a seemingly malicious verifier.

Technically, if we wanted to get rid of this requirement we could use an IND-CCA-secure KEM that supports public keys of the form g^a in elliptic curves, *e. g.*, the PSEC-KEM that is standardized by the ISO [120] and proven to be secure by Shoup [208] in the random oracle model.

[13] Observe that the probability that two parties following the protocol honestly publish the same public key is $\frac{1}{|\mathsf{Curve25519}|}$

Claim 1. *If there is an attacker $\mathcal{A}$ that (t, ϵ)-wins the above game (cf. Figure 3.6) then there is an algorithm $\mathcal{B}$ that (t', ϵ')-breaks the* DDH *assumption in* Curve25519 *where $t \approx t'$ and $\epsilon \leq \epsilon'$.*

Proof. Suppose $\mathcal{A}$ wins the game with probability ϵ. We construct a DDH-distinguisher $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine. $\mathcal{B}$ gets as input (g, g^a, g^b, g^γ) and wants to distinguish whether $\gamma = ab$ or not. $\mathcal{B}$ simulates $\mathcal{C}$ as follows: It creates a QR code containing g^b. If $\mathcal{A}$ creates a QR code containing g^γ then $\gamma = ab$ and thus $\mathcal{B}$ is able to solve DDH. □

3.4.2 $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ is authentic.

Now let us assume public keys are unique. We argue that in this case $k = (\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ authenticates sender and receiver since these are the only parties to compute k. Here, we show that the sender $(\mathcal{P}_a)$ is authenticated. The proof for the receiver is similar. To this end, we define the following algorithm that reflects key derivation (for the first message sent during a session) on the side of the receiver $(\mathcal{P}_b)$ where g^a, $g^{\bar{x}_{a,0}}$ and $g^{\bar{x}_{a,2}}$ are long-term and ephemeral public keys of $\mathcal{P}_a$, b is the long-term secret of $\mathcal{P}_b$ and z is a pointer to the prekey pair $(x_{b,z}, g^{x_{b,z}})$ (cf. Figure 3.3).

Algorithm 2 $\mathsf{Key.derive}\left(g^a, g^{\bar{x}_{a,0}}, g^{\bar{x}_{a,2}}, b, z\right)$

$\mathsf{secret} \leftarrow f\left(g^{x_{b,z} \cdot a}, g^{b \cdot \bar{x}_{a,0}}, g^{x_{b,z} \cdot \bar{x}_{a,0}}\right)$
$(k_{BA,r}, k_{BA,c}) \leftarrow f\left(\mathsf{secret}, \mathsf{const}_0, \mathsf{const}_R\right)$
$k_{\text{shared}} \leftarrow g^{\bar{x}_{a,2} \cdot x_{b,z}}$
$(k_{AB,r}, k_{AB,c}) \leftarrow f\left(k_{\text{shared}}, k_{AB,r}, \mathsf{const}_R\right)$
$k \leftarrow f\left(MAC(k_{AB,c}, \mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K\right)$

Let us now consider the security game that is depicted in Figure 3.7. We say that $\mathcal{A}$ (t, ϵ)-wins the security game if it runs in time at most t and $\mathcal{C}$ returns 1 with probability at least ϵ. We argue that this security game acutally models authenticity of k. To this end, suppose again that g^a is the public key of $\mathcal{P}_a \neq \mathcal{A}$. Note that this public key is unique. It is given to $\mathcal{A}$, together with the public key g^b and a prekey $g^{x_{b,z}}$ of party $\mathcal{P}_b$ by computing k. The goal of $\mathcal{A}$ is to impersonate $\mathcal{A}$ to $\mathcal{P}_b$. If $\mathcal{A}$ is able to succeed, then it may obviously break the authenticity property. On the other hand, if the probability for $\mathcal{A}$ to succeed is small it is very unlikely for $\mathcal{A}$ to compute the shared key of two honest parties $\mathcal{P}_a$ and $\mathcal{P}_b$. By the uniqueness of public keys this gives authenticity.

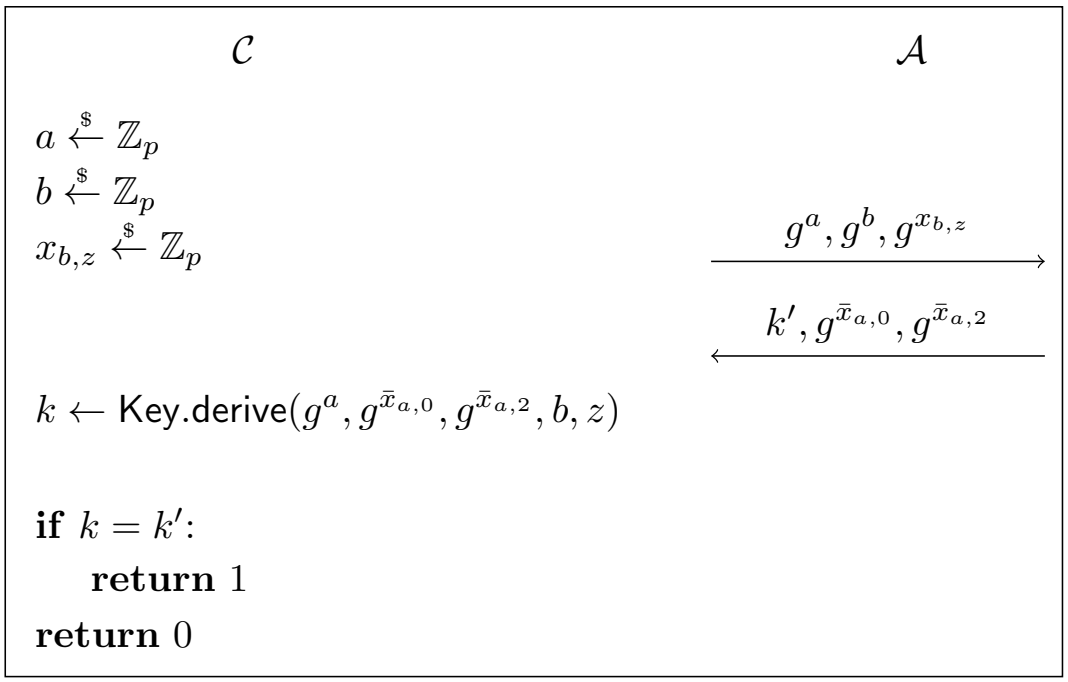


Figure 3.7: Authenticity of k security game.

Claim 2. *If an attacker $\mathcal{A}$ (t, ϵ)-wins the above security game (cf. figure 3.7), then there is an algorithm $\mathcal{B}$ that (t', ϵ')-breaks the* CDH*-assumption in* Curve25519 *where $t \approx t'$ and $\epsilon \leq \epsilon' + \frac{t'}{2^{512}}$.*

Remark 2. *Following Krawczyk [136], the analysis will view the functions f and MAC as a* non-programmable *random oracle [87] for the proof of claim 2.*

Proof. We describe a CDH-forger $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine. $\mathcal{B}$ gets as input $(g, g^a, g^{x_{b,z}})$ and wants to compute $g^{a \cdot x_{b,z}}$. It samples $b \xleftarrow{\$} \mathbb{Z}_p$ and sends $(g^a, g^b, g^{x_{b,z}})$ to $\mathcal{A}$. Note that $\mathcal{B}$ is not able to compute Key.derive. Instead we let $\mathcal{B}$ always return 0. We argue that with overwhelming probability this will not be detected by $\mathcal{A}$: We observe that since f is modeled as a random oracle (and thus the image of f is uniformly distributed over $\{0,1\}^{512}$) for $\mathcal{A}$ to tell the value of k it needs to query f on $\mathrm{MAC}_{k_{AB,c}}((\mathsf{const}_1), \mathsf{const}_0, \mathsf{const}_K)$ since otherwise the value of k is information-theoretically hidden from $\mathcal{A}$. The same argument applies for the value of $\mathrm{MAC}_{k_{AB,c}}$ (which is needed to compute k), $k_{BA,r}$ (which is needed to compute $k_{AB,c}$) and secret (which is needed to compute $k_{BA,r}$). Now, suppose $\mathcal{A}$ queries f on $\mathsf{secret} = (g^{a \cdot x_{b,z}}, g^{b \cdot \bar{x}_{a,0}}, g^{x_{b,z} \cdot \bar{x}_{a,0}})$ for some $\bar{x}_{a,0}$. Since f is modeled as a random oracle, the

query of $\mathcal{A}$ is actually public and thus $\mathcal{B}$ can extract $g^{a \cdot x_{b,z}}$, the solution to the CDH instance. Thus, if CDH is hard in Curve25519 for $\mathcal{A}$, to be successful, $\mathcal{A}$ needs to correctly guess a bitstring of length 512. This probability can be neglected. □

We immediately obtain that k is not only authentic but also uniformly distributed. Now, a similar (information theoretic) argument applies to keys that are derived from k for the next messages to be sent and received.

We stress that for our proof to be bootstrapped to the setting with more than one prekey of $\mathcal{P}_b$, these have to be pairwise distinct (*i. e.*, unique) since otherwise replay attacks become possible. In particular, Claim 2 does not apply to keys exchanged relying on the *last resort key*.

3.4.3 TextSecure Encryption is One-time Authenticated

Next, we prove the actual encryption of TextSecure to be one-time authenticated encryption. We stress that one-time security suffices for TextSecure since, here, the actual encryption and MAC keys are updated (and can be seen as fresh, cf. previous section) with every message to be sent.

3.4.3.1 Cryptographic Primitives

We shortly recall the cryptographic primitives that are used by the TextSecure encryption procedure.

A *message authentication code* is a pair of PPT algorithms $\mathsf{MAC} = (\mathsf{Tag}, \mathsf{Vfy})$ such that $\mathsf{tag} \xleftarrow{\$} \mathsf{Tag}(k, m)$ on input a key k and a message m returns tag for that message. The algorithm $\{0,1\} \leftarrow \mathsf{Vfy}(k, m, \mathsf{tag})$ returns $\mathsf{Tag}(k, m) \stackrel{?}{=} \mathsf{tag}$. We require the usual correctness properties. Following Bellare et al. [29, 33] we say that an attacker $\mathcal{A}$ (t, ϵ)-breaks the strong one-time security of MAC if it runs in time t and

$$\mathsf{Pr}\left[(\mathsf{tag}, m) \xleftarrow{\$} \mathcal{A}^{\mathsf{Tag}(k,)} : \begin{array}{l} \mathsf{Vfy}(k, m, \mathsf{tag}) = 1 \\ \wedge\, (\mathsf{tag}, m) \neq (\mathsf{tag}', m') \end{array}\right] \geq \epsilon$$

where $\mathcal{A}$ is allowed to query Tag at most one time (the query is denoted by m' and the response by tag').

A *symmetric encryption scheme* is a pair of PPT algorithms $\mathsf{SE} = (\mathsf{Enc}, \mathsf{Dec})$ such that $c \xleftarrow{\$} \mathsf{Enc}(k, m)$ on input a key k and a message m returns a ciphertext c and $m \leftarrow \mathsf{Dec}(k, c)$ on input a key k and a ciphertext c outputs m. We require the

usual correctness properties. We say that an attacker $\mathcal{A}$ (t, ϵ)-breaks the one-time IND-CPA-security [30] of SE if it runs in time t and

$$\mathsf{Pr}\left[b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Encrypt}(\cdot,\cdot)} : b = b'\right] \geq \epsilon$$

where $\mathcal{A}$ is allowed to query Encrypt at most one time on two messages m_0 and m_1 of equal length and Encrypt samples a uniformly random bit b and returns $c \xleftarrow{\$} \mathsf{Enc}(k, m_b)$.

3.4.3.2 Authenticated Encryption

An authenticated encryption scheme is a symmetric encryption scheme that also provides authenticity. Security of a Authenticated Encryption scheme AE is captured through a security game that is played between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$ [32]

- The challenger samples $b \xleftarrow{\$} \{0, 1\}$ and a key $k \xleftarrow{\$} \mathcal{K}$ and initializes a set $\mathcal{S} = \emptyset$.
- The adversary may then query each of the Encrypt and Decrypt oracles once and they respond as depicted in Figure 3.8.
- Finally, $\mathcal{A}$ outputs a bit b' and wins if $b = b'$.

$\mathsf{Encrypt}(m_0, m_1, hd)$
$(C^0) \xleftarrow{\$} \mathsf{AE.Enc}(k, hd, m_0)$
$(C^1) \xleftarrow{\$} \mathsf{AE.Enc}(k, hd, m_1)$
if $C^0 = \perp \vee C^1 = \perp$: **return** $\perp$
$\mathcal{S} \leftarrow \mathcal{S} \cup \left\{C^b\right\}$
return C

$\mathsf{Decrypt}(C, hd)$
$m \leftarrow \mathsf{AE.Dec}(k, hd, C)$
if $C \in \mathcal{S}$: $\mathsf{div} \leftarrow 1$
if $\mathsf{div} = 1 \wedge b = 1$ **return** m
return $\perp$

Figure 3.8: Encrypt and Decrypt Oracles in the Authenticated Encryption security game.

Since the behaviour of oracle Decrypt might not be clear at first sight, we will shortly explain it. First, we recall that the goal of the adversary is to determine the bit b. Stated otherwise, if the adversary queries Encrypt on two *distinct* messages of equal

length its goal is to determine which message is encrypted. Oracle Decrypt will reveal this information to $\mathcal{A}$ if either $\mathcal{A}$ manages to query Decrypt for a ciphertext that is not authentic, i.e., $C \notin \mathcal{S}$ and that nonetheless does not incur in a decryption error.

3.4.3.3 Authenticated Encryption in TextSecure

For TextSecure we let hd contain at least $(g^{\bar{x}_{a,2}}, \mathsf{regID}_a, \mathsf{regID}_b, ctr, pctr)$, the ephemeral public key of party $\mathcal{P}_a$, the identifiers of both parties and the counters ctr and pctr that allow the receiver to determine which key is to be used for decryption. The key is set to be $k = (\mathsf{k}^x_{\mathsf{Enc}}, \mathsf{k}^x_{\mathsf{MAC}})$ handed over by the ratchet protocol.

Algorithm 3 $\mathsf{TS.Enc}(k, hd, m)$

$c \leftarrow \mathsf{Enc}_{\mathsf{k}^x_{\mathsf{Enc}}}(m)$
$\chi \leftarrow (v, g^{\bar{x}_{a,2}}, \mathsf{ctr}_a, \mathsf{pctr}_a, c)$
$\mathsf{tag} \leftarrow \mathsf{MAC}_{\mathsf{k}^x_{\mathsf{MAC}}}(\chi)$
return $\mathbf{C} \leftarrow (\chi, \mathsf{tag})$

Algorithm 4 $\mathsf{TS.Dec}(k, hd, \mathbf{C})$

parse $\mathbf{C}$ **as** $\mathbf{C} = (\chi', \mathsf{tag}')$
parse χ' **as** $\chi' = (v', (g^{\bar{x}_{a,2}})', \mathsf{ctr}'_a, \mathsf{pctr}'_a, c')$
$\mathsf{tag}'' \leftarrow \mathsf{MAC}_{\mathsf{k}^x_{\mathsf{MAC}}}(\chi')$
if $\mathsf{tag}'' \neq \mathsf{tag}'$ **return** $(\perp, \perp)$
$m \leftarrow \mathsf{Dec}_{\mathsf{k}^x_{\mathsf{Enc}}}(c)$
if $m = \perp$ **return** $(\perp, \perp)$
return m

Theorem 1. *If there is an attacker $\mathcal{A}$ that (t, ϵ)-breaks the one-time authenticated encryption security of* TS.Ste *then there is an attacker $\mathcal{B}_{\mathsf{MAC}}$ that (t_1, ϵ_1)-breaks the strong one-time security of* MAC *and an attacker $\mathcal{B}_{\mathsf{Enc}}$ that (t_2, ϵ_2)-breaks the one-time* IND-CPA *security of the encryption scheme where $t \approx t_1 \approx t_2$ and*

$$\epsilon \leq \epsilon_1 + \epsilon_2$$

Proof. The proof is by a sequence of games. First, we will modify the Decrypt oracle such that its response will be independent of b. If the MAC scheme is secure this change will be undetected by $\mathcal{A}$. After that we will rely on the security of Enc to argue that $\mathcal{A}$ has only negligible advantage in telling the value of b. By ζ_i we will denote the event that $\mathcal{A}$ is successful in Game i.

GAME 0. This is the real Authenticated Encryption security game as described above. Thus, we have:

$$\mathsf{Pr}[\zeta_0] = \epsilon$$

GAME 1. In Game 1, the challenger proceeds as follows: When the attacker queries a ciphertext C to the decryption oracle, $\mathcal{C}$ always outputs $\perp$. Except for this, $\mathcal{C}$ proceeds exactly as in Game 0. Note that Decrypt will return always $\perp$ if $\mathsf{div} = 0$ (i.e., conditioned on $\mathsf{div} = 0$ Game 0 and Game 1 are identical) and will return m only if $\mathsf{div} = b = 1$. Now, if $\mathsf{div} = 1$ then $C \notin \mathcal{S}$. In this case the ciphertext C that was queried to Decrypt by the adversary contains a valid tag which was not computed before: Either $\chi \neq \chi'$ and thus tag authenticates another message than the one that queried by oracle Encrypt to compute the tag (if there was a query it all). Or $\chi = \chi'$ but $\mathsf{tag} \neq \mathsf{tag}'$. By the *strong* one-time security of MAC (Recall that strong one-time security requires that it is computationally infeasible to compute a valid tag, tag', for message m, even if a valid tag, tag, for message m is known.) we have:

$$\mathsf{Pr}[\zeta_0] - \mathsf{Pr}[\zeta_1] \leq \epsilon_1$$

GAME 2. In Game 2, instead of encrypting m_b, $\mathcal{C}$ samples a random message m, computes $C \xleftarrow{\$} \mathsf{TS.Enc}(k, hd, m)$ and returns C. This change does not affect oracle Decrypt since due to Game 1 it will return $\perp$ anyway. Now, by the IND-CPA-security of Enc this goes unnoticed from the adversary. Thus, we have:

$$\mathsf{Pr}[\zeta_1] - \mathsf{Pr}[\zeta_2] \leq \epsilon_2$$

Claim 3. $\mathsf{Pr}[\zeta_2] = 0$.

Proof. In Game 2 the Decrypt oracle reveals no information about b due to Game 1. Neither does the Encrypt-oracle due to Game 2. The claim follows. □

3.5 Summary

In this chapter we provided an extensive review and a detailed security analysis of TEXTSECURE's protocol. We have shown peculiarities of TEXTSECURE and potential weaknesses, and that our improvement of the TEXTSECURE protocol mitigates the UKS attacks. Therefore we have shown, that the protocol from section 3.3.6.2 proves long-term public keys to be unique, *i. e.*, that with overwhelming probability no two parties share the same public key (except if they collude). Then we proved that $(\mathsf{k}_{\mathsf{Enc}}, \mathsf{k}_{\mathsf{MAC}})$ are bound to the respective long-term keys of the parties which

gives authentic keys (that are also uniformly distributed). Finally, we established that TextSecure's encryption is actually one-time authenticated. To the best of our knowledge, this is the first formal verification of the security guarantees offered by the tool. In the larger context of this thesis, we offered a contribution to advance on the *precondition B* posited in the introduction. The insight that can be attained from our research allows to make a more informed decision on whether to use TextSecure and possibly in which context, as well as, whether the trust a user extends to TextSecure is justified insofar that it achieves its self-set goals.

4 Detection and Mitigation of Malicious Code on Websites

Distrust & caution are the parents of security.
—Benjamin Franklin, *Poor Richard's Almanack*

In their current state of imperfectness we must assume that systems are flawed and that some flaws are exploitable. However, while for the time being we may need to acknowledge that attackers will try to compromise a system and may need to accept that they sometimes succeed, what we can do is to retrofit existing systems with means of detection and—in the best case—mitigation of attacks. In this chapter we thus approach the *precondition C* and introduce a way to detect and mitigate drive-by download attacks and other malicious code directly in the browser.

4.1 Introduction

During the last few years, we observed a shift in attacks against end users: instead of attacking network services, many of today's attacks focus on vulnerabilities in client applications. Especially the web browser is a popular target for attackers. There are many different kinds of threats and attack vectors against current browsers, such as:

- *Drive-by download attacks* in which a vulnerability in the web browser or one of its components/extensions (*e. g.*, Acrobat Reader or Flash plugins) is exploited to execute code of the attacker's choice [199]. Often, the goal of a drive-by download attack is to download (and execute) a so-called *loader* [95], the first

stage of a malware infection of the victims computer, which aims at disabling potential defenses and then downloads and installs the actual malware before removing itself from the now infected system.

- *Cross-Site Scripting (XSS) vulnerabilities* that enable an attacker to inject arbitrary client-side scripts into web pages [131, 151, 226]. XSS vulnerabilities offer a wide variety of uses to the attacker, among them the theft of private information.
- *Clickjacking* (also known as *UI redressing*) is a technique in which an attacker tricks a web site visitor into clicking on an element of a different page that is only barely (or not at all) visible [21].

