universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

„Swimming in Murky Waters: Exploring Protocols, Security Features And Practices On The Internet“

verfasst von / submitted by

Olamide Omolola

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl

Acknowledgements

My most profound appreciation goes to God, the author of everything and the One who stood by me and comforted me during the entirety of my doctorate journey. I cannot begin to describe everything He did for me, but one thing is sure, I could not have made it this far without Him.

Further, I am grateful to my current Advisor Prof. Edgar Weippl for his support at the end of this thesis. I am also thankful to my first advisor, Peter Lipp, who believed in me and helped me steer my initial research direction. Thank you for granting me the autonomy to explore and design my thesis. In addition, I want to thank my first colleagues i.e., Georg Wagner, Sebastian Ramacher, and Stefan More. Thank you for helping me settle into the position at Graz and the fruitful collaboration that led to the success of LIGHTest.

Also, I would like to say a big thank you to Prof. Kévin Huguenin, Prof Alan Mislove, Ass. Prof. Tobias Fiebig, Assoc. Prof Bertil Chapuis for collaborating with me and helping me shape my research direction during this thesis. I want to specially acknowledge Ass. Prof. Tobias Fiebig for going beyond and above the call of duty to help me by not just collaborating but taking an interest in my well-being as well. Thank you, Tobias!

Sometimes we are fortunate, and we meet friends that become family. I am lucky that I met Sem Shabani, Lisette Shabani, Stanley, Deji Ademola, Henintsoa, Njara, and Benedict Oguah. Thank you for your prayers, support, and always being present to listen. You have all shaped me in different ways, which has helped me become who I am today.

The past year has been challenging with a lot of uncertainty. That period was one of the toughest in my life. However, there was a reason I kept standing up every day, and that was to work with Rob Prokop, Mike Laplume, and Kevin Galasinski. Thank you for teaching me how to build but far more importantly, for showing me what it means to have character. Thank you! I cannot forget to mention Mary Prokop as well. Thank you for your wise words.

Additionally, I am deeply grateful to Rev Niyi and Debbie Dahunsi. You took me under your wing, taught me, and always cared for me. Thank you for the spiritual leadership and training you gave me. You prepared me ahead of time.

Space would limit the effort to acknowledge everyone who helped me, prayed for me, stood by me in one way or the other. But even if I could not mention your name here, it does not diminish your impact on my life and how grateful I am for that.

Finally, I am incredibly grateful to my parents, Adekunle and Mary Omolola. You have always been there, and I do not take that for granted. I cannot thank my parents without thanking my brother Adewale Omolola. Bro Wale, you set the pace, and you are a blessing to me. Thank you

Abstract

Today, the Internet is essential for many aspects of life– academia, relationships, business, governmental administration, and much more. Many of these activities require secure communication protocols. However, as security was not a priority when the Internet was initially designed, several core protocols used today suffer from security challenges. Still, not only the old foundations of the Internet are affected by this. Also, newly designed protocols regularly suffer from security issues. Take Transport Layer Security (TLS), which arrived in its 5th iteration with TLS 1.3. The predecessors, SSLv2, SSLv3, TLS 1.0, and TLS 1.2, are now considered obsolete due to inherent security issues in these protocols.

In this thesis, we investigate current protocols and discover that some lack privacy guarantees. Primarily, we focus on a major component of the web ecosystem–(1). HTTP (HyperText Transfer Protocol) which encompasses TLS (including its certificate transparency extension) and Sub-resource Integrity.

We enhance these non-privacy preserving protocols to become privacy-preserving. Methodologically, we also utilize large-scale Internet measurements to analyze and audit the practical implementation of these protocols on the Internet. We present and analyze how these protocols changed over time and utilize measurements to gain a deeper understanding of how the *technical* specification process of protocols influences their adoption and security.

Overall, our results document–and contribute to–the constant improvement of the TLS ecosystem and how it moves to user-centric design.

Kurzfassung

Das Internet ist heute für viele Bereiche des Lebens unverzichtbar - für die Wissenschaft, für Beziehungen, für die Wirtschaft, für die staatliche Verwaltung und für vieles mehr. Für viele dieser Aktivitäten sind sichere Kommunikationsprotokolle erforderlich. Da jedoch bei der Entwicklung des Internets die Sicherheit nicht im Vordergrund stand, leiden mehrere der heute verwendeten Kernprotokolle unter Sicherheitsmängeln. Doch nicht nur die alten Grundlagen des Internets sind davon betroffen. Auch neu entwickelte Protokolle haben regelmäßig mit Sicherheitsproblemen zu kämpfen. Nehmen wir die Transport Layer Security (TLS), die mit TLS 1.3 in ihrer fünften Iteration angekommen ist, als Beispiel. Die Vorgänger, SSLv2, SSLv3, TLS 1.0 und TLS 1.2, gelten aufgrund inhärenter Sicherheitsprobleme in diesen Protokollen inzwischen als veraltet.

In dieser Arbeit untersuchen wir die aktuellen Protokolle und stellen fest, dass einige keine Datenschutzgarantien bieten. In erster Linie konzentrieren wir uns auf eine wichtige Komponente des Web-Ökosystems -(1). HTTP (HyperText Transfer Protocol), das TLS (einschließlich seiner Zertifikattransparenz-Erweiterung) und Sub-Ressourcen-Integrität umfasst.

Wir verbessern diese Protokolle, die die Privatsphäre nicht schützen, so dass sie die Privatsphäre schützen. Methodisch nutzen wir auch groß angelegte Internet-Messungen, um die praktische Umsetzung dieser Protokolle im Internet zu analysieren und zu überprüfen. Wir stellen dar und analysieren, wie sich diese Protokolle im Laufe der Zeit verändert haben, und nutzen Messungen, um ein tieferes Verständnis dafür zu gewinnen, wie der technische Spezifikationsprozess von Protokollen ihre Annahme und Sicherheit beeinflusst.

Insgesamt dokumentieren unsere Ergebnisse - und tragen dazu bei - die ständige Verbesserung des TLS-Ökosystems und seine Entwicklung hin zu einem benutzerzentrierten Design.

Publications

1. Included Publications

- Daniel Kales, Olamide Omolola and Sebastian Ramacher. "Revisiting User Privacy for Certificate Transparency". In: *4th IEEE European Symposium on Security and Privacy*. 2019

  Co-first author, initiated this work and a major contributor to the solution

- Bertil Chapuis, Olamide Omolola, Mauro Cherubini, Mathias Humbert and Kévin Huguenin. "An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources". In: *WWW*. ACM / IW3C2, 2020, pp. 34–45

  Second author and major contributor to this work. Olamide led the web developer study section

- Paul Plessing and Olamide Omolola. "Revisiting Privacy-aware Blockchain Public Key Infrastructure". In: *ICISSP*. SCITEPRESS, 2020, pp. 415–423

  Co-first author, initiated this work and a major contributor to the solution

- Olamide Omolola, Richard Roberts, Md. Ishtiaq Ashiq, Taejoong Chung, Dave Levin and Alan Mislove. "Measurement and Analysis of Automated Certificate Reissuance". In: Lecture Notes in Computer Science 12671 (2021), pp. 161–174

  First author, initiated this work and a major contributor to the solution

2. Other Publications

- Andreas Abraham, Felix Hörandner, Olamide Omolola and Sebastian Ramacher. "Privacy-Preserving eID Derivation for Self-Sovereign Identity Systems". In: *ICICS*. vol. 11999. Lecture Notes in Computer Science. Springer, 2019, pp. 307–323

  Author and contributor to the solution

- Waqar Aqeel, Zachary Hanif, James Larisch, Olamide Omolola, Taejoong Chung, Dave Levin, Bruce Maggs, Alan Mislove, Bryan Parno and Christo Wilson. "Assertion-Carrying Certificates". English. In: *Workshop on Foundations of Computer Security 2020*. June 2020

  Co-first author and a major contributor to the solution

- Olamide Omolola, Stefan Josef More, Edona Fasllija, Georg Wagner and Lukas Alber. "Policy-based Access Control for the IoT and Smart Cities". In: *Open Identity Summit*. Mar. 2019

  First author, initiated this work and a major contributor to the solution

- Stephanie Weinhardt and Olamide Omolola. "Usability of Policy Authoring tools: A layered approach". In: *International Conference on Information Systems Security and Privacy*. Feb. 2019

  First author, initiated this work and a major contributor to the solution

- Georg Wagner, Olamide Omolola and Stefan Josef More. "Harmonizing Delegation Data Formats". In: *Lecture Notes in Informatics*. Vol. 2017. Germany: Gesellschaft für Informatik, Oct. 2017. ISBN: 978-3-88579-671-8. DOI: 20.500.12116/3578

  Author and contributor to the solution

Contents

1. Introduction

As of 2016, the United Nations reported that more than 3 billion people use the Internet[1]. Many of these users carry out private or sensitive activities on the Internet. Therefore they require secure services and protocols to enable their activities.

However, security was not a priority when the Internet and many of its vital services such as emails [51] and protocols such as HyperText Transfer Protocol (HTTP) [34], Border Gateway Protocol (BGP) [47], Domain Name System (DNS) [50], and Teletype Network (Telnet) [57] were designed. Communication using email, DNS, HTTP, and Telnet are unencrypted and vulnerable to eavesdropping and man-in-the-middle attacks. BGP is at the same time vulnerable to BGP hijacking[2].

Nowadays, the industry and academia are revisiting insecure protocols, improving them, and creating features to enhance their security guarantees. Some security extensions are proposed for protocols that cannot be changed without bringing down the internet while others are abandoned, and replacements are designed from scratch.

Services like emails are now secured with a myriad of protocols [39, 4, 59], HTTP is now secured with TLS [59], DNS Security Extensions (DNSSEC) [6] ensures the integrity of DNS responses. Telnet is no longer recommended for use and was replaced by Secure Shell (SSH) [69] while BGP is undergoing security evaluation, and new versions are in use [46].

1.1. Problem Statement

The following section outlines the research problem and clarifies its domain and context. It also defines the core research questions to be answered in the course of the research.

1.1.1. Problem Domain and Context

In recent years, many activities that were often carried out on paper or in person have evolved to being internet-based, thereby creating a need for secure communication than ever before. Furthermore, the exploitation of inherent weaknesses in the existing communication protocols is a major source of concern for the stakeholders on the internet. In this thesis, we narrow down the problem space and focus on analyzing security protocols and features and proposing possible improvements. Our work spans the trust management spectrum between collaborating entities on the internet with a specific focus on the HTTP ecosystem.

[1] https://news.un.org/en/story/2016/09/539112-nearly-47-cent-global-population-now-online-un-report

[2] BGP hijacking is when the Internet traffic is maliciously rerouted

HTTP is a protocol for transmitting hypermedia documents such as HTML and more. Within a HTML page, more documents such as scripts, pictures and files can be linked. This possibility comes at a cost since there is a likelihood that the linked document has been compromised. SRI (Subresource integrity) is a new HTTP security feature that allows users to verify that the linked document has not been modified from the original that was included.

Furthermore, within the HTTP ecosystem, TLS enables the authentication of communicating parties using certificates signed by designated Certificate Authorities (CAs). The certificate is a signed attestation to the identity of an entity from a CA. CAs are usually third-parties that the communicating parties trust.

In recent times, some CAs were compromised and behaved maliciously by issuing certificates for different entities without the request from those entities. These misissued certificates were used to steal private data or decrypt information flow thereby causing a loss of privacy. As a result of the unrestricted ability for CAs to issue certificates, countermeasures such as Certificate Transparency (CT), Certificate Authority Authorization (CAA) were created.

1.1.2. Research Questions

This thesis aims to answer the following questions based on the problem domain and context in the Subsection 1.1.1: We aim to answer the following research questions in this Ph.D.

1. **RQ1** How does the technical specification of protocols and complexity of usage influence their adoption?

2. **RQ2** Do protocol enhancements fulfill their security guarantees in practice without causing other vulnerabilities?

1.2. Contributions

We provide short descriptions of our contributions in this section.

We enhance the security guarantees of Certificate Transparency (CT) in the first contribution of this thesis [37], having discovered that though CT prevents mis-issuance of certificates in the ecosystem, it could lead to leakage of private data. Our contribution therefore protects users against tracking activities in CT by preventing eavesdropping of the online activities of the user

Our next contribution to the field of TLS was to understand how automated re-issuance of certificates has affected the TLS ecosystem [55]. Previously, certificates were expensive costing $50 and above. Furthermore, services such as revocation and re-issuance came at a premium. Although the TLS protocol provided security guarantees, it was not well adopted as a result of the cost. The cost of certificates affected reissues and revocations and caused several bugs relating to certificates go unchecked for a long time. As a result, the security guarantee of the TLS ecosystem was weak since it wasn't well adopted. The

advent of Let's Encrypt changed this narrative with free valid certificates and automatic certificate management. In this work, we show that the security guarantees of the TLS ecosystem have improved because of better adoption and better certificate management techniques.

In the third contribution of this thesis, we study a privacy enhancing PKI implementation and discovered that it does not fulfil its privacy guarantees. We redesigned the protocol to improve its privacy guarantees and implemented a proof of concept [56].

Another contribution is the analysis of Sub-resource Integrity (SRI) [14]. SRI is a recent W3C recommendation that creates a digest of sub-resources included in a webpage. SRI can protect against the corruption of sub-resources. We perform the first empirical study analyzing SRI and its usage. We also find in a survey that the technical specification of the recommendation is confusing to developers and the lack of automation that prevents faster adoption.

1.2.1. Internet Measurements

Internet measurements play a role in my contributions. Internet measurements help security researchers and protocol designers understand how security tools are deployed in the wild. Many insights have been gained into this field such as the discovery of invalid certificates and ability to track devices using invalid certificates [16]; the discovery that CAs do not honor Certificate Authority Authorization (CAA) and many certificates were found to be mis-issued [61]; wide-spread mismanagement of the DNSSEC infrastructure [18]; the insecure practice of CDNs controlling the secret keys of client's certificates [13] and many more.

These insights have exposed the many weaknesses in security deployments and usage. It can be argued that a useful security feature is useless when improperly deployed or mismanaged. Therefore, Internet measurement is a critical security specialization.

1.3. Organization of Thesis

The rest of this proposal is structured as follows: Chapter 2 describes our work on certificate transparency, Chapter 3 outlines the analysis and improvement made to a blockchain PKI and Chapter 4 describes our work on subresource integrity. Chapter 5 outlines the analysis of automated certificate reissuance while Chapter 6 gives a summary and conclusion of the thesis. Note that the introduction and conclusion are concise to avoid repetition in subsequent chapters.

2. Revisiting User Privacy for Certificate Transparency

The subsequent paper has been published as follows:

Daniel Kales, Olamide Omolola and Sebastian Ramacher. "Revisiting User Privacy for Certificate Transparency". In: *4th IEEE European Symposium on Security and Privacy*. 2019

In this paper, we devise a privacy-preserving approach for auditors (browsers) to verify Signed Certificate Timestamps (SCTs). Our approach leverages Private Information Retrieval (PIR) to retrieve membership proofs.

Abstract. Public key infrastructure (PKI) based on certificate authorities is one of the cornerstones of secure communication over the internet. Certificates issued as part of this PKI provide authentication of web servers among others. Yet, the PKI ecosystem is susceptible to certificate misissuance and misuse attacks. To prevent those attacks, Certificate Transparency (CT) facilitates auditing of issued certificates and detecting certificates issued without authorization. Users that want to verify inclusion of certificates on CT log servers contact the CT server directly to retrieve inclusion proofs. This direct contact with the log server creates a privacy problem since the users' browsing activities could be recorded by the log server owner.

Lueks and Goldberg (FC 2015) suggested the use of Private Information Retrieval (PIR) in order to protect the users' privacy in the CT ecosystem. With the immense amount of certificates included on CT log servers, their approach runs into performance issues, however. Nevertheless, we build on this approach and extend it using multi-tier Merkle trees, and render it practical using multi-server PIR protocols based on distributed point functions (DPFs). Our approach leads to a scalable design suitable to handle the increasing number of certificates and is, in addition, generic allowing instantiations using secure accumulators and PIRs.

We implement and test this mechanism for privacy-preserving membership proof retrieval and show that it can be integrated without disrupting existing CT infrastructure. Most importantly, even for larger CT logs containing 2^{31}certificates, our approach using sub-accumulators can provide privacy with a performance overhead of less than 9 milliseconds in total.

Keywords: Certificate Transparency · User Privacy · Private Information Retrieval

1 Introduction

Nowadays Transport Layer Security (TLS) [Res18] is the de-factor standard for secure communication over the internet. In general, TLS enables two parties—a client and a server—to agree on a shared secret key which can then be used to encrypt payload data. During the handshake that is responsible to perform the key agreement, the client most commonly verifies the server's identity based on the server's X.509 certificate [CSF+08] issued by some trusted certificate authority (CA). However, in the standard certificate ecosystem, there is still room for misuse, as multiple certificates may be issued for the same domain name. The most prominent

examples of such incidents include CAs like Comodo[1] or DigiNotar[2] issuing certificates for, among others, subdomains of `google.com`. In the latter case of DigiNotar, these fraudulent certificates were used for man-in-the-middle attacks against users.

To this end, countermeasures like Certificate Transparency (CT) [Lau14, LLK13, DGHS16] have received a lot of attention recently. In CT, all issued TLS certificates are publicly logged. Its goal is to allow any party to audit the public log and find suspicious certificates or check the integrity of the log itself. The ultimate goal of CT is to eventually have clients refuse connections for certificates that are not included in a public log. Google began enforcing this policy for certificates issued after April 2018[3] in its browser. Also, other big browser vendors such as Apple and Mozilla are in the process of enforcing this policy.

In a logging system, web clients need to ensure that log servers do not hand out promises of certificate inclusion in the log without actually doing so. To combat misbehaving log servers, web clients act as auditors, verifying that any certificates they received are actually publicly logged. Although this is an important role, it has negative privacy implications for clients performing such an auditing role, as verifying the inclusion of a certificate reveals the browsing behavior of the client to the log server.

1.1 Overview of the Certificate Transparency Ecosystem

Certificate Transparency is a large ecosystem with many participants. First, there are so-called *submitters*, who submit certificates or precertificates[4] to the log server and receive a Signed Certificate Timestamp (SCT). An SCT is a log server's promise that the certificate it was issued for will be included in the log server after a particular time period called Maximum Merge Delay (MMD). This SCT is then either directly included in the certificate or exchanged with a web client during the TLS handshake. Submitters are usually CAs, but, in general, anyone can submit certificates to the log server.

[1] https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html
[2] https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html
[3] https://www.section.io/blog/chrome-ct-compliance/
[4] Precertificates are certificates provided to the log before the issuance of the actual certificate. They contain a special critical poison extension that renders the certificate unusable in TLS connections.

Log servers receive certificate chains and issue SCTs for them. They add those chains (together with the respective SCTs) to a data structure which allows to store elements and to later produce succinct witnesses to attest the membership of certain values within this data structure. Conceptually, such a data structure realizes what is formalized by cryptographic accumulators (see [DHS15] for an overview). Technically, it is realized using Merkle trees [Mer89].

The role of *monitors* is to watch the log servers and audit their behavior by verifying the validity and consistency of the accumulator over time. Monitors also can check for misissued certificates and alert domain owners when they detect a potentially malicious certificate in the log.

Finally, there are so-called *auditors*, who verify the SCTs' signature and check that the accompanying certificate is present in the log by requesting a witness attesting their membership in the accumulator and verifying it against the current accumulator value. They additionally can request a proof of consistency with respect to changes in the accumulator over time. An auditor is an essential part of a TLS client, but it could also be a secondary function of a monitor. Auditors built into TLS clients do not necessarily perform this inclusion checks in real time when visiting a website; usually, only the SCT signatures are verified against the public keys of trusted log servers. The Merkle-tree inclusion proofs are then retrieved asynchronously at a later time to audit if the log server is well-behaved. Chromium-based browsers already have a built-in component that validate SCTs by sending them to a Google resolver that validates inclusion proofs. A misbehaving log server, issuing SCTs for certificates that are not logged by the server, will then get reported to big browser vendors and will subsequently be removed from the list of trusted log servers.