These and similar attack techniques target different vulnerabilities within a browser or one of its components. The root cause of this problem is the fact that an attacker can compromise the integrity of almost all DOM properties of a website by injecting malicious JavaScript code into the website's source code. Current defense mechanisms, as well as, reactive analysis and forensic approaches are often slow or complicated to set up and conduct, since an attacker can use many different ways to obfuscate the code or make it hard to obtain a copy of the code in the first place. Several techniques attempting to address this problem have been proposed. On the one hand, there are analysis frameworks such as Wepawet [64], performing an offline analysis of a given page in order to detect drive-by download attacks. Cujo [203] performs an online analysis, but introduces an overhead of more than 1.5 seconds on JavaScript-heavy sites such as Facebook, which negatively impacts the user experience. On the other hand, there is a huge body of work in which different techniques are proposed to avoid attacks in the first place [97, 158, 224]. Approaches such as Gatekeeper [97] or *Google Caja* [158] attempt to find a way to execute arbitrary JavaScript in a secure environment. Such attempts typically require working on a subset of the complete JavaScript specification, *e. g.*, Gatekeeper removes language constructs such as `eval()` and `document.write()` from the JavaScript specification for their analysis. Complementary to these approaches are novel browser designs, such as Gazelle [224], constructed to address these problems from the ground up. However, as such approaches tend to focus on a limited range of attack vectors or lack compatibility with the current infrastructure, many do not effectively mitigate current threats for the user.

4.1.1 Contribution

In this chapter, we introduce IceShield, a novel approach to perform light-weight instrumentation of JavaScript, detecting a diverse set of attacks against the DOM

tree, and protecting users against such attacks. The instrumentation is light-weight in the sense that IceShield runs directly *within* the browser, as it is implemented in JavaScript and does not require any external components. Thus, the runtime overhead is low and IceShield even works on embedded browsers used, for example, in modern smartphones. By performing dynamic analysis, we do not need to worry about obfuscation since we can inspect the attack attempt during runtime, exactly at the point where the payload is being decoded, *i. e.*, we can interpret its plain-text. Furthermore, our approach does not rely on proprietary features and is thus applicable with every major browser, since the detection is implemented in JavaScript. It is thus also portable across platforms.

Special care needs to be taken to implement the instrumentation in a robust and tamper resistant way: since the tool is implemented in JavaScript, an attacker could try to overwrite our analysis functions during runtime. We rely on a new way for tamper resistant meta programming in modern browsers, based on safely overwriting JavaScript core methods and DOM properties with a minimal performance overhead [104, pp. 123-133]. The basic idea is to take advantage of a new feature available since ECMA Script 5 (ES5) called `Object.defineProperty()` [171]. This allows us to *freeze* object properties, as well as, methods and native DOM properties and prevent them from being modified afterwards. This enables us to mitigate attacks against our instrumentation, where an attacker tries to overwrite the respective method again, reset an overwritten method or access the native methods we have overwritten to bypass the inspection and detection process. This approach works on all modern browsers supporting ES5 and later.

As IceShield performs the analysis directly in the browser, it can also mitigate attacks and protect users and websites utilizing the tool. It allows us to identify the suspicious elements within a page and change them accordingly. Aiming for a minimal impact in case of false positives, we pad the parameters in question to destroy the payload of the potential exploit, but avoid visible impact on the rendered website. This enables us to actually protect users from attacks, with only a very low perceivable percentage of false positives.

We have devised a prototypical implementation of IceShield, which we evaluated in different system contexts: an average workstation, a low-performance netbook and a smartphone. The runtime overhead of IceShield averages below 12 ms for the workstation and at 89 ms for the smartphone, and we achieved a detection rate of 98% using live malicious websites. What is more, we also were able to detect three exploits that the tool could not have seen before, as they were significantly newer than the training set, and show that attacks can be mitigated successfully.

4.1.2 Outline

We start out with an introduction to the problem space and also discuss related work, before delving into our solution and its implementation. The remainder of the chapter is structured as follows: in Section 4.2 we introduce the technical background and basic assumptions underlying our approach, as well as, discuss related work. In Section 4.3 we describe our system design, detail heuristics and scoring metric that form the basis for detecting malicious code, and elaborate on our implementation. We end this chapter with an evaluation of our system and a discussion of its limitations.

4.2 Technical Background

In this section we present the technical background that motivates our approach and present our basic idea. We also outline our detection and protection framework and illustrate its detection range with attack patterns we observed in the wild. At the end of the section, we discuss related work.

4.2.1 Motivation and Basic Idea

We assume that almost every JavaScript based attack will have to use native methods at some point in order to prepare necessary data structures (*e. g.*, to store the shellcode on the heap or stack) and afterwards perform the actual exploit by triggering a vulnerable function. This is true for heap and JIT spraying attacks, exploits against vulnerabilities in a browser plug-in or the user agent itself, as well as security issues in particular websites. The data set of malicious code samples we assembled during the testing phase of IceShield showed that most malicious scripts use native JavaScript methods such as `concat()`, `unescape()`, `substring()`, and similar string functions [174] during preparation and deployment of their malicious payload. The exploit code utilizing these functions is usually heavily obfuscated, making static code analysis and detection cumbersome and difficult. The four JavaScript code examples shown in Listing 4.1 illustrate several novel obfuscation techniques introduced and discussed on `sla.ckers.org`. These code snippets are meant to be a proof-of-concept, thus performing nothing more than a simple call to `alert(/* some data */)`.

Listing 4.1: Obfuscated JavaScript code samples executing the alert() method

```
1) ({0:#0=alert/#0#/#0#(1)});
2) (1..__proto__.e0=alert)(1.e0);
```

```
3) a=a setter=alert;
4) _=[[$,__,,$$,,_$,$_,_$_,,,$_$]=!''+[!{}]+{}]
     [_$_+$_$+__+$],_()[_$+$_+$$+__+$](-~$)
```

Especially the last of the four examples in Listing 4.1 is hard to analyze since it takes advantage of non alpha-numeric characters. It is an example of so-called non-alnum obfuscation [104, pp. 67-68], as first introduced by Hasegawa[14]. This demonstrates the enormous versatility and flexibility of JavaScript and underlines the difficulty of static JavaScript code analysis. Furthermore, JavaScript allows an attacker to create morphing code, a fact that has recently been demonstrated by Heyes et al. [113]. This suggests that an attacker can render any signature based malware detection lacking advanced de-obfuscation routines useless, similar to the limitations of signature based shellcode [211] and malware [183] detection. In addition, filtering mechanisms working on a layer different than the layer to actually protect against attacks are not capable of detecting obfuscated code as for example demonstrated by the large amount of bypasses against the Webkit XSS Auditor [23] and the Internet Explorer 8 XSS filter [135].

With IceShield, we introduce a new approach to detect and mitigate attacks against web browsers and to protect the integrity of the DOM. We do not rely on any form of static code analysis, but rather the creation of an alternative and light-weight execution context that can be deployed as a script on arbitrary websites or as a browser extension. We use inline code analysis such that obfuscation does not hamper our inspection: we can perform the analysis after the de-obfuscation has taken place and can analyze the exploit attempt in clear text. The analysis itself is based on detecting attack patterns of suspicious behavior. We describe these patterns in heuristics similar to the ones proposed by Wepawet [64] and Cujo [203], but we demonstrate how such features can be extended to cover other attack vectors and be used in a live analysis rather than in an offline setting. IceShield can be run in a low prioritized execution context such as being included on a website protecting the user of this website from attacks embedded in the website (*e. g.*, via banner advertisements). The tool can also be deployed as a browser extension or injected via a proxy to provide a better protection range and independence from the individual websites potentially including IceShield. Our approach aims to have minimal footprint and overhead, and we propose a novel way of JavaScript property mimicking which we discuss in detail in Section 4.3.

[14] Hasegawa, Y., Re: New XSS vectors/Unusual Javascript, `http://sla.ckers.org/forum/read.php?2,15812,28465#msg-28465` (June 2009, accessed 03.Oct.2014)

4.2.2 Dynamic Detection and Protection Framework

IceShield attempts to accomplish several different goals. The first and most important is to provide the possibility to analyze drive-by download attempts at the time a malicious websites tries to execute code in the context of the victim's browser. By performing this analysis *within the context* of an actual browser, we are able to analyze the code dynamically. Thus, IceShield is not affected by any level of code obfuscation since it can analyze the code *after* the decoding/decrypting has finished. This is achieved by dynamically instrumenting objects and functions, and providing an execution context in which we can analyze their behavior. The instrumentation enables us to perform parameter analysis allowing inspection of the called methods and their parameters during runtime. With a set of heuristics and a scoring based attestation trained with data mining techniques, IceShield can determine, if the combination of method call and parameter setup indicates malicious intent. To illustrate the expressiveness of the approach, we use a set of heuristics to detect different kinds of attacks. Besides new features, we use several heuristics similar to the ones implemented in Wepawet. The set of heuristics can easily be extended to enhance IceShield's detection features in case completely novel attack vectors become known.

Second, we aim at protecting users against malicious websites: once IceShield has detected an exploitation attempt, we are able to manipulate potentially malicious code before an attack takes place. This can, for example, be achieved by modifying or removing malicious content from the DOM tree. This enables us to protect the victim from the full consequences of an attack and provide detailed information on the attack technique itself. Preliminary results suggest that this approach is effective in practice and enables us to effectively mitigate attacks.

The third goal is to implement the instrumentation in a light-weight and tamper resistant manner. On the one hand, the overhead of our analysis framework should be low such that the temporal impact is small and hardly noticeable by a user. On the other hand, an attacker should not be able to remove our instrumentation since this would enable a way to bypass our system. We achieve these two objectives by implementing our instrumentation in JavaScript and introducing a novel way to use latest features of ES5 [104, p. 133]. If the browser correctly implements ES5 functionality, it is hard for an attacker to bypass the system.

In empirical measurements, we show that the overhead is small: on average, our instrumentation has an overhead of a few tens of milliseconds even on low-end systems, which is significantly small compared to the loading time of a web page. The framework can be used on different browsers and it is portable since IceShield does not depend on specific features or proprietary extensions.

We successfully tested IceShield with all modern major browsers such as Firefox 33, Chrome 37, Safari 5, and Internet Explorer 9. This enables a deployment of IceShield on many different devices in diversity and number. For each page a user visits, IceShield monitors the behavior of this site by dynamically analyzing the code that was supposed to be executed. In case this is an exploitation attempt, a sanitized version of the analysis reports generated by IceShield could in theory be sent to a central logging site. This enables us to gather information about latest attack vectors since we can take advantage of the diverse user base that uses IceShield. Existing tools are either customized to one specific environment (*e. g.*, HoneyMonkey uses a series of machines with different configurations [225]) or emulate different browsers like Wepawet does. The shortcoming of emulation is that it is hard (if not impossible) to mimic all aspects of different browsers, *e. g.*, the large number of browser plugins significantly increases the attack surface that needs to be covered. In contrast, IceShield can be executed within the context of a diverse set of browsers, and thus can spot attacks even in obscure settings, as long as at least one user is affected by such attacks and runs IceShield.

4.2.3 Detection Range

Our approach aims to detect a wide array of possible client side attack techniques including malware deployments as well as XSS and related attacks. The general attack patterns that can be detected by our approach can be categorized into the following items and we designed our detection heuristics accordingly (see Section 4.3.1):

1. Read access to sensitive DOM properties such as `document.cookie`, URL tokens, Form and link tokens, and sensitive HTML element attribute values

2. Modification of existing DOM properties, form actions, link targets, `src`, `href`, `data` and comparable attributes, `document.cookie`, event flow modifications and modification of arbitrary `innerHTML` / `outerHTML` properties

3. Creation of outbound information channels such as elements loading off-domain resources and modifications of existing `src`, `href`, `data` and comparable attributes

4. Loading external resources such as dangerous tags via `createElement` / `createElementNS`, modifications of existing `src`, `href`, `data` and comparable attributes, as well as, modification of arbitrary `innerHTML` / `outerHTML` properties

5. Overlapping of existing elements, framing of targeted websites, markup injections and absolute positioning, changing of element dimensions and conditional display
6. Creation of invisible DOM elements such as zero height/width elements, elements outside the view port, transparent elements, changing element alignment/z-index and conditional display

IceShield is capable of monitoring arbitrary HTML element attributes which enables us to detect even advanced attacks. It is possible to use regular expression based patterns to monitor certain attribute classes and groups, such as the new HTML5 custom data attributes [221], which were originally meant for web developers to add arbitrary attributes to their documents without destroying the markup validity.

4.2.4 Related Work

We are not the first to propose techniques to address the problem of malicious code on the web. We briefly discuss related work in this section and compare the different approaches to the one we present in this chapter.

In the last few years, several different kinds of low- or high-interaction honeyclients were introduced such as for example HoneyMonkey [225], Capture-HPC[15], SpyProxy [167], Monkey-Spider [118], or PhoneyC [181]. All of them can only be used in an (offline) analysis setting and are not capable of actually protecting end-users due to their high runtime overhead and the complexity involved when using them.

Wepawet/JSAND [64] and Cujo [203] are closely related to our approach. Wepawet is a framework to detect and analyze malicious JavaScript code in an offline setting. The tool combines anomaly detection techniques and dynamic emulation to analyze a given piece of code. Cujo uses similar heuristics to detect drive-by download, but performs the analysis on a web proxy. This approach introduces on average an analysis overhead of 500 ms and JavaScript-heavy sites such as Facebook might even introduce an overhead of more than 1.5 seconds.

Compared to these two tools, we use a similar set of detection heuristics, but IceShield can analyze the actual DOM tree within the browser and thus perform a more fine-grained analysis. Furthermore, the overhead is an order of magnitude lower compared to Cujo. In addition, our tool protects users from attacks since we can modify parameters passed to native methods to mitigate potential attacks.

[15] `http://projects.honeynet.org/capture-hpc`

An advantage of our approach compared to recent proposals such as Zozzle [67] is the light-weight implementation and the portability. However, our current prototype has a higher false-positive rate which could potentially be lowered by using more elaborated machine learning techniques.

IceShield neither necessarily requires user decision nor interaction to protect. Comparable solutions, like NoScript[16], rely on their users' decision-making abilities for effective protection—in case the user trusts a domain that spreads malicious code, the protection is rendered useless and will be bypassed.

4.3 System Design

In this section, we provide a detailed overview of the dynamic instrumentation and detection techniques used by IceShield. We discuss how such an instrumentation can be implemented in a robust way and present the different components and analysis techniques used by the tool.

4.3.1 Heuristics to Identify Suspicious Sites

The set of heuristics and rules can be kept slim, as the parameters we need to inspect are typically de-obfuscated by the script we observe, before they are assessed by the rules. This reduces the overhead to a minimum and allows us to conduct a more detailed analysis of malicious code, than any other form of de-obfuscation. We developed the heuristics by manually analyzing recent attacks and tailored the heuristics to allow for a detection of a wide variety of attack patterns, by generalizing what we observed before. Some heuristics are applied in a comparable way by Wepawet [64]. We enlarge the feature coverage, for instance, by also observing behavior like the creation of potentially dangerous elements. The existing heuristics serve as a proof-of-concept; new rules can easily be added to the system. However, when we initially published this research, our features already covered all relevant attack vectors. The list below describes the heuristics used in the prototype. The different features correspond to the attack patterns we have discussed in Section 4.2.3 and illustrate the range of attacks we can discover with IceShield. The individual rules are internally being applied with identifiers to protect the users privacy and ease later database-based analysis and cumulation:

[16] `https://addons.mozilla.org/en-US/firefox/addon/noscript/`

1. *External domain injection*: Having a script inject an external domain into existing HTML elements can indicate malicious activity that aims at commandeer existing links or forms. The prototype differentiates between injection of `<embed>`, `<object>`, `<applet>`, and `<script>` tags, as well as, `<iframe>` injections.

2. *Dangerous MIME type injection*: We highlight when a script sets a MIME type such as `application/java-deployment-toolkit` that changes how a browser interprets an existing DOM object.

3. *Suspicious Unicode characters*: When arguments to a native method contain characters like `%u0b0c` or `%u0c0c`, this can indicate a code execution attempt.

4. *Suspicious decoding results*: We record if decoding functions such as `unescape()` or `decodeURIComponent()` contain suspicious characters.

5. *Overlong decoding results*: When a decoding function receives an especially long argument, this can indicate the preparation of a payload. The prototype uses a threshold of 4096 characters, based on our previous evaluation of existing attacks and, in contrast, the behavior of benign sites.

6. *Dangerous element creation*: Some elements are found often in malicious contexts, such as `<iframe>`, `<script>`, `<applet>`. We record their use and distinguish between elements being created with and without an explicit namespace context.

7. *URI/CLSID pattern in attribute setter*: We note when an element attribute is set to an external URI, data/JavaScript URI or a Class ID (CLSID) string.

8. *Dangerous use of the* `innerHTML` *property*: We record when a script tries to inject a string containing HTML elements such as `<iframe>`, `<object>`, `<script>`, or `<applet>` into the value of an existing element.

4.3.2 Dynamic Instrumentation and Detection

We use inline code overwriting and hooking as the basic techniques to perform the instrumentation such that we can check for the heuristics introduced above. We create a context that allows us to inspect name and parameters of called functions dynamically at runtime, wrap the native JavaScript methods into this context and overwrite the original. We retain the original method inside IceShield's scope for later use, a form of overwriting also sucessfully applied in other contexts, such as binary analysis [83, 228].

The pseudo-code shown in Listing 4.2 demonstrates the basic principle of method overwriting and wrapping we utilize. If our heuristic analysis does not flag an attack attempt, we will call the original method without modifying its parameters. If IceShield's internal scoring mechanism indicates that a certain threshold has been reached, we can either completely block the method call in question, or can modify the set of arguments only and leave the intended code flow intact while still preventing the attack. For example, a typically long string of shellcode would simply be nulled before it can be passed to the original version of the method `unescape()`. Besides being able to mitigate attacks, this approach also allows us to better understand the actual flow of malicious code, although it may be heavily obfuscated.

Listing 4.2: Pseudo-code illustrating the instrumentation performed by us.

```
var method_to_overwrite_with = function() {
  var original = window.native_method;
  var result = analyze(
    this.name, this.arguments
  );

  if(result === 'malicious') {
    return defend_user();
  }

  return original.call(this, this.arguments);
}

Object.defineProperty(window, 'native_method',
{
  value:method_to_overwrite_with,
  configurable:false
});
```

As outlined before, IceShield makes use of an ECM Script 5 feature called `Object.defineProperty()` [171] to provide a robust implementation of the instrumentation. `Object.defineProperty()` allows to define new object properties, as well as, overwrite existing ones. Its capabilities also extend to methods and native DOM properties, while allowing us to provide a descriptor literal that specifies the option that should apply for the respective property. This descriptor literal can be be used with six different descriptors, which are:

1. *value*: The value that will be associated with the defined property.
2. *get*: A function can be assigned that will serve as a *getter*—it will be called as soon as get access to the property occurs.
3. *set*: A function can be assigned for serving as a *setter*—it will be called as soon as set access to the property occurs.

4. *writeable*: This descriptor influences if the property is read-only or not, default permission is read-write access.
5. *enumerable*: If set to false, the property will not be enumerated by a for-in loop such as `for(var a in object)`.
6. *configurable*: If this descriptor is set to false, it will make sure that the property is frozen and cannot be modified anymore. Neither can the property be deleted nor is the browser supposed to restore the original state on deletion.

The most relevant descriptor for IceShield is *configurable* and the possibility to set it to `false`, thereby *freezing* the property state. Freezing means that no other script can modify the property or any of its child properties again. It will not even be affected by a `delete` operation, which makes our approach tamper resistant against an attack, which tries to change or reset overwritten methods, or tries to bypass our inspection and detection process by accessing the original native methods. The same is true for property retrieval maneuvers that work on Gecko based browsers such as `Components.lookupMethod(top, 'alert')`—an attacker is unable to use this technique to bypass IceShield. The object freezing can also be accomplished by using the method `Object.freeze()`. Batch processing of several objects to be frozen at once can be accomplished by using `Object.defineProperties()` [170]. Heiderich elaborated on ECMA Script 5 Object Extensions and their protective potential in his PhD thesis [104, pp. 127-128, 133].

All modern user agents such as Firefox 4 and up, Chrome 6-10 and up, and Internet Explorer 9 and higher versions support object freezing. However, older or obscure browsers that do not fully support ES5 will not provide reliable tamper resistance for IceShield, which means that an attacker can potentially bypass the system. However, as most browsers have by now successfully implemented (semi-)automatic update mechanisms, modern browser variants, such as Chrome 38, Firefox 33, and Internet Explorer 11 make up the vast majority of the market [62], such that older, deprecated browser versions that do not allow object freezing are less relevant today. Additionally, we performed several tests to verify the degree to which also older browsers support the standard. Initially, some of the early user agents we tested, such as Safari 5 7533.16 allow to overwrite a frozen object property. These artifacts can be considered to be software bugs: we tested later versions of the Webkit engine noticing the problem does not exist anymore. Consequentially, our approach allows to retrofit protective mechanisms also to rather old browsers.