Current State of the Certificate Transparency Ecosystem Due to the efforts of big browser vendors, especially Google, the CT ecosystem is growing rapidly. As of August 2018, several of the big CT log servers (e.g., Google Argon) have more than 250 million certificates in their log. Even the smaller log servers have more than 10 million active certificates, with a certificate issuance rate of $\approx$ 53000 new certificates per hour. Two months later (October, 2018), the certificate issuance rate jumped to $\approx$ 105000 new certificates per hour and several of the big CT log servers (e.g., Google Argon) have more than 400 million certificates in their log

servers. Cloudflare's CT statistics website[5] publishes live statistics about the current state of CT log servers.

A big factor in these numbers is the growing popularity of Let's Encrypt,[6] a free, automated certificate authority that accounts for more than 72% of all certificates in CT logs (cf. Table 1) This huge number of certificates makes the use of simple privacy-preserving techniques, such as simply downloading the full log, impossible in practice.

Root CA	Certificates	Percentage
DigiCert	64,226,041	5%
Let's Encrypt	941,016,262	72%
Sectigo	246,484,842	19%
Other	62,114,615	5%

Table 1. Number of certificates per CAs tracked as part of CT. Numbers are based on data from https://merkle.town as of April 8th, 2019.

1.2 Privacy Challenges with CT

The auditors' role in CT is essential because they verify that a log server did not issue an SCT for a certificate that is not included in the public log after the MMD. However, this vital process of auditing SCTs in CT can violate a user's privacy if the auditor is, e.g., a TLS client.

The CT auditor checks that the corresponding certificate of each valid SCT is included in the log server. This is done by requesting a membership proof for the certificate hash from the log server and verifying it against the accumulator value (the Signed Tree Hash (STH)) of the log server. The downside to this approach is that it reveals the browsing behavior of the specific auditor (which is usually a TLS client) to the log server because having a particular SCT means that the auditor visited this website. A malicious log server can choose to record this browsing behavior and sell this browsing history to interested third parties, like advertising agencies.

On the other hand, the privacy problem stated above can also weaken the integrity of the CT ecosystem, since TLS clients are discouraged to audit sites which they may not want to be associated with, e.g., sites

[5] https://merkle.town/
[6] https://letsencrypt.org/

of political, religious or sexual nature. In turn, if no-one is auditing the validity of SCTs for these certificates, they can more likely become targets for adversaries, as they could convince a malicious log server to issue an SCT for a malicious certificate and not include it into its log. If a potential victim of a man-in-the-middle attack using this malicious certificate is not likely to audit the SCT because he does not want his browsing behavior known, this attack is much more likely to succeed unnoticed.

Furthermore, the privacy problem is transferred to applications or protocols that use CT as a basis or follow the same architectural design. One such application is DECIM [YRC18], which aims to detect the compromise of endpoints in messaging scenarios. DECIM provides a key management protocol based on CT and enables users to refresh and manage keys in a transparent manner. Users of this system query keys from log servers and can thereby leak their communication partners. Thus, the authors of DECIM suggest the use of spoof queries over an anonymous channel such as Tor to hide the actual user queries from the log servers. Our proposed solutions can also be directly applied to DECIM.

1.3 Our Contribution

In this work, we tackle the privacy issues within the Certificate Transparency ecosystem. Our contributions are as follows:

- We build on top of Lueks and Goldberg's approach [LG15] for privacy-preserving retrieval of inclusion proofs from CT log servers. To achieve privacy there, clients fetch inclusion proofs using a multi-server private information retrieval (PIR) protocol. We, however, present a more scalable design for logging a huge number of certificates, which allows us to include small static partial inclusion proofs in an SCT, a server's certificate or as a TLS extension. The client can then check the inclusion based on the partial proof and by fetching the missing parts of the proof using a PIR-based approach.
- We verify the practicality of our approach by extending Google's CT log server implementation and performing experiments on realistic log server sizes. Even without using the approach of sub-trees, we report practical performance numbers and improve both runtime and communication compared to previous approaches. For our multi-tier approach, we report a client runtime overhead of less than a millisecond in total, a server runtime overhead of less than 9 milliseconds, and total communication overhead of around 7 KB for 2^{31} certificates

when using hourly sub-trees, and even below 1 milliseconds for clients and servers for sub-trees accumulating certificates per minute.

Specifically, our goal is to tackle the privacy issue without any changes to the TLS server side to ease the possibility of a fast deployment. In our approach, we split the Merkle tree containing all certificates into multiple tiers of smaller Merkle trees where the trees at the bottom contain certificates. This split can, for example, be based on a parameterizable time interval or a maximum number of certificates. The sub-trees, respectively their roots, are then combined into the larger tree containing all certificates. This separation of the certificates into smaller sub-trees then allows us to embed membership proofs concerning the sub-trees in an extension field of the SCT or as an X.509v3 extension [CSF+08] into the certificate itself. As the height of the larger tree is now considerably smaller than a single tree containing all certificates, the approach by Lueks and Goldberg [LG15] using PIR to fetch the membership proofs, becomes practical again.

We formalize our approach in more general terms using accumulators and sub-accumulators, where we consider the smaller sub-trees as accumulators and then the full tree as an accumulator of accumulators. Using this abstraction, we discuss different types of accumulators including Merkle-tree accumulators as well as RSA and bilinear pairing based ones in terms of their performance characteristics as well as their consequences on the security on the CT ecosystem.

Additionally, we use a different two-server PIR solution as an alternative to the PIR scheme used by Lueks and Goldberg. We make use of the work on distributed point functions by Gilboa and Ishai [GI14] to build an efficient two-party computationally secure PIR system and present a highly performant implementation. For this, the client needs to know the index i of the item it wants to retrieve in the database. At the moment, there no such index exists in the SCTs that are returned by the log server. Therefore, we propose to include such an index in the `CtExtensions` field of an SCT. Alternatively, this static piece of information can also be included in a TLS extension.

Finally, our approach is general and can also be applied to other systems based on CT-like architectures. In particular, it could be used to replace the spoof queries over Tor as proposed in DECIM for hiding a user's communication partners.

1.4 Related Work

Different approaches have been proposed to solve various privacy issues in the context of CT, and we discuss some of them below as well as known privacy-preserving techniques.

Tor [DMS04] and AN.ON [BFK00], two open and privacy-enhancing networks can provide the needed infrastructure to solve the privacy problem in CT. In both networks, the client requesting an inclusion proof is anonymous through a series of complex routing mechanisms. However, Tor suffers from unpredictable performance and AN.ON has limited bandwidth and has no load balancing mechanisms [WHF07].

Another suggestion allows the clients to receive the inclusion proofs using special DNS records through their DNS resolvers [Lau16]. In this case, the log server operates DNS name servers which serve authoritative answers to special queries from web clients. One of the pitfalls [Mes17] of this approach is that the browsing history is still observable since DNS requests are mostly sent in plaintext over UDP.

The draft of version 2 of the CT RFC [LLK+18] discusses the privacy issues in CT and presents three mechanisms to retrieve the Merkle inclusion proofs in a privacy-preserving way. The first mechanism involves a new TLS extension where the TLS servers send the inclusion proofs and SCTs in the process of communication with the client. The inclusion proofs and the SCTs can be updated on the fly in this case. This approach puts additional load on the server, i.e., the server has to continually update the inclusion proofs it has in storage. The second mechanism involves the Online Certificate Status Protocol (OCSP) [SMA+13]. A user contacts the OCSP service of a certificate authority to check whether a certificate was revoked, thus leaking the browsing behavior to the CA. The OCSP can also be used in this manner to deliver inclusion proofs to the client. However, OCSP does not solve the privacy problem but simply shifts the information leakage to a certificate authority. OCSP stapling [III11], which was initially designed to offload computational costs to the servers, also helps to address privacy issues, since the client no longer needs to contact the CA themselves, but verifies the time-stamped OCSP response appended by the server to the initial TLS handshake. The OCSP stapling approach also adds additional load on the server because the time-stamped OCSP response has to be continuously changed and updated. The final mechanism involves adding the inclusion proofs and the SCTs directly as an X509v3 certificate extension. This extension

can not regularly be updated, and while it provides privacy, it quickly de-synchronizes with the log servers.

Lueks and Goldberg [LG15] propose to store membership proofs in a PIR database optimized for multi-user queries. The database stores a record containing the membership proof for each certificate; thus for storing 2^ℓ certificates, the database is required to store $\ell \cdot 2^\ell$ hashes. For log servers storing a million of certificates, the performance is reasonable; however, current CT log servers contain a hundred times more certificates than assumed by Lueks and Goldberg, rendering their approach impracticable.

Eskandarian et al. [EMBB17] address another privacy issue, which is not the focus of this work. In case a misbehaving log server is identified, an auditor is required to publish the offending SCT to indicate the log server's misbehavior. Naturally, the incident together with the SCT would then be reported to browser vendors managing the list of trusted log servers. However, this again leaks the client's browsing behavior to a third party. Eskandarian et al. tackle this issue by constructing zero-knowledge proofs of exclusion, proving that an SCT has been excluded from a log whereas the verifier only learns that an entry has been excluded. Their techniques fundamentally rely on efficient proofs of knowledge of signatures together with suitable signature schemes for signing timestamps.

A recent proposal [NGR18] addresses the issues related to gossiping in Certificate Transparency. Gossiping is the sharing of information about log servers between clients. The authors propose three protocols for gossiping SCTs and Signed Tree Heads (STHs) amongst web clients. The protocols necessitate the exchange of sensitive information that can be used to aggregate network activities of different clients or track clients across different origins. The authors proposed measures to ensure that the possibility of such a privacy breach is minimal. However, the protocols and the privacy measures of this draft focus primarily on the gossiping protocols and do not address the privacy concerns that come with the fetching of inclusion proofs.

Demmler et al. [DRRT18] use a PIR based on distributed point functions (DPF) to construct a multi-server private set-intersection protocol optimized for unbalanced set sizes. In addition to a performant implementation, they also touch on deployment considerations, which we also discuss in Section 5.4.

Splinter [WYG+17], a system built on Function Secret Sharing and DPFs, provides privacy for users querying a public database. The query from a

user is split and sent to multiple servers that have a copy of the same data. Splinter cannot only retrieve data in a PIR-like fashion but also enables a user to compute functions such as MAX or TOPK over ranges of the public data without non-colluding servers learning any information about the query.

A different approach to PIR is called oblivious RAM (ORAM), where a client can read and write to a database stored on a server without the server learning about the location or content of the reads and writes. The original work of Goldreich [Gol87] has spawned an extensive line of work for different ORAM constructions and improvements, a recent example being [SvDS+18]. However, while ORAM is a more powerful primitive than PIR, it is not well suited for the scenario of CT, since the database is read-only and public, with many different clients wanting to retrieve data.

A different line of work investigates privacy-preserving key directories, which are similar to the logging infrastructure used in CT but additionally hide the contents of the key directory. Examples of such systems include CONIKS [MBB+15], EthIKS [Bon16], Catena [TD17], and the generalization of *Verifiable Key Directories* by Chase et al. [CDG18].

2 Preliminaries

In this section, we introduce cryptographic primitives and constructions that we subsequently use as building blocks. Notation-wise, let $[n] := \{1, \ldots, n\}$ for $n \in \mathbb{N}$. For an algorithm $\mathcal{A}$, we write $\mathcal{A}(\cdots; r)$ to make the random coins explicit. We say that an algorithm is efficient, if it runs in probabilistic polynomial time (PPT).

2.1 Accumulators

We rely on the formalization of accumulators by Derler et al. [DHS15]. Based on this formalization, we then state the Merkle tree, the RSA, and the bilinear accumulators within this framework. We start with the definition of a static accumulator.

Definition 1 (Static Accumulator). *A static accumulator is a tuple of efficient algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify})$ *which are defined as follows:*

$\mathsf{Gen}(1^\kappa, t)$: *This algorithm takes a security parameter* κ *and a parameter* t*. If* $t \neq \infty$*, then* t *is an upper bound on the number of elements to be accumulated. It returns a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, where* $\mathsf{sk}_\Lambda = \emptyset$ *if no trapdoor exists. We assume that the accumulator public key* pk_Λ *implicitly defines the accumulation domain* D_Λ*.*

$\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$ *and a set* $\mathcal{X}$ *to be accumulated and returns an accumulator* $\Lambda_\mathcal{X}$ *together with some auxiliary information* aux*.*

$\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x_i)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, an accumulator* $\Lambda_\mathcal{X}$*, auxiliary information* aux *and a value* x_i*. It returns* $\perp$*, if* $x_i \notin \mathcal{X}$*, and a witness* wit_{x_i} *for* x_i *otherwise.*

$\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_{x_i}, x_i)$: *This algorithm takes a public key* pk_Λ*, an accumulator* $\Lambda_\mathcal{X}$*, a witness* wit_{x_i} *and a value* x_i*. It returns* 1 *if* wit_{x_i} *is a witness for* $x_i \in \mathcal{X}$ *and* 0 *otherwise.*

We now define a dynamic accumulator, but adapt it to our use-case. We only allow additions of elements to the accumulator.

Definition 2 (Dynamic Accumulator). *A dynamic accumulator is a static accumulator with an additional tuple of efficient algorithms* $(\mathsf{Add}, \mathsf{WitUpdate})$ *which are defined as follows:*

$\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This deterministic algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, an accumulator* $\Lambda_\mathcal{X}$*, auxiliary information* aux*, as well as an element* x *to be added. If* $x \in \mathcal{X}$*, it returns* $\perp$*. Otherwise, it returns the updated accumulator* $\Lambda_{\mathcal{X}'}$ *with* $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$ *and updated auxiliary information* aux'*.*

$\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, a witness* wit_{x_i} *to be updated, auxiliary information* aux *and an* x *which was added to the accumulator. It returns an updated witness* wit'_{x_i} *on success and* $\perp$ *otherwise.*

Note that the formalization of accumulators by Derler et al. gives access to a trapdoor if it exists. Giving those algorithms access to the trapdoor can often be beneficial performance-wise, but requires additional trust assumptions. We will discuss the consequences for instantiating our schemes in Section 3.3.

Finally, we recall the notion of collision freeness:

Definition 3 (Collision Freeness). *A cryptographic accumulator is* collision-free*, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\varepsilon(\cdot)$ such that:*

$$\Pr\left[\begin{array}{l}(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda) \leftarrow \mathsf{Gen}(1^\kappa, t),\\ (\mathsf{wit}^\star_{x_i}, x_i^\star, \mathcal{X}^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_\Lambda)\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda^\star, \mathsf{wit}^\star_{x_i}, x_i^\star) = 1\\ \wedge\ x_i^\star \notin \mathcal{X}^\star\end{array}\right] \leq \varepsilon(\kappa),$$

where $\Lambda^\star \leftarrow \mathsf{Eval}_{r^\star}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X}^\star)$ and the adversary gets access to the oracles

$$\mathcal{O} = \{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot), \mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot)\}$$

and, if the accumulator is dynamic, additionally to

$$\{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot), \mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot)\}.$$

2.2 Merkle-tree Accumulator

In Scheme 1, we cast the Merkle-tree accumulator in the framework of [DHS15]. Correctness can easily be verified. We restate the well-known fact that this accumulator is collision free.

Lemma 1. *If $\{H_k\}_{k\in\mathsf{K}^\kappa}$ is a family of collision resistant hash functions, the static accumulator in Scheme 1 is collision free.*

In the current CT log server implementation, H_k is instantiated using SHA-256. Also, in practical instantiation, the requirement that Eval only works on sets of a size that is a power of 2 can be dropped. It is always possible to repeat the last element until the tree is of the correct size.

2.3 Dynamic Public-Key Accumulators

Besides hash based-based constructions, major lines of work investigated accumulators in the hidden order groups, i.e. RSA-based, and the known order groups, i.e. discrete logarithm-based, setting. The first collision-free RSA-based accumulator is due to Barić and Pfitzmann [BP97]. The accumulator in this construction consists of a generator raised to the product of all elements of the set. Then witnesses essentially consist of the same value skipping the respective elements in the product. Thereby, the witness can easily be verified by raising the power of the witness to

$\underline{\mathsf{Gen}}(1^\kappa, t)$: Fix a family of hash functions $\{H_k\}_{k\in\mathsf{K}^\kappa}$ with $H_k : \{0,1\}^* \to \{0,1\}^\kappa \ \forall\ k \in \mathsf{K}^\kappa$. Choose $k \xleftarrow{R} \mathsf{K}^\kappa$ and return $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda) \leftarrow (\emptyset, H_k)$.

$\underline{\mathsf{Eval}}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})$: Parse pk_Λ as H_k and $\mathcal{X}$ as $(x_0, \ldots, x_{n-1})$. If $\nexists\ k \in \mathbb{N}$ so that $n = 2^k$ return $\perp$. Otherwise, let $\ell_{u,v}$ refer to the u-th leaf (the leftmost leaf is indexed by 0) in the v-th layer (the root is indexed by 0) of a perfect binary tree. Return $\Lambda_\mathcal{X} \leftarrow \ell_{0,0}$ and $\mathsf{aux} \leftarrow ((\ell_{u,v})_{u\in[n/2^{k-v}]})_{v\in[k]}$, where

$$\ell_{u,v} \leftarrow \begin{cases} H_k(\ell_{2u,v+1} || \ell_{2u+1,v+1}) & \text{if } v < k, \text{ and} \\ H_k(x_i) & \text{if } v = k. \end{cases}$$

$\underline{\mathsf{WitCreate}}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x_i)$: Parse aux as $((\ell_{u,v})_{u\in[n/2^{k-v}]})_{v\in[k]}$ and return wit_{x_i} where

$$\mathsf{wit}_{x_i} \leftarrow (\ell_{\lfloor i/2^v \rfloor + \eta, k-v})_{0\le v\le k}, \text{ where } \eta = \begin{cases} 1 & \text{if } \lfloor i/2^v \rfloor \pmod 2 = 0 \\ -1 & \text{otherwise.} \end{cases}$$

$\underline{\mathsf{Verify}}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_{x_i}, x_i)$: Parse pk_Λ as H_k, $\Lambda_\mathcal{X}$ as $\ell_{0,0}$, set $\ell_{i,k} \leftarrow H_k(x_i)$. Recursively check for all $0 < v < k$ whether the following holds and return 1 if so. Otherwise return 0.

$$\ell_{\lfloor i/2^{v+1} \rfloor, k-(v+1)} = \begin{cases} H_k(\ell_{\lfloor i/2^v \rfloor, k-v} || \ell_{\lfloor i/2^v \rfloor + 1, k-v}) & \text{if } \lfloor i/2^v \rfloor \pmod 2 = 0 \\ H_k(\ell_{\lfloor i/2^v \rfloor - 1, k-v} || \ell_{\lfloor i/2^v \rfloor, k-v}) & \text{otherwise.} \end{cases}$$

Scheme 1: Merkle-tree accumulator.

the element and checking if result matches the accumulator. We recall the RSA-based accumulator in Scheme 2. Note however, that we define $\mathsf{WitCreate}$ in a way that does not require the factorization of N, i.e. no secret key is required. Correctness can easily verified, and collision freeness follows from the strong RSA assumption:

Lemma 2 ([BP97]). *If the strong RSA assumption holds, Scheme 2 is collision-free.*

Additionally, we recall the t-SDH-based accumulator from Nguyen [Ngu05]. The idea here is to encode the accumulated elements in a polynomial. This polynomial is then evaluated for a fixed element and the result is randomized to obtain the accumulator. Similar to the RSA-based accumulator, a witness consists of the evaluation of the same polynomial with the term corresponding to the respective element cancelled out. For verification a pairing is used to check whether the polynomial encoded in the witness is a factor of the one encoded in the accumulator. The scheme is depicted in Scheme 3. Again we define the accumulator in a way that no secret key, i.e. s, is required. Correctness is again obvious, whereas collision freeness follows from the t-SDH assumption:

$\underline{\mathsf{Gen}(1^\kappa, t)}$: Fix a hash functions H with $H : \{0,1\}^* \to \mathbb{P}$. Choose an RSA modulus $N = p \cdot q$ with two large safe primes p, q, and let g be a random quadratic residue mod N. Set $\mathsf{sk}_\Lambda \leftarrow \emptyset$ and $\mathsf{pk}_\Lambda \leftarrow (N, g, H)$

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as (N, g, H). Return $\Lambda_\mathcal{X} \leftarrow g^{\prod_{x \in \mathcal{X}} H(x)} \mod N$ and $\mathsf{aux} \leftarrow \mathcal{X}$.