Our tool will not attempt to modify the user agent protected `location` object [175]. The majority of modern browsers forbid getter access to this object, as well as, its child nodes for the sake of user privacy and to avoid security problems. If JavaScript is executed via direct location object access—for example, via the vector

`location=name`—it will be executed within the scope we control, so we do not need to apply additional protection mechanisms. This is the same for location methods like `replace()`, `apply()` or the property `document.URL` [169].

To make sure that IceShield will notice even more exotic code execution attempts, it turned out to be not sufficient to just intercept calls to native methods relating to `window` and `window.document`, but also monitor read and write access for several DOM properties. The same is necessary for dynamic creation and manipulation of HTML elements and tags. Consequently, we overwrite the setter and getter methods of several HTML element prototypes, as we detailed in the original paper [106].

4.3.3 Scoring Metric

We use techniques from the area of machine learning to decide whether or not a given site is malicious. Specifically, we use the features discussed in Section 4.3.1 as input for a decision function F. We treat these heuristics observed by IceShield when visiting the site as vector x of the form $(f_1, f_2, \ldots, f_n)$ and define a linear decision function $F(x)$ using a weight vector w and a bias term b as

$$F(x) = \begin{cases} w^T x - b > 0 & \text{if } x \text{ is a malicious site} \\ w^T x - b \leq 0 & \text{if } x \text{ is a benign site} \end{cases}$$

The decision surface underlying F is the hyperplane $w^T x + b = 0$, which also induces a way to distinguish between instances of benign and malicious sites based on the behavior observed by IceShield. In our proof-of-concept implementation we use Linear Discriminant Analysis (LDA [101]) to find a linear combination of weights that separate the two classes, but other machine learning algorithms could be used as well. To find the optimal weights w and bias term b, we use a corpus of labeled benign and malicious sites as our training set (see Section 4.4).

The decision function $F(x)$ induces a scoring metric $f(x)$ that we can use to actually detect malicious sites. The scoring metric is defined as $f(x) = w^T x$ and $f(x) > b$ indicates an instance of a malicious site, while $f(x) \leq b$ denotes a benign site. We can also use the scoring metric as some kind of *ranking*: higher values of $f(x)$ indicate a site that tries to exploit multiple vulnerabilities of a visiting browser. As noted above, other scoring metrics can be integrated into IceShield, we just chose LDA due to its simplicity and to demonstrate how an actual metric and data mining algorithm can be incorporated into the tool.

4.3.4 User Protection

IceShield is also capable of changing the parameters passed to native methods in case the heuristic analysis indicates a malicious attempt. The easiest way to do so is to just overwrite the suspicious argument with an empty string or add randomly dimensioned padding to maliciously looking strings before passing them to the actual method. To avoid interference with the user experience, we null the payload of the possible exploit, which mitigates the danger to the user, but in most cases has no visible impact. The IceShield prototype currently defuses a possible exploit payload in case the heuristics indicate any form of overflow or heap spray. This means that strings longer than 4096 bytes containing suspicious characters, as well as, suspicious MIME types and CLSID strings assigned to new and existing DOM elements, are being modified.

Unlike approaches either completely allowing or disallowing JavaScript execution such as NoScript or the Internet Explorer XSS Filter, IceShield has minimal impact on the user experience since only the critical function call is being defused, whereas the rest of the (possibly benign) JavaScript codeflow is not affected at all. This also minimizes the negative effects of false positives our tool might have in practice.

4.3.5 Implementation as Browser Extension

The purely JavaScript based approach that we introduced so far has a few limitations which we discuss next. We found several ways to circumvent and attack our own tool while testing our approach, but we also came up with new techniques to be able to harden it against those detection bypasses. In the following, we first discuss several limitations, before we present a robust design of the general approach as a browser extension. Note that this reduces the portability since IceShield needs to be customized for each browser, but the tool is better hardened against tampering attempts targeting our instrumentation. While the extension is browser-specific, each extension is still portable across operating systems and hardware platforms. Furthermore, the core technology of our approach remains the same for each browser.

4.3.5.1 Iframes

One of the biggest challenges for our JavaScript approach and comparable tools are `<iframe>` tags pointing to JavaScript URIs [222] or resources using the `data` protocol handler (so called *data URIs* as defined in RFC 1998 [152]). An iframe containing a `src` attribute pointing to such an URL executes the JavaScript or similar code contained in the URL as soon as the user agent's parser has reached this position in the DOM tree. The JavaScript is not being executed in the window context

we can control with our tool, but in an implicitly created fresh context. This, of course, renders our approach useless since there is no way we are able to monitor the execution in the previously described manner. Listing 4.3 illustrates this problem, and we verified this behavior in all major browsers.

Listing 4.3: Iframe and object tag setup to bypass analysis

```
<iframe src="javascript:evil()"></iframe>
<object data="data:x,%3cscript>evil()%3c/script>"></object>
```

The same effect can be observed for `<object>` tags since most user agents have them behave similarly to `<iframe>` tags depending on what source they point to. The example in Listing 4.3 also shows how an object tag using a data attribute acts equivalently to an `<iframe>` with a `src` attribute.

4.3.5.2 Links

Similar to the previously described iframe problem, an `<a>` tag applied with a target attribute either set as `_blank`, `_top`, or just a bogus value and a JavaScript or data URI as `href` attribute value will have the given code be executed in a new window context. This again bypasses the detection mechanism and renders an implementation in pure JavaScript bypassable. The target attribute is usually used to specify if a link should open in the same or rather a new window. The target attribute can also be used to open a link in a specifically named window context.

This feature is necessary for websites making heavy use of frame sets, frames, and pop-up windows. In case the user agent receives a target attribute value that does not exist in the currently existing scope, the link will open in the same window, but a new window context.

4.3.5.3 META Redirects

Many user agents provide the possibility to emulate HTTP header information inline by using `<meta>` tags combined with the `http-equiv` and the `content` attributes. An attacker can abuse this feature by forcing the user agent to perform a redirect after a given amount of time ranging from 0 to n seconds as shown in Listing 4.4.

Listing 4.4: META refresh example bypassing analysis

```
<meta http-equiv="refresh" content="0;url=javascript:x()" />
```

Again, JavaScript and data URIs are being used to execute script code. It strongly depends on the user agent in how far this kind of attack is capable of bypassing our approach. Browsers based on the Gecko layout engine [172] do not allow META

redirects to JavaScript URIs anymore, but they still support data URIs to be used instead. All other tested browsers such as Chrome, Opera and Internet Explorer still support JavaScript URIs in this use case. While some of them execute the JavaScript code in the scope our tool controls, all browsers supporting data URIs can use those as a working bypass.

4.3.5.4 DOM Element Surveillance

The solution to the problems discussed above can be found in scanning and analyzing the website's markup during parsing of the DOM tree. This can be accomplished by using two user agent features: the DOM event `DOMContentLoaded` and the possibility to select all existing DOM elements with the query `document.getElementsByTagName('*')` [173]. Before the document is actually loaded and rendered, the script can loop over the existing DOM elements and check assorted tag attribute combinations such as `<iframe>` and `src` or `<a>` and `href` or the mentioned `<meta>` and `content`. Listing 4.5 illustrates how this pre-evaluation of JavaScript code can be implemented.

Listing 4.5: Example for markup analysis before execution

```
document.addEventListener("DOMContentLoaded", function(){
    var elements = document.getElementsByTagName('*');
    for(var i in elements) {analyze(elements[i].src);}
}, false);
```

In case the protocol handlers `javascript:` or `data:` appear at the very beginning of the strings to check, a pre-evaluation can take place: the code can be executed in an environment again controlled by our tool. Most user agents allow line-breaks, tabs and several more control characters merged into the protocol handler so a pre-filtering is mandatory.

To avoid interferences with the website's functionality and user experience, this can be done in a cloned version of the existing DOM. After evaluation and analysis, the results can be channeled back to the tool's logging components and be merged with the already existing scoring. Tests have shown that this approach works very well in practice already with most passive attack vectors requiring user interaction. Active JavaScript execution via `<iframe>` and `src` combinations can be intercepted too, but most user agents besides Chrome add unnecessary limitations. Note that such an approach is not affected by heavy obfuscation either since the relevant data is being taken and analyzed directly from the already existing DOM tree and not the raw markup itself. The script accesses the code that has already been de-obfuscated and normalized by the user agent itself.

Nava demonstrated with *Active Content Signatures* (ACS) [180] how a `<plaintext>` tag can be used to render all markup following after an arbitrary branch in the DOM tree can be rendered inactive for thorough inspection, modification, and sanitization before being inserted in the DOM tree again. This approach can be used to effectively deal with the mentioned problems around `<iframe>`, `<object>` and similar tags. This way, no race conditions can appear since the plaintext tag is turning every element into a single passive text-only DOM element providing unlimited amount of time for analysis and removal of malicious code.

4.3.5.5 Browser Extensions

Phung et al. [193] showed how similar approaches can be used to protect specific websites and applications against JavaScript based attacks such as XSS, CSRF and other attacks targeting the users of the attacked website or application [126]. Their approach encapsulates the native JavaScript methods and properties with an Aspect Oriented Programming (AOP) related approach based on a specific policy tailored to the website's features and specifics [75]. We suggest to move further and create browser-specific extensions such as a Firefox plug-in or an Internet Explorer Browser Helper Object (BHO) to provide more generic protection as well as gain better hardening against tampering attempts against our solution by attacker-provided code.

Extensions for Google Chrome are easy to create, but do not provide the amount of flexibility necessary for our tool to work. This is due to the technique of using *isolated worlds*, meaning a read-only mirroring for important and security critical DOM properties [25]. Our approach requires the ability to overwrite DOM elements of the website to protect users against attacks. An extension for Gecko based browsers fulfills all requirements necessary to make our approach work from within the browser as well as BHOs for the Internet Explorer. Besides the described JavaScript based version of IceShield, we have also implemented a Greasemonkey user script and a browser extension for Firefox that performs basically the same task.

4.3.6 Fingerprinting

IceShield is designed to be hard to detect by an attacker. We consider this to be important since many drive-by download attacks we observed fingerprinted the visiting user agent and deployed their payload conditionally. The same behavior is shown by several current exploit kits [178]. As a first step to be stealth, our tool consists exclusively of JavaScript code and does not make use of any external resources such as style sheets or images. Thus, an attacker has no possibility to read style sheet information via `window.getComputedStyles()` or utilize image tags and

error handlers to find out about the existence of our tool. IceShield also does not pollute the global scope such as the OWASP ESAPI tool [187] or other comparable libraries. Instead, we use an architecture wrapped in an anonymous function. Any declared variable will reside inside this function scope, and thus does not leak into the global scope.

Since the tool is making heavy use of overwritten native methods, an attacker could easily find out about its existence via several child properties of those methods if no further precautions are met. Let `window.alert` be overwritten by a custom function. An attacker can call the `toString()` or `valueOf()` method of `window.alert` which will result in leaking the source code of the overwriting function, instead of the string `function alert() { [native code] }`.

The solution to avoid leakage via `toString` and its child nodes, is to overwrite the `window.alert.toString.toString` with its parent method `window.alert.toString`, as shown in Listing 4.6.

Listing 4.6: Approach for effective toString mimicking

```
alert.toString = function(){
  return 'function alert() { [native code] }'
}
alert.toString.toString = function(){
  return 'function toString() { [native code] }'
}
alert.toString.toString.toString = alert.toString.toString;
```

This effectively overwrites the whole `toString()` chain and implicitly affects `valueOf()`, too. Thus an attacker will not be able to detect the presence of our tool by using these two methods or a combination thereof. This approach works well in all tested browsers. Note that an adversary capable of executing arbitrary JavaScript in the attacked DOM might always find ways to detect the presence of IceShield. Thus the tamper resistance established via the ES5 object capabilities is of immane importance for our approach.

A major aspect of fingerprinting are *timing attacks*, which are in general a very hard problem to deal with. This aspect can be considered as a limitation of IceShield that we have so far not managed to get around: an attacker can make use of the fact that functional string concatenation and operator based string concatenation will have a completely different code flow as soon as the `String.concat()` method has been overwritten. An attacker can thus perform two concatenation operations: if the timing value for the first one (*i. e.*, done functionally with `concat()`) differs significantly from the second one (*e. g.*, performed with the + operator), then a method modification must have taken place. This could cause the attacker to not

deploy the payload to avoid detection, and thus waste precious attack code, possibly containing exploits against unreported vulnerabilities.

4.4 Evaluation

In this section, we describe the settings and datasets we used to evaluate the prototype version of IceShield. We also present an overview of the detection and performance results obtained during several experiments.

4.4.1 Evaluation Environment

We compiled two datasets for the evaluation of IceShield: Our *known-good dataset* consists of the top 61,554 websites chosen from the top list of the Alexa traffic ranking [11]. To minimize the possibility that malicious sites exist in this set, we checked all URLs against the `malwaredomainlist.com` (MDL) blocklist [142], which lists currently active malicious sites. The *known-bad dataset* is composed of 81 sites selected from MDL [142], each identified by a unique URL. While the number of URLs may seem to be small, all URLs in this dataset point to *exploit kits* like for example *Phoenix*, *Neosploit*, or *Eleonore*. An exploit kit is a framework to serve a variety of pre-built exploits to the unsuspecting user to initiate a drive-by attack [199]. We chose to focus on exploit kits as each instance of an exploit kit represents a whole class of exploits, and Curtsinger et al. showed that such a set is representative for current attacks [67]. Given this result, we can use a smaller known-bad set to test for a much larger number of actual malicious sites. Since the quality of the results we can achieve with a supervised learning algorithm depend on the training set, we manually verified all URLs in our malicious training set to confirm that they are indeed pointing to malicious websites that perform some kind of attack.

To demonstrate the versatility of our approach, we evaluated IceShield on three different devices:

- An average workstation equipped with an Intel Core i7-870 processor and 8 GB RAM, running Ubuntu 10.04 Linux and Firefox 3.6.8
- As an example of a typical mid-range system, we used a netbook ASUS EeePC 1000H with an Intel Atom N270 and 1 GB RAM, running Ubuntu 10 Linux distribution and Firefox 3.6.12.
- To evaluate the performance of our tool on a low-end device, we performed tests on a Nokia n900 smartphone with a 600 MHz ARM7 Cortex-A8 processor and

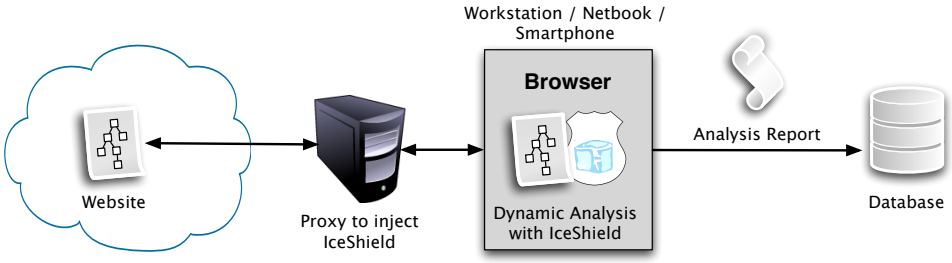


Figure 4.1: Evaluation setup for IceShield: We inject the instrumentation code via a proxy and send the result to a database.

256 MB RAM, running a Maemo Linux distribution and Firefox 3.5 Maemo Browser 1.5.6 RX-51

We performed tests on all three devices and did not have to adjust IceShield for any of them: as long as the browser on the device supports the features we require, the underlying platform is not relevant.

The evaluation environment is completed by a proxy server to inject IceShield into the HTML context of the visited pages, and a logging infrastructure, as depicted in Figure 4.1. Once a website has been successfully loaded in the browser, we log the following data points: the URL visited, execution time of IceShield and *onload* time of the respective page as well as the features observed in this website as discussed in the previous section. Furthermore, we log whether the URL belongs to the malicious or the benign set.

4.4.2 Classification Results

For the instantiation of our classification algorithm, we used the top 50 sites from the Alexa traffic ranking as benign and 30 samples. Consequentially, the test set consists of the 61,504 sites ranked below the top 50 sites used in the training set, and the remaining 51 instances of exploit kits found in the known-bad dataset.

The resulting model allows for the detection of 50 of the 51 malicious sites in the known-bad set. At the same time we achieved a false positive rate of 2.17%. A manual investigation of the one missed malicious sample yielded that this exploit relied on a Java file (instead of JavaScript) to be executed by a browser plugin to set a DOM variable, which was required for the execution of the exploit. As we did not execute Java in our evaluation environment, the de-obfuscation routine was unable to read the necessary value and the execution stopped. Hence, we were unable to observe any relevant feature, except that the site accessed `document.cookie` two

times. When retesting this site with a browser that had Java enabled, we could indeed observe a relevant feature set, which allowed for a successful detection of this exploit, too.

At first glance, a false positive rate of 2.17% may look prohibitivly high. However, as IceShield does not rely on completely blocking access to the respective site, but only disarms questionable elements in the DOM tree, the user will in most cases not notice a false positive, as the benign elements of the site will continue to work as expected.

We drew a 10% random sample from the result set of the false positives (resulting in 134 sites) to evaluate the usability of sites that have parts of the DOM removed. For 82.9% of the sample, the removal of DOM elements was not noticeable by human testers, while 9.2% of the sites remained partially usable, while, *e. g.*, banner ads were not displayed correctly. A mere 7.5% of the false positives suffered severe usability issues as a consequence of an incomplete DOM tree. In consequence, we can determine an *effective false positive rate*, *i. e.*, the percentage of cases, where the tool's presence is noticed by the user in a negative fashion, at only about 0.37%.

4.4.3 Detecting Unknown Exploits

We also evaluated, whether IceShield is capable to detect previously unseen attack vectors. To this end we cataloged websites that served individual exploits against browsers and plugins, such as an Internet Explorer exploit (CVE 2010-3962) or the so-called `_Marshaled_pUnk` exploit, which abused a memory corruption flaw in Apple Quicktime's QTPlugin.ocx ActiveX control (CVE 2010-1818). We explicitly verified that both samples were *not* contained in the known-bad set. When exposing IceShield to these completely unknown samples, both attack vectors were clearly labeled as a malicous by our heuristics and model, and consequentially mitigated. This example underlines that our approach offers significant flexibility and can detect and mitigate novel, as well as, widespread older threads.

Similarly positive results were obtained when testing against an exploit delivered via MHTML (CVE-2011-0096). Payloads deployed by this method are known to bypass most available filter mechanisms, as the subset of characters necessary to execute JavaScript is very small and does completely omit quotes and parentheses. However, while the payload was delivered Base64-encoded, the exploit necessarily needed to fall back to a set of native functions that IceShield monitored during decoding and execution. The encouraging suggestion we can draw from these results it that IceShield is indeed capable to detect novel attacks that were unknown at the time of training the system.

4.4.4 Performance Results

Under the aspects of usability on the one hand and stealthiness on the other, it is important to keep the execution time of IceShield low. As *execution time*, we log the time difference between the execution of the first line of code and the time immediately after we have overwritten and wrapped all required methods and objects. This is accurate since the first line that is executed is `var timestamp = Date.now();`, as IceShield is injected such that it is executed first in the browser. We measure the *onload time* as the difference between the execution of the first line of code and the moment when the process of rewriting the document is finished, *i. e.*, the DOM is ready. We define the overhead as the percentage of the onload time that is needed to execute IceShield.

We recorded all times on the high-end workstation. Analyzing the Alexa data set, we found that the execution time ranges from 2 ms to 760 ms. While the maximum execution time seems high, it stays well below 25 ms in the large majority of the cases, as can be seen in Figure 4.2: the average execution time measured over all samples is 11.6 ms, which corresponds to an average overhead of 6.27%. The 99.5th percentile is 25 ms. In summary, these results indicate that the execution time and overhead is very low for the vast majority of websites and hardly noticeable by the user in practice given the typical time it requires to load a web page.

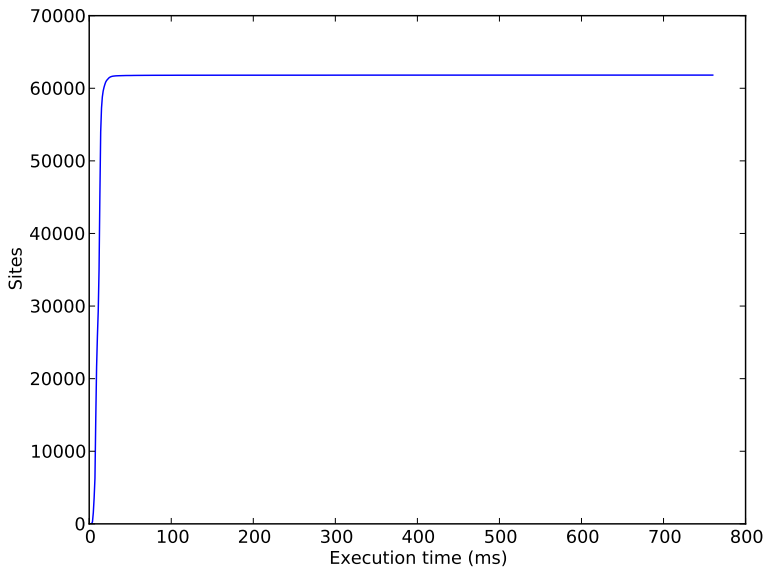


Figure 4.2: Cumulative distribution of the execution time of IceShield

We also evaluated the performance of IceShield against several common JavaScript benchmarks such as SunSpider [227], Google's V8 Benchmark [93], and the SlickSpeed [165] benchmark. Only the V8 benchmark showed a significant performance loss due to its excessive use of native functions: the benchmark result on the tested workstation changed from 376 points without using IceShield to 222 points with having the tool observing the DOM. However, we believe that this is not very relevant in practice, since the V8 benchmark focuses on rendering and number crunching tasks, rather than representing real-life web application test scenarios. SlickTest did not show any noticeable performance changes while the confidence interval displayed in the SunSpider results insignificantly changed from 2.7% to 4.4% when having IceShield active and running.