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Return $\mathsf{wit}_x \leftarrow g^{\prod_{x' \in \mathcal{X} \setminus \{x\}} H(x')} \mod N$.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as (N, g, H). If $\mathsf{wit}_x^{H(x)} = \Lambda_\mathcal{X} \mod N$ holds, return 1, otherwise return 0.

$\underline{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g, H) and aux as $\mathcal{X}$. Set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$, $\mathsf{aux}' \leftarrow X'$, and $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{H(x)} \mod N$. Return $\Lambda_{\mathcal{X}'}$ and aux'.

$\underline{\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g, H). Return $\mathsf{wit}_{x_i}^{H(x)} \mod N$.

Scheme 2: RSA-based accumulator.

$\underline{\mathsf{Gen}(1^\kappa, t)}$: Let $\mathbb{G}$ be a prime order group p generated by g with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Choose $s \in \mathbb{Z}_p^*$ and return $\mathsf{sk}_\Lambda \leftarrow \emptyset$ and $\mathsf{pk}_\Lambda \leftarrow (\mathbb{G}, e, (g^{s^i})_{i=0}^t)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as $(\mathbb{G}, e, (g^{s^i})_{i=0}^t)$ and $\mathcal{X}$ as subset of $\mathbb{Z}_p$. Expand the polynomial $\prod_{x \in \mathcal{X}}(x + X) = \sum_{i=0}^n a_i X^i$, choose $r \xleftarrow{R} \mathbb{Z}_p^*$ and return $\Lambda_\mathcal{X} \leftarrow (\prod_{i=0}^n g^{s^i})^{a_i})^r$ and $\mathsf{aux} \leftarrow (r, \mathcal{X})$.

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse aux as $(r, \mathcal{X})$, run $(\mathsf{wit}_x, \ldots) \leftarrow \mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X} \setminus \{\mathsf{x}\}; \mathsf{r})$, and return wit_x.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as $(\mathbb{G}, e, (g^{s^i})_{i=0}^t)$. If $e(\Lambda_\mathcal{X}, g) = e(\mathsf{wit}_x, g^x \cdot g^s)$ holds, return 1, otherwise return 0.

$\underline{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ and aux as $(r, \mathcal{X})$. Set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$ and return $\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X}'; r)$.

$\underline{\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Return $\Lambda_\mathcal{X} \mathsf{wit}_{x_i}^{x - x_i}$.

Scheme 3: t-SDH-based accumulator.

Lemma 3 ([Ngu05]). *If the t-SDH assumption holds, Scheme 3 is collision-free.*

2.4 Distributed Point Functions

Distributed Point Functions (DPFs) were introduced by Gilboa and Ishai [GI14] and later generalized and improved by Boyle, Gilboa, and Ishai [BGI15, BGI16] in a concept called Function Secret Sharing (FSS). A point function $P_{x,y}$ is a function defined for $x, y \in \{0,1\}^*$, so that

$$P_{x,y}(x') = \begin{cases} y & \text{if } x' = x \\ 0^{|y|} & \text{otherwise.} \end{cases}$$

A DPF is a keyed function family F_k, where given x, y we can generate n keyshares $(k_0, k_1, \ldots, k_n)$ so that $\sum_{i=0}^n F_{k_i} = P_{x,y}$ and F_{k_i} completely

hides x and y. We focus on the case of two parties because efficient DPF constructions exist for $n = 2$, where the sizes of the key-shares k_i are logarithmic in the domain of the DPF input, whereas the best generic multi-party construction of DPFs have key-share sizes in the order of the square root of the domain.

The interface of a DPF is given as a tuple of functions $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ in [BGI16], and is defined for general y, however for our use, we restrict it to $y = 1$. We also fix the number of parties to two, and then use AES as an efficiently computable PRF, as suggested by [BGI16]. In the following, N refers to the domain of the DPF. We describe the interface in the following:

$\mathsf{DPF.Gen}(x)$: Given an index $x \in [N]$, this algorithm returns a key pair (k_0, k_1).

$\mathsf{DPF.Eval}(k_b)$: Given a key k_b, which is the result of a previous call to $\mathsf{DPF.Gen}(x)$, this algorithm produces a keystream K_b of length N. Given $K_0 = \mathsf{DPF.Eval}(k_0)$ and $K_1 = \mathsf{DPF.Eval}(k_1)$,

$$(K_0 \oplus K_1)[x'] = \begin{cases} 1 & \text{if } x' = x \\ 0 & \text{otherwise.} \end{cases}$$

2.5 Private Information Retrieval

Private Information Retrieval (PIR) is a primitive originally introduced by Chor et al. [CKGS98], that allows a client to retrieve an item from a server database without the server learning anything about the item requested. The server's privacy is not a concern in PIR schemes, and the database may even be public, only the client's query is considered private.

Computational PIR is a flavor of PIR where the client's query is hidden from a polynomially bounded server. Such PIR schemes can, for example, be built from fully homomorphic encryption (FHE). Information-theoretic PIR protects the client's query even against a computationally unbounded server. Such schemes usually rely on multiple non-colluding servers to provide such strong privacy guarantees and usually offer more performance than single-server PIR schemes. Since the introduction of PIR in the 1990s by Chor et al. many works have improved the communication and computational complexity of PIR schemes, for example [DHS14, ACLS18, GCM+16, MBFK16, Gol07, DGH12, BS07, LG15].

An efficient 2-server computational PIR scheme can be constructed from DPFs in a straight-forward way as shown in [GI14]. Before we discuss the

instantiation, we recall their definition of a private information retrieval (PIR) protocol:

Definition 4 (2-server PIR). *A 2-server PIR protocol involving two servers S_0, S_1 holding the same $n-$bit database z and a user consists of three algorithms (Q, A, M) with query domain D_Q and answer domain D_A and are defined as follows:*

$Q(n, i)$: *On input of an index i, client returns queries $(q_0, q_1) \in D_Q^2$.*

$A(z, q)$: *On input of a query q and a database z, server b returns an answer a_b.*

$M(i, a_0, a_1)$: *In input of an index i and two answers a_0, a_1, recovers and returns the i-th database entry z_i.*

We note that [GI14, Definition 2] explicitly handless random coins, but we simply omit them for the sake of brevity.

Definition 5 (Correctness). *A 2-server PIR scheme is correct if for every $n \in \mathbb{N}$, every $z \in \{0,1\}^n$ and every $i \in [n]$, it holds that*

$$\Pr\left[(q_0, q_1) \leftarrow Q(n, i) \colon M(i, (A(j, z, q_j))_{j \in \{0,1\}}) = z_i\right] = 1.$$

Definition 6 (Computational Secrecy). *Let $D_{b,\lceil \log n \rceil, i}, b \in 0, 1, n \in \mathbb{N}$ and $i \in [n]$ denote the probability distribution on q_b induced by Q. A 2-server PIR scheme provides computational secrecy if there exists a PPT algorithm* Sim *such that the following two distributions*

$$\{\textsc{Sim}(b, \lceil \log n \rceil)\}_{b \in \{0,1\}, n \in \mathbb{N}} \text{ and } \{D_{b,\lceil \log n \rceil, i}\}_{b \in \{0,1\}, n \in \mathbb{N}, i \in [n]}$$

are computationally indistinguishable.

We now give a short intuition of a 2-server PIR construction from a DPF. There, the client calls $(k_0, k_1) \leftarrow \mathsf{DPF.Gen}(q)$ and sends k_0 to server 0 and k_1 to server 1. Both servers 0 and 1 call $K_i \leftarrow \mathsf{DPF.Eval}(k_i)$ and perform an inner product between the expanded keystream and the database items, $X_i = \bigoplus_{l=0}^{N} K_i[l] \cdot \mathrm{DB}[l]$. The servers finally return X_0 and X_1 to the client who can recover the requested item $x_q = X_0 \oplus X_1$. The correctness of this PIR scheme follows from the correctness of the used DPF scheme. The privacy of the PIR scheme follows from the privacy of the used DPF scheme ([GI14, Theorem 2]), but requires that the two servers do not collude.

3 Modeling Append-Only Logs and Membership Proofs for CT

In this section, we give a model of the append-only[7] log functionality that is used in the CT ecosystem. We then extend the append-only log by also allowing for privacy-preserving membership proofs.

3.1 Append-Only Logs

An append-only log has to provide several functionalities: (i) adding new items, (ii) proving membership of a item in the log, (iii) proving consistency of the append-only property between two versions of the log. We closely model append-only logs on the definition of accumulators, but take care of the interactive nature. In the following, we define the syntax of the append-only log protocol between a client and a server closely resembling the CT protocol.

Definition 7 (Append-Only Log). *An append-only log is an interactive protocol of a global* Setup *algorithm, a client with algorithms* VerifyMember *and a server with algorithms* (Append, GetAcc, ProveMember) *which are defined as follows:*

$\mathsf{Setup}(1^\kappa, t)$: *This algorithm takes a security parameter* κ *and a parameter* t*. If* $t \neq \infty$*, then* t *is an upper bound on the number of elements to be accumulated in the log. It returns public parameters* pp.[8]

$\mathsf{Append}(x_i)$: *This algorithm takes new item* x_i *and appends it to the log.*

$\mathsf{GetAcc}()$: *This algorithm returns the current log accumulator value* $\Lambda_{\mathcal{X}}$ *to the client.*

$\mathsf{ProveMember}(x_i)$: *This algorithm value* x_i*. It returns* $\perp$*, if* $x_i \notin \mathcal{X}$*, and a witness* wit_{x_i} *for* x_i *otherwise.*

$\mathsf{VerifyMember}(\Lambda_{\mathcal{X}}, \mathsf{wit}_{x_i}, x_i)$: *This algorithm takes an accumulator* $\Lambda_{\mathcal{X}}$*, a witness* wit_{x_i} *and a value* x_i*. It returns* 1 *if* wit_{x_i} *is a witness for* $x_i \in \mathcal{X}$ *and* 0 *otherwise.*

[7] Although the generalized functionality might be more accurately called "add-only", since the order of the elements is not preserved in general, we choose to go with "append-only", since it is consistent with the terminology used, e.g., by the Certificate Transparency RFC [LLK13].

[8] We assume that these public parameters are available implicitly in all algorithms.

The server starts off with an initially empty log. Optionally, a server can provide an additional algorithm Gen *and the client an additional algorithm* VerifyAcc *defined as follows:*

$\mathsf{Gen}()$: *This algorithm generates a secret signing key* sk *and a verification key* pk.

$\mathsf{VerifyAcc}(\mathsf{pk}, \Lambda_{\mathcal{X}}, \sigma)$: *This algorithm takes the server public key* pk*, an accumulator* $\Lambda_{\mathcal{X}}$*, a signature on the accumulator* σ*. It returns* 1 *if* σ *is valid, and* 0 *otherwise.*

It these two algorithms are available, GetAcc *additionally returns a signature on the accumulator.*

The additional algorithms Gen and VerifyAcc provide the functionality of signed tree head, i.e., Gen creates the signing key material on the server side and VerifyAcc verifies the signature on the accumulator.

For correctness of the log, we require that for every $\kappa \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\kappa}, \mathsf{t})$, that for every x appended to the log using $\mathsf{Append}(\mathsf{x})$, for all $\Lambda \leftarrow \mathsf{GetAcc}()$, it holds that

$$\mathsf{VerifyMember}(\Lambda, \mathsf{ProveMember}(x), x) = 1.$$

This essentially captures that the membership proof for every element added to the log can be verified. If the append-only log also provides the optional algorithms Gen and VerifyAcc, then we additionally require for correctness, that for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}()$ and all $(\Lambda, \sigma) \leftarrow \mathsf{GetAcc}()$, it also holds that

$$\mathsf{VerifyAcc}(\mathsf{pk}, \Lambda, \sigma) = 1.$$

A variant of the append-only log is one with privacy-preserving membership proofs, which allow a client to retrieve a membership proof for a certain item without the server learning the item for which the proof was requested. This property is useful in many applications such as CT, where it allows a client to hide its browsing behavior from the log server.

Definition 8 (Append-Only Log with Privacy-Preserving Membership Proofs). *The append-only log with privacy-preserving membership proofs additionally extends Definition 7 with algorithms* $(\mathsf{PMQuery}, \mathsf{PMReconstruct})$ *for the client the server with* PMAnswer *algorithm which are defined as follows:*

$\mathsf{PMQuery}(x_i, i, n)$: *This algorithm takes an item x_i with its corresponding index i and returns queries $(q_j)_{j\in[n]}$ for n servers.*

$\mathsf{PMAnswer}(j, q_j)$: *This algorithm takes a query q_j for the j-th servers and returns an answer a_j.*

$\mathsf{PMReconstruct}(i, (a_j)_{j\in[n]})$: *Given answers $(a_j)_{j\in[n]}$ for index i, it reconstructs the witness wit_{x_i}.*

The client may use n servers to request membership proofs.

The proof returned by PMReconstruct can be verified as normal using VerifyMember. While the algorithms in this definition are closely modeled after those of PIR protocols, we note that this does not necessarily restrict instantiations to PIR based ones.

For correctness, we require first of all, that it satisfies the correctness of append-only logs. Additionally, we require that for all items x append to the log with their corresponding index i and n servers, for all $\Lambda \leftarrow \mathsf{GetAcc}()$, it holds that

$$\mathsf{VerifyMember}(\Lambda, \mathsf{PMReconstruct}(i, (a_j)_{j\in[n]})) = 1$$

where

$$a_j \leftarrow \mathsf{PMAnswer}(j, \mathsf{PMQuery}(x_i, i, n)) \text{ for } j \in [n].$$

Thereby we ensure that reconstructed witness verify. This definition allows the clients to contact multiple servers to obtain the membership proof. In the following, we will focus on the case $n = 2$.

Finally, we discuss the security notions. However, since our main concern are privacy issues, we only discuss the first two properties briefly. Inspired by an accumulator's collision freeness, the append-only log is collision-free if servers can only produce witnesses for elements that were included in the accumulator. Secondly, we require that adversaries cannot forge signatures on accumulators, i.e. that the append-only log is unforgeable. The third notion is geared towards the client's privacy when requesting proofs for logged elements and ensures that the queries do not leak any information on the queried elements. More formally, we define it in the same vein as computational secrecy of PIRs (cf. Definition 6):

Definition 9 (Computational Secrecy). *Let $D_{b,\lceil \log N\rceil,i}, b \in \{0,1\}, n \in \mathbb{N}$ and $i \in [n]$ denote the probability distribution on q_b induced by* PMQuery.

An append-only log scheme provides computational secrecy if there exists a PPT algorithm Sim *such that the following two distributions*

$$\{\text{SIM}(b, \lceil \log n \rceil)\}_{b\in\{0,1\},n\in\mathbb{N}} \text{ and } \{D_{b,\lceil \log n \rceil,i}\}_{b\in\{0,1\},n\in\mathbb{N},i\in[n]}$$

are computationally indistinguishable.

3.2 CT as Append-Only Log

We now show that the existing CT logging ecosystem implements an append-only log according to Definition 7. It also provides the optional algorithms based on signature schemes, which we formally recall in Appendix B. We note that Scheme 4 uses a yet undefined algorithm of the

Let MT be a Merkle-tree accumulator, Σ be a signature scheme, and let $\mathcal{X} \leftarrow \emptyset$ be the initially empty log on the server.

$\underline{\mathsf{Setup}(1^\kappa, t)}$: Call $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda) \leftarrow \mathsf{MT.Gen}(1^\kappa, t)$, set $\mathsf{pp} \leftarrow (1^\kappa, t, \mathsf{pk}_\Lambda)$, and return pp.
$\underline{\mathsf{Gen}()}$: Return $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$.
$\underline{\mathsf{Append}(x_i)}$: Set $\mathcal{X} \leftarrow \mathcal{X} \cup \{x_i\}$, and update the internal state of the accumulator $(\Lambda, \mathsf{aux}) \leftarrow \mathsf{MT.Add}((\emptyset, \mathsf{pk}_\Lambda), \Lambda, \mathsf{aux}, x_i)$ or $(\Lambda, \mathsf{aux}) \leftarrow \mathsf{MT.Eval}((\emptyset, \mathsf{pk}_\Lambda), \Lambda, \mathsf{aux}, x_i)$ if x_i is the first element appended.
$\underline{\mathsf{GetAcc}()}$: Set $\sigma = \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, \Lambda)$ and return (Λ, σ).
$\underline{\mathsf{VerifyAcc}(\mathsf{pk}_\Sigma, \Lambda, \sigma)}$: : Return $\Sigma.\mathsf{Verify}(\mathsf{pk}_\Sigma, \Lambda, \sigma)$.
$\underline{\mathsf{ProveMember}(x_i)}$: Return $\mathsf{MT.WitCreate}((\emptyset, \mathsf{pk}_\Lambda), \Lambda, \mathsf{aux}, x_i)$.
$\underline{\mathsf{VerifyMember}(\Lambda, \mathsf{wit}_{x_i}, x_i)}$: Return $\mathsf{MT.Verify}(\mathsf{pk}_\Lambda, \Lambda, \mathsf{wit}_{x_i}, x_i)$.

Scheme 4: Certificate Transparency Logging as append-only log.

Merkle-tree, namely $\mathsf{MT.Add}$, yet no function to update witnesses is used. Especially if $\mathsf{UpdateWitness}$ is not defined to achieve a dynamic accumulator, Add is easily implemented by simply recomputing the accumulator value.

As the correctness, collision freeness and unforgeability are straight-forward to check for Scheme 4, we only give a sketch of the proof:

Lemma 4. *Scheme 4 is correct. Additionally, if the accumulator is collision-free and the signature scheme Σ is unforgeable, Scheme 4 is collision-free and unforgeable, respectively, as well.*

Proof (Sketch of proof). Correctness follows easily from the correctness of the accumulator and the signature scheme. Collision freeness follows with

a straightforward reduction to the collision freeness of the accumulator, and unforgeability follows from the EUF-CMA security of the signature scheme.

The existing CT ecosystem does not implement an append-only log with privacy-preserving membership proofs according to Definition 8. Thus we extend the existing CT system with privacy-preserving membership proofs using PIR in Scheme 5.

Let PIR be a private information retrieval scheme where the witnesses are stored in the PIR databases.

$\mathsf{PMQuery}(x_i, i, n)$: For each Merkle-tree level $v \in [k]$ run $(q_j^{(v)})_{j\in[n]} \leftarrow \mathsf{PIR.Q}(n, \lfloor 1/2^v \rfloor + \eta)$ where $\eta = 1$ if $\lfloor 1/2^v \rfloor = 0 \pmod 2$ and $\eta = -1$ otherwise. Return $((q_j^{(v)})_{v\in[k]})_{j\in[n]}$.

$\mathsf{PMAnswer}(j, q_j)$: Parse q_j as $(q_j^{(v)})$ and run $a_j^{(v)} \leftarrow \mathsf{PIR.A}(j, q_j^{(v)})$ for each Merkle-tree level $v \in [k]$. Return $(a_j^{(v)})_{v\in[k]}$.

$\mathsf{PMReconstruct}(i, (a_j)_{j\in[n]})$: Parse $(a_j)_{j\in[n]}$ as $(a_j^{(v)})_{v\in[k]}$ and run $\mathsf{wit}_{x_i}[v] \leftarrow \mathsf{PIR.M}(i, (a_j^{(v)})_{j\in[n]})$ for each $v \in [k]$. Return wit_{x_i}.

Scheme 5: CT log with privacy-preserving membership proofs.

For the case with $n = 2$, i.e. two servers, we specialise in Scheme 6 the scheme using the DPF-based PIR from Section 2.5. For both schemes, the client traverses each level of the tree and calculates the index of the element he needs to retrieve for the Merkle-tree witness, with the server input to the PIR functionality being the hashes in the current tree level. The privacy guarantees of Scheme 6 follow from the privacy guarantees of the used PIR scheme.

Theorem 1. *If* PIR *is computationally secret, then Scheme 5 is computationally secret too.*

Proof. Indeed, the k-fold application of PIR's Sim algorithm induces a simulation algorithm on the combined distribution of successive queries.

For the DPF-based instantiation, this means that the scheme provides secrecy if the two servers do not collude.

Remark 1 (Database Representation). While this scheme as presented has the advantage that the structure of the PIR database closely resem-

Client		**Server j**
Input: index i, Merkle-tree size N		Input: Merkle-tree ℓ with size N
$\mathsf{PMQuery}(\emptyset, i, 2)$:		
for $v = 0$ to $\lceil \log_2 N \rceil$:		
$\eta \leftarrow 1 - 2 \cdot (\lfloor i/2^v \rfloor \pmod 2)$		
$k_1^v, k_2^v \leftarrow \mathsf{DPF.Gen}(\lfloor i/2^v \rfloor + \eta)$		
$q_1 \leftarrow (k_1^v)_{v \in [\lceil \log_2 N \rceil]}$		
$q_2 \leftarrow (k_2^v)_{v \in [\lceil \log_2 N \rceil]}$		
return (q_1, q_2)		
for $j = 1$ to 2:		
$\mathsf{PMAnswer}(j, q_j)$:		
	$q_j \longrightarrow$	
		Parse q_j as $\left(k_j^0, k_j^1, \ldots, k_j^{\lceil \log_2 N \rceil}\right)$
		for $v = 0$ to $\lceil \log_2 N \rceil$:
		$K \leftarrow \mathsf{DPF.Eval}(k_j^v)$
		$a_j^v = \bigoplus_{k=0}^{\lfloor N/2^v \rfloor} \ell_{k,v} \cdot K[k]$
		$a_j \leftarrow \left(a_j^v\right)_{v \in [\lceil \log_2 N \rceil]}$
	$\longleftarrow a_j$	
return a_j		
$\mathsf{PMReconstruct}(2, \{a_1, a_2\})$:		
return $(a_1^v \oplus a_2^v)_{v \in [\lceil \log_2 N \rceil]}$		

Scheme 6: Privacy-Preserving Retrieval of Membership Proofs instantiated with DPF-based PIR.

bles the Merkle-tree and does not induce much storage-overhead, we also now discuss a possible alternative representation as used by Lueks and Goldberg [LG15], where they precompute the full Merkle-tree proof for each item and store it in a separate database. This reduces the amount of PIR queries to 1, which can improve performance if the PIR scheme is the performance bottleneck. However, such a representation has the disadvantage that updates to the Merkle-tree accumulator are much more costly, since the precomputed proofs need to be updated if the accumulator changes, increasing the cost of updates to $\mathcal{O}(n)$, where n is the number of total items in the accumulator. Furthermore, for our DPF-based PIR implementation, the actual cost of the PIR is composed of (i) the DPF evaluation and (ii) the inner product of the database items. Our highly

performant DPF implementation results in the inner product dominating this time (see Section 5.2). If we perform one PIR per tree level, we calculate an inner product with a database containing a single hash value per item, whereas, for the separate database of full proofs, we perform the inner product with a database containing k hash values per item, where k is the tree height. This results in the total time spent on the inner product being longer in the case of the separate database since in the tree-based PIR approach the number of items in the PIR database is halved each level. Therefore, and due to the costly updates and more substantial memory requirements, we use the tree-based PIR approach over the separate database of precomputed proofs. In Section 5.3, we give a comparison between these two approaches and give evidence that the tree-based database structure is superior.

3.3 Using Public-Key Accumulators

Scheme 4 only requires that the underlying accumulator's Eval and WitCreate algorithms only rely on public keys and public parameters. Scheme 6 does not require any special properties. While the latter is defined to efficiently fetch the witnesses of a Merkle-tree accumulator, for any other accumulator it can be defined by retrieving witnesses stored in a database using a PIR protocol. Hence it is also possible to instantiate the append-only log using public-key accumulators, e.g., with Scheme 2 and Scheme 3. We discuss the performance characteristics of instantiations using different kinds of accumulators in Section 5.2.

However, if Scheme 4 would allow one to use accumulators where servers also have access to the accumulator's secret key, servers could produce witnesses for elements that were not added to the accumulator (c.f. [DHS15, Section 3]). In Appendix A we discuss this fact for the RSA accumulator (Scheme 2). A similar fact can also be observed for the bilinear accumulator presented in Scheme 3. In that case, knowledge of the exponent s would allow the server to fabricate witnesses for non-accumulated elements. Thus, when using public-key accumulators to instantiate the append-only log, it is essential that the accumulator is set up by a trusted third party.

4 Sub-Accumulators in CT-Logs

Using private information retrieval (PIR) to retrieve CT log membership proofs comes with increased computation and communication complexity,

especially for the log server. In this section, we explore options that can reduce this complexity and make privacy-preserving membership queries more practical.

In the current CT logging ecosystem, adding new certificates to the log server does not happen instantly. Instead, the submitting parties get a signed promise of inclusion into the log, and all submitted certificates are only appended to the log at certain time intervals. The length of these intervals is not specified by the standard, but a certificate must be included in the log after the maximum merge delay (MMD) set by the log operator (usually 24 hours). This process allows us to restructure the Merkle-tree to reduce the overall depth of the tree that has to be traversed during the PIR protocol.

In Section 1.4 we discussed some of the methods outlined by the designers to increase the user's privacy when requesting membership proofs from the log server. One of the proposed solutions is to embed the proof in a certificate extension; however, such a proof would quickly get out of sync with the current accumulator value of the log. We, therefore, suggest using a hybrid approach of static and dynamic accumulators instead. A static accumulator requires the whole set of elements $\mathcal{X}$ to be accumulated to be available when building the accumulator, with no further updates permitted. Even though the CT ecosystem continually receives updates for new certificate chains, we can still make use of static accumulators. We collect all new certificates for a specified time interval into a set $\mathcal{X}_t$ and build a static accumulator $\Lambda_{\mathcal{X}_t}$. Since the accumulator for this small set $\mathcal{X}_t$ is static, we can generate a witness wit_{x_i} for each $x_i \in \mathcal{X}_t$, proving membership of x_i in $\Lambda_{\mathcal{X}_t}$, and attach this witness to the SCT or embed it in the certificate, since we do not require any updates to the witness in the future.

These small static accumulators for a given time interval are then in turn accumulated in a dynamic accumulator, which as a whole can be seen as the equivalent of the current CT log. This process helps to reduce the size of the dynamic accumulator, which in turn reduces the complexity of the PIR approach. A client only needs to fetch the inclusion proof for the dynamic accumulator using PIR and verifies the membership of the certificate in the sub-accumulator and the membership of the sub-accumulator in the dynamic accumulator.

Example 1 (New sub-tree every hour). The largest CT log servers, e.g., Google Argon, have an average throughput of $\approx$ 60000 certificates per

hour. Thus building a sub-tree per hour means that we need to accumulate about 2^{16} elements in the static sub-accumulators. In turn, if we assume a runtime of 3 years, this would result in a total of $24 \cdot 365 \cdot 3 = 26280$ items in the dynamic accumulator. If we instantiate this dynamic accumulator using a Merkle-tree accumulator, we have a tree depth of 15, which is very feasible to retrieve using multi-server PIR.

For the choice of static sub-accumulators, we consider two possibilities: using static Merkle-tree accumulators or using public-key based static accumulators.

Merkle-Tree Sub-Accumulators A straight-forward implementation is to also use Merkle-tree accumulators to instantiate the sub-accumulators. This essentially amounts to a conceptual categorization of some sub-tree of the original accumulator as static sub-accumulators, with the only change to the original accumulator being the guarantee that a sub-tree is static and does not accept any additional values.

Public-Key Sub-Accumulators An alternative to using static Merkle-tree accumulators the leaves of our big Merkle-tree would be to use static public-key based accumulators instead. These public-key accumulators have different trade-offs compared to Merkle-tree accumulators. They usually offer a constant-size membership proof and accumulation value, compared to the logarithmic proof size of Merkle-tree accumulators. However, both the generation and verification algorithms of public-key accumulators usually require more computationally expensive public-key operations. Furthermore, public-key accumulators require a trusted setup phase as we discussed in Section 3.3.

From a web server point of view, the constant size proofs of public-key accumulators are beneficial in theory, as the required communication only grows by a small, fixed amount. Furthermore, the web server does not actually have to perform any public-key operations but only relays the witness to the client, which then performs the verification algorithm. However, we are considering sub-accumulator sizes, where the combined size of the membership proof and accumulator value are very similar for Merkle-tree accumulators, RSA accumulators, and bilinear accumulators. This fact, combined with the setup requirements and the lower performance, makes the use of public-key accumulators less attractive in our setting.

We now discuss our approach more formally and show that the so obtained append-only log still provides secrecy. The sub-accumulator approach can

be interpreted as an accumulator of accumulators. We cast our approach in into the accumulator framework in Scheme 7 where we use the second argument of the Gen algorithm to define the size of the sub-accumulators.

Let OA and IA be accumulators.

$\underline{\mathsf{Gen}(1^\kappa, T)}$: Let $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{IA.Gen}(1^\kappa, T)$ and $(\mathsf{sk}_o, \mathsf{pk}_o) \leftarrow \mathsf{OA.Gen}(1^\kappa, \infty)$. Set $\mathsf{sk}_\Lambda \leftarrow \emptyset$ and $\mathsf{pk}_\Lambda \leftarrow (\mathsf{pk}_i, \mathsf{pk}_o, T)$. Return $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as $(\mathsf{pk}_i, \mathsf{pk}_o, T)$. Partition $\mathcal{X}$ into T-sized subsets $\mathcal{X}_1, \ldots, X_\ell$ with X_ℓ having potentitally less than T elements. For $j \in [\ell]$ compute $(\Lambda_j, \mathsf{aux}_j) \leftarrow \mathsf{IA.Eval}((\emptyset, \mathsf{pk}_i), \mathcal{X}_j)$ and $(\Lambda_\mathcal{X}, \mathsf{aux}) \leftarrow \mathsf{OA.Eval}((\emptyset, \mathsf{pk}_i), (\Lambda_j)_{j\in[\ell]})$. Set $\mathsf{aux}_\mathcal{X} \leftarrow ((\mathcal{X}_j, \Lambda_j, \mathsf{aux}_j)_{j\in[\ell]}, \mathsf{aux})$ and return $\Lambda_\mathcal{X}, \mathsf{aux}_\mathcal{X}$.

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}_\mathcal{X}, x)}$: Parse pk_Λ as $(\mathsf{pk}_i, \mathsf{pk}_o, T)$ and $\mathsf{aux}_\mathcal{X}$ as $((\mathcal{X}_j, \Lambda_j, \mathsf{aux}_j)_{j\in[\ell]}, \mathsf{aux})$. Find $j \in [\ell]$ such that $x \in \mathcal{X}_j$. Now compute $\mathsf{wit}_i \leftarrow \mathsf{IA.WitCreate}((\emptyset, \mathsf{pk}_i), \Lambda_{\mathcal{X}_j}, \mathsf{aux}_j, x)$ and $\mathsf{wit}_o \leftarrow \mathsf{OA.WitCreate}((\emptyset, \mathsf{pk}_o), \Lambda_\mathcal{X}, \mathsf{aux}, \Lambda_{\mathcal{X}_j})$. Return $(\mathsf{wit}_i, \Lambda_i, \mathsf{wit}_o)$.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as $(\mathsf{pk}_i, \mathsf{pk}_o, T)$ and wit_x as $(\mathsf{wit}_i, \Lambda_i, \mathsf{wit}_o)$. If both $\mathsf{IA.Verify}(\mathsf{pk}_i, \Lambda_i, \mathsf{wit}_i) = 1$ and $\mathsf{OI.Verify}(\mathsf{pk}_o, \Lambda_\mathcal{X}, \mathsf{wit}_o) = 1$, return 1, otherwise return 0.

Scheme 7: Sub-Accumulator based Accumulator.

Lemma 5. *If both* IA *and* OA *are collision-free, then Scheme 7 provides collision freeness.*

Proof (Sketch of Proof). Assume that $\mathsf{wit}_x = (\mathsf{wit}_i, \Lambda_i, \mathsf{wit}_o)$ is a verifiying witness for $x \notin \mathcal{X}$. Then either $x \notin X_j$ for all $j \in [\ell]$ and hence (x, wit_i) breaks collision freeness of IA, or Λ_i was not accumulated in $\Lambda_\mathcal{X}$, thus $(\Lambda_i, \mathsf{wit}_o)$ breaks collision freeness of OA.

Making this accumulator dynamic or at least providing an Add algorithm is more involved, though. If one adds one element at a time, it is necessary to add to $\mathcal{X}_\ell$ and update its accumulator until the $\mathcal{X}_\ell$ is also of size T. However, then, one has to remove the old accumulator value from the outer accumulator and add the new one. Hence it is more efficient to gather T elements and then add them at once. In that case, it is sufficient to add one accumulator to the outer accumulator. Alternatively, one could also add sets with less than T elements with one additional sub-accumulator without touching any of the old accumulator values at the cost of a larger outer accumulator.

Integration into the append-log scheme with privacy-preserving membership proof using this approach of adding T elements together is straightforward provided that the outer accumulator is a Merkle-tree or provides an Add algorithm. Consequently, we obtain such a scheme providing computational secrecy with non-colluding servers.

Corollary 1. *Scheme 6 instantiated with Scheme 7 provides computational secrecy if the PIR servers are non-colluding and all sub-accumulator witnesses have the same size.*

Proof. This follows from Theorem 1. If the sub-accumulator witnesses do not have the same size, it is possible to distinguish queries for witnesses which do not have the same size.

For improved efficiency in the context of certificate transparency, we make use of the fact that we can include parts of the proof into extension fields of the SCT or the certificate. Note that, throughout its lifetime, wit_i and Λ_i stay constant, and only wit_o needs to be updated after adding new elements to the append-only log. Hence we add wit_i and Λ_i to `CtExtensions`. Then only wit_o needs to be retrieved using the PIR protocol, thus greatly reducing its cost. With this approach, we can always avoid the restriction of requiring equal-sized sub-accumulator witnesses.

4.1 Additional Considerations

As discussed in Section 3.3, public-key accumulators usually require a setup phase involving a trusted third party. Otherwise, the party holding the accumulator might have access to the secret trapdoor information, allowing the creation of witnesses for elements that are not actually contained in the accumulator. A popular alternative to a trusted third party is the use of multi-party computation to compute the public parameters. One prominent example of such an approach is the "ceremony" of the cryptocurrency Zcash based on [BCG+14], where a multi-party computation was performed including hundreds of participants in a scalable multi-party protocol to generate the public parameters for the used proof system [BGM17].

We leverage the non-collusion property of the servers to generate the parameters for the used public-key accumulator using multi-party computation protocols. In recent years, more and more efficient solutions for distributed parameter generation have emerged, e.g., for distributed RSA

key generation [FLOP18], where the authors report a time of 134 seconds on a single core per party to generate a distributed RSA key pair, or for distributed ECDSA key generation [Lin17]. Similar techniques can be employed to generate public parameters for a bilinear accumulator in a distributed fashion.

5 Implementation & Evaluation

In this section, we describe our implementation of the DPF-based PIR to retrieve Merkle-tree inclusion proofs. We integrate our implementation into the existing CT log server infrastructure provided by Google and then evaluate its performance.

Using the DPF construction of Boyle et al. [BGI15] and its extensions [BGI16], we can efficiently generate and evaluate the DPF using only AES, which is very performant when using the AES-NI instructions in modern x86-64 CPUs. Like Wang et al. [WYG+17], we use the Matyas-Meyer-Oseas one-way compression function [MMO85], defined as $H(x) = E_{k_0}(x) \oplus x$. The fixed-key property of this construction allows us to benefit from the fact we only have to perform the AES key schedule once for maximum performance. Furthermore, we use the full-domain evaluation algorithm proposed by [BGI16] to avoid calculating intermediate results multiple times and optimize the implementation with respect to AES and vector pipelining. Additionally, the inner product of the expanded DPF keystream and the 256-bit long SHA-256 hash values in the Merkle-tree can be efficiently calculated using AVX vector operations, making our multi-server PIR suitable for large log sizes. For experiments using public-key accumulators, we base our implementation on the work of Tremel[9]. We report microbenchmarks on the performance of our implementations in Section 5.2.

5.1 Integration into existing CT log server infrastructure

The Google CT team provides two open-source implementations of a CT log server. The original prototype implementation[10] written in C++, and their new CT log server[11] written in Go, using Trillian[12], a scalable implementation of a Merkle-tree accumulator with separate data storage layers, as a backend.

[9] https://github.com/etremel/crypto-accumulators/
[10] https://github.com/google/certificate-transparency
[11] https://github.com/google/certificate-transparency-go
[12] https://github.com/google/trillian

To show the practicality of our solution, we integrate a prototype into the C++ implementation[13] and provide libraries for DPF-based PIR for both, C++[14] and Go.[15] We added new HTTP endpoints for retrieving proofs using DPF, given the index of the hashes in the SCTs, and extended the existing client software to verify retrieved SCTs against two servers. This new API was then used to verify the inclusion of several certificates in the log. We believe that the integration of the DPF-based PIR into the C++ log server is easily adaptable to the Go CT log server. We refer to the microbenchmarks in the following section for the performance overhead compared to the existing approach.

5.2 Performance Evaluation

To show the practicality of our solution, we evaluated both a DPF-based PIR on a standard Merkle-tree as currently used in CT logs, as well as DPF-based PIR on a Merkle-tree with hourly sub-accumulators to reduce the tree depth and complexity of the PIR query. We consider multiple different log server sizes based on existing log servers, more concretely, we perform benchmarks on log servers with $N \in \{2^{20}, 2^{22}, 2^{24}, 2^{26}, 2^{28}\}$ certificates.[16] All experiments are performed on a desktop PC equipped with an Intel® Core™ i7-4790 CPU @ 3.60GHz and 16 GB of RAM. We perform microbenchmarks on the different parts of the protocol. All tests are performed using a single-threaded implementation only; however, we remark that the server-side operations, the DPF.Eval algorithm and the inner product calculation, are trivially and perfectly parallelize-able, e.g., when using 4 threads, we observe a speedup of $\approx$ 4x.