Fast execution and a low overhead is even more relevant on devices that rely on battery power. Thus, we conducted performance tests on a netbook and a smartphone (and again on a high-end workstation for comparison). As test cases, we selected seven interactive, high-profile websites. We accessed each URL ten times with each device and present in Table 4.1 the average over all runs. Even wth limited hardware, IceShield manages to perform reasonably fast. The execution time exceeds 100ms only on `twitter.com` and stays below in all other test cases. On average, our tool executed in 8.7 ms on a high-end workstation, in 50.4 ms on a netbook, and in 89.3 ms on a smartphone.

Table 4.1: Execution times on different platforms

Site (#DOM nodes)	High-End PC	Netbook	Smartphone
Google.com (113)	8.2 ms	48.9 ms	80.9 ms
Google Maps (436)	8.0 ms	50.1 ms	93.4 ms
Twitter.com (1032)	8.1 ms	49.4 ms	102.4 ms
Facebook (195)	11.6 ms	56.3 ms	92.6 ms
Yahoo! (818)	8.4 ms	48.5 ms	92.4 ms
Youtube (745)	7.9 ms	50.7 ms	79.8 ms
Baidu (52)	8.4 ms	48.7 ms	83.6 ms
Average	8.7 ms	50.4 ms	89.3 ms

Note that in recent months we have observed a huge improvement in the performance of JavaScript engines in the different browsers. If this trend continues, we can expect that the performance of IceShield even increases in the future.

4.5 Discussion

There are several limitations IceShield is faced with in its current proof-of-concept state. In case an attacker deploys a malicious PDF, Java Applet, or Flash file without using any native DOM methods to create the necessary tags and attributes, the heuristics used by IceShield might not collect enough information to deliver an adequate score. A malicious website containing nothing more than `<embed src="evil .pdf"/>` and avoiding utilization of native DOM methods will still be able to deploy and execute its payload.

Another limitation of the prototype is the lack of heuristic coverage on ActiveX based attacks. This is merely due to the fact that legacy versions of Internet Explorer are not capable of executing the IceShield code. These problems do not apply for the Internet Explorer 9 Beta we tested on. Note that this limitation is merely a matter of implementation and not a substantial problem of scope such as the aforementioned issue. Another limitation of IceShield, deployed in the JavaScript version by a website, is given by the *Same Origin Policy* (SOP). In an attack scenario, where an exploit will be deployed *after* redirecting the victim to another domain, a new window context will be loaded and the protective mechanisms of our approach cannot work anymore: IceShield cannot "stick" to the users window context since the domain borders have been crossed. To mitigate this limitation, we can run the tool on a higher level of execution privileges than the usual website context, for example, with a Firefox extension or a user script running on Greasemonkey. The Firefox extension we created successfully addresses this limitation. The Greasemonkey user script we created is also not affected by this.

The lack of tamper resistance support for older user agents such as Firefox 3, Internet Explorer 8 and Opera 10 is another limitation. These older browsers do not support features such as `Object.defineProperty()`, and need workarounds like `obj.__noSuchMethod__`. The features necessary for making our approach work safe and successfully have been implemented in the new versions of these user agents, which support the latest ECMA Script specification as discussed in Section 4.3.

The heuristics we used to detect attacks as introduced in Section 4.3.1 already cover a diverse set of possible attacks, as also illustrated by the fact that we detected three attacks with IceShield that the tool had not seen before. The heuristics are not complete in a sense of them covering each possible attack vector. Depending on the actual exploit, our heuristics might be bypassed and allow sophisticated attackers to deploy their payload. However, IceShield can be easily extended to include more heuristics that then cover more attack vectors.

4.6 Summary

In this chapter, we presented IceShield, a tool to perform light-weight dynamic analysis of JavaScript code without needing to leave the context of a browser in order to detect and prevent attacks. We extended the available repertoire of mechanisms to detect and mitigate a compromise, contributing to *precondition C* posited in Section 1.1. IceShield achieves its goals by leveraging inline code analysis and hooking to wrap native JavaScript methods into a context that enables us to dynamically analyze the behavior of these methods. We use methodologies from the area of machine learning to derive a model of malicious behavior and are able to efficiently apply this model during runtime. We took intricate care to showcase a robust implementation that cannot easily be overwritten by an attacker attempting to interfere with our analysis code. We achieve this goal by applying a novel technique using features available in the ECMA Script 5 standard and later, which allow us to *freeze* object properties. Our empirical evaluation of IceShield against malicious and benign websites shows that the tool achieves a accuracy of at least 98% in detecting attacks against the browser and was able to mitigate three previously unknown attack vectors. Its effective false positive rate is low: a user experiences a reduced functionality of websites only in roughly 0.37% of the sample set. Test on resource-limited devices have shown that the performance overhead is low, making IceShield suitable for deployment even on small devices, like smartphones and netbooks.

5

Locational Privacy in the Absence of Anonymous Payments

> Let all Men know thee, but no man know thee
> thoroughly: Men freely ford that see the shallows.
>
> —Benjamin Franklin, *Poor Richard's Almanack*

The idea of *privacy by design* has been described as a set of principles that aim to root privacy-enhancing technology deeply within technical solutions and organizational structures [53]: privacy should not be an afterthought, but be deeply embedded into technology, without hampering its functionality. While the latter may sound straightforward in theory, it may pose significant hurdles in practice. Privacy thus is taking a similar route *security* already took: security properites, too, have long been handled as one of many features of a product or a technology. Slowly, the realization begins to set in that it must be considered as a foundation instead—a fact that security researchers have been preaching for years.

For privacy to become a foundation of design, it needs not only to overcome the notion that it (seemingly) hampers functionality, but also that in a given technological context it contradicts security efforts. However, often enough security and privacy are just two sides of the same coin, *e. g.*, customer data that is not retained in the first place cannot be compromised by an attacker later on. Thus, for a more holistic approach, we rephrase the concept of *privacy by design* to *security and privacy by design*, a principle that the European Commission, for instance, sees as a foundation for RFID technology [82].

We apply this concept—which has also found application in recent research on RFID-based systems [17] and cloud security [47]—to the field of post-paid electronic acquisition of goods or services. To this end, we propose a system that respects both the vendors' need for authenticity of billing-relevant data and the customers' requirement for effective protection of personal and personally identifiable data.

5.1 Introduction

Blumberg and Eckersley define locational privacy as "the ability of an individual to move in public space with the expectation that under normal circumstances their location will not be systematically and secretly recorded for later use" [39]. In a world of Big Data, where any fact about an individuals life, once revealed, will potentially be stored indefinitely, it is important to limit the data that is created or revealed in the first place. While completely anonymous systems would be desirable in many cases from a customer's side, legitimate business interests on the side of the vendor may prevent the adoption of a technical solution that relies on complete anonymity. In the context of financial transactions, the prevalent academic approach to protecting a user's locational privacy is to protect the user's *identity* and thus indirectly conceal their location. Various anonymous electronic cash (e-cash) schemes have been published (*e. g.*, [45, 51, 57, 58]) since Chaum published his seminal paper *Blind Signatures for Untraceable Payments* [56] in 1982. However, none has been (widely) adopted. Besides posing technical hurdles, e-cash often makes it hard for the vendor to walk the established path of resolving a dispute with a customer on front of a court of law, as the customer is not known—although many schemes reveal the customer's identity in case of *double spending*, but only then. Anonymous payment schemes also forfeit the option of post-paid good and services, where the customer needs to be billed and thus is typically known. Finally, there may be applications where regulations and legal restrictions prohibit the customer from being anonymous. A vendor in this market will be unable to provide anonymous payment services to its customers.

Under the premise that the customer must be identifiable, we thus must conceptionally deviate from the widespread paradigm of anonymizing customers in privacy-enhancing systems. Our new approach to this problem is to *anonymize locations*. In our solution, the user's identity is explicitly known during a transaction, yet the user's *location* is concealed, effectively hindering the creation of a movement profile based on financial transactions. We use the increasingly relevant example of recharging electric vehicles and paying for energy on the go to showcase our approach. We do not exclude the possibility that our approach can be adapted to other settings that require the customer to be known during such a transaction.

5.1.1 The Example of Electric Vehicles

In a world where oil-dependent mobility comes increasingly under scrutiny, electric vehicles (EVs) are one possible alternative (again). While around the year 1900 about 38% of the cars in the US were powered electrically, in the following decades the EV had been marginalized by cars with internal combustion engines. This fact has often been attributed to the introduction of the electric starter motor that made combustion engine-driven vehicles as easy to use as electric ones [92]. Today, electric vehicles are once again a promising concept for solving some of the environmental and transport challenges we are facing as a civilization.

In the context of privacy-preserving non-anonymous transactions, the EV scenario offers several interesting constraints. First of all, the proliferation of vehicles and infrastructure is limited, but rapidly increasing. Today, many major car manufacturers offer a series of EVs or plan to do so in the next one or two years. Market research predicts up to 3.4 million annual world-wide sales of plug-in hybrid (PHEV) and battery electric vehicles (BEV) in 2020 [195]. While we are aware that most people leave a cornucopia of movement traces due to their use of existing technology (*e. g.*, cell phones), we think that technical solutions for emerging fields, like electric mobility, should be designed with privacy in mind.

Second, at least for the time being, the capacity of most electric vehicle batteries is rather limited, thus most EVs require relatively frequent charging, both to prevent deep battery discharge and to counter the users' *range anxiety*, *i. e.*, the fear not to reach a destination on the remaining charge. Many drivers charge whenever they can. Both in North America and Europe there are currently more than ten thousand charging stations (CS) accessible to the public [139, 196] and the numbers are on the rise: the European Commission proposed a minimum target of 795,000 publicly accessible charging stations throughout the EU [52].

The increasing availability of public charging stations is positive and necessary for the success of EVs. However, in combination with the need to charge frequently, it renders vehicle movement profiles more detailed than those derived from fossile fuel not paid with cash. A vehicle's movement profile allows to infer habitual behavior of its owner, both correctly and incorrectly so. "Did a person see an AIDS counselor? Did an employee skip lunch to pitch a new invention to venture capital investors? Did she meet with safety regulators to tip them off about work conditions?" [39] If a person regularly charges her EV in front of a rehabilitation clinic, an entity interpreting available location data might (falsely) suspect a history of drug abuse. Persons who park their car in a red light district on a regular basis may simply want to keep this information to themselves [188].

Third, cash is not an option for almost all utilities. While even in the US cash accounts for a significant minority of 12.6% of domestic retail payments [10] (Canada: 54% of retail payment volume in 2012 [15]), cash accounted for 78% of the retail payments in Europe in 2008 and would solve the privacy problem at hand. However, in most parts of the developed world, utilities deliver energy either based on a subscription (post-paid) or pre-paid model where the customer's name is known. The common element in both scenarios is that an identity is linked to each payment process. For the foreseeable future, charging an electric vehicle will take much longer than refueling a traditional car, which is yet another reason to charge where the vehicle is anyway. As a result, EV charging infrastructure will be much more distributed, in contrast to the current network of fuel stations. This makes cash logistics prohibitively expensive: even cost benefits of existing billing infrastructure aside, maintaining a low cash level in all stations does not scale well in a widely distributed infrastructure, while retaining a larger amount of cash in charging stations solicits theft or at least vandalism.

Fourth, the sales of electric energy is tightly regulated in many countries. Many of these regulations aim at making the market more transparent to the customer. However, when applied to the relatively new EV scenario some of these requirements can compromise the locational privacy of the customer. These regulations stem from the time where electric energy was explicitly sold to households or companies, *i. e.*, non-mobile entities, and result in requirements for points of sale and storage of transaction documentation that can compromise the locational privacy of the customer.

5.1.2 Contribution

In this chapter, we propose a system to authenticate non-anonymous transactions, while preserving the users' locational privacy. We use the example of electric vehicle charging, as it offers several interesting constraints. More precisely, we make the following contributions:

- We adapt a carefully chosen group signature scheme without compromising its strong security properties to allow for full compliance with regulations and legal requirements. These requirements were identified with the help of experts in the field of commercial law and energy law. The privacy mechanisms protecting the user's location data are very strong: not only is it impossible to decide whether a user has charged her vehicle at a specific CS, it is even impossible to decide whether a user has *ever* been charging at one or several CSs more than once. In compliance with local laws, the system still allows a trusted

third party to revoke the location anonymity of past billing processes in cases of a dispute.

- Our solution is complete, in that it covers the charging process from after authentication to providing all information necessary for the clearing process. It closely fits existing clearing and billing structures and can be implemented efficiently on a large scale.
- To the best of our knowledge, we are the first to also offer an implementation of a practical charging and billing system for electric vehicles that provides strong protection of the customer's locational privacy. Our implementation performs well even on the limited hardware of a CS, while we are able to process more than one million charging processes per hour using off-the-shelf hardware at the backend (BE), thus providing a cost-effective way to process billing information from a large network of CSes.

5.1.3 Outline

The remainder of this chapter is structured as follows: in the next section we provide technical background and outline our approach, as well as, the design decisions that followed. We then provide a detailed description of all processes within the resulting system, present the evaluation of our prototypical implementation and conclude with a discussion of our approach.

5.2 Technical Background

We begin this section with a definition of the problem space, before outlining our scheme, which consists of three main phases: (1) authenticating the customer, (2) authenticating the tuple of customer identity and energy consumption data, and (3) transmitting this data to a clearing house, all without compromising the customer's locational privacy. We end this section with a discussion of related work.

5.2.1 Problem Space

We define the problem space as follows: Electric utility companies that are honest but curious and want to learn about past, present, and future locations of vehicles, or any entity obtaining (billing) records from utilities, can infer a movement profile for every customer, based on these records.

Under the assumption that

a) the creation of movement profiles without explicit consent of the subject is undesired and the existence of unnecessary data is to be avoided,

b) anonymous payments are not an option,

c) the solution should integrate well with existing billing infrastructure and processes

we explore how the creation of movement profiles can be prevented, while integrity, authenticity, and, in parts, the confidentiality of the data transmitted between a point of sale (*i. e.*, a CS) and a backend is provided. Conceptually, we thus must deviate from the widespread paradigm of anonymizing customers in privacy-enhancing systems. Our new approach to this problem is to *anonymize locations*, *i. e.*, charging stations. In this context we identify three core issues:

One way to cryptographically bind a customer identity to metering data are digital signature algorithms, as they achieve non-repudiation. However, the location where a charging process took place can be directly inferred, as classical digital signatures not only guarantee the authenticity of the signed data, but also authenticate entities, *i. e.*, the respective charging station (*Issue 1*). Location-bound tokens, like the identifier of the energy meter used for the measurement, naturally compromise the customer's locations, but utilities are legally required to retain this information in many European countries (*Issue 2*). The location of a transaction can also be inferred from network-based identifiers (*Issue 3*), primarily the charging station's IP address, *e. g.*, by correlating BE server access logs with billing data timestamps.

Furthermore, an entity may have access to the network that connects the backend or a CS to the Internet. Such an attacker might try to infer the origin of a message by using a timing side channel, but must be unable to attribute the connection to a specific user. We assume that all attackers are computationally bound and accordingly unable to break computationally hard cryptographic primitives.

Attacks against the point of sales itself are out of scope of this chapter.

5.2.2 Approach

We address *Issue 1* by employing a group signature scheme with strong security properties that provides very efficient verification procedures for large numbers of signatures as a central building block of our system. The scheme allows for the *conditional* identification of a signer, while in the default case allowing him to remain anonymous. For every entity that is not in possession of the so-called *opening key*,

the actual signer of a message is indistinguishable from every other potential signer within the same group. Thus, while a customer's transaction is always linked to his customer account, our system guarantees unlinkability with respect to location and time of a transaction.

We address *Issue 2* by modifying the signature scheme such that information that is required by law or regulations, but would compromise the customers' locational privacy, is also only *conditionally* available. In normal operation this information is as strongly protected, as the signer's identity itself. In case of a legal dispute, where this information must be produced by the utility in front of a court of law, such that an independent entity can assess the proper calibration of the energy meter, the identifier of charging station and energy meter can be revealed by a trusted third party. Legally required information was identified with the help of our colleagues from the faculty of law, who specialize in commercial law and energy law.

We address *Issue 3* by anonymizing the sender of billing-relevant information on the network level. As the communication between charging station and backend is not highly time critical, we could in principle use high-latency mix networks, such as Mixminion [72] or Mixmaster [164]. However, as network availability is an issue, we chose to use the, at the time being, most popular anonymity network, which increasingly offers good redundancy due to its high number of nodes: the Tor network [78]. As Tor does not provide protection against exploiting timing side channels, especially in the presence of low traffic volume, we also discuss how these kinds of attacks can be mitigated in our application context.

In summary, the authentication and charging process we propose is as follows (cf. Figure 5.1):

1. The CS authenticates towards the customer and vice versa. The CS retains the authenticated customer identity.
2. Upon successful authentication, the CS's power outlet is unlocked and/or put on-line. Charging begins as soon as the EV is connected.
3. When the power line connection between the CS and the EV is interrupted, the CS generates a tuple containing all information required for the billing process (*i. e.*, the authenticated customer identity stored from Step 1, the amount of energy provided to the user, a timestamp indicating the beginning of the charging process and a timestamp indicating its end). Each location-bound token that is legally required is encrypted to the single entity in possession of the opening key. The tuple is signed using the group secret x_i of the respective CS and the data is transmitted to the BE via the Tor network. To ensure confidentiality of the transmitted data, we establish a TLS tunnel between CS and BE prior to transmission.

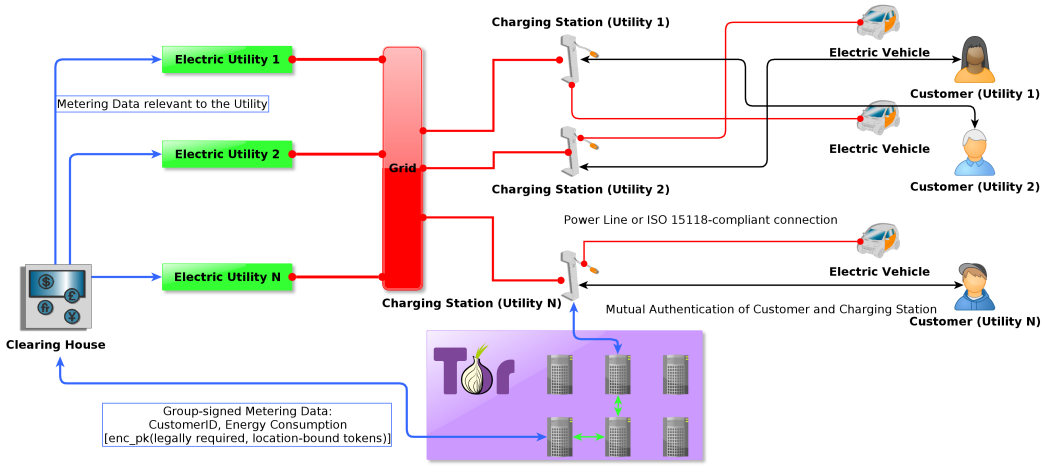


Figure 5.1: Charging and Transmission of Metering Data

We are aware that in being compliant with legal regulation, our system is also dependent of legal protection: A high legal hurdle must be placed before the identification of a signer (*i. e.*, the respective CS) and the disclosure of location-bound tokens. This could mean, for example, that a court order or the customer's consent is required.

5.2.3 Roaming

While a significant part of customers can still only charge at CSes owned by the utility they have a contract with, roaming is desired by most market participants. In Figure 5.1 the concept is represented by the introduction of a clearing house. Following the example of the banking and telecommunications sector, efforts are already under way in the energy sector to establish a clearing house to back a roaming-enabled charging infrastructure for electric vehicles. As the clearing house aggregates and verifies metering data from all the CSs, it is capable to provide either only data clearing or also financial clearing to the associated electric utility companies, which in turn allows each utility's customers to roam freely between all other utilities cooperating with the clearing house. At the time of writing, clearing services for EV charging in Europe are neither unified nor sufficiently developed—first approaches to facilitate the use of charging stations operated by several providers using only one authentication token are made, for example, by HubJect[17] and e-clearing.net.

[17] http://www.hubject.com

5.2.4 Group Signatures and XSGS

Group signature schemes are an essential part of our approach and thus we explain in the following how we utilize and adapt this concept and why we chose the *eXtremely Short Group Signature* (XSGS) scheme.