In Table 2, we present the microbenchmarks when using DPF-based PIR to retrieve the Merkle-tree inclusion proof. We observe that even for a server with 2^{28} ($\approx$ 270 million) certificates, the total work for the server is just 1.067 seconds, whereas the client workload is less than 1 millisecond. The total communication between the parties is less than 6 KB. The inner product of the SHA-256 hashes is dominating the runtime. One possible future optimization to further speed up this inner product step would be the use of AVX512 instructions to process two hash values at once.

Table 3 shows the performance of our sub-tree approach. We observe that the total execution of the DPF-based PIR with a reduced tree size

[13] https://github.com/dkales/certificate-transparency
[14] https://github.com/dkales/dpf-cpp
[15] https://github.com/dkales/dpf-go/tree/master/dpf
[16] For larger log server sizes, the data no longer fits in the RAM.

N	*Client* DPF.Gen	*Server* DPF.Eval	*Server* Inner Prod.	*Client* Verification	Communication $C \rightarrow S_i$	Communication $C \leftarrow S_i$
2^{20}	0.05	0.32	4.28	< 0.01	2298	640
2^{22}	0.07	1.23	16.72	< 0.01	2886	704
2^{24}	0.08	4.78	64.49	< 0.01	3546	768
2^{26}	0.09	19.22	251.32	< 0.01	4278	832
2^{28}	0.11	78.41	988.93	< 0.01	5082	896

Table 2. Performance of DPF-based PIR when retrieving privacy-preserving membership proofs from a standard Merkle-tree based log server containing N certificates. Time in milliseconds, communication in bytes per server. Merkle-trees for larger log servers no longer fit into the main memory of our test machine.

of 15 levels results in a total runtime of 130 μs. The client verification time is slower when using PK accumulators in sub-trees, but still in the order of milliseconds. We also list the additional communication of the sub-accumulator and the corresponding witness in the "extra" column, as the web server needs to send this information included in the SCT. This approach is very performant, even for logs containing a total of 2^{31} certificates.

While the client verification times for the used public-key accumulators are very fast, a problem manifests on the server side. Since we set up the public-key accumulators on public-parameters only, we cannot use the secret trapdoor information to speed up accumulation and witness generation for the RSA and bilinear accumulators. This results in much worse performance for these two operations, especially generating witnesses for each element. Table 4 shows the performance of these two operations for 2^{10} and 2^{16} elements, which roughly correspond to creating one sub-accumulator per minute and hour on larger log servers respectively. We observe that for the public-key accumulators we evaluated (using a security level of 128 bits), the only realistic parameter set is using bilinear accumulators for sub-trees of size 2^{10}, which roughly corresponds to one sub-tree per minute. For the other options, accumulating all elements and generating the witnesses would take longer than the intended time-frame of one hour for 2^{16} elements or one minute for 2^{10} elements. A possible solution would be to retain the secret parameters and keep them split into shares, with servers engaging in a multi-party computation protocol to compute witnesses using the shares of the secret trapdoor information. The design and implementation of an efficient protocol for this task is an interesting avenue for future work. However, for our current system and

N	$N_\wedge$	Sub-acc. type	N_{sub}	*Client* DPF.Gen	*Server* DPF.Eval	*Server* Inner Prod.	*Client* Verification	Communication $C \to S_i$	Communication $C \leftarrow S_i$	Communication extra
2^{31}	2^{15}	RSA	2^{16}	0.03	0.01	0.09	3.97	1143	480	384
2^{31}	2^{15}	Bilinear	2^{16}	0.03	0.01	0.09	2.81	1143	480	768
2^{31}	2^{15}	Merkle	2^{16}	0.03	0.01	0.09	< 0.01	1143	480	512
2^{31}	2^{21}	RSA	2^{10}	0.06	0.62	7.68	3.97	2583	672	384
2^{31}	2^{21}	Bilinear	2^{10}	0.06	0.62	7.68	2.81	2583	672	768
2^{31}	2^{21}	Merkle	2^{10}	0.06	0.62	7.68	< 0.01	2583	672	320

Table 3. Performance of DPF-based PIR when retrieving privacy-preserving membership proofs from a log server with sub-accumulators. The number of sub-accumulators in the upper tree is $N_\wedge$, the maximum number of elements per sub-accumulator is N_{sub}. Time in milliseconds, Communication in bytes per server.

implementation, we recommend using Merkle-tree accumulators for the sub-accumulators.

Accumulator	N_{sub}	Accumulation	Witness gen.
RSA	2^{16}	63.47	≈ 1000000
	2^{10}	1.06	264.15
Bilinear	2^{16}	2.99	95672.12
	2^{10}	0.12	24.43
Merkle	2^{16}	0.03	0.09
	2^{10}	< 0.01	< 0.01

Table 4. Performance of one-time server-side sub-accumulator operations (without trapdoor information). Time in seconds.

5.3 Comparison to Lueks and Goldberg [LG15]

The only previous work aiming to improve the privacy of retrieving CT log membership proofs is by Lueks and Goldberg [LG15], where the authors optimize the PIR scheme of Goldberg [Gol07, DGH12] to allow for efficient batching of multiple queries. The PIR scheme used in [LG15] provides information-theoretic security and it is robust, meaning it can be extended so that some servers are allowed to misbehave, while still allowing the client to recover the item. Furthermore, it can be scaled up to more than two servers. In comparison, the DPF-based PIR we use only provides computational security and does not provide robustness, but can be instantiated very efficiently for two servers. We argue that the robustness property is not critical in the case of retrieving Merkle-tree inclusion proofs, since the validity of the retrieved item is later verified against the Merkle-tree head, allowing for detection of wrong results. We therefore compare to the scheme of Lueks and Goldberg in its simplest form, using two servers and providing robustness against 0 misbehaving servers. In Table 5, we give concrete performance numbers for both our implementation and the implementation of [LG15], which has been integrated into `Percy++`.[17] Since both implementations could benefit from our sub-accumulator approach, we only benchmark performance of retrieving standard membership proofs. For [LG15], we perform 2^8 queries in parallel to make use of their proposed optimizations. We give numbers for both, the precalculated database of membership proofs and the

[17] `http://percy.sourceforge.net/`

tree-based approach we discuss in Remark 1. For [LG15], we follow the recommendation of the authors and arrange the database in square-root sized blocks to minimize communication.

Our DPF based implementation outperforms the PIR scheme of Lueks and Goldberg in both runtime and communication in all tested configurations, where we can especially notice the logarithmic communication of the DPF-based PIR. Furthermore, we observe that using our approach of arranging the database to make use of the tree structure of the Merkle-tree does also improve the runtime and communication considerably when using the PIR scheme of Lueks and Goldberg, mostly due to the reduced overall size of the database. This also means we can keep the whole database for the tree-based representation in memory, resulting in much better performance. We can observe the jump in runtime from $N = 2^{24}$ to $N = 2^{26}$ when using a database of precomputed proofs for the DPF-based PIR, which is due to the fact that the database no longer fits into the available RAM.

N	Protocol	DB structure	Time/Query	Comm.
2^{22}	DPF	Tree	0.02	3.5
		Precomputed	0.28	0.98
	[LG15]	Tree	0.08	77.6
		Precomputed	0.59	108.7
2^{24}	DPF	Tree	0.06	4.2
		Precomputed	1.23	1.08
	[LG15]	Tree	0.24	155.0
		Precomputed	3.54	222.4
2^{26}	DPF	Tree	0.25	4.99
		Precomputed	82.61	1.17
	[LG15]	Tree	0.78	312.0
		Precomputed	–	–
2^{28}	DPF	Tree	1.03	5.8
		Precomputed	450.51	1.28
	[LG15]	Tree	3.57	625.5
		Precomputed	–	–

Table 5. Comparison of different PIR protocols when retrieving a membership proof from a log of N certificates. Time in seconds, Communication in KiB per server. A value of $-$ indicates the implementation ran out of memory.

Remark 2 (Batch processing of client queries). The main contribution of Lueks and Goldberg [LG15] was the optimized batch processing of queries, where a server can process multiple queries at an asymptotically lower cost than processing each query individually. Their approach even manages to batch queries from different clients together, which is beneficial in systems such as CT. For our DPF-based PIR, we cannot batch queries for different clients, but can still optimize multiple queries from the same client. This scenario is realistic due to two factors. First, when a client connects to a website, he usually does not only retrieve one certificate, but instead connects to multiple different web servers hosting stylesheets, Javascript files, images, or other resources, verifying each certificate. Second, the auditor in each TLS client can collect multiple certificates to audit them in batches at a later time. Demmler et al. [DRRT18] use a binning approach for queries in their PIR-PSI protocol, which can also be applied to our use-case. The main idea is to partition the database into β bins of N/β items each. When the indices of queries are uniformly distributed, the maximum number of items per bin can be bounded probabilistically. All bins are then padded to the maximum number of items, and multiple smaller queries are performed for each bin. The overall runtime is expected to decrease, with a slight increase in communication, depending on the choice of β. We refer to [DRRT18] for a more detailed discussion.

5.4 Deployment Considerations

With the enforcement of CT logging by big browser vendors in early 2018, the CT infrastructure has grown considerably and logged hundreds of millions of certificates. Any changes to the ecosystem should, therefore, be critically analyzed, as these changes may require widespread updates to server and client software. Our log with privacy-preserving membership proofs has the advantage that it can co-exist alongside existing log servers and does not require significant changes. Embedding proofs for the sub-accumulators in an extension field of the SCT means that a web server does not require any changes to support our proposed changes, as his job is to provide the SCTs to the client. For the client, the auditor code has to be extended to distinguish a log with privacy-preserving membership proofs from a standard log, and to use the new API endpoint to retrieve the proof from the two servers. The log server obviously requires more substantial changes, but its API is still compatible to a standard log server and both servers answering DPF-based PIR queries can still answer membership queries in a standard way if no privacy is required.

In addition to these considerations regarding the disruption of the existing ecosystem, we also require a non-collusion assumption between the two servers participating in the PIR query. This non-collusion assumption can be solved by hosting the second server on a cloud platform potentially run by a competitor of the first log server provider, as was also proposed by [DRRT18]. To maintain their reputation, the cloud providers have a significant incentive not to collude. The second server could also be hosted by privacy-conscious organizations and advocacy groups such as the European Digital Rights (EDRi) or the Electronic Frontier Foundation (EFF). Furthermore, the system is not strictly limited to two parties. Several such non-colluding servers could exist, and an auditor-client could pick any two of them to perform a privacy-preserving membership proof.

Remark 3 (Cloning Existing Log Servers). We now describe another approach to facilitate better integration into the existing CT logging ecosystem. Instead of setting up a new log server and accepting the submission of new certificates, we rely on the CT ecosystem and clone the data of an already existing log server. In addition to monitoring the cloned log server for consistency, we can now restructure the certificates contained in the cloned log server in sub-trees. Furthermore, other monitors can verify the consistency of our new log servers against the cloned one. The new sub-accumulator based log servers then hand out their own SCTs (including PIR index i and sub-accumulator witness wit_i) to existing domain owners that want to provide privacy to their users. These SCTs can then be delivered to clients by the web server, and clients can choose to perform a privacy-preserving membership proof against the new log servers instead of a regular one.

6 Conclusion

In this work, we have reiterated potential privacy problems for the CT ecosystem and presented a solution based on two-server PIR that offers competitive performance for real-world parameters. Furthermore, we present an approach using sub-accumulators that reduces the complexity of the PIR queries to a point where a single server could handle multiple thousands of requests per second, and show how such an approach can be set up by mirroring existing log servers, providing a privacy-preserving alternative for auditors. We have shown the practicality of our solution by integrating it into the existing CT log server implementation and performed a performance evaluation for several different parameter sets. We

believe our approach could offer privacy-conscious users an alternative and further strengthen the existing CT ecosystem.

Acknowledgments We thank David Derler and Daniel Slamanig for discussions and valuable comments. We thank all anonymous reviewers for their valuable comments. The authors have been supported by iov42, EU H2020 Project LIGHTest, grant agreement n°700321, and EU H2020 Project Safe-DEED, grant agreement n°825225.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society, 2018.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.

[BFK00] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "anonymity and unobservability in the internet". In *Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions*, CFP '00, pages 57–65, New York, NY, USA, 2000. ACM.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367. Springer, 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM Conference on Computer and Communications Security*, pages 1292–1303. ACM, 2016.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptology ePrint Archive*, 2017:1050, 2017.

[Bon16] Joseph Bonneau. Ethiks: Using ethereum to audit a CONIKS key transparency log. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 95–105. Springer, 2016.

[BP97] Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 1997.

[BS07] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. *J. Cryptology*, 20(3):295–321, 2007.

[CDG18] Melissa Chase, Apoorvaa Deshpande, and Esha Ghosh. Privacy preserving verifiable key directories. *IACR Cryptology ePrint Archive*, 2018:607, 2018.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CSF+08] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and W. Timothy Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. *RFC*, 5280:1–151, 2008.

[DGH12] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *USENIX Security Symposium*, pages 269–283. USENIX Association, 2012.

[DGHS16] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *ESORICS (2)*, volume 9879 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2016.

[DHS14] Daniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: practical multi-server PIR. In *CCSW*, pages 45–56. ACM, 2014.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, volume 9048 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2015.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *PoPETs*, 2018(4):159–178, 2018.

[EMBB17] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate transparency with privacy. *PoPETs*, 2017(4):329–344, 2017.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 331–361. Springer, 2018.

[GCM+16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *NSDI*, pages 91–107. USENIX Association, 2016.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, 2014.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC*, pages 182–194. ACM, 1987.

[Gol07] Ian Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 131–148. IEEE Computer Society, 2007.

[III11] Donald E. Eastlake III. Transport layer security (TLS) extensions: Extension definitions. *RFC*, 6066:1–25, 2011.

[Lau14] Ben Laurie. Certificate transparency. *ACM Queue*, 12(8):10–19, 2014.

[Lau16] Ben Laurie. Certificate transparency over DNS, 2016.

[LG15] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *Financial Cryptography*, volume 8975 of *Lecture Notes in Computer Science*, pages 168–186. Springer, 2015.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO (2)*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.

[LLK13] Ben Laurie, Adam Langley, and Emilia Käsper. Certificate transparency. *RFC*, 6962:1–27, 2013.

[LLK+18] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. Certificate transparency version 2.0 (draft-ietf-trans-rfc6962-28), 2018.

[MBB+15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In *USENIX Security Symposium*, pages 383–398. USENIX Association, 2015.

[MBFK16] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR : Private information retrieval for everyone. *PoPETs*, 2016(2):155–174, 2016.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[Mes17] Eran Messeri. Privacy implications of certificate transparency's DNS-based protocol, 2017.

[MMO85] Stephen M. Matyas, Carl H. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithms. *IBM Technical Disclosure Bulletin*, 27(10):5658–5659, 1985.

[NGR18] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-Draft draft-ietf-trans-gossip-05, Internet Engineering Task Force, January 2018. Work in Progress.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

[Res18] Eric Rescorla. The transport layer security (TLS) protocol version 1.3. *RFC*, 8446:1–160, 2018.

[SMA+13] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 internet public key infrastructure online certificate status protocol - OCSP. *RFC*, 6960:1–41, 2013.

[SvDS+18] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. *J. ACM*, 65(4):18:1–18:26, 2018.

[TD17] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *IEEE Symposium on Security and Privacy*, pages 393–409. IEEE Computer Society, 2017.

[WHF07] Rolf Wendolsky, Dominik Herrmann, and Hannes Federrath. Performance comparison of low-latency anonymisation services from a user perspective. In *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 2007.

[WYG+17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI*, pages 299–313. USENIX Association, 2017.

[YRC18] Jiangshan Yu, Mark Ryan, and Cas Cremers. DECIM: detecting endpoint compromise in messaging. *IEEE Trans. Information Forensics and Security*, 13(1):106–118, 2018.

A Membership Witnesses for Non-Accumulated Elements

We consider Scheme 2 with $\mathsf{pk}_\Lambda = (N, g, H)$ and $\mathsf{sk}_\Lambda = (p, q)$ where $N = p \cdot q$. Let $\mathcal{X}$ be some set and $x \notin \mathcal{X}$ such that $H(x)$ is invertible $\mod (p-$

$1)\cdot(q-1)$. Now, the accumulator for $\mathcal{X}$ is computed as $\Lambda_{\mathcal{X}} = g^{\prod_{x'\in\mathcal{X}} H(x')} \mod N$. Yet, as the factorization of N is known, the server can compute $\mathsf{wit}_x = \Lambda_{\mathcal{X}}^{H(x)^{-1}} \mod N$. Although x is not member of $\mathcal{X}$, $\mathsf{wit}_x^{H(x)} = \Lambda_{\mathcal{X}} \pmod N$ holds and thus the verification succeeds.

Assuming that p and q are κ bit primes $p-1$ and $q-1$ have at most $\approx \kappa-1/\log(\kappa-1)$ prime factors and if the have a large prime factor, upper bound is a lot smaller. $H(x)$ is invertible if $H(x)$ is not one of the prime factors of M. Hence, the chance of a random element x with $H(x)$ being non-invertible $\mod M$ is approximately

$$\frac{2\frac{\kappa-1}{\log(\kappa-1)}}{\frac{2^{2\kappa}}{2\kappa}} = \frac{\kappa(\kappa-1)}{4^{\kappa-1}\log(\kappa-1)} \leq \frac{\kappa(\kappa-1)^2}{4^{\kappa-1}(\kappa-2)}.$$

This gives the server opportunity to produce membership witnesses for non-accumulated elements with high probability.

B Signature Schemes

In this section, we shortly recall the standard definition of signature schemes.

Definition 10 (Signature Scheme). *A signature scheme Σ is a triple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *of PPT algorithms, which are defined as follows:*

$\mathsf{Gen}(1^\kappa)$: *On input of a security parameter, this algorithm outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$ *consisting of a secret signing key* sk *and a public verification key* pk.[18]

$\mathsf{Sign}(\mathsf{sk}, m)$: *On input of a secret key* sk *and a message m, this algorithm outputs a signature σ.*

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: *On input of a public key* pk*, a message m and a signature σ, this algorithm outputs a bit b.*

For correctness, we require that for all security parameters $\kappa \in \mathbb{N}$, for all key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$, for all messages $m \in \mathcal{M}$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1\right] = 1.$$

Additionally, we require them to be $\mathsf{EUF\text{-}CMA}$-secure.

[18] We assume that pk implicitly defines the message space $\mathcal{M}$.

Definition 11 (EUF-CMA). *The advantage* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{EUF\text{-}CMA}}(\cdot)$ *of an adversary* $\mathcal{A}$ *in the* EUF-CMA *experiment is defined as*

$$\Pr\left[\begin{matrix}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa), (m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathsf{S}(\mathsf{sk},\cdot)}(\mathsf{pk}) : \\ m^* \notin \mathcal{Q}^{\mathsf{S}} \ \wedge \ \mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1\end{matrix}\right],$$

where the environment maintains an initially empty list $\mathcal{Q}^{\mathsf{S}}$ *and the oracles are defined as follows:*

$\mathsf{S}(\mathsf{sk}, m)$: *Set* $\mathcal{Q}^{\mathsf{S}} \leftarrow \mathcal{Q}^{\mathsf{S}} \cup \{m\}$ *and return* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$.

A signature scheme is existentially unforgeable under random message attacks, if for every PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{EUF\text{-}CMA}}(\cdot)$ *is bounded by a negligible function in the security parameter* κ.

3. Revisiting Privacy-aware Blockchain Public Key Infrastructure

The subsequent paper has been published as follows:

Paul Plessing and Olamide Omolola. "Revisiting Privacy-aware Blockchain Public Key Infrastructure". In: *ICISSP*. SCITEPRESS, 2020, pp. 415–423

In this paper, we devise a privacy-aware approach for PKI on the blockchain. We analyse the privacy guarantees of PB-PKI, a proposed privacy-aware PKI and discover that it leaks private data. Our proposed privacy-aware PKI on the blockchain leverages ring signature for key authentication and verfication.

Abstract. Privacy-aware Blockchain Public Key Infrastructure (PB-PKI) is a recent proposal by Louise Axon (2017) to create a privacy-preserving Public Key Infrastructure on the Blockchain. However, PB-PKI suffers from operational problems. We found that the most important change, i.e., the key update process proposed in PB-PKI for privacy is broken. Other issues include authenticating a user during key update and ensuring proper key revocation. In this paper, we provide solutions to the problems of PB-PKI. We suggest generating fresh keys during key update. Furthermore, we use ring signatures for authenticating the user requesting key updates and use Asynchronous accumulators to handle the deletion of revoked keys. We show that the approach is feasible and implement a proof of concept.

Keywords: Blockchain, Public Key Infrastructure, Privacy, RSA.

1 Introduction

Nowadays, Public Key Infrastructure (PKI) plays a major role in ensuring secure communication. PKI works on the principle that a trusted third-party organization called Certificate Authority (CA) can sign certificates and vouch for the authenticity of the link between the public key and the subject name contained within the certificate. The trusted third-party verifies a client's identity and confirms the client's identity before signing. This third-party signs a certificate with its private key[1]. The certificate of the trusted third-party is assumed to be widely known, and another entity can verify the signed client certificate by verifying the signature of the trusted third-party organization affixed to it.

However, PKI is a centralized infrastructure and attacks like those carried out on Comodo[2] or DigiNotar[3] compromises the CAs, and that can compromise the integrity of the certificates issued thereby compromising the whole infrastructure.

Phil Zimmerman proposed the Web of Trust (WoT) in 1992 [15] as a decentralized alternative to PKIs. The initial processes leading to trust in WoT is different from that in PKI. For example, if Alice knows the public key of Bob and trusts him, and Bob knows the public key of Claire, then Alice can ask Bob for Claire's key and trust that it is indeed Claire's key. Alice and Bob exchange their keys initially in person. This exchange event is called a "Key Signing Party", where people exchange their keys with each other in person. However, this personal exchange poses two problems. The first problem is an

[1] The public key is included in the certificate

[2] https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html(last accessed on 20/03/2019)

[3] https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html(last accessed on 20/03/2019)

efficiency problem as only a few keys can be exchanged initially in a particular time frame. The second problem is a trust chain problem as it is possible that Alice is unable to find a trust chain that connects with Claire, thereby giving rise to isolated trust communities.

Researchers are continually trying to improve the two mechanisms above in different ways. Blockchain has recently come to the center stage of the research community [10], and many researchers have proposed using the Blockchain to solve some of the PKI and WoT problems. One of such proposals is the Privacy-aware Blockchain-based PKI (PB-PKI) [1]. PB-PKI[4] aims to use the Blockchain to solve the problems of WoT and also ensure privacy.

However, some of the goals were not achieved, and we discovered some problems with the key update process and the key revocation in the proposal.

Our contributions in this paper include the following:

1. **We show that the key update process in PB-PKI [1] does not ensure privacy.**
2. **We propose the use of ring signatures to solve the problem of authenticating registered members of the blockchain during key update to ensure that only registered members can perform key updates.**
3. **We propose a revocation mechanism that involves key deletion from the blockchain for PB-PKI.**
4. **We implement a PoC and show that it is feasible in practice.**

The rest of the paper is structured as follows: Section 2 gives a short overview of previous research; Section 3 gives a short introduction into Asynchronous accumulators and Ring Signatures; Section 4 describes the privacy notions in PB-PKI; Section 5 describes PB-PKI; Section 6 discusses the problems of PB-PKI and provides our solutions to them; we evaluate our ideas in Section 7; Section 8 discusses the implications of our changes to PB-PKI and we conclude in Section 9.

2 Related Work

Blockchain became popular in 2009 with the introduction of Bitcoin [10]. Blockchains are decentralized and store transactions between parties. All the transactions are publicly auditable by all participants, and once a transaction is recorded and confirmed, it is practically immutable. These characteristics have led researchers to propose implementing PKIs and WoT on the Blockchain [4, 6–8, 13, 14]. One example of such a proposal is Certcoin [4]. Certcoin is a blockchain variant built upon Namecoin[5] that also functions as a WoT.

[4] PB-PKI is actually an implementation of WoT on the Blockchain

[5] Namecoin is a fork of Bitcoin

The simplest version of Certcoin uses Namecoin as a bulletin board where blockchain posts and blockchain traversals support its functionalities. Data structures such as Asynchronous Accumulators [11] and Kademlia Distributed Hash Table (DHT) [9] were introduced in successive versions so that Certcoin is time- and space-efficient. However, Certcoin was not built with privacy taken into consideration.

Certcoin transactions store information about public key events. The public key events are registration, update, revocation, and verification. Certcoin mandates every entity to register two key pairs. The entity uses the first key pair called online key pair for communication and uses the second key pair called offline key pair for security purposes such as key revocation[6] and update. The offline key pair is stored offline to protect it.

The authors of Certcoin provide an incomplete implementation of Certcoin in the language Go. The implementation uses RSA keys and the Asynchronous Accumulator for key verification. PB-PKI builds on the foundation of Certcoin.

3 Preliminaries

This section briefly describes the algorithms that are used in the rest of the paper

3.1 Asynchronous Accumulator

Reyzin and Yakoubov proposed the Asynchronous accumulator(AA) [11]. AA is the dynamic form of the accumulator called Merkle tree [5], and it provides algorithms for adding or deleting elements from the original set. AA (see figure 1) is a container of several Merkle roots, i.e., an array of Merkle roots. Every index of the AA can contain only the root of one Merkle tree at a particular time.

AA preserves efficiency in an environment, where modifications of a Merkle tree happen often due to frequent addition of leaves. Recalculating the whole tree after a modification requires $\log_n$ time, therefore, it is desirable to optimize this process. AA works similar to a Merkle tree regarding verification.

1. Initially, the AA is empty
2. To add a message m_1, m_1 is hashed to H_1 and put into the first slot of the AA at index 0.
3. Suppose a second message m_2 needs to be added. Since index 0 is occupied by H_1, H_1 and H_2[7] are concatenated and hashed to H_{12}. The hash H_{12} is put at index 1 of the AA. Simultaneously, H_1 gets removed from index 0. To

[6] The offline key pair can revoke the online key by signing a revoke-event

[7] H_2 is the hash of message m_2

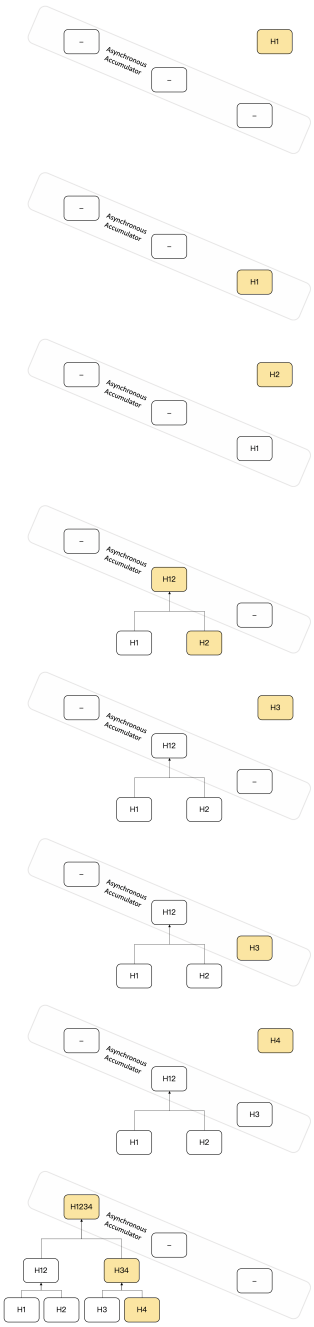


Fig. 1. Illustration of an Asynchronous Accumulator with its empty (”-”) and filled (”H****”) Merkle roots and their corresponding Merkle trees.

verify that message m_1 is part of the AA, the prover has to remember m_1 and H_2. With this knowledge, he can reconstruct the Merkle root at index 1.
4. When a third message m_3 arrives, it is hashed to H_3 and placed at index 0 since index 0 is empty.
5. The next message m_4 is also hashed to H_4. However, index 0 contains H_3. Therefore, H_3 and H_4 are concatenated and hashed into H_{34}. H_{34} should be placed at index 1, but it is occupied by H_{12}. Thus, H_{12} and H_{34} are concatenated and hashed into H_{1234}. H_{1234} is then placed at index 2. To verify the existence of message m_1, only message m_1, H_2 and H_{34} need to be remembered. Simultaneously, H_3 and H_{12} are removed from index 0 and index 1.

The worst case scenario for an addition to AA is $\log_n$ operations (concatenating and hashing), with n leaves of the biggest tree. In the best case, the addition can be done after one hash if index 0 is free. The trade-off of faster additions is that more space is needed for storing an AA compared to one Merkle root. The storage requirement of AA for n leaves is $\log_n$.

3.2 Ring Signatures

Ring signatures allow a user to sign a message and specify a set of possible signers without revealing which member actually signed the message [12]. The user can choose any set of possible signers that includes himself and sign by using his secret key and the other's public keys without getting their approval or assistance. A ring signature does not need any set-up, and the ring signature scheme is defined by two procedures:

1. **Sign**$(m, P_1, \ldots, P_r, S_s)$ which produces a ring signature σ for the message m, given the public keys $P_1, P_2, \ldots, P_r$ of the r ring members, together with the secret key S_s of the s-th member (who is the actual signer).
2. **Verify**(m, σ) which accepts a message m and a signature σ (the signature includes the public keys of all the possible signers), and outputs either true or false

4 Privacy Concerns

Users sometimes assume that public blockchains are anonymous. This assumption is true to the extent that personal information is not connected to the accounts of the blockchain user. An account is the hash of the public key of a user's key pair on the blockchain. Two possibilities for tracking users exist. For example, it is possible to find out the IP-address of a blockchain address by observing the blockchain network. A user can prevent this possibility by masking

his IP-address using anonymizing technologies like Tor. The second violation of privacy happens by design because the public can audit Blockchains. Therefore, the public knows what data is in a blockchain address' wallet, the meta information of the data such as its origin and more. A naive solution would be to use a new public key for each transaction. However, this has no real privacy gain since the mapping between the public key and the account is available for the public.

In terms of blockchain based PKIs, this violation of privacy would translate to linking any public key and its transactions even when the public key has been updated. At the moment a public key is used in any service, an adversary could track its activities across services. PB-PKI provides unlinkability between an updated public key and its identity without compromising the ability to verify that a specific public key is authorized to take an action.

5 PB-PKI

In this section, we give a short introduction into the original PB-PKI [1].

PB-PKI modifies part of Certcoin to achieve privacy. Registering, revoking and verifying a key in PB-PKI is the same as that of Certcoin. An entity registers its identity by posting its public key on the blockchain. The main difference between Certcoin and PB-PKI is the key update process. The unique key update procedure in PB-PKI aims to provide untrackability and provide a way to disclose the link between an identity and its key by the entity at a later point. This user-controlled disclosure enables the entity to prove that a message is signed with its unlinkable key which is connected to the certified key. PB-PKI hides the link between an identity represented by its current public key and its previous actions as well as keys, while still retaining the authenticity of the key.

Since the PB-PKI Key update process is the main improvement over Certcoin, we will focus on it in the rest of this paper. The PB-PKI Key update process (RSA keys) involves two steps:

1. Generate a new offline key pair, pkf_n and skf_n (offline public key at time n and offline secret key at time n), where:
2. Compute the new online key pair pkn_n and skn_n (online public key at time n, and online secret key at time n) in the following manner:
   With this formula, the new keys are a valid RSA key pair, where:

$$\mathsf{pkn}_\mathsf{n} \cdot \mathsf{skn}_\mathsf{n} = 1 \pmod{\mathsf{N}_\mathsf{n}}$$

The two steps form a chain of keys as shown in figure 2 after every update. When a user wishes to disclose the public keys, the user has to publish all offline

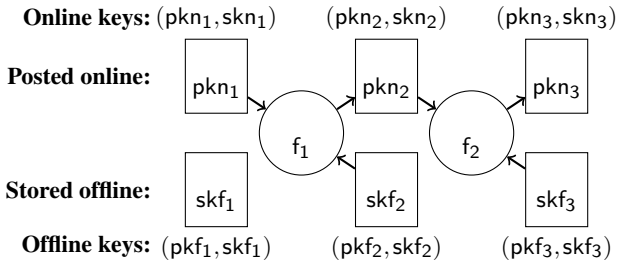


Fig. 2. Key update procedure to facilitate user-controlled disclosure.

public keys. With that knowledge, anyone can recompute the chain of public keys and verify that a specific key leads back to a particular identity.

Regarding the key update transaction, it must be guaranteed that this new unlinkable public key is from a registered member. Louise Axon [1] proposed that the identity that wants to update must provide a signature signed with the current key. However, this would mean a public linkage of all keys. Therefore, the author proposed that the link is disguised by encrypting the signature with the public keys of a randomly chosen subset of the network members. These network members are then included in the verification process because they can decrypt the signature and verify that it is indeed from a key that was already on the network. This of course still enables the verifiers to track that two keys belong to the same person. However, since the subset is chosen randomly for every key update, a verifier can only track the identity link between two keys.

Furthermore, the changed usage of the offline key pair means that it cannot be used the same way as in Certcoin. In Certcoin, the offline key provided a way to revoke compromised keys. In case of online key theft, a user can still prevent the malicious usage of the stolen online key by revoking it with the offline key. However, the user stores the offline keys securely after its generation in PB-PKI and only publishes them to disclose its identity. To obtain the same revocation abilities as Certcoin, PB-PKI introduces the so-called “Master Key”. The Master key functions the same way as offline keys in Certcoin, i.e., it can revoke a current compromised online key.

6 Enhancements to PB-PKI

This section describes the problems encountered in the implementation of PB-PKI and the solutions to these problems[8]. Some of the problems required a total

[8] We refer to our improved version of PB-PKI as Enhanced PB-PKI.

change of procedure while the others needed some modifications to the existing procedures. The issues and their solutions are given below:

6.1 Creation of new Keys

In PB-PKI [1] new keys are generated with the novel update procedure described in section 5. This update procedure should ensure unlinkable keys while allowing user-controlled disclosure at the same time. However, this novel procedure has a side effect - it uses the same modulus for every updated key pair. An attack where encrypted messages can be decrypted when two public keys have the same modulus was documented by Dan Boneh [2]. We present equations below that involves two different public keys $\mathsf{e}_1, \mathsf{e}_2$, and two cipher texts of the same message encrypted by the two public keys A, B. The equations show that the message can be decrypted even without the private keys.
Given:

$$\mathsf{A} = \mathsf{M}^{\mathsf{e}_1} \pmod{\mathsf{n}}$$

$$\mathsf{B} = \mathsf{M}^{\mathsf{e}_2} \pmod{\mathsf{n}}$$

If $mathsf\gcd(e_1, e_2) = 1$,there exists some x, y such that $\mathsf{x}\mathsf{e}_1 + \mathsf{y}\mathsf{e}_2 = 1$
Therefore:

$$\begin{aligned}
\mathsf{A}^{\mathsf{x}} \cdot \mathsf{B}^{\mathsf{y}} &= \mathsf{M}^{\mathsf{e}_1 \mathsf{x}} \cdot \mathsf{M}^{\mathsf{e}_2 \mathsf{y}} \\
&= \mathsf{M}^{\mathsf{x}\mathsf{e}_1} \cdot \mathsf{M}^{\mathsf{y}\mathsf{e}_2} \\
&= \mathsf{M}^{\mathsf{x}\mathsf{e}_1 + \mathsf{y}\mathsf{e}_2} \\
&= \mathsf{M}^{1} \\
&= \mathsf{M}
\end{aligned}$$

The attacker could also trick the user to sign sensitive information twice with a past and current key. The attacker can then decrypt the message using the process we described. Furthermore, an adversary can scan the blockchain for public keys with equal modulus, and be sure that these keys belong to the same user. Instead of user-controlled disclosure, the novel update process actually ensures linkability by design.

Solution As the proposed update procedure could not disguise links between keys, we sought another mechanism. We found that the key update mechanism in PB-PKI is not critical to ensuring privacy. The mechanism helps in disclosure since a user can provide the link between his keys to a verifier. However, there

Table 1. Comparison of PB-PKI with NOOB-PKI regarding key operations.

PB-PKI	Enhanced PB-PKI
Setup Phase User U generates an online, offline and master RSA key pair.[9]	**Setup Phase** User U generates an online and offline RSA key pair.
Key Registration: U posts a registration transaction with – his identity – a timestamp – the public part of the generated online key – a signature of the generated online public key – a signature of the master key It has to be verified that the identity and the online key have not been registered previously and that the signature of the online key is valid.	**Key Registration:** U posts a registration transaction with – his identity – a timestamp – the public part of the generated online key (registration key) – a signature of the online public key – the witness of the online public key – the public part of the generated offline key – a signature of the offline public key It has to be verified that the identity and the public online key have not been registered previously. The signatures of the online and offline keys and the witness are also validated.
Key Update: User U generates a new offline RSA key pair with public part pkf and private part skf. To generate the new online key with public part pkn and private part skn, U calculates: $\mathsf{pkn_n} = \mathsf{pkn_{n-1}} \cdot \mathsf{skf_n}(\mathsf{ModNn})$ $\mathsf{skn_n} = \mathsf{skn_{n-1}} \div \mathsf{skf_n}(\mathsf{ModNn})$ U then posts an update transaction with – a timestamp – the public part of the newly calculated online key – a signature of the online public key – a signature of the previous online public key to ensure that U is already part of the network. This signature is encrypted with the public keys of a subset of network members. The subset is chosen randomly at each update and has to verify the signature of U's previous online key. U secret shares the new offline public key between a majority of the network members. It has to be verified that the online public key has not been registered previously, that the first signature is valid, and that the second signature is valid and done with a currently valid public key on the network.	**Key Update:** User U generates a new online and offline RSA key pair and posts an update transaction with: – a timestamp – the public part of the new online key – a signature of the new online public key – the witness of the new online public key – the public part of the new offline key – a signature of the new offline public key – a ring signature by use of a randomly selected subset of registration keys and U's registration key – all registration keys used for the ring signature The network members verify that the online public key has not been registered previously, the signatures of the online and offline keys are valid, and the witness is valid. Additionally, the network verifies that the ring signature is valid and that all used registration keys exist.
Key Revocation: User U posts a revocation transaction with – a timestamp – the public part of the to be revoked public key – a signature of the public key Key revocation can be executed either by the key holder.	**Key Revocation:** User U posts a revocation transaction with – a timestamp – the public part of the to be revoked key – a signature of the public key – the witness of the public key – the new ancestors of the revoked public key The network members verify that the revoked key is indeed part of the network and that the signature, the witness, the ancestors are valid. The ancestors of a given message are all its parent nodes in a Merkle tree. The revoked key is replaced with "$\perp$" in the Merkle tree, and the ancestors of "$\perp$" have to be posted so that other users can update their witnesses accordingly.

are other ways that one can use to prove the ownership of a key without linking one's keys. One such method is by signing the public key.

We propose generating a new random key pair at each update event. A user can still disclose that a specific key belongs to him. He does it by providing a signature with the key to be disclosed and his registration key. Thus, a verifier can be sure of the ownership of the disclosed key.