The idea of group signature schemes has first been introduced by Chaum and van Hejst in 1991 [59]. A group signature scheme is a digital signature scheme that (additionally) provides a (strong) form of sender anonymity. Unlike in classical signature schemes where each signature is produced by a single signer, in a group signature scheme each signature is produced on behalf of a group. For the verifier it is easy to check whether the signature has been produced by one of the current *group members*. However finding out who exactly produced the group signature is impossible. Intuitively, the larger the group is, the better are the anonymity guarantees provided for each group member—an ideal property for our scenario.

5.2.4.1 Anonymity: Pseudonyms vs. Group Signatures

Group signatures provide a very strong form of anonymity that is usually referred to as *unlinkability*: it is not only impossible to map a signature to its creator—this could be achieved by pseudonyms alone. Unlinkability also implies that noone, except for a dedicated trusted party called *opener*, is able to decide whether two group signatures have been produced by the same signer. We believe that for our application this property is crucial[18]. When using pseudonyms for CSs alone to protect the user's locational privacy, the verifier could easily build up customer profiles for every CS which, with more and more user-dependent billing data, could possibly be narrowed down to a single CS. In this way one could easily reveal the true CSs behind the pseudonyms. As a consequence, the verifier could easily follow where and when each user charged its vehicle. Group signatures on the other hand do not even reveal whether two signature belong to the same CS. So users who constantly charge their vehicle at the same CS are indistinguishable from those who travel a lot and often use CSs that they have never visited before.

5.2.4.2 Design Features of the XSGS Scheme

Group signatures vary in the extent of functionality they offer and in the security guarantees they provide for group members and verifiers. In our work, we utilize

[18] We recall once again that user identities have to be known to the verifier for a proper billing process. Thus it is not possible to anonymize user identities in the bills.

the XSGS scheme by Delerablee and Pointcheval [76]. The XSGS scheme is an extended variant of the well-known group signature scheme by Boneh, Boyen and Shacham (BBS) which achieves very high efficiency with respect to both signature size and speed [41]. It modifies the BBS scheme in two ways. First, it adds improved protection of group members against collusions of (corrupted) members who try to frame a user. In XSGS, even if the issuer itself is corrupted and takes part in that collusion, its honest group members cannot be framed. Second, XSGS guarantees unlinkability of signatures to even hold against an adversary that can convince the opener to open all other signatures. BBS does in general not cover such attacks (not even when the adversary may convince the opener only once). As a theoretical benefit of these extensions, the XSGS scheme can be proven secure in the very strong security model of Bellare, Shi, and Zhang [35]. We believe that these extended properties of XSGS are necessary in our application. In particular, they allow to implement the issuer at the same place as the (only) verifier (*i. e.*, the clearing house), without risking the CS's anonymity.

5.2.4.3 Support for Batch Verification

An important design restriction of our solution is that we consider a single verifier that has to verify a huge number of signatures. The group members, on the other hand, do only have to generate a moderate amount of signatures each day. Thus our group signature scheme should ideally feature very fast verification procedures. Kim et al. showed that XSGS supports batch verification [130]. In general, batch verification [31, 84] identifies the most expensive operation in the verification of a signature scheme and combines the verification procedures of several signatures into a single one such that this operation is only executed for a (small) constant number of times (which is independent of the number of signatures). This can greatly improve efficiency.

For security reasons, the combination process is setup in such a way that adversaries cannot produce a combination of invalid signatures which pass the batch verification test[19]. In the XSGS verification, the most expensive operation is the evaluation of a bilinear operation (the so-called pairing) executed on elements of certain elliptic curves. This operation is usually applied in each signature verification. The batch verifier for the verification process only requires the pairing to be called once, independent of the number of signatures.

Batch verifiers pay off when the (expected) number of invalid signatures per batch is very small. In theses cases one can easily apply a divide-and-conquer approach

[19] The batch verifier of Kim et al. uses the so-called small exponent test [31].

to identify the invalid signatures. One recursively divides the batch into two halves and applies batch verification to these halves. Every batch that does not successfully pass the batch verification is again divided up into two new batches of half the size. This process incurs at most $\log_2(n)$ (where n is the size of the original batch) subprocesses per invalid signature. However, for larger number of invalid signatures, this process quickly performs inferior to the separate verification of each signature.

5.2.4.4 Dynamic Groups

XSGS allows for dynamic groups, *i. e.*, group members can be added and removed without re-initializing the whole scheme. Also, member joins do not require updates of the group public key (GPK), which is of significant advantage, if we expect the system (at least in the first years of installment) to often add new group members, while we consider revocations of credentials to be less frequently required. We stress that if member joins do not require modifications of GPK, it is impossible to *not* modify the group public key when revoking users—we need to signal the verifier that the set of legal signers has changed. However, the approach underlying XSGS is very efficient. It is based on dynamic accumulators [50, 182]. In case of a revocation the group manager modifies the GPK and publishes a set of values that help all but the revoked users to recalculate the GPK. This information does not have to be transferred in secret. Instead the group manager can simply make it publicly available.

5.2.4.5 Group Signatures in an EV Charging Infrastructure

In an e-mobility scenario the application of group signatures to ensure the authenticity of metering data results in the following setup:

N utilities choose to cooperate by utilizing a certain clearing house. Each utility i provides m_i charging stations to the public. A clearing house can act as the group manager within the group signature scheme, that allows entities to join the group and can revoke individual signing keys. The group manager cannot, however, identify individual signers within the group.

Upon setup of a CS, the clearing house allows the CS to join the group. Thus all m_i stations form one group. Anyone (*e. g.*, clearing house, utilities, customers, a judge, etc.) is able to verify that a tuple $(\textit{metering data}, \textit{customer id})$ has been signed by a CS within the group and that the signed data has not been altered. If the group signature scheme includes an opening phase, one entity exists that is able to attribute any signature to the signer within the group. In the case at hand this could be an

independent notary or a newly purposely formed institution, *i. e.*, an entity that can be trusted to act lawfully and is trusted by all market participants. However, while the *metering data* will already contain at least the energy consumption and a timestamp for each the beginning and the end of the charging process, in some legislations this is not enough to identify the CS for dispute resolution in court. The energy meter within a CS is calibrated and often sealed, so a for metering data to stand up in court, also, *e. g.*, the identification number of the meter (meterID) is required. As the meterID can be attributed to a certain CS, it is a location-bound token that needs additional protection, lest it will compromise the customer's location privacy.

There is already one entity all participants have to trust (the Opener), as it can disclose the signer, and, consequentially, where a charging process took place. We thus use an asymmetric encryption scheme to encrypt any location-bound token with the Opener's public key. Thus the message sent from the CS to the clearing house will be formed as shown in Listing 5.1. The Opener may only decrypt the encrypted message part when the legal constraints for opening the signature are fulfilled.

Listing 5.1: Payload of message used to transmit metering data

$$\texttt{metering data, customer id, } enc_{OPK}(\texttt{location-bound token}),\ groupsign(\texttt{metering data}, \texttt{customer id}, enc_{OPK}(\texttt{location-bound token}))$$

5.2.5 Related Work

Locational privacy has been recognized as being desirable as early as 1996 [121]. Its importance has been recognized for example in the field of pervasive computing [14] and also in the context of location-based mobile applications [198]. The importance of location privacy in the context of transportation is underlined by numerous publications that aim at preserving location privacy in various applications like vehicular communication systems [79, 88, 116, 205, 237], ticketing for public transport systems [20, 38, 112], and electronic road toll collection [19, 154, 197]. In the latter context, Chen et al. [233] propose the use of a group signature scheme to enhance the users' privacy. However, the authors did not implement the proposed solution and fail to evaluate the feasibility of their approach in the given scenario.

A limited amount of publications have considered locational privacy in the context of e-mobility so far: Chao Li [54] implement a merchant entity of the Compact e-Cash scheme [49] aimed at the application within a charging station. Liu et al. [140] propose an anonymous electronic payment scheme that supports two-way anonymous payments. Stegelmann's and Kesdogan's approach [213] aims at providing locational privacy in the presence of a smart grid that actively manages EVs as

energy buffers. We are not aware of an implementation that allows to evaluate the practicality. While their design incorporates optional anonymity revocation, it relies on an anonymous electronic cash scheme for billing. None of these approaches can be used when anonymous electronic payments are not tolerated by legislation or even just undesired by the vendor.

5.3 System Design

In this section we describe all processes that constitute our system. We describe every process in the lifecycle of a charging station and also a protocol for user authentication.

5.3.1 Bootstrapping the System

Before we can start authenticating users, charging vehicles, and securely transmitting energy consumption data, we have to set up the infrastructure.

The clearing house acts as the *group manager* within the XSGS scheme. It can add a new CS to the group by issuing a certificate (credential) $UCert$ to CS. A CS with a valid $UCert$ is also referred to as *group member*. The clearing house can also revoke the ability of group members to sign on behalf of the group. An entity sufficiently independent of clearing house and utilities serves as the *opener*. In our scenario N electric utilities choose to cooperate by utilizing a certain clearing house. Each utility i provides m_i charging stations to the public.

In order to bootstrap the XSGS scheme, the group manager first needs to generate the group (curve) parameters of a bilinear group (including group descriptions, generators, and pairing specification). Technically, the bilinear group consists of two elliptic curve groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with random generators $G_1 \in \mathbb{G}_1$ and $H, G_2 \in \mathbb{G}_2$ and the description of a non-degenerated bilinear pairing $e : G_1 \times G_2 \to G_t$ such that $e(G_1^a, G_2^b) = e(G_1, G_2)^{ab}$ for every $a, b \in \mathbb{Z}_p$. For more details we refer to [41]. Next it generates a secret Diffie-Hellman key $IK \in \mathbb{Z}_p$ (called issuer key) with its corresponding public key $W = G_2^{IK}$. The issuer key IK is used to generate certificates for new group members. Given these values, the opener generates a private key of a chosen-ciphertext secure encryption system, the opening key OK. The corresponding public encryption key is denoted as OPK. The public key OPK is used in the signing process of the group signature scheme to encrypt the signer's certificate $UCert$. This enables the opener to reveal which CS has actually created a given group signature. On a technical level OK consist of two independent secret keys of an ELGamal encryption system. OPK contains the

corresponding public keys. It is well known that ElGamal is only chosen-plaintext secure. However, the system applies the well-known Naor-Yung transformation [177] which encrypts a given messsage under both ElGamal keys resulting in ciphertext Z_1 and Z_2. Additionally, it generates a NIZK proof P of equality of plaintexts in Z_1 and Z_2. The ciphertext Z consist of $Z = (Z_1, Z_2, P)$. The group public key GPK consist of the paramaters of the bilinear group, W, and OPK.

5.3.2 Setting Up New Charging Stations

Each new CS must join the group before it can sign metering data. Now that group manager and opener are set up, the group manager can add new charging stations to the group. Note that all charging stations, independent of the utility that operates them, will be members of the same group.

The group manager starts the join process by transmitting the GPK to the CS. The CS draws its private signing key $UK \in \mathbb{Z}_p$ and computes a commitment $C = H^{UK}$ of UK. Then it sends C together with a NIZK proof of knowledge of UK to the group manager. On successful verification of this proof, the group manager selects a random signing key $x \in \mathbb{Z}_p$ for the CS and calculates the group member identifier

$$A = (G_1 \cdot C)^{\frac{1}{IK+x}} \Leftrightarrow e(A, W \cdot G_2^x) = e(G_1 \cdot C, G_2). \tag{5.1}$$

The values A and x constitute the certificate $UCert$ of the CS. Intuitively, $UCert$ is a digital signature over x that can only be computed with the help of IK. The group manager first sends A to the CS and proves that it knows a corresponding x that fulfills the above equation. Knowing that its communication partner can indeed issue certificates, the CS produces a classical signature S using its USK over A as $S = Sign_{USK}(A)$ and sends $(S, cert_{CS})$ to the issuer. This pair is important when resolving disputes as it binds the anonymous certificate $UCert$ to a concrete CS that can be identified via the classical PKI. If the signature is valid, the group manager sends x to the CS and registers the entry $(UCert, C, cert_{CS}, S)$ in a database.

Now since $C = H^{UK}$ and UK is known to the CS we get that

$$A = (G_1 \cdot H^{UK})^{\frac{1}{IK+x}} \Leftrightarrow e(A, W \cdot G_2^x) = e(G_1 \cdot H^{UK}, G_2). \tag{5.2}$$

5.3.3 Decommission of Charging Stations

Occasionally it may be necessary to remove a CS from the group, be it because it is replaced by a CS of a newer generation or to deal with a compromise. We consider the revocation of a group member's credentials to be a less frequent event than the

joining of a new member. Thus, while $UCert$ and UK remain unchanged upon the joining of a new member, removing a member from the group requires that all remaining group members receive information on how to re-calculate their group identifiers A.

Assume the group manager wants to revoke a CS with $UCert' = (A', x')$. First, it publishes an updated version of the GPK. For example G_1, G_2, and H are substituted by $G_1^* = G_1^{\frac{1}{IK+x'}}$, $G_2^* = G_2^{\frac{1}{IK+x'}}$, and $H^* = H^{\frac{1}{IK+x'}}$. Next each group member with $UCert = (A, x)$ and secret key UK except for the one to be revoked has to update its group identifier A to $A^* = A^{\frac{1}{IK+x'}}$. To this end it is sufficient that the group manager simply publishes x'.

$$A^* = A^{\frac{1}{IK+x'}} = \left(G_1^* \cdot H^{*UK} \cdot A^{-1}\right)^{\frac{1}{(x-x')}}. \tag{5.3}$$

Next, each charging station computes a new signature $S^* = Sign_{USK}(A^*)$ over the new group member identifier A^* and sends it to the group manager. The group manager verifies S^* from each CS and, on success, updates the existing database entries with the new values for A^*, C^* and S^*. Note that the CSs do not have to save an incremental revocation list of all revoked members to decide on the validity of newly signed metering data. However, it might be necessary for the group manager to retain a limited set of old group credentials for the time span that the respective jurisdiction sets for the resolution of disputes concerning past charging processes.

5.3.4 User Authentication

Before the CS channels electricity into the EV, the customers need to authenticate themselves towards the CS so they can be billed. At the same time the CS should authenticate itself towards the customer to prove that it is genuine. This may thwart attacks where a rogue CS may be setup that exploits access to an EV in a malicious way.

Besides acting as a group manager in the context of the group signature scheme, the clearing house could also serve as trust anchor (RootCA) within the PKI used for user authentication. Each electric utility that cooperates in this system serves as Intermediary CA and can thus issue certificates to its customers.

We assume that each customer holds a secret signing key sk_{CUS} together with its corresponding public verification key pk_{CUS}, a certificate $cert_{CUS}$, and the public verification key of the certification authority pk_{CA}. $cert_{CUS}$ consists of a signature over pk_{CUS} and the user's identity (which has been computed using the CA's secret signing key corresponding to pk_{CA}). Similar to before, each CS holds a secret key

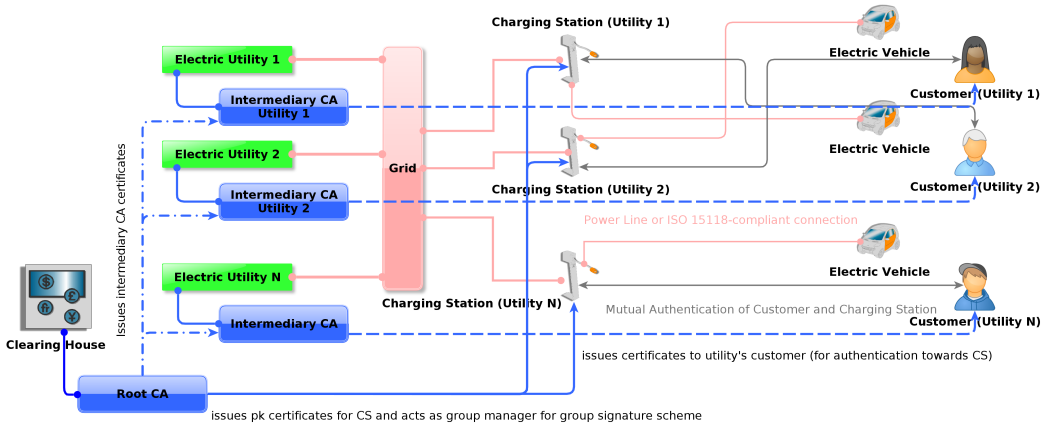


Figure 5.2: PKI for User Authentication

USK_{CS}, a public key UPK_{CS}, a certificate $cert_{CS}$ over UPK_{CS} issued by the same CA, and pk_{CA}.

To authenticate each other, the customer can, for instance, be provided with a smartcard (equipped with sk_{CUS}, pk_{CUS}, $cert_{CUS}$, and pk_{CA}) and a CS can engage in a classical challenge and response protocol as outlined in Listing 5.2. Intuitively, it exploits that only the holder of the secret signing key can produce valid signatures for arbitrary messages. The intial exchange of nonces thwarts replay attacks.

Instead of equipping customers with smartcards, the smartcard can also be bound to and incorporated in the EV to allow for ISO 15118 compliant authentication [119], where the vehicle (and not the customer) authenticates towards the CS. Communication then takes places using a specially equipped power cable instead of a contactless interface.

Listing 5.2: Challenge Response Protocol for User Authentication

```
1. CS:  r_CS ∈_R {0,1}^l
        send r_CS to CUS
2. CUS: r_CUS ∈_R {0,1}^l
        send r_CUS to CS
3. CS:  s_CS = Sign(USK, r_CS||r_CUS)
        send s_CS, cert_CS to CUS
4. CUS: verify cert_CS with respect to PK_CA
        IF successful:
              verify s_CS with respect to UPK
              IF successful:
                    compute s_CUS = Sign(sk, r_CS||r_CUS||s_CS)
                    send s_CUS, cert_CUS to CS
```

```
                ELSE: abort
          ELSE: abort
5. CS:  verfify cert_CUS with respect to PK_CA
        IF successful:
            verify s_CUS with respect to pk_CUS
            IF successful:
                start charging process
            ELSE: abort
        ELSE: abort
l: security parameter
```

5.3.5 Ensuring Authenticity of Metering Data

When the charging process is terminated (*i. e.*, the customer chooses to end the process or the cable connection between EV and CS is severed), the CS creates a message M consisting of the authenticated customer identity, the amount of energy consumed by the customer, two timestamps marking the beginning and the end of the charging process, and a string that identifies the utility owning the CS. As discussed above, legal regulations may require transmission and storage of the identifier ($meterID$) of the calibrated energy meter or of other certified components of a point of sale. These identifiers would reveal the physical location of the transaction. To avoid this, we have to adapt the group signature scheme slightly. Instead of being sent in the clear, the $meterID$ is probabilistically encrypted using the opener's encryption key OPK before being added to M. In the same way other location-critical information can be incorporated into the group signature. Only the opener can decrypt these values using its secret decryption key OSK. We stress that while the $meterID$ is always encrypted with the opener's public key and never transmitted in the clear, it is not neccessary to prove that the *correct* $meterID$ has been incorporated into the ciphertext. The opener can uniquely identify the CS and any incorrect information of a CS on its $meterID$ can thus easily be revealed. It is reasonable that each charging station holds exactly one meterID, such that the mapping of group signature keys UK and meterID is one-to-one. Thus opening a signature will reveal a unique CS identity and a unique meterID. We may even assume that the mapping between UK and meterID is publicly available without compromising the user's locational privacy. As sketched above, CS's group signature s on M consists of an encryption Z of $UCert$ and a message-dependent NIZK proof showing that CS knows a valid $UCert$ with corresponding UK which fullfil Equation 5.2 and that $UCert$ has been encrypted correctly under public key OPK in the ciphertext Z (which is part of s). Intuitively, these types of message-dependent proofs work like signatures. Generating them on new messages requires the creator to know A, x, and UK. They are often referred to as signatures of knowledge [55]. XSGS uses particularly efficient

NIZKs that can be computed by applying the well-known Fiat-Shamir heuristic [85] to interactive zero-knowledge protocols in the random oracle model [34].

On a more technical level the proof shows that its creator knows several discrete logarithms; x, UK, and the secret exponents used to encrypt A with the ElGamal encryption systems. Intuitively, it proves that if the ElGamal ciphertexts were decrypted, then the resulting plaintext would, together with x, UK fulfill Equation 5.2. For more details on the computations of the group signature, we refer to the literature [76, 130].

5.3.6 Transmission of Metering Data

To prevent the disclosure of the CS's network location, the CS first connects to the Tor [78] network and establishes a routing circuit. It then starts a TLS session with the backend (BE) and in the process verifies the certificate presented by BE. We use a ciphersuite based on Ephemeral Diffie-Hellman (DHE) with CBC-MAC, as it offers perfect forward secrecy and because of its cryptographic security properties: it has recently be shown to be provably secure in a strong security model [122]. We rely on TLS to guarantee that each transmission from a CS reaches the backend. The BE acknowledges the successful submission by sending the string `ACK` and a timestamp. We rely on TLS for the authenticity of the reply.