6.2 Key Deletion

In previous papers about Certcoin [4] and PB-PKI [1], fast key verification is achieved with the Merkle root of a Merkle tree containing all valid public keys. Specifically, many Merkle roots are involved because Certcoin and PB-PKI use the AA as explained in Subsection 3.1. To quickly verify whether a key is valid, a verifier takes the key as well as its witness and computes its Merkle root. Then the verifier takes the Merkle roots of the latest block and checks whether one of them match the computed one. This process is quick because it needs at most $\log_n$ operations (for n number of keys in the Merkle tree) to calculate the Merkle root. In addition, it is space efficient, because the verifier only needs to store the latest blockheader plus the public key and the witness[10].

Adding keys is very efficient as well because of the Asynchronous Accumulator. The maximum of operations needed per addition of a key is $\log_n$ (for n number of all keys in the Merkle tree). Consequently, the Merkle tree does not need to be fully recalculated but it is merged with other trees of the same size, i.e., concatenating the roots and hashing them.

This works very well when one adds keys only. However, the event of a key deletion in case of key revocation was not discussed intensively by the authors of Certcoin and PB-PKI. Even though in both papers it is mentioned that deletion is possible, only Certcoin gives a short explanation of how deletions can be implemented using a Merkle tree. To make the term clear: *deleting* a key from a Merkle tree means to modify an entry of a Merkle tree. The entry to be deleted gets replaced with an entry that indicates that the former public key is deleted.

Solution In the Enhanced version of PB-PKI, we use the sign "$\perp$" for deleted entries. With such a modification, we have to recalculate $\log_n$ parents of the Merkle tree. Therefore, while adding keys takes at worst case $\log_n$ operations, deletion of keys always takes $\log_n$ operations.

Furthermore, the time required for key holders to update their witness needs to be considered as well. Every witness in this Merkle tree has to be updated.

[10] Witnesses are the missing hashes needed by a verifier to construct a Merkle root

The number of operations for witness update is in the worst case in direct correlation to the number of operations needed to update the AA. In the best case, a key holder only has to do one operation to bring his witness up to date. Therefore for both addition and deletion, a key holder needs to do at most $\log_n$ operations.

6.3 Space Efficiency

The second issue regarding entry deletion from Merkle trees is space efficiency. If there are additions to the AA only, the only information that is needed to update any witness are the resulting Merkle roots after every addition. However, in the case of deletion, one needs to know the former witness of the deleted element in order to update the other witnesses of a Merkle tree. This means that the witness of every revoked and deleted key has to be stored and published. The result of this is more storage is used.

Solution The AA changes with every transaction, but only the final accumulator after all transactions in a block are taken into account gets shown. This means that for key additions it is mandatory to publish the witness of each added key as well. The result is the publication of every change of the AA.

As a result of the changes made in the preceding paragraphs, a prover with an old witness has to go back to the last block where his public key plus witness was valid. Then he traverses all transactions of all blocks leading to the most current block, and he adjusts his witness according to the occurred key events. Finally, he has a witness that provides proof when using the latest accumulator.

6.4 User Authentication

The third challenge was the authentication of a pending key update. At key update, it should be ensured that the user requesting the key update already owns a key on the network while securing his identity. In the proposed PB-PKI this is done through a signature of the current key. However, this would enable a linking of the keys. Therefore, the signature is encrypted. It is encrypted with a subset of total public keys available on the network. These members then have to decrypt the signature and verify that it is indeed done with a registered key.

This practice sounds nice in theory but is hard to implement. The first set of challenges arise: how can the subset of network members announce their verification of the signature? Post it on the blockchain? How can one trust that they are telling the truth? Moreover, how can one guarantee that they actually verify the signature and have not been offline for two years?

This set of questions remains unanswered but even if there were answers to them, another question arises: How to find out a subset of current public

keys? This would either require global storage of n public keys (for n network members), like the AA, but instead of a few Merkle roots we needed to record all valid keys, millions of it in every block. This would be fast but space intensive (totally unusable). Alternatively, we traverse the blockchain looking for valid keys, which would be space friendly but time intensive. Additionally, we cannot find out which public keys that were posted onto the blockchain are still valid and which are not, because no keys are linked to each other.

Solution We decided to find another way of solving the authentication problem. We chose to use ring signatures. Ring signatures are explained in Subsection 3.2. In the Enhanced PB-PKI, a ring signature is crafted and posted in the update transaction. The public keys are randomly selected out of the public keys published at registration event. The user's public key that was created during registration is added to the set of keys. With these selected keys, he performs the ring signature and puts it onto the transaction. That way, anyone can verify that this transaction must be from one of the already registered holders of the used public keys. However, nobody knows who it was from this group of keys. Moreover, by carefully selecting different large sets at each update, an attacker cannot tell which key belongs to which identity. We used a ring size of 6 possible signers for the proof of concept because having a higher ring size is space inefficient.

7 Evaluation

For the proof of concept, we used an Asus Laptop with 4GB RAM and an Intel Core processor i5 of the 5th generation. The installed operating system was Ubuntu 16.04. The PoC was written in the programming language Go, and the database we used to store the blocks was LevelDB.

With this setup, we measured how long it takes to execute the different operations:

- Register a Key: ~80 milliseconds
- Update a Key: ~120 milliseconds
- Revoke a Key: ~10 milliseconds
- Verify a Key: ~15 microseconds
- Update a Witness: ~90 milliseconds

Verifying a key is the most critical activity because users often have to verify whether a key is valid. In Enhanced PB-PKI, verifying a key is the fastest operation, because it only involves hashing and concatenation without any write operation. The bottleneck of posting transactions is the need to mine new blocks

and verify them. Setting the mining time or process is beyond the scope of this paper. The Enhanced PB-PKI is a full blockchain and it was deployed locally on the laptop for the experiments.

8 Discussion

In this section, we consider the possible cases of key compromise and the options that the Enhanced PB-PKI provides to the victim. Key compromise happens when an adversary gets access to or steals the private part of the online or the offline key. Getting access to a key means an adversary knows a key while the victim still has access to the key too. Stealing a key means an adversary takes the key away from the victim so that the adversary knows the key while the victim does not.

8.1 Security Model

We consider a model where PB-PKI is accessible to the public. In this model, we assume that an adversary has access to the blockchain. The adversary is also part of every operation involving registration, key update and key revocation and able to tamper with any of these operations.

The security goals of the system are as follows:

1. The adversary cannot link a key to the owner.
2. Only the owner of a public online key can prove the ownership.

Depending on which private keys were stolen or accessed, we consider six different cases:

1. **Online secret key accessed only:** The adversary can take part in the network by using the accessed online key. The victim can revoke his online key and update it. The update creates a new online key he can safely use again.
2. **Online and offline secret keys accessed:** The adversary can take part in the network by using the accessed online key. The adversary can also perform key update, but this is not useful because when the user revokes the online key and performs key update, then the adversary can no longer impersonate the victim. After key update, the adversary does not have any knowledge of either the new online or offline key.
3. **Online secret key stolen only:** The adversary can take part in the network by using the stolen online key. To recover from this case, it requires that the victim stored the previous online key. With this previous online key and the current offline key, he can recalculate the stolen online key. Then he revokes and updates the stolen key.

4. **Online and offline secret keys stolen:** The adversary can take part in the network by using the stolen online key. In this case, the victim can do nothing against the adversary. However, the victim can update the previous online key, and stay part of the network. The possibilities for the adversary are limited to the use of the stolen online key only. To mitigate this scenario, the introduction of an expiration date of public keys would limit the key abuse of the adversary to a certain time frame.

As long as keys are not stolen and key compromise is detected early, an adversary can be kept under control. To avoid key theft, keys should always be copied to a safe location.

9 Conclusion

In this paper, we have shown that the initial proposal of PB-PKI is fraught with challenges. Some of the problems include authentication during key update, how revoked keys can be deleted successfully and the key update mechanism is not as secure as earlier expected. Enhanced PB-PKI simplifies the update process by requiring the user to generate fresh keys without using the key updates mechanism by Louise Axon et al. [1]. The new key update mechanism removed the privacy problem that was introduced by PB-PKI . The Enhanced PB-PKI solved the problem of user authentication by introducing ring signatures to PB-PKI. We also adapt the use of AA in order to aid key deletion and ensure that storage space is maximized after deletion of a revoked key. The Certcoin usage of offline keys for key revocation was retained instead of using it for key updates as in the initial PB-PKI construction. We developed the proof of concept using the Go programming language and show that the solutions we proposed are feasible.

In the future, we plan to implement optimizations to the ring signature scheme. New ring signature schemes such as forward-secure linkable ring signatures [3] will also be investigated as this supports forward security and the blockchain user can prove ownership of a ring signature using this scheme.

Acknowledgments

We acknowledge David Derler, Sebastian Ramacher, Peter Lipp and Clemens Brunner for fruitful discussions during the period of this work. We acknowledge Sebastian Ramacher, Peter Lipp and Clemens Brunner for their thorough reviews.

This research is part of the LIGHTest project funded by the European Union's Horizon 2020 research and innovation programme under G.A. No 700321.

References

1. L. Axon and M. Goldsmith. Pb-pki: A privacy-aware blockchain-based pki. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications - Volume 6: SECRYPT, (ICETE 2017)*, pages 311–318. INSTICC, SciTePress, 2017.
2. D. Boneh. Twenty years of attacks on the rsa cryptosystem. 1998.
3. X. Boyen and T. Haines. Forward-secure linkable ring signatures from bilinear maps. *Cryptography*, 2(4):35, 2018.
4. S. Y. Conner Fromknecht, Dragos Velicanu. Certcoin: A namecoin based decentralized authentication system, 2014.
5. D. Derler, C. Hanser, and D. Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, pages 127–144, 2015.
6. J. A. Garay, A. Kiayias, N. Leonardos, and G. Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, pages 465–495, 2018.
7. E. Karaarslan and E. Adiguzel. Blockchain based DNS and PKI solutions. *IEEE Communications Standards Magazine*, 2(3):52–57, 2018.
8. S. Matsumoto and R. M. Reischuk. IKP: turning a PKI around with blockchains. *IACR Cryptology ePrint Archive*, 2016:1018, 2016.
9. P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.
10. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
11. L. Reyzin and S. Yakoubov. Efficient asynchronous accumulators for distributed pki. *IACR Cryptology ePrint Archive*, 2015:718, 2015.
12. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 552–565, 2001.
13. A. Singla and E. Bertino. Blockchain-based PKI solutions for iot. In *4th IEEE International Conference on Collaboration and Internet Computing, CIC 2018, Philadelphia, PA, USA, October 18-20, 2018*, pages 9–15, 2018.
14. A. Yakubov, W. M. Shbair, A. Wallbom, D. Sanda, and R. State. A blockchain-based PKI management framework. In *2018 IEEE/IFIP Network Operations and Management Symposium, NOMS 2018, Taipei, Taiwan, April 23-27, 2018*, pages 1–6, 2018.
15. P. Zimmermann. Pgp user's guide, volume i: Essential topics, 1994.

4. An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources

The subsequent paper has been published as follows:

Bertil Chapuis, Olamide Omolola, Mauro Cherubini, Mathias Humbert and Kévin Huguenin. "An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources". In: *WWW*. ACM / IW3C2, 2020, pp. 34–45

In this paper, we perform the first empirical study of the use of Subresource Integrity (SRI) for web subresources. We conduct a large-scale measurement as well as a user study to understand the use of SRI in the wild.

Abstract. Web developers can (and do) include subresources such as scripts, stylesheets and images in their webpages. Such subresources might be stored on content delivery networks (CDNs). This practice creates security and privacy risks, should a subresource be corrupted. The subresource integrity (SRI) recommendation, released in mid-2016 by the W3C, enables developers to include digests in their webpages in order for web browsers to verify the integrity of subresources before loading them. In this paper, we conduct the first large-scale longitudinal study of the use of SRI on the Web by analyzing massive crawls (≈3B URLs) of the Web over the last 3.5 years. Our results show that the adoption of SRI is modest (approx. 3.40%), but grows at an increasing rate and is highly influenced by the practices of popular library developers (e.g., Bootstrap) and CDN operators (e.g., jsDelivr). We complement our analysis about SRI with a survey of web developers (N=227):It shows that a substantial proportion of developers know SRI and understand its basic functioning, but most of them ignore important aspects of the recommendation. The results of the survey also show that the integration of SRI by developers is mostly manual – hence not scalable and error prone. This calls for a better integration of SRI in build tools.

Keywords: web security; subresource integrity; common crawl

1 Introduction

The Web is a set of interlinked resources identified by their URLs. A significant portion of these resources consists of HTML webpages that include navigable links and subresources, such as scripts, stylesheets, images or videos. A change in a subresource can affect the webpage that includes it.

With the advent of content delivery networks (CDNs), an increasing number of subresources are hosted by third-party providers (in this paper, we will refer to these as *external subresources*, also called cross-domain subresources). The advantages provided by such platforms include reduced costs and latency as well as increased reliability. However, their usage comes at a security price: A subresource can be altered (accidentally or not) upon transmission from these third-party providers or directly on them, as it was the case for the British Airways in 2018 Eden (2018). The consequences can be dramatic, including the theft of user credentials (i.e., on login pages) and credit card data (i.e., on payment pages), malware injection, and website defacement (i.e., modification of the content). In general, when an external subresource is included in a webpage, there is no guarantee that its content will remain the same.

The subresource integrity (SRI) recommendation W3C (2016), released in mid-2016 by the W3C, addresses this issue by enabling web developers/webmasters to include digests in order for web browsers to verify the integrity of subresources before loading them. SRI is implemented in the vast majority of desktop and mobile browsers. Unfortunately, no in-depth analysis about the adoption and the understanding of SRI (by web developers) has been performed so far. Most existing works Shah and Patil (2018); Kumar et al. (2017) focus on modest-size datasets of webpages, focus on one snapshot only, and only look at the basic statistics, e.g., they do not study the main *factors* behind the adoption/usage of SRI. Our work fills this gap (i) by conducting the first large-scale longitudinal study on the adoption/usage of SRI on the Web, and (ii) by surveying web developers regarding their understanding and usage of SRI.

Contributions. By relying on a massive dataset of about 3B URLs, we first thoroughly analyze the use of the SRI recommendation on the Web over the last 3.5 years. We began our analysis in May 2016, right before the official release of the SRI recommendation and took a snapshot approximately every six months. Our analysis of the 3B webpages shows the following: first, we measure the extent of the most typical threat, i.e., the use of external subresources, and we find that more than 80% of the webpages include at least one external subresource; second, we study the adoption and usage of SRI over time and observe an increasing, but still modest usage, from less than 1% in October 2017 to 3.40% in September 2019. We observe that the use of SRI is linked to a number of popular libraries and CDN operators, including Bootstrap and jsDelivr, that provide snippets of code that use SRI for including their subresources.

Given the results of our large-scale analysis of SRI adoption and related security mechanisms, we evaluated the extent of knowledge of web developers about SRI. To do so, we conducted an online survey with 227 respondents asking if web developers are aware of the risks, are aware of SRI, to what extent they understand the implementation of SRI, and what their current practices are when using SRI. Particular care was put in the survey instrument that was designed and tested iteratively before deployment. Panelists were recruited among the Wordpress and NPM package repository contributors. Our results show that about two thirds of the respondents could identify the main threats in subresource usage, such as malicious code injection or cross-site scripting, but most of them ignore some important aspects of SRI implementation, e.g., the cases where the digests used in SRI are malformed or multiple. They also show that the integration of SRI by developers is mostly manual – hence

not scalable and error prone. This calls for a better integration of SRI in build tools.

2 System and Threat Model

We consider a webpage hosted on a given server (`www.server.com`). The webpage includes a number of subresources such as scripts (JavaScript, through a `script` element), stylesheets (CSS, through a `link`), and images (through an `img`). The inclusion is done *by reference*, i.e., the subresources are not copied in the webpage. The subresources can be internal (stored on the same server as the webpage, possibly in a different volume or folder, e.g., `scripts/`) or external/cross-domain (stored on a different server, e.g., `www.otherserver.com`). Typical external subresources include subresources found on another website or hosted on a mirror or content delivery network for reliability and performance reasons (e.g., the jQuery library). This last case opens the door to cross-domain script inclusion risks. When a user visits the webpage, the browser first fetches the webpage from the server and then it fetches its subresources. Finally, it renders the webpage to the user.

We consider the threat where the content of the subresource is altered, meaning that it did not correspond to the initial content of the subresource when it was included in the webpage (i.e., what the web developer intended to include). Such a situation can occur in multiple cases: (i) because the communication channel between the client and the server was compromised by an adversary (e.g., an Internet service provider), (ii) because the storage of the server was compromised by an adversary (e.g., a hacker who broke into the server or a malicious mirror operator), or (iii) simply because the subresource was changed by its maintainer. A subresource integrity threat can have important consequences. For instance, a corrupted script can – among other things – compromise the visitors' devices (e.g., by redirecting them to a malicious website), steal their private data (e.g., passwords and credit card information), track them, or shock them (and compromise the reputation of the author) by changing the content of the webpage (i.e., defacing). Figure 1 depicts the system and threat model for subresource integrity.

3 The W3C SRI Recommendation

SRI W3C (2016) enables web developers to specify an integrity attribute for some types of subresources they include in their webpages, in such a

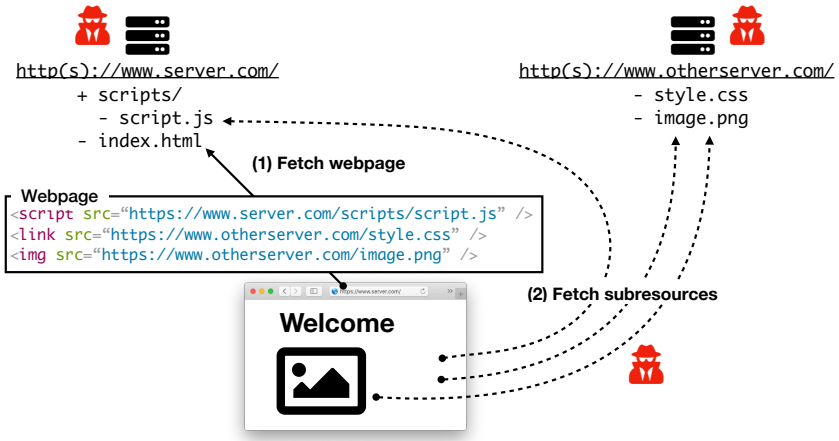


Fig. 1. System and threat model for subresource integrity.

way that the user agent can verify their integrity before loading them. This guarantees that the content of the subresources corresponds to what the developers intended to include, specifically that it has not changed. As of September 2019, SRI covers `script` (i.e., JavaScript) and `link` (i.e., CSS) elements and it is fully supported by Chrome, Firefox, Opera, Safari, and partially by Edge. A typical use of SRI is as follows (in the `head` section of the HTML page):

```
<script src="https://www.server.com/script.js"
        integrity="sha256-47D...␣sha512-8HB..." />
```

An integrity attribute contains one or multiple space-separated hash expressions. Each hash expression is composed of the name of a hashing algorithm (i.e., `sha256`, `sha384` or `sha512`) and a base64-encoded digest generated with the corresponding algorithm. The content of a subresource is said to match a hash expression if the digest of the subresource is equal to the digest specified in the expression. When rendering the HTML snippet above, the browser first fetches the subresource (i.e., `script.js`). The browser tries to match the content of the subresource to the different hash expressions specified in the integrity attribute and loads the subresource according to the following rules: (1) When the attribute contains a *single* hash expression, the subresource is loaded if it matches it; (2) when the attribute contains *multiple* expressions generated with the *same* hash algorithm, the subresource is loaded if it matches one of them; and (3) when the attribute contains *multiple* expressions generated with *different* hash algorithms, the subresource is loaded if it matches one of the hash expressions with the *strongest* digest (`sha512` ¿ `sha384` ¿ `sha256`). If the

integrity attribute is empty or malformed, the subresource is loaded nevertheless. Note that since SRI "fixes" the content of the subresources, it is not appropriate for subresources that can change (e.g., latest version of a library).