Although Tor provides sender anonymity, a possible timing side channel exists: if there is only sporadic network traffic within the system of CSes and BE, an attacker observing both the network at a CS and the BE could correlate these events with charging timestamps (somehow obtained) from the clearing house. As the transmission of billing-relevant data is not time-critical in the example of EV charging, we can prevent correlation as follows:

Each CS is scheduled to send a transmission of a given size once per 15 minutes. Each charging process results in one message of typically less than 1000 byte. If we fix the size of the transmission, for instance, at 5 kByte, it fits several messages (M, s). We pad each transmission with random data to the maximum size. If a message (M, s) does not fit in the current transmission anymore, it is scheduled for the next. If no charging process has been finished within the time window, we just transmit the string `empty` and pad the transmission to the defined maximum size. As all transmissions are of equal size and are encrypted as described above, an attacker observing the network is unable to distinguish between transmissions that contain billing data and those that do not. Thus, the attacker gains nothing.

5.3.7 Verification of Metering Data

When the BE at the clearing house has received (M, s) it verifies the group signature s by checking the NIZK proof with respect to the GPK and thus determines whether the consumption data that is bound to the identity of a customer is valid. For details on the computations, we refer to [76, 130]. If the signature does not verify, it discards the message as it cannot stem from a CS within the group. On success, the signed tuple M is passed on to the clearing service for processing. As there is one central verifier in the system that verifies all metering data, batch verification of group signatures offers a significant efficiency gain.

5.3.8 Dispute Resolution

In the case of a dispute, the opener can craft a non-repudiable publicly verifiable proof of the actual creator of a given group signature. The opener will act so only upon the request of a judge or with the consent of the customer. Note that even after a message M_i has been subject to the opening process, it is impossible to decide, whether a CS who signed M_i also signed a different message M_j, *i. e.*, the location of other, potentially unrelated charging events remains hidden. To open the signature s, the opener uses its secret opening key OK to decrypt the ciphertext Z and obtain the certificate $UCert$ of the signer. Next it uses its access to the registration database to obtain UPK and S which correspond to $UCert$. From this information it computes a publicly verifiable NIZK proof that $UCert$ is actually encrypted in Z. Together with the database entry $A, cert_{CS}, S$ this convincingly reveals the identity of the signer in a non-repudiable way. A judge can verify the proof using GPK and in the process also obtains a proof that the Opener as acted honestly and has not framed any other participant. In legislations where a meterID or similar needs to be stored with each transaction, the opener decrypts this information from the tuple t contained in M and also transmits it to the judge. Upon receiving this information the judge uses the RSA public key of the RootCA pk_{CA} to verify the certificate for the CS UPK as $Vrfy_{pk_{CA}}(UPK)$ and the CS's signature of A, S as $Vrfy_{UPK}(A, S)$. He has thus both obtained a proof that the opener acted correctly and the information, which the CS issued the signature on, is contained in the disputed message. For more details on the computations performed in the opening and judging process, please refer to Delerablee and Pointcheval [76].

If all proofs verified in the process outlined above, evidence exists that the metering information has not been altered during transmission and storage. As the signing CS (and the meterID) are known at this point, it is also possible to verify the correct function of the energy meter in the respective CS. Thus, the process offers a high

amount of legal certainty for all parties involved: the clearing house, electric utilities and the customer.

5.4 Evaluation

In this section, we describe how we evaluated our prototype implementation. We also present an overview of the performance results obtained both for the various operations of the XSGS scheme and the transmission of data from a CS to the BE.

5.4.1 Evaluation Environment

We aimed at evaluating our approach in a realistic environment. Thus, we implemented XSGS and tested the creation of signed messages, the setup process for adding new charging stations, and the procedure to decommission charging stations on a prototype of a CS for EVs built at our department. The CS contains an inexpensive industrial-grade Intel Atom platform (CS1, cf. Table 5.1) as control unit that interacts with the energy flow control subsystems within the CS and acts as a front-end to the user. Additionally, we evaluated our implementation on a Freescale i.MX53, which is an implementation of an ARM A8 core. Comparable platforms to both variants can be found in CSes in the market or under development today.

As BE we chose an Intel server platform (cf. Table 5.1). We used this platform to evaluate all XSGS operations typically performed by the group manager, opener, judge, or any entity that wishes to verify a signature. We also created signatures and performed join operations as a comparison to the measurements on the actual CS. While the Tor network is widely used and considered usable for non-time critical applications, we also used this platform to evaluate if latency and throughput are acceptable in our application scenario.

Table 5.1: Evaluation Environment

	Hardware Platform		OS
CS_1	Intel Atom D2550,	1GB RAM	Ubuntu 12.04
CS_2	Freescale i.MX53,	1GB RAM	Ubuntu 10.04
BE	Intel Xeon X5650,	2GB RAM	Ubuntu 12.04

5.4.2 Evaluation Results

We performed the setup procedure required for adding a new CS 100 times. The computations necessary on the CS are performed on average in 757.4 ms on CS_1

and 1077.3 ms on CS_2, while the computations on the BE took 55 ms on average. Accordingly, we performed 100 decommission procedures: on average, the computations performed on CS_1 take 49.0 ms (resp. 77.8 ms on CS_2), the computations on the BE take 20 ms. We also performed 100 dispute resolution procedures on the BE: on average opening a message takes 8.2 ms, while judging takes 6.9 ms.

We evaluate the time required to prepare a message to transmit the metering data to the BE. Preparing a message 1000 bytes (taken from `/dev/urandom`) takes 28.5 ms on average on CS_1; on CS_2 the process takes 41.5 ms. Preparing a message that allows for batch verification on the BE takes slightly longer: 28.8 ms on CS_1 or 43.1 ms on CS_2. In both cases the time required for preparing the messagess scales linearly with the amount of messages. For a message size up to 100,000 bytes message creation takes less than 33 ms on CS_1 and 54.2 ms on CS_2. Figure 5.3 shows that the size of the message only has a limited impact on the time required to create a valid signature, as we only sign a hash of the message. Creating a signed message of one million bytes takes 66.7 ms on average on CS_1 and 161.1 ms on CS_2. These results show that ensuring the authenticity of messages by means of group signatures is feasible on the limited hardware found in a CS. Even more so, as we only need to generate one signature for each charging process. As even quick charging takes about 20 to 30 minutes for an 80% charge today and will take at least minutes in the foreseeable future, the amount of time required for signing the customer's energy consumption data is insignificantly small.

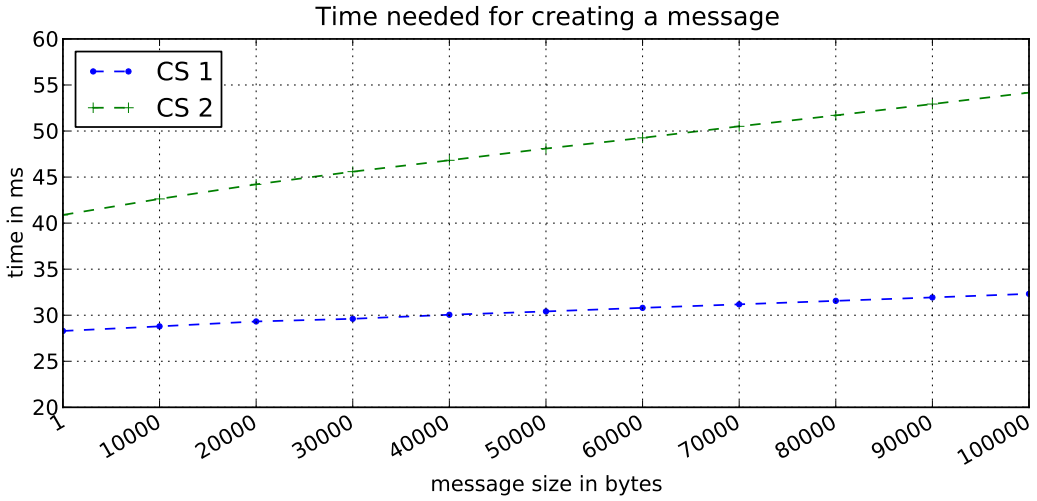


Figure 5.3: Time required for message creation by size

Being able to batch verify messages offers a significant performance increase. While a CS will typically only create one message every few minutes or every few hours,

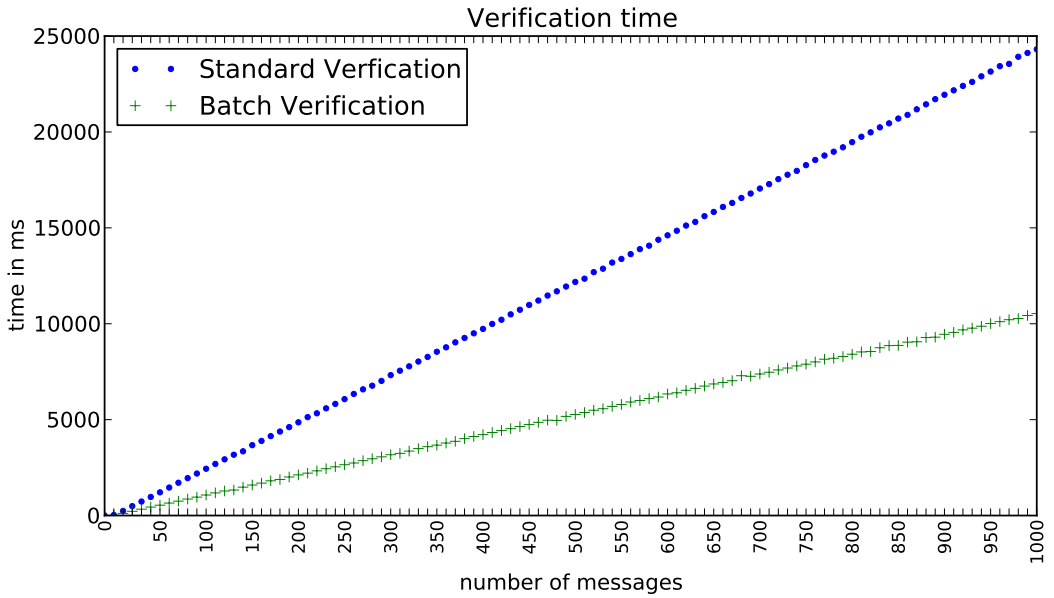


Figure 5.4: Time required for verification by #messages received

each message has to be verified by the BE. The verification of a normal message takes 30 ms, a single batch-enabled one is verified in about the same time. Figure 5.4 shows that verification time increases linearly with the number of messages. Standard verification allows for processing 41 messages per second on the BE, while batch verification allows for processing of 93 messages in the same time. When comparing the time required for verifying one thousand messages, batch verification is about 2.3 times faster. In a worst case scenario, where a batch contains so many invalid signatures that it is faster to verify each individual message, we can still process 147,600 messages per hour using a single CPU core. As the process can be parallelized at will, a comparable server with eight CPUs cores instead of one is sufficient for processing more than one million messages per hour.

As transmission times vary due to network latency, we evaluate the network performance separately: We used `iperf`[20] to measure whether the Tor network offers enough bandwidth for transmitting metering data from the CS to the BE. We controlled that the bandwidth between the host running the `iperf` server and and the one running the client is not the limiting factor and repeated our measurements at various times of the day, building a new Tor circuit for each iteration. We were able to transfer a minimum of 373 kbit per second and a maximum of 1.07 Mbits per second through the Tor network. While the actual throughput may vary depending

[20] http://iperf.sourceforge.net/

on the time of day and the chosen circuit, our evaluation shows that it is reasonable to assume that we can transfer metering data through the Tor network, especially as the communication between CS and BE is not subject to real-time requirements. Also note that the BE is not affected by Tor's limited bandwidth, as there is no need to obscure the BE's location and only CSes communicate via Tor.

In summary, we found that our approach performed well on all tested platforms and, most importantly, is fast enough for our application.

5.5 Discussion

We now discuss possible attacks against both the authenticity of billing-relevant data and against the user's locational privacy.

5.5.1 Malicious Customer

While our system is well equipped to counter attackers with capabilities as described in Section 5.2.1, there exists the theoretical possibility that an attacker, who is a valid customer in the system, could force a CS_1 offline before a revocation of a different CS_2 takes place. Thus CS_1 does not realize that the group credentials have changed and must be recomputed. The attacker then authenticates himself and charges his EV at CS_1, which is possible as user authentication works offline. The CS signs the metering data with its current credentials. At some point in the future, when the CS is online again, it transmits the data to the BE. It will then also receive new group credentials and will again be able to create valid messages. Still, the BE will discard the delayed metering data from the CS as it has been generated with the old credentials. Hence, the attacker was able to charge for free in the meantime. There are at least two counters to this attack. First, if the CS is up and running again, it may simply re-sign all the unsent metering data with the updated credential. Second, if the CS is for some reason not able to continue signature generation, we can still retain old credentials for verification and use the old group signature to bill the customer correctly.

To be able to resolve disputes that concern metering data that was signed at some point in the past before one or more revocation events took place, we have to store old group credentials for the period regulations or utility terms and conditions dictate for dispute resolution anyway. Also, for operative reasons, most utilities will want to define a maximum time a CS may be offline before it is marked as faulty in a monitoring system and a repair crew is sent out. While the system is designed to be tolerant towards network outages, long periods where a CS is offline will be

undesirable for management reasons. As a consequence, there will be an archived GPK available to verify the incoming message, as the dispute resolution period is longer than the maximum offline time tolerated by the utility operating the CS.

5.5.2 Tracking and Localization Attacks

Ma et al. show that if a set of traces of time and corresponding location of mobiles nodes exist, where "[t]he traces are anonymous in that the true identity of a participant has been replaced by a random and unique identifier" [141], a small amount of side information is sufficient for an attacker to infer the true identity of a user. The work of de Montjoye et al. [74] supports these claims and shows that even datasets with coarse traces provide little anonymity, in such that four spatio-temporal points are enough to uniquely identify an individual with a probability of 95%.

However, neither approach is applicable to our system. We do not conceal the identity of the user, but cryptographically protect their location. All information is transmitted encrypted with a provably secure TLS variant. Thus, the attacker needs to be a legitimate receiver of the data, *i. e.*, the clearing house or a utility. Both receive the following information: customer A of utility B consumed N kWh of energy, starting from timestamp X, ending at timestamp Y. Every location-bound token, like the CS's public key and the meterID, is encrypted only to the opener and thus never leaked to any other party. This encrypted data is also transmitted to both the clearing house and the respective utility. However it is meaningful to neither party as both lack the appropriate key to decrypt the data.

The data available to an adversary thus does not contain the location of the user, nor can the attacker use the amount of energy consumed to infer the distance the user has driven between two charging events, due to external factors that influence power consumtion, like driving style, speed, etc. For instance, wind resistance increases with the speed of a vehicle, such that a user can cover a long distance at lower speed or a shorter distance at higher speed while consuming the same amount of energy. An attacker may infer a limited amount of information from the timestamps written at the beginning and the end of the charging process, namely how often a user charged her vehicle and how long the individual charging processes took. However, it is indiscernible to the attacker whether these charging processes took place at different CSs or always at the same CS. The attacker also still lacks the information of where the relevant CSes are located (assuming there is more than one CS within reachable distance of the user).

Shokri et al. [200] propose a metric to quantify the performance of a location privacy protection mechanism (LPPM) that, given a trace of spatio-temporal locations, pro-

tects the user from localization attacks, meeting disclosure attacks and aggregated presence attacks by reducing the accuracy and/or precision of the events' spatio-temporal information. In their termini our systems applies location hiding as an online LPPM in a distributed architecture, *i. e.*, we only look at the current event at the time of its creation and hide all location-bound information by encrypting it to the opener. As argued above, while records of user interaction exists for billing purposes, they do not contain any spatio-temporal locations or references to such data. An adversary, who knows the location of every CS, may determine the location where the EV *could have been charged* with a high accuracy (as it was necessarily at the location of a CS), but he is unable to achieve a high correctness as to where the EV *was actually charged*.

A potential information leak could exist, if the billing data the clearing house receives from a charging station not only contained the information that a user is a customer of a given utility, but also contained the information which utility owns the CS this customer just used. For example, given two charge events at two different utilities separated by 2 hours, there might be only one possible pair of charging stations for which this would have been feasible. However, the clearing house does not need to receive this information. It must merely receive the information that a user is a registered customer of a given utility. At the end of the clearing period the clearinghouse also receives the accumulated amount of energy each utility dispensed via its CSs and can thus balance claims against each other.

5.6 Summary

In this chapter we showed that it is possible to follow the principle of *security and privacy by design* in a tightly regulated environment while allowing market participants to continue to use existing processes. We introduced a system that protect customers' locational privacy during financial transactions, without relying on anonymous payments. We focused on the example of re-charging electric vehicles and are able to protect the customer's locational privacy during the whole charging process. Our system also fully supports all requirements needed to bill the customer after the charging process and enables users to roam between different CSes provided by different electric utilities. As such, it covers all relevant aspects required for the charging of EVs.

The basic idea of our approach is to adapt a group key signature scheme to the tightly regulated setting of selling electric energy as means of propulsion. We described all protocol steps and outlined how the system can be deployed in practice.

In an empirical evaluation, we also demonstrated that the solution has a low overhead and can scale to millions of charging processes per hour—even on off-the-shelf hardware.

6 Conclusion

"See, I told you they'd listen to Reason,"
Fisheye says, shutting down the whirling gun.
—Neal Stephenson, *Snow Crash*

In this thesis, we presented several approaches to advance the security of software and systems. While the benefits of improving security can be manifold and our approaches have as much merit in a corporate context, our intentions are to improve the situation of the many millions of computer users, who process and store their private data on mobile and stationary computer systems.

In **Chapter 2** we introduced a mitigation technique for novel attacks that are based on Scalable Vector Graphics, building on offensive research that allowed us to be proactively aware of attack vectors. SVGs were of limited relevance on the web until the introduction of the HTML5 standard, which made it mandatory for browsers to support the format. Being rather one-file-webapplications than simply images the complexity of a living standard proved and sometimes still proves to be overwhelming for developers to follow completely. We counter the threats emerging from this fact by providing *SVGPurifier*, which removes malicious content from a given SVG file, without damaging benign file structure and contents. We have tested our prototype with over 100,000 SVG images and found that we can filter 98.5% of the files without causing differences in the visual appearance. For the remaining 1.5% we determined the visual deviation to be negligible in more than half of the cases. While the risks of SVGs have been mitigated, preemptive offensive research is a continuous effort that spans fields and disciplines. In the context of web browsers, we have followed up by presenting a new class of attacks, called *mutation-based XSS* [111], while Heiderich et al. also presented a class of attacks that do not require scripts at all [110]. Others have shown attacks on various approaches that aim

to counter Return Oriented Programming [206], attacked Acoustic Captchas [155], SAML implementations [160], the XML-Security standard [209], digital locking systems [214], One-Time Password Generators [186], or demonstrated the feasibility and effectiveness of hardware trojans [28].

In **Chapter 3** we investigated whether the goals that the authors of an open source application claim to achieve are fully reflected in the applications key exchange and messaging protocol and are indeed be achieved by the application. In the introduction we cited the example of the Heartbleed bug in OpenSSL to support the notion that the mere availability of source code is not necessarily positive in itself, but only insofar as it facilitates the review and analysis of software. Just recently we needed to add Bash[21] to the list of widespread open-source software, whose source code did not receive the appropriate amount of attention in a timely manner. In the chapter at hand we aim at giving TextSecure the attention a supposedly secure instant messaging application with more than 500,000 users deserves. We are the first to formally describe and analyse its complete protocol. In the course of this analysis we discover several potential weaknesses and show that it is possible to conduct an Unknown Key-Share attack against users of TextSecure. In the following we discuss how these issues can be mitigated and show that with our proposed amendments in place TextSecure does indeed achieve the goals of authenticity and confidentiality. However, while important, our analysis targets just one element of the vast Open Source landscape. As we have argued and exemplified, more and continuous effort is needed to ensure that users do not base their security and privacy expectations on empty promises. On the other hand it would be applaudable, if software developers would abstain from simply designing cryptographic protocols from scratch, following a best-effort approach. Using well-understood protocols that have received extensive scrutiny from the academic community definitively has its merits, although new protocols are necessary to better apply to specific requirements. Before coming up with a sparsely documented or even closed source implementation, these developers should first show their protocols' soundness in theory in the open.

In **Chapter 4** we presented a novel approach to detect and mitigate malicious code on websites. To this end we create an alternative execution context and use inline code analysis to the advantage of not needing to worry about obfuscation, as the code needs to be in its de-obfuscated form directly before it can be executed anyway. We then inspect the websites' in-browser representation for suspicious behavior that we have described in heuristics. To arrive at these heuristics we analyzed a significant set of malicious and benign websites to develop meaningful features. We then use these features to classify code with the help of a supervised learning algorithm. Our empirical evaluation on different classes of computing devices shows that the

[21] CVE-2014-6271, -7169, -6277, -6278, -7186, and -7187

IceShield's average runtime overhead is below 12 ms on workstations and 89 ms on a smartphone, while achieving a detection accuracy of 98% using live malicious websites. We also were able to detect three exploits that the tool had never seen before and demonstrate how attacks can be mitigated successfully. If we want to remain lightweight and directly within the scope of web browsers, there are two basic paths to follow, that may converge eventually: hardening the DOM on the one hand and continuously detecting also novel malicious behavior. Following the first direction, Heiderich [104] has shown how the DOM can be retrofitted with tamper-proof fine-grained controls to create a trusted and capability-controlled DOM, while, for instance, Phung et al. [192] have shown how ECMA Script 5 can be used to sandbox untrusted JavaScript. Following the second direction, it is plausible that the quality of the detection models of supervised learning approaches degrades as malicious code evolves over time. Schwenk et al. [207] argue that a lot of work remains to be done to arrive at autonomous, continuous learning approaches, that do not require human involvement for defining a ground truth. Leaving the context of browsers completely, Corona et al. [63] have shown that behavior-based statistical learning approaches as presented in this chapter can also be successfully applied in different fields, such as the detection of malicious JavaScript in PDF files. Back in the context of browsers, other approaches follow the path of hardening and compartmentalizing (elements of) the browser itself—as implemented in Google Chrome or Microsoft Internet Explorer—or also involve the respective website to deliver strict definitions to the browser of which kind of resources it may access from which sources, such as Content Security Policy (CSP). However, none of these approaches can currently be considered bulletproof in every situation [46, 159, 179, 236].

In **Chapter 5** we follow an alternative route. Instead of amending an existing technology we follow the principle of *security and privacy by design* to integrate the functional requirement of providing data for post-paid billing with the requirements of authenticity and prevention of customer movement profiles. We propose a locational privacy-preserving architecture using the example of a roaming-enabled charging infrastructure for electric vehicles. By introducing a system that protects customers' locational privacy without relying on anonymous payments we show that privacy-friendly solutions can be integrated with existing business processes and also provide sufficient flexibility to follow regulatory requirements. An empirical evaluation of our approach shows that it works effectively on limited hardware and can scale to millions of transactions per hour even on off-the-shelf general purpose hardware. Our approach has also found interest beyond the academic community. Efforts are under way to integrate it into a commercial PKI solution, which underlines our claim that security, privacy, and functionality must not be considered conflicting goals, but together can help to create novel, sustainable solutions that help us solve today's and tomorrow's challenges.

Outlook and Future Work In this thesis, we do explicitly not adopt the position that our work provides the one way to achieve a better protection of private, or for that matter, arbitrary data. We offer but one contribution to improve the status quo in an ever-evolving context. Although we proactively developed mitigation techniques for novel attack vectors we followed the line of reactive security, creating an amendment to the imperfect situation users find themselves in with modern and regrettably often enough not so modern browsers. We thus necessarily furthered the arms race that defines large parts of todays computer security. While one could argue that such an arms race is not sustainable, there are currently no clear indications that it will end anytime soon, as this would require abandoning most existing systems and infrastructures and starting over. We can rather expect gradual changes and a rather long migration path towards more secure and privacy friendly systems and infrastructures. The process of raising the stakes for attackers is a continuous one and requires continuous innovation for our defenses to be effective.

We also analyzed privacy-enhancing technology—that does indeed represent a positive element of this gradual change—with the aim to amend its shortcomings and to smoothen the path towards privacy-enhancing technologies that are actually usable for a not technically inclined person. Narrowing the scope to the concrete field of cryptographic messaging protocols TextSecure is a step forward and we thank its developers for their applaudable effort. However, new—purportedly secure—instant messenger apps are published almost on a weekly basis, and almost none is ever reviewed. And while diversity can be helpful, it also requires interoperability. The path towards secure and private communication for larger parts of the populace thus requires a common standard. It should define an efficient, secure, privacy-enhancing instant messaging protocol that matches todays usability requirements and can be proven to be secure in a strong security model and allows for a well-defined federated infrastructure. Such a standardized or even standardizable protocol for instant messaging communication is a valuable medium-term goal. In the long run however, secure communication and storage solutions are futile without appropriately secure underlying systems and infrastructures.

Widening the scope again to look at the larger picture, analyzing existing systems and novel threats, and pragmatically amending existing technologies to improve their security properties will be an important task for years to come. However, when engineering new architectures for emerging fields of technology, it is mandatory to do better than just “functional”. When rolling out new infrastructures or designing new systems, security *and* privacy must be fundamental building blocks of each and every step in the process—from the initial idea to the details of the actual implementation.

List of Figures

List of Tables

Listings

Bibliography

[1] "National vulnerability database (NVD) (CVE-2007-1765)," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-1765, Mar. 2007.

[2] "National vulnerability database (NVD) (CVE-2008-3702)," http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-3702, Aug. 2008.

[3] "Fonts – SVG 1.1 (Second Edition)," http://www.w3.org/TR/SVG/fonts.html, Jun. 2010. [Online]. Available: http://www.w3.org/TR/SVG/fonts.html

[4] "National vulnerability database (NVD) (CVE-2010-3113)," http://web.nvd.nist.gov/view/vuln/detail?vulnId=cve-2010-3113, Aug. 2010.

[5] "Scalable vector graphics (SVG) 1.1 (Second edition)," http://www.w3.org/TR/SVG11/, Jun. 2010. [Online]. Available: http://www.w3.org/TR/SVG11/

[6] "Svgpurifier: inaccurately converted images," http://svgpurifier.nds.rub.de/, May 2011. [Online]. Available: http://svgpurifier.nds.rub.de/

[7] "Edward snowden and ACLU at SXSW," Mar. 2014. [Online]. Available: https://www.youtube.com/watch?v=UIhS9aB-qgU&t=14m24s

[8] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 340–353.

[9] Adobe Systems Inc., "Illustrator 10 XML Extensions Guide," Sep. 2001. [Online]. Available: ftp://leonxiii.no-ip.org/Diseno%20Grafico/.../AI10XMLGrammar.pdf

[10] Aite Group, LLC, "The less-cash society: Forecasting cash usage in the united states," Tech. Rep., 2011.

[11] Alexa, the Web Information Company, "Top 1,000,000 Sites," November 2010, http://www.alexa.com/topsites.

[12] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of rc4 in tls," in *USENIX Security*, 2013.

[13] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *IEEE Symposium on Security and Privacy*, 2013.

[14] A.R. Beresford and F. Stajano, "Location privacy in pervasive computing," *IEEE Pervasive Computing*, vol. 2, no. 1, pp. 46 – 55, Mar. 2003.

[15] C. Arango, D. Hogg, and A. Lee, "Why is cash (still) so entrenched? Insights from the Bank of Canada's 2009 methods-of-payment survey," Bank of Canada, Tech. Rep., 2012.

[16] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Usenix WOOT*, 2010.

[17] G. Avoine, M. Bingol, X. Carpent, and S. Yalcin, "Privacy-friendly authentication in RFID systems: On sublinear protocols based on symmetric-key cryptography," *IEEE Transactions on Mobile Computing*, vol. 12, no. 10, pp. 2037–2049, Oct. 2013.

[18] M. Backes, M. Durmuth, and D. Unruh, "Information Flow in the Peer-Reviewing Process," in *IEEE Symposium on Security and Privacy*, 2007.

[19] J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede, "PrETP: privacy-preserving electronic toll pricing," in *19th USENIX Security Symposium*, 2010.

[20] F. Baldimtsi, G. Hinterwalder, A. Rupp, A. Lysyanskaya, C. Paar, and W. Burleson, "Pay as you go," in *HotPETs*, 2012.

[21] M. Balduzzi, "New insights into clickjacking," in *OWASP AppSec Research*, Jun. 2010.

[22] E. Barker, D. Johnson, and M. Schmid, "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)," NIST Special Publication 800-56a, National Institute of Standards and Technology, Tech. Rep., 2007.

[23] A. Barth, "Bug 29278 – XSSAuditor bypasses from sla.ckers.org," Sep. 2009, https://bugs.webkit.org/show_bug.cgi?id=29278.

[24] A. Barth, J. Caballero, and D. Song, "Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves," in *IEEE Symposium on Security and Privacy*, 2009.

[25] A. Barth, A. P. Felt, P. Saxena, and A. Boodman, "Protecting browsers from extension vulnerabilities," in *Proc. of the 17th Network and Distributed System Security Symposium*, 2009.

[26] A. Barth, C. Jackson, C. Reis, and Google Chrome Team, "The Security Architecture of the Chromium Browser," 2008, http://seclab.stanford.edu/websec/chromium/.

[27] B. Beck, "LibreSSL - an OpenSSL replacement. the first 30 days, and where we go from here," 2014. [Online]. Available: http://www.openbsd.org/papers/bsdcan14-libressl/index.html

[28] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, ser. Lecture Notes in Computer Science, G. Bertoni and J.-S. Coron, Eds. Springer Berlin Heidelberg, Jan. 2013, no. 8086, pp. 197–214.

[29] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology – CRYPTO'96*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 18–22, 1996, pp. 1–15.

[30] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *38th Annual Symposium on Foundations of Computer Science*. Miami Beach, Florida: IEEE Computer Society Press, Oct. 19–22, 1997, pp. 394–403.

[31] M. Bellare, J. A. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *EUROCRYPT*, 1998, pp. 236–250.

[32] M. Bellare and S. Keelveedhi, "Authenticated and misuse-resistant encryption of key-dependent data," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 14–18, 2011, pp. 610–629.

[33] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *Advances in Cryptology – ASIACRYPT 2000*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed., vol. 1976. Kyoto, Japan: Springer, Berlin, Germany, Dec. 3–7, 2000, pp. 531–545.

[34] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS*, 1993.

[35] M. Bellare, H. Shi, and C. Zhang, "Foundations of group signatures: The case of dynamic groups," in *CT-RSA*, 2005, pp. 136–153.

[36] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *PKC*, 2006. [Online]. Available: http://cr.yp.to/ecdh/curve25519-20060209.pdf

[37] P. Bisht and V. N. Venkatakrishnan, "XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks," in *Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2008.

[38] E.-O. Blass, A. Kurmus, R. Molva, and T. Strufe, "PSP: private and secure payment with RFID," in *WPES*, 2009.

[39] A. J. Blumberg and P. Eckersley, "On locational privacy, and how to avoid losing it forever," Electronic Frontier Foundation, Tech. Rep., 2009. [Online]. Available: https://www.eff.org/wp/locational-privacy

[40] H. Bojinov, E. Bursztein, and D. Boneh, "XCS: Cross Channel Scripting and its Impact on Web Applications," in *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[41] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *CRYPTO*, 2004, pp. 41–55.

[42] N. Borisov, I. Goldberg, and E. A. Brewer, "Off-the-record communication, or, why not to use pgp," in *WPES*, 2004, pp. 77–84.

[43] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment.* Springer, 2003.

[44] d. boyd and A. E. Marwick, "Social privacy in networked publics: Teens' attitudes, practices, and strategies," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 1925128, Sep. 2011.

[45] S. Brands, "Electronic cash systems based on the representation problem in groups of prime order," in *CRYPTO*, 1993.

[46] F. Braun and M. Heiderich, "X-frame-options: All about clickjacking?" Tech. Rep., 2014. [Online]. Available: https://frederik-braun.com/xfo-clickjacking.pdf

[47] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider, "AmazonIA: When elasticity snaps back," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 389–400.

[48] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," RFC 4880 (Proposed Standard), Internet Engineering Task Force, Nov. 2007, updated by RFC 5581. [Online]. Available: http://www.ietf.org/rfc/rfc4880.txt

[49] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Advances in Cryptology - EUROCRYPT*, 2005.

[50] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO*, 2002, pp. 61–76.

[51] J. L. Camenisch, J.-M. Piveteau, and M. A. Stadler, "An efficient electronic payment system protecting privacy," in *ESORICS*, 1994.

[52] cars21.com, "EU proposes minimum of 8 million EV charging points by 2020," http://beta.cars21.com/news/view/5171, 2013.

[53] A. Cavoukian, "Privacy by design. the 7 foundational principles: Implementation and mapping of fair information practices," Tech. Rep., 2010.

[54] Chao Li, "Anonymous payment mechanisms for electric car infrastructure," Master's thesis, LU Leuven, Leuven, 2011.

[55] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *CRYPTO*, 2006, pp. 78–96.

[56] D. Chaum, "Blind signatures for untraceable payments," in *dvances in Cryptology: Proceedings of CRYPTO '82*, 1982.

[57] ——, "Security without identification: transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, no. 10, p. 1030–1044, Oct. 1985.

[58] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology - CRYPTO*, 1988.

[59] D. Chaum and E. van Heyst, "Group signatures," in *EUROCRYPT*, 1991, pp. 257–265.

[60] S. Checkoway, H. Shacham, and E. Rescorla, "Are Text-only Data Formats Safe? or, Use This LaTeX Class File to Pwn Your Computer," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[61] J. Clark, "XSL transformations (XSLT)," http://www.w3.org/TR/xslt, Nov. 1999.

[62] clicky.com, "Web browsers by version (global marketshare)," Oct. 2014. [Online]. Available: http://clicky.com/marketshare/global/web-browsers/versions/

[63] I. Corona, D. Maiorca, D. Ariu, and G. Giacinto, "Lux0r: Detection of malicious PDF-embedded JavaScript code through discriminant analysis of API references," in *Proceedings of the 7th ACM Workshop on Artificial Intelligence and Security*, 2014.

[64] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *19th international conference on World Wide Web*, 2010.

[65] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen, "A Safety-Oriented Platform for Web Applications," in *IEEE Symposium on Security and Privacy*, 2006.

[66] M. Crider, "CyanogenMod is now installed on over 10 million android devices," Dec. 2013. [Online]. Available: http://www.androidpolice.com/2013/12/22/cyanogenmod-is-now-installed-on-over-10-million-android-devices/

[67] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Fast and Precise In-Browser JavaScript Malware Detection," in *USENIX Security Symposium*, 2011.

[68] A. Dabirsiaghi, "The OWASP AntiSamy project," http://code.google.com/p/owaspantisamy/, Apr. 2011.

[69] E. Dahlström, "SVG and HTML," http://dev.w3.org/SVG/proposals/svg-html/svg-html-proposal.html, Jul. 2008.

[70] J. Dahse, N. Krein, and T. Holz, "Code reuse attacks in PHP: Automated POP chain generation," in *21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[71] E. Damiani, S. De Capitani di Vimercati, E. Fernandez-Medina, and P. Samarati, "An access control system for SVG documents," *King's College, University of Cambridge, UK*, pp. 29–31, 2002.

[72] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, 2003.

[73] L. Davi, A. Dmitrienko, M. Egele, T. Fischer, T. Holz, R. Hund, S. Nürnberger, and A.-R. Sadeghi, "MoCFI: A framework to mitigate control-flow attacks on smartphones," in *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[74] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific Reports*, 2013. [Online]. Available: http://www.nature.com/srep/2013/130325/srep01376/full/srep01376.html

[75] M. Deiters, "Aspect-Oriented programming," Jul. 2010, http://msdn.microsoft.com/en-us/library/aa288717(VS.71).aspx.

[76] C. Delerablée and D. Pointcheval, "Dynamic fully anonymous short group signatures," in *VIETCRYPT*, 2006, pp. 193–210.

[77] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Des. Codes Cryptography*, 1992.

[78] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *13th USENIX Security Symposium*, 2004.

[79] F. Dötzer, "Privacy issues in vehicular ad hoc networks," in *Privacy Enhancing Technologies*, 2006.

[80] B. Driessen, R. Hund, C. Willems, C. Paar, and T. Holz, "Don't trust satellite phones: A security analysis of two satphone standards," in *2012 IEEE Symposium on Security and Privacy (SP)*, May 2012, pp. 128–142.

[81] EFF Twitter account, "Bruce schneier: "one-click encryption is one click too many." #TrustyCon pic.twitter.com/HtrZWHGZTE," Feb. 2014. [Online]. Available: https://twitter.com/EFF/status/439126918663643136/photo/1

[82] European Commission, "COMMISSION RECOMMENDATION on the implementation of privacy and data protection principles in applications supported by radio-frequency identification," Tech. Rep. C(2009) 3200, 2009.

[83] H. Father, "Hooking Windows API - Technics of Hooking API functions on Windows," *The CodeBreakers Journal*, vol. 1, no. 2, 2004.

[84] A. Fiat, "Batch RSA," in *CRYPTO*, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 175–185.

[85] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986, pp. 186–194.

[86] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication," RFC 7235 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7235.txt

[87] M. Fischlin, A. Lehmann, T. Ristenpart, T. Shrimpton, M. Stam, and S. Tessaro, "Random oracles with(out) programmability," in *Advances in Cryptology – ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 6477. Singapore: Springer, Berlin, Germany, Dec. 5–9, 2010, pp. 303–320.

[88] J. Freudiger, M. Raya, M. Félegyházi, P. Papadimitratos *et al.*, "Mix-zones for location privacy in vehicular networks," in *Win-ITS*, 2007.

[89] T. Frosch, M. Kührer, and T. Holz, "Predentifier: Detecting botnet c&c domains from passive DNS data," in *Advances in IT Early Warning*. Darmstadt: Fraunhofer Verlag, 2013.

[90] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is TextSecure?" in *IN SUBMISSION*, 2014.

[91] T. Frosch, S. Schäge, M. Goll, and T. Holz, "Improving location privacy for the electric vehicle masses," Horst Görtz Institute for IT Security, Technical Report TR-HGI-2013-001, 2013. [Online]. Available: https://www.hgi.rub.de/media/attachments/files/2013/11/TR-HGI-2013-001.pdf

[92] Gijs Mom, *The Electric Vehicle: Technology and Expectations in the Automobile Age*. John Hopkins University Press, 2004.

[93] Google, "V8 benchmark suite," 2011, http://v8.googlecode.com/svn/data/benchmarks/v6/run.html.

[94] M. Green, "Noodling about IM protocols," Jul. 2014. [Online]. Available: http://blog.cryptographyengineering.com/2014/07/noodling-about-im-protocols.html

[95] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker, "Manufacturing compromise: The emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 821–832.

[96] C. Grier, S. Tang, and S. T. King, "Secure Web Browsing with the OP Web Browser," in *IEEE Symposium on Security and Privacy*, 2008.

[97] S. Guarnieri and B. Livshits, "GATEKEEPER: Mostly Static Enforcement of Security and Reliability Policies for JavaScript Code," in *USENIX Security Symposium*, 2009.

[98] M. V. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks," in *Symposium on Network and Distributed System Security (NDSS)*, 2009.

[99] U. Harnhammar, "kses - PHP HTML/XHTML filter," http://sourceforge.net/projects/kses/, Mar. 2010. [Online]. Available: http://sourceforge.net/projects/kses/

[100] Harris Interactive, "A generation unplugged," Tech. Rep., 2008.

[101] T. Hastie, R. Tibshirani, and R. Friedman, "Linear discriminant analysis," in *The Elements of Statistical Learning*. Springer, 2001, p. 84 ff.

[102] M. Heiderich, "Opera SVG AII testcase," http://heideri.ch/opera/, 2011.

[103] ——, "SVG chameleon via XSLT - HTML5 Security Cheatsheet," http://html5sec.org/#125, Mar. 2011.

[104] ——, "Towards elimination of XSS attacks with a trusted and capability controlled DOM," Ph.D. dissertation, Ruhr-Universität Bochum, Bochum, 2012.

[105] M. Heiderich and T. Frosch, "SVGpurifier smoketest," http://heideri.ch/svgpurifier/SVGPurifier/, Apr. 2011.

[106] M. Heiderich, T. Frosch, and T. Holz, "IceShield: Detection and mitigation of malicious websites with a frozen DOM," Menlo Park, CA, 2011.

[107] M. Heiderich, T. Frosch, M. Jensen, and T. Holz, "Crouching tiger – hidden payload: Security risks of scalable vector graphics," Chicago, IL, USA, 2011.

[108] M. Heiderich, T. Frosch, M. Niemietz, and J. Schwenk, "The bug that made me president: A browser- and web-security case study on helios voting," Tallinn, Estonia, 2011.

[109] M. Heiderich, K. Kotowicz, and M. Rupp, "Cure53 public pentest report: Cryptocat 2," Tech. Rep., 2012. [Online]. Available: https://blog.crypto.cat/wp-content/uploads/2012/11/Cryptocat-2-Pentest-Report.pdf

[110] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: Stealing the pie without touching the sill," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 760–771. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382276

[111] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang, "mXSS attacks: Attacking well-secured web-applications by using innerHTML mutations," Berlin, Germany, 2013.

[112] T. S. Heydt-Benjamin, H.-J. Chae, B. Defend, and K. Fu, "Privacy for public transportation," in *Privacy Enhancing Technologies*, 2006.

[113] G. Heyes, "Polymorphic javascript," Jul. 2010, http://www.thespanner.co.uk/2008/02/27/polymorphic-javascript/.

[114] I. Hickson, "HTML standard — the map element," http://whatwg.org/specs/web-apps/current-work/multipage/the-map-element.html#svg-0, Apr. 2011.

[115] L. Huang, Z. Weinberg, C. Evans, and C. Jackson, "Protecting browsers from Cross-Origin CSS attacks," in *ACM Conference on Computer and Communications Security (CCS) 2010)*, 2010.

[116] J.-P. Hubaux, S. Capkun, and J. Luo, "The security and privacy of smart vehicles," *Security & Privacy, IEEE*, vol. 2, no. 3, pp. 49–55, 2004.

[117] R. Hund, T. Holz, and F. C. Freiling, "Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 383–398.

[118] A. Ikinci, T. Holz, and F. C. Freiling, "Monkey-spider: Detecting malicious websites with low-interaction honeyclients," in *Sicherheit*, 2009.

[119] International Organization for Standardization, "ISO/IEC DIS 15118 - road vehicles – vehicle to grid communication interface," 2012.

[120] ISO/IEC, "Iso/iec 18033-2:2006, information technology – security techniques – encryption algorithms – part 2: Asymmetric ciphers," International Organization for Standardization, Tech. Rep., 2006.

[121] I. Jackson, "Anonymous addresses and confidentiality of location," in *Information Hiding*, 1996.

[122] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Advances in Cryptology - CRYPTO*, 2012.

[123] T. Jager, K. G. Paterson, and J. Somorovsky, "One bad apple: Backwards compatibility attacks on state-of-the-art cryptography," in *NDSS Symposium*, 2013.

[124] T. Jager, S. Schinzel, and J. Somorovski, "Bleichenbacher's attack strikes again: breaking PKCS# 1 v1. 5 in XML encryption," in *ESORICS*, 2012.

[125] T. Jager and J. Somorovsky, "How to break XML encryption," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 413–422. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046756

[126] M. Johns, "Code injection vulnerabilities in web applications - exemplified at cross-site scripting," Ph.D. dissertation, University of Passau, Passau, Jul. 2009.

[127] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodová, C. Taylor, V. Frolov, E. Reeber, and A. Naik, "Replacing testing with formal verification in Intel Core i7 processor execution engine validation," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. Bouajjani and O. Maler, Eds. Springer Berlin Heidelberg, Jan. 2009, no. 5643, pp. 414–429.

[128] A. Karjalainen and N. Mehta, "The heartbleed bug," 2014. [Online]. Available: http://heartbleed123.com/

[129] 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)*, 2007.

[130] K. Kim, I. Yie, S. Lim, and D. Nyang, "Batch verification and finding invalid signatures in a group signature scheme," *I. J. Network Security*, vol. 13, no. 2, pp. 61–70, 2011.

[131] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating Cross-Site scripting attacks," in *ACM Symposium on Applied Computing (SAC)*, 2006.

[132] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 207–220.

[133] C. Kolbitsch, E. Kirda, and C. Kruegel, "The power of procrastination: Detection and mitigation of execution-stalling malicious code," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 285–296.

[134] G. Kopf and B. Brehm, "Phrack magazine: Secure function evaluation vs. deniability in OTR and similar protocols," Apr. 2012. [Online]. Available: http://phrack.org/issues/68/14.html

[135] A. Kouzemchenko, "Examining and bypassing the IE8 XSS filter," 2009, http://www.slideshare.net/kuza55/examining-the-ie8-xss-filter.

[136] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology – CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Santa Barbara, CA, USA: Springer, Berlin, Germany, Aug. 15–19, 2010, pp. 631–648.

[137] B. Krebs, "The scrap value of a hacked PC, revisited — krebs on security," 2012. [Online]. Available: http://krebsonsecurity.com/2012/10/the-scrap-value-of-a-hacked-pc-revisited/

[138] K.-H. Krempels, C. Terwelp, S. Wüller, T. Frosch, and S. Gökay, "Communication reduced interaction protocol between customer, charging station, and charging station management system," in *SMARTGREENS 2014*, Barcelona, Spain, 2014.

[139] LEMnet, "Map of charging stations," http://www.lemnet.org/LEMnet_Map.asp, 2013.

[140] J. Liu, M. Au, W. Susilo, and J. Zhou, "Enhancing location privacy for electric vehicles (at the right time)," in *ESORICS*, 2012.

[141] C. Y. Ma, D. K. Yau, N. K. Yip, and N. S. Rao, "Privacy vulnerability of published anonymous mobility traces," in *MobiCom '10*, 2010.

[142] Malware Domain List, "Mdl complete database in csv format," Nov. 2010, http://www.malwaredomainlist.com/mdlcsv.php.

[143] M. Mannan and P. C. van Oorschot, "A protocol for secure public instant messaging," in *Financial Cryptography and Data Security*, 2006.

[144] M. Marlinspike, "Internet explorer SSL vulnerability," May 2002. [Online]. Available: http://www.thoughtcrime.org/ie-ssl-chain.txt

[145] ——, "More tricks for defeating ssl in practice," Black Hat USA, 2009.

[146] ——, "Convergence | beta," 2011. [Online]. Available: http://convergence.io/details.html#

[147] ——, "sslstrip," 2011. [Online]. Available: http://www.thoughtcrime.org/software/sslstrip/

[148] ——, "chapcrack," 2012. [Online]. Available: https://github.com/moxie0/chapcrack

[149] M. Marlinspike and T. Perinn, "Trust assertions for certificate keys draft-perrin-tls-tack-02.txt," 2013. [Online]. Available: http://tack.io/draft.html

[150] J. Marquardt, *Das Privatleben der Römer*, 2nd ed. Leipzig: S. Hirzel, 1896.

[151] M. Martin and M. S. Lam, "Automatic generation of XSS and SQL injection attacks with Goal-Directed model checking," in *USENIX Security Symposium*, 2008.

[152] L. Masinter, "RFC 2397 - the "data" URL scheme," Aug. 1998.

[153] J. Mason, S. Small, F. Monrose, and G. MacManus, "English Shellcode," in *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[154] S. Meiklejohn, K. Mowery, S. Checkoway, and H. Shacham, "The phantom tollbooth: Privacy-preserving electronic toll collection in the presence of driver collusion," in *20th USENIX Security Symposium*, 2011.

[155] H. Meutzner, V. H. Nguyen, T. Holz, and D. Kolossa, "Using automatic speech recognition for attacking acoustic CAPTCHAs: The trade-off between usability and security," in *Annual Computer Security Applications Conference (ACSAC)*, 2014.

[156] C. Meyer and J. Schwenk, "Sok: Lessons learned from ssl/tls attacks," in *WISA*, 2013, pp. 189–209.

[157] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting ssl/tls implementations: New bleichenbacher side channels and attacks," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[158] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay, "Caja - safe active content in sanitized javascript," 2007, http://code.google.com/p/google-caja/.

[159] M. Mimoso, "Vupen cashes in four times at pwn2own," 2014. [Online]. Available: http://threatpost.com/vupen-cashes-in-four-times-at-pwn2own/104754

[160] V. Mladenov, C. Mainka, F. Feldmann, J. Krautwald, and J. Schwenk, "Your software at my service," in *ACM CCSW*, 2014.

[161] S. Mohammed, L. Chamarette, J. Fiaidhi, and S. Osborn, "A Safe RSS Approach for Securely Sharing Mobile SVG Biomedical Images for Web 2.0," in *12th IEEE International Conference on Computational Science and Engineering*, 2009.

[162] S. Mohammed, J. Fiaidhi, H. Ghenniwa, and M. Hahn, "Developing a Secure Web Service Architecture for SVG Image Delivery," *Journal of Computer Science*, vol. 2, no. 2, pp. 171–179, 2006.

[163] S. M. A. Mohammed and J. A. W. Fiadhi, "Developing Secure Transcoding Intermediary for SVG Medical Images within Peer-to-Peer Ubiquitous Environment," in *CNSR '05 Proceedings of the 3rd Annual Communication Networks and Services Research Conference*, 2005.

[164] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster Protocol — Version 2," http://www.abditum.com/mixmaster-spec.txt, 2003.

[165] MooTools, "Slickspeed speed/validity selectors test for frameworks," 2010, http://mootools.net/slickspeed/.

[166] V. Moscaritolo, G. Belvin, and P. Zimmermann, "Silent circle instant messaging protocol protocol specification," 2012.

[167] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy, "SpyProxy: execution-based detection of malicious web content," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Boston, MA: USENIX Association, 2007, pp. 1–16.

[168] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, "A crawler-based study of spyware on the web," in *NDSS*, 2006.

[169] Mozilla, "document.URL - MDC," 2010, https://developer.mozilla.org/en/document.URL.

[170] ——, "defineProperties - MDC," 2011, https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperties.

[171] ——, "defineProperty - MDC," 2011, https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineProperty.

[172] ——, "Gecko - MDC," 2011, https://developer.mozilla.org/en/Gecko.

[173] ——, "Gecko-Specific DOM events - MDC," 2011, https://developer.mozilla.org/en/Gecko-Specific_DOM_Events.

[174] ——, "String - MDC," 2011, https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/String#Methods_2.

[175] ——, "window.location - MDC," 2011, https://developer.mozilla.org/en/window.location.

[176] Y. Nadji, P. Saxena, and D. Song, "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense," in *Symposium on Network and Distributed System Security (NDSS)*, 2009.

[177] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *STOC*, 1989, pp. 33–43.

[178] R. Naraine, "Drive-by downloads. the web under siege - securelist," Apr. 2009, http://www.securelist.com/en/analysis?pubid=204792056.

[179] ——, "Pwn2own 2012: Google chrome browser sandbox first to fall," 2012. [Online]. Available: http://www.zdnet.com/blog/security/pwn2own-2012-google-chrome-browser-sandbox-first-to-fall/10588

[180] E. V. Nava, "ACS - active content signatures," *PST_WEBZINE_0X04*, no. 4, Dec. 2006.

[181] J. Nazario, "PhoneyC: a virtual client honeypot," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, 2009.

[182] L. Nguyen, "Accumulators from bilinear pairings and applications," in *CT-RSA*, 2005, pp. 275–292.

[183] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-Version Antivirus in the Network Cloud," in *USENIX Security Symposium*, 2008.

[184] Open WhisperSystems, "TextSecure, now with 10 million more users," Dec. 2013. [Online]. Available: https://whispersystems.org/blog/cyanogen-integration/

[185] Open WhisperSystems, "The new TextSecure: Privacy beyond SMS," Feb. 2014. [Online]. Available: https://whispersystems.org/blog/the-new-textsecure/

[186] D. Oswald, B. Richter, and C. Paar, "Side-channel attacks on the yubikey 2 one-time password generator," in *Research in Attacks, Intrusions, and Defenses*, ser. Lecture Notes in Computer Science, S. J. Stolfo, A. Stavrou, and C. V. Wright, Eds. Springer Berlin Heidelberg, Jan. 2013, no. 8145, pp. 204–222.

[187] OWASP, "Enterprise security API," Mar. 2011, http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.

[188] C. Paar, K. Schramm, A. Weimerskirch, and W. Burleson, "Securing green cars: IT security in next-generation electric vehicle systems," in *Annual Meeting and Exposition of the Intelligent Transportation Society of America*, 2009.

[189] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the TLS record protocol," in *Advances in Cryptology – ASIACRYPT 2011*, ser. Lecture Notes in Computer Science, D. H. Lee and X. Wang, Eds., vol. 7073. Seoul, South Korea: Springer, Berlin, Germany, Dec. 4–8, 2011, pp. 372–389.

[190] S. Patnaik, "htmLawed," http://www.bioinformatics.org/phplabware/internal_utilities/htmLawed/.

[191] T. Perinn, "Axolotl ratchet," Jun. 2014. [Online]. Available: https://github.com/trevp/axolotl

[192] P. H. Phung and L. Desmet, "A two-tier sandbox architecture for untrusted JavaScript," in *Proceedings of the Workshop on JavaScript Tools*, ser. JSTools '12. New York, NY, USA: ACM, 2012, pp. 1–10.

[193] P. H. Phung, D. Sands, and A. Chudnov, "Lightweight Self-Protecting javascript," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, vol. March 2009, 2009.

[194] T. Pietraszek and C. V. Berghe, "Defending Against Injection Attacks Through Context-Sensitive String Evaluation," in *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[195] Pike Research, "Electric vehicle market forecasts," http://www.pikeresearch.com/research/electric-vehicle-market-forecasts, 2013.

[196] PlugShare, "FAQ," http://www.plugshare.com/tpl/faq_popup.html, 2013.

[197] R. A. Popa, H. Balakrishnan, and A. Blumberg, "VPriv: protecting privacy in location-based vehicular services," in *USENIX Security Symposium*, 2009.

[198] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li, "Privacy and accountability for location-based aggregate statistics," in *ACM CCS*, 2011.

[199] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iFRAMEs point to us," in *USENIX Security Symposium*, 2008.

[200] R. Shokri, G. Theodorakopoulos, J. Le Boudec, and J. Hubaux, "Quantifying location privacy," in *2011 IEEE Symposium on Security and Privacy (SP)*, May 2011.

[201] M. D. Raimondo, R. Gennaro, and H. Krawczyk, "Secure off-the-record messaging," in *WPES*, 2005, pp. 81–89.

[202] B. Ramsdell and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification," RFC 5751 (Proposed Standard), Internet Engineering Task Force, Jan. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5751.txt

[203] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks," in *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[204] P. Riikonen, "Secure internet live conferencing protocol specification DRAFT," 2007. [Online]. Available: http://tools.ietf.org/id/draft-riikonen-silc-spec-09.txt

[205] K. Sampigethaya, M. Li, L. Huang, and R. Poovendran, "AMOEBA: robust location privacy scheme for VANET," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 8, pp. 1569 –1589, Oct. 2007.

[206] F. Schuster, T. Tendyck, J. Pewny, A. Maaß, M. Steegmanns, M. Contag, and T. Holz, "Evaluating the effectiveness of current anti-ROP defenses," in *Research in Attacks, Intrusions and Defenses*, ser. Lecture Notes in Computer Science, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Springer International Publishing, Jan. 2014, no. 8688, pp. 88–108.

[207] G. Schwenk, A. Bikadorov, T. Krueger, and K. Rieck, "Autonomous learning for detection of JavaScript attacks: Vision or reality?" in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, ser. AISec '12. New York, NY, USA: ACM, 2012, pp. 93–104.

[208] V. Shoup, "A proposal for an iso standard for public key encryption," Cryptology ePrint Archive, Report 2001/112, 2001, http://eprint.iacr.org/.

[209] J. Somorovsky, "On the insecurity of XML-security," Ph.D. dissertation, Ruhr-Universität Bochum, Bochum, 2013.

[210] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On breaking SAML: Be whoever you want to be," in *USENIX Security Symposium*, 2012.

[211] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," *Mach. Learn.*, vol. 81, no. 2, 2010.

[212] S. Stamm, "Public key pinning released in firefox," Sep. 2014. [Online]. Available: https://blog.mozilla.org/security/2014/09/02/public-key-pinning/

[213] M. Stegelmann and D. Kesdogan, "Design and evaluation of a privacy-preserving architecture for vehicle-to-grid interaction," in *EuroPKI*, 2012.

[214] D. Strobel, B. Driessen, T. Kasper, G. Leander, D. Oswald, F. Schellenberg, and C. Paar, "Fuming acid and cryptanalysis: Handy tools for overcoming a digital locking and access control system," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds. Springer Berlin Heidelberg, Jan. 2013, no. 8042, pp. 147–164.

[215] surespot, "encrypted messenger." [Online]. Available: https://surespot.me/

[216] S. Thomas, "DecryptoCat," 2013. [Online]. Available: http://tobtu.com/decryptocat-old.php

[217] Threema, "Seriously secure mobile messaging." [Online]. Available: https://threema.ch/de/

[218] S. Uellenbeck, M. Dürmuth, C. Wolf, and T. Holz, "Quantifying the security of graphical passwords: The case of android unlock patterns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, p. 161–172.

[219] Unkown, "FiSH – secure communications with internet relay chat," 2007. [Online]. Available: http://ultrx.net/doc/fish/

[220] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," in *Symposium on Network and Distributed System Security (NDSS)*, 2007.

[221] W3C, "3.2 elements — HTML5," http://dev.w3.org/html5/spec/elements.html#embedding-custom-non-visible-data.

[222] ——, "Client-side scripting techniques for WCAG 2.0," 2004, http://www.w3.org/TR/2004/WD-WCAG20-SCRIPT-TECHS-20041119/.

[223] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The Multi-Principal OS Construction of the Gazelle Web Browser," in *USENIX Security Symposium*, 2009.

[224] ——, "The Multi-Principal OS Construction of the Gazelle Web Browser," in *USENIX Security Symposium*, 2009.

[225] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities," in *Network and Distributed System Security Symposium (NDSS)*, 2006.

[226] G. Wassermann and Z. Su, "Static detection of Cross-Site scripting vulnerabilities," in *International Conference on Software Engineering (ICSE)*, 2008.

[227] WebKit, "Sunspider javascript benchmark," 2011, http://www2.webkit.org/perf/sunspider-0.9.1/sunspider-0.9.1/driver.html.

[228] C. Willems, T. Holz, and F. Freiling, "CWSandbox: Towards Automated Dynamic Binary Analysis," *IEEE Security and Privacy*, vol. 5, no. 2, 2007.

[229] C. Willems, F. C. Freiling, and T. Holz, "Using memory management to detect and extract illegitimate code for malware analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 179–188.

[230] C. Willems, R. Hund, A. Fobian, D. Felsch, T. Holz, and A. Vasudevan, "Down to the bare metal: Using processor features for binary analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 189–198.

[231] M. Winandy, "Security and trust architectures for protecting sensitive data on commodity computing platforms," Ph.D. dissertation, Ruhr-Universität Bochum, Bochum, 2012.

[232] M. E. Winstead, "To those of you looking at photos i took with my husband years ago in the privacy of our home, hope you feel great about yourselves." Aug. 2014. [Online]. Available: https://twitter.com/M_E_Winstead/status/506197725285998592

[233] Xihui Chen, G. Lenzini, S. Mauw, and J. Pang, "A group signature based electronic toll pricing system," in *ARES*, 2012.

[234] F. Yamaguchi, M. Lottmann, and K. Rieck, "Generalized vulnerability extrapolation using abstract syntax trees." in *Proc. of the 28th Annual Computer Security Applications Conference (ACSAC)*, 2012.

[235] E. Z. Yang, "HTML Purifier," http://htmlpurifier.org/, Mar. 2011. [Online]. Available: http://htmlpurifier.org/

[236] Zero Day Initiative, "Google chrome directory traversal sandbox escape vulnerability," 2014. [Online]. Available: http://www.zerodayinitiative.com/advisories/ZDI-14-089/

[237] Zhendong Ma, "Location privacy in vehicular communication systems: a measurement approach," Ph.D. dissertation, University of Ulm, Ulm, 2011.

Curriculum Vitae

Contact Details

Electronic Mail **tilman dot frosch at rub dot de**.

Education

2012–2014 **Doctoral Studies**, *Chair for Systems Security, Department of Electrical Engineering and Information Technology, Ruhr-Universität Bochum.*

2009–2011 **Master of Science IT Security**, *Ruhr-Universität Bochum.*

2006–2009 **Bachelor of Science Applied Computer Science**, *Ruhr-Universität Bochum.*

List of Publications

Peer Reviewed Publications

How Secure is TextSecure?
Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz
under review

Communication Reduced Interaction Protocol Between Customer, Charging Station, and Charging Station Management System
Karl-Heinz Krempels, Christoph Terwelp, Stefan Wüller, Tilman Frosch, and Sevket Gökay
3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS), 2014

mXSS Attacks: Attacking well-secured Web-Applications using innerHTML Mutations
Mario Heiderich, Jörg Schwenk, Tilman Frosch, Jonas Magazinius, and Edward Z. Young
20th ACM Conference on Computer and Communications Security (CCS), 2013

Crouching Tiger – Hidden Payload: Security Risks of Scalable Vector Graphics
Mario Heiderich, Tilman Frosch, Meiko Jensen, and Thorsten Holz
18th ACM Conference on Computer and Communication Security (CCS), 2011

The Bug that made me President
Mario Heiderich, Tilman Frosch, Marcus Niemietz, and Jörg Schwenk
International Conference on E-Voting and Identity (VoteID), 2011

IceShield**: Detection and Mitigation of Malicious Websites with a Frozen DOM**
Mario Heiderich, Tilman Frosch, and Thorsten Holz
14th International Symposium on Recent Advances in Intrusion Detection (RAID), 2011

Other Publications

Ich weiss, wo du letzten Sommer geladen hast – RUB-Forscher entwickeln datenschutzfreundliche Lösungen zum Laden von E-Autos
Tilman Frosch
RUBIN, Ruhr-Universität Bochum, 2014

I know where you charged last summer – RUB researchers develop privacy-enhancing solutions for charging electric cars
Tilman Frosch
RUBIN International Edition, Ruhr-Universität Bochum, 2014

Improving Location Privacy for the Electric Vehicle Masses
Tilman Frosch, Sven Schäge, Martin Goll, and Thorsten Holz
HGI Technical Report, HGI-TR 2013-001, 2013

Predentifier: Detecting Botnet C&C Domains from Passive DNS Data
Tilman Frosch, Marc Kührer, and Thorsten Holz
Bookchapter in Advances in IT Early Warning, Fraunhofer Verlag. 2013

List of Conference and Workshop Participation

- SPRING 9 – GI SIDAR Graduierten-Workshop über Reaktive Sicherheit, 2014 (invited)
- 2. Stralsunder Sicherheitskonferenz, 2013 (invited)
- 18th ACM Conference on Computer and Communication Security (CCS), 2011
- 14th International Symposium on Recent Advances in Intrusion Detection (RAID), 2011
- GI Wireless Security, 2009
- Various non-academic conferences in the years 2000 to 2014