Content Security Policy (CSP) directives (specified in HTML `meta` elements or HTTP headers, e.g., `Content-Security-Policy: require-sri-for script;`) can be used to force web developers to specify a *valid* integrity attribute for each subresource. In this case, subresources without an integrity attribute or with a malformed one are *not* loaded. Such a mechanism enables the separation of concerns between web developers and system administrators. More recently, these directives are being abandoned by Web browsers and their removal from the recommendation is scheduled Braun (2019).

4 Large-Scale Analysis

In this section, we report on the large-scale analysis of the use of SRI on the Web. We describe our data sources and methodology and then report on the results.

4.1 Data sources

We rely on two main data sources: large-scale crawls of the web and a popularity-based ranking of domains.

Web Crawl: Common Crawl. The Common Crawl (*CC*) dataset is a collection of snapshots of the Web Common Crawl (2019a). It contains one snapshot per month, since 2011-01. Each snapshot is available in the WARC format, which contains the raw data of the webpages (HTTP headers and HTML content but not the content of the subresources). The latest snapshot of Common Crawl (2019-09) contains 2,954,836,069 URLs Common Crawl (2019b). Note that the number of URLs for a given website does not necessarily reflect its number of webpages, rather it depends on its architecture (i.e., single page vs. multi-page applications Mikowski and Powell (2013)). Each snapshot (in each of the format) is divided into multiple archives so as to enable parallel and distributed processing of the dataset. The *CC* dataset is publicly available on `Amazon S3`, where it can be processed and analyzed using `Amazon EMR`.

Domain Name Ranking: Cisco Umbrella 1 Million (Top1m). The Cisco Umbrella 1 Million dataset (*Top1m*) ranks popular domain names based on statistics on DNS queries (≈100 billion requests/day) and client IPs (≈65 million unique users) across 165 countries Hubbard (2019).

4.2 Methodology

To study the use of SRI, we parse the HTML content of the webpages and identify the subresources included in webpages. Parsing the content of all the webpages contained in a snapshot of *CC* is time consuming. As our analysis focuses mainly on the use of SRI, we rely on a simple filter to detect webpages that include subresources with an integrity attribute. More specifically, we keep only the webpages that contain the string "integrity=". Note that this filtering is done on the *static* (i.e., returned by the server, without any client-side manipulation such as JavaScript execution) *raw* (i.e., in the binary format, that is before decoding) content of the webpages; this can lead to false negatives (i.e., filtering out webpages that do include subresources with an integrity attribute). We further detect the encoding of the webpages and parse them by using the Python `beautifulsoup` library (v4.7.1) with the `lxml` parser (v4.3.3). This enables us to extract the subresources of the webpages (and their attributes) and to filter out the webpages that do *not* include any subresource with an integrity attribute. Indeed, some webpages might not have been filtered out because they do contain the string "integrity=", but somewhere else in the webpage, e.g., in the text of the webpage on SRI on the W3C website. This constitutes the set of webpages on which we conduct our analysis, specifically the "webpages that contain at least one SRI" set (*CC-SRI*). For each webpage in the *CC-SRI* dataset, we extract its URL, content security policy (CSP) (from the HTTP header) and all its `link` and `script` elements/subresources. In order to study not only the current use of SRI but also its evolution over time, we process a total of 13 *CC* snapshots from 2016-05 (i.e., before the release of the SRI specification in late June 2016) to 2019-09 (more specifically 2016-05, 2016-10, 2017-02, 2017-05, 2017-08, 2017-11, 2018-02, 2018-05, 2018-08, 2018-11, 2019-03, 2019-06, 2019-09). In order to study the use of subresources on the web in general (not only for pages that do use SRI), we also extract a sample (*CC-all-1%*) that contains 1% of the latest snapshot (2019-09) of the *CC* dataset. For reproducibility purposes, the source code of our analysis scripts is available online at `https://github.com/isplab-unil/cc-sri`.

4.3 Results

We now present the results of our analysis of the use of SRI, based on the *CC* dataset.

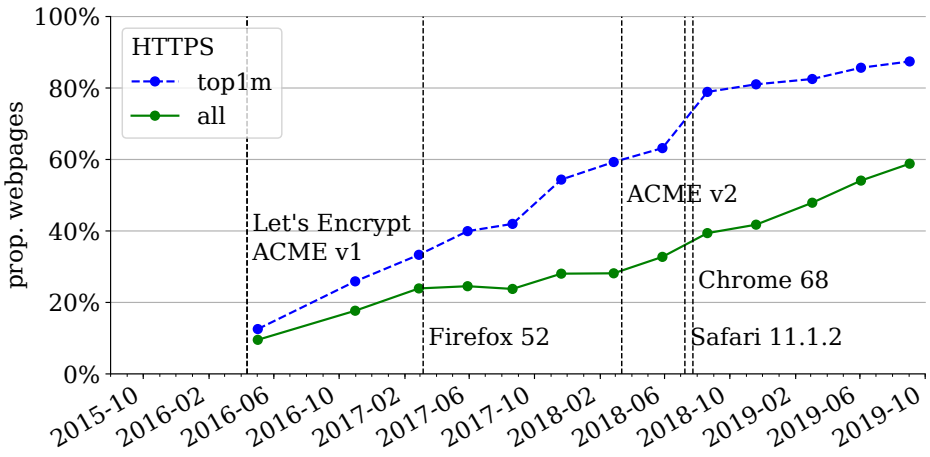


Fig. 2. Evolution of the proportion of webpages served over HTTPS (in *CC*).

HTTPS Adoption The TLS protocol used in HTTPS provides, in addition to authentication, channel integrity. As such, it is related and sometimes complementary to SRI that provides channel and storage integrity (i.e., it also protects against adversaries who tamper with the content stored on the external server). Therefore, we start by measuring the adoption of TLS/HTTPS, in the *CC* dataset. We identify the protocol used (i.e., HTTP vs. HTTPS) based on the URL of the webpage.

Fig. 2 depicts the evolution of the proportion of webpages served over HTTPS for the *CC* dataset and its subset (*Top1m*). It also features the important milestones in the development and deployment of HTTPS, including the release of `Let's encrypt` and of the `ACME` protocols v1 and v2 (which respectively automate the generation/distribution of certificates and the deployment of PKIs) as well as the introduction of security warnings for non-HTTPS webpages in major web browsers.

In the latest *CC* snapshot (2019-09), 58.82% of the webpages are served over HTTPS. The use of HTTPS is substantially higher among the most popular webpages (87.43% in *Top1m*).

Extent of the Threat We measure the extent of the most typical threat to subresource integrity: the case where a webpage includes an external subresource. To do so, we compute the proportion of webpages that include at least one external subresource, in the *CC-all-1%* dataset. A subresource is called external if the host that serves it (in terms of its fully qualified domain name, e.g., www.otherserver.com) is *different* than the host that serves the webpage. Note that this is a heuristic: A same

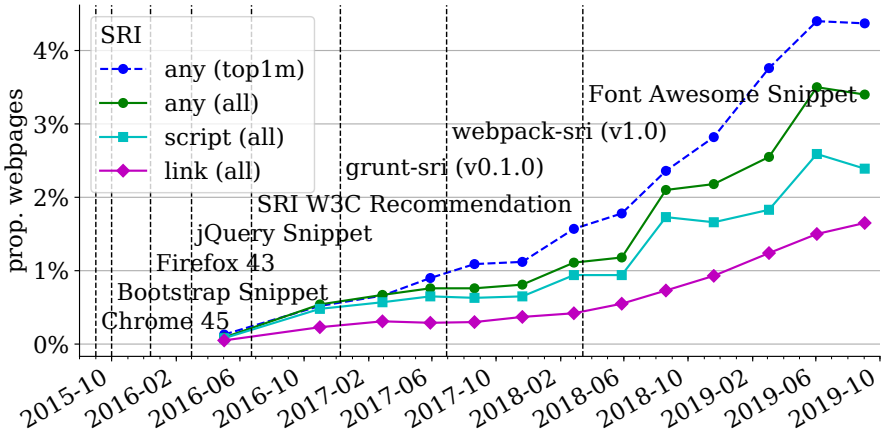


Fig. 3. Evolution of the proportion of webpages containing subresource(s) w/ an integrity attribute (in *CC*).

hostname can point to different servers (e.g., reverse proxy) and different hostnames can point to the same server. We found that 82.76% of webpages include at least one link or script external subresource[1] and that these webpages include on average 8.24 ± 14.71 such subresources. All these webpages are potentially exposed to threats to subresource integrity and can benefit from SRI (some do, as explained below). For images (i.e., `img` elements), which are common subresources but not covered by SRI, the proportion of webpages including an external image is 54.37%.

SRI Adoption We measure the adoption of SRI by counting the proportion of webpages that include *at least* one subresource with an integrity attribute (i.e., the size of the *CC-SRI* dataset). We further distinguish between the types of subresources: link and script, which are the only covered by SRI. Figure 3 depicts the evolution over time of the proportion of webpages containing at least one element (link, script or any) with an integrity attribute. It also features some important milestones in the development and adoption of SRI; in particular, we include the dates at which some popular libraries (e.g., Bootstrap) began to include integrity attributes in the code snippets provided on their webpages, typically in the "Quick Start" section. Note that CDNs hosting popular libraries (e.g., jsDelivr) also include such snippets.

[1] We focus our analysis mainly on link and script elements as these are the only covered by the SRI recommendation at the moment. We discuss this in Sections 5 and 7.

It can be observed that some websites began to use SRI before the recommendation was officially released. This can be explained by the fact that the draft of the recommendation was available before its release and, more importantly, some libraries (e.g., Bootstrap) included integrity attributes in their code snippets as early as in late 2015 (see Table 1). Additionally, SRI was implemented in some web browsers before its release, as early as 2015-09 for Chrome (v45) and 2015-12 for Firefox (v43), because developers from both Google and Mozilla were involved in the edition of the SRI recommendation.

We find that the overall adoption of SRI is modest, with only 3.40% of all webpages in *CC*, but it grows at an increasing rate (the increase in 2018 is twice as large as in 2017). The adoption of SRI is highly influenced by the inclusion of the integrity attribute in code snippets provided by library developers on their websites. Another factor that could accelerate the adoption of SRI is the automatic inclusion of integrity attributes by build tools; we discuss this in Section 7. Although this cannot be directly concluded from Fig. 3, it becomes clear when analyzing the targets of the subresources with an integrity attribute (as explained below). As hosting subresources on third-party servers comes with risks that major websites are probably reluctant to take, the adoption of the SRI recommendation by the *Top1m* websites is faster than for the rest of the Web. As mentioned above, the adoption of SRI is influenced by library developers and CDN operators (this is confirmed by the results of our survey of web developers, as a large fraction of developers report including integrity attributes by copy-pasting snippets; see Section 5). In order to better understand this point, we analyze (1) the main subresources (i.e., libraries, add-ons) used on the web and whether the corresponding websites promote SRI and (2) the main domains hosting subresources for which SRI is used.

Subresources. We compute the list of the most popular subresources in the *CC-all-1%* dataset and check whether they include code snippets and – if yes – whether these snippets use SRI (i.e., include an integrity attribute). For snippets that use SRI, we determine the date at which they began to do so by relying on the Wayback Machine, an online archive of the Web Internet Archive (2019). As the same subresources (e.g., JQuery) can be hosted on multiple domains with different URL formats, we devised a heuristic to identify them. The URL of the subresource is contained in the `target` attribute of the element. We grouped the subresources by domains and selected the top-100 domains (in terms of number of URLs); the top-100 covers 62.71% of the subresources present in *CC-all-1%*. For these domains, we manually identify patterns and build regular expressions

Rank	Prop.	Name	Resources	Type	Snippet	SRI	Adoption
1	11.72%	Google Syndication	JS	Add-on	✓		
2	6.39%	jQuery	JS/CSS	Library	✓	✓	Mar.16
3	5.04%	Wordpress	JS/CSS	Platform			
4	2.62%	Google APIs	JS	Add-on	✓		
5	2.40%	Blogger	JS/CSS	Platform			
6	2.21%	FontAwesome	CSS	Library	✓	✓	Mar. 18
7	1.41%	TripAdvisor CDN	JS/CSS	Platform			
8	1.38%	Twitter	JS/CSS	Add-on	✓		
9	1.29%	SmugMug	JS/CSS	Platform			
10	1.21%	Squarespace	JS/CSS	Platform			
11	1.21%	Bootstrap	JS/CSS	Library	✓	✓	Oct.15
12	1.19%	WIX	JS/CSS	Platform			
13	1.13%	Google Ad Services	JS	Add-on	✓		
14	0.94%	Google Tag Services	JS	Add-on	✓		
15	0.94%	jQueryUI	JS/CSS	Library	✓	✓	Mar.16

Table 1. Most popular subresources (in *CC-all-1%*).

to extract the names of the subresources from the URLs (e.g., "`https://cdn.jsdelivr.net/npm/jquery@3.2.1/dist/jquery.min.js`").

Table 1 lists the top-15 subresources (in terms of number of inclusions) found with our heuristic. It can be observed that a few subresources account for a substantial portion of the subresources in *CC-all-1%*, e.g., 11.72% for Google Syndication and 6.39% for jQuery. Note that some popular subresources, such as those of the Facebook ad network, do *not* appear in our results. This is because these subresources are loaded asynchronously using JavaScript. We observe that only a few subresources include snippets with SRI. In particular, add-ons do not use SRI; this is because the corresponding subresources are often transparently updated (i.e., without changing the URL: `http://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js`) by the providers, as noted by Lauinger et al. (2017). SRI is not well suited for such subresources.

Domains. We compute, in each snapshot of the *CC-SRI* dataset, the popularity of domains in terms of the number of subresources *with* an integrity attribute that are hosted on the considered domain (e.g., based on the URL in the `target` attribute). Figure 4 depicts the evolution over time of the top-10 domains found in the *CC-SRI* dataset. The top-10 domains cover 94.23% of all the subresources with an integrity attribute in *CC-SRI*. All the domains, except `Shopify`, provide snippets with SRI on their websites. Shopify does not because it is a platform, not a library included in other websites; but it uses SRI for its platform, which is very popular.

SRI Usage We analyze the current practices of web developers when they use SRI.

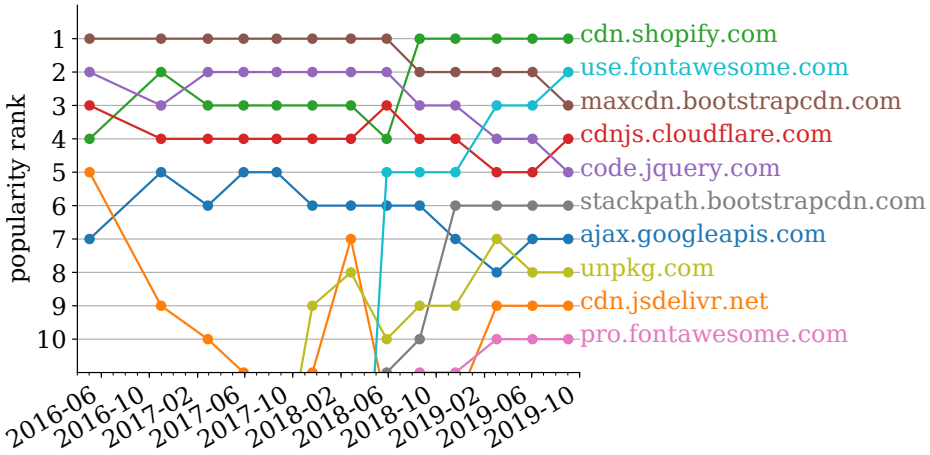


Fig. 4. Evolution of the top-10 domains targeted by subresources with an integrity attribute (in *CC-SRI*).

Number of Subresources. Even though a webpage uses SRI for some of its subresources, it does not necessarily use SRI for all of them. We compute the number of resources with and without an integrity attribute in the snapshots of the *CC-SRI* dataset. Fig. 5 depicts the number (mean and standard deviation) of subresources per webpage with the integrity attribute and the total number of subresources per webpage (w/ or w/o an integrity attribute). In the latest snapshot, webpages contain an average of 41.61 subresources and the number of subresources varies highly across webpages. The average number of subresources per webpage with the integrity attribute is much lower at 1.79.

Hash Algorithms. We compute the distribution of the hash algorithms (e.g., sha384) and of the number of hash expressions (i.e., digests) used in integrity attributes (for `link` and `scripts` elements, without distinctions), in the latest snapshot. To do so, we parse all the integrity attributes according to the format specified in the SRI recommendation. If the parsing fails, the attribute is considered malformed. In our analysis, we distinguish between malformed and empty attributes even though in practice, both are loaded by the browser (unless a CSP directive specifies otherwise). For the well-formed attributes, we extract the hash algorithm and label the attribute accordingly. When an integrity attribute contains multiple digests, e.g., sha256 and sha384, we label it with the different algorithms sorted by increasing strength (e.g., "sha256+384"): We use different labels for the different combinations of hash algorithms. Because a large proportion of web developers simply copy-paste snippets

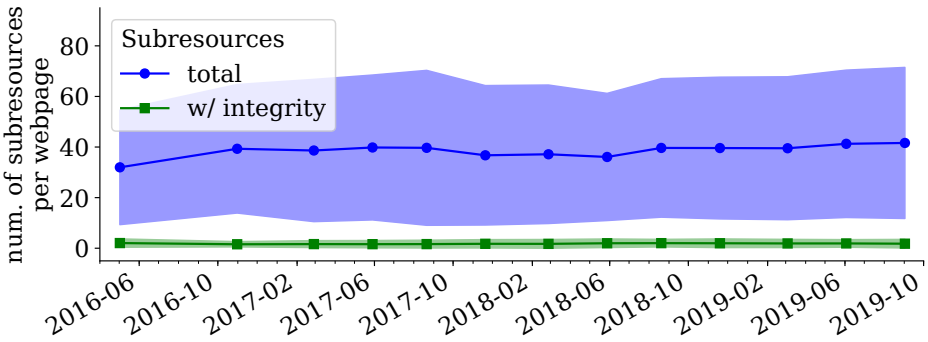


Fig. 5. Mean and standard deviation of the number of subresources (all or with an integrity attribute) per webpage (in *CC-SRI*) that contains at least one subresource w/ an integrity attribute.

from websites (e.g., libraries and CDNs), the observations on the usage of SRI apply to a large extent to the developers of the included libraries and to the operators of the CDNs.

Figure 6 depicts the distribution of the hash algorithms across all the subresources with an integrity attribute. Most integrity attributes contain a single digest, and the most popular algorithms are sha384, sha256 and sha512 (in that order). Only 1.16% of the integrity attributes contain more than one digest with *different* hash algorithm. Note that, as browsers consider only the digests generated with the *strongest* hash algorithms and as the browsers that support SRI all support all the hash algorithms, such a practice does not make sense in practice. A possible explanation is that some web developers misunderstood how the case of multiple digests is handled by browsers (this is confirmed by the results of our survey; see Section 5): They might have (erroneously) thought that having more than one digest increases the security of the integrity verification. We observed even fewer (0.0004%; they are part of the "other" bar of the histogram) integrity attributes that contain more than one digest with the *same hash* algorithm (e.g., two sha256 digests). Such a practice enables developers to support multiple versions of a subresource by including the digest of each version (with the same hash algorithm); though convenient, this practice is very marginal.

For the malformed integrity attributes (1.41%), we manually investigate them; the causes include: missing hash algorithms, unsupported hash algorithms (i.e., "md5"), mistyped hash algorithms (e.g., "ha256"), and (possibly failed) injection through templates (e.g. "{{CHECKSUM}}"). This last example might be valid if the value of the attribute is correct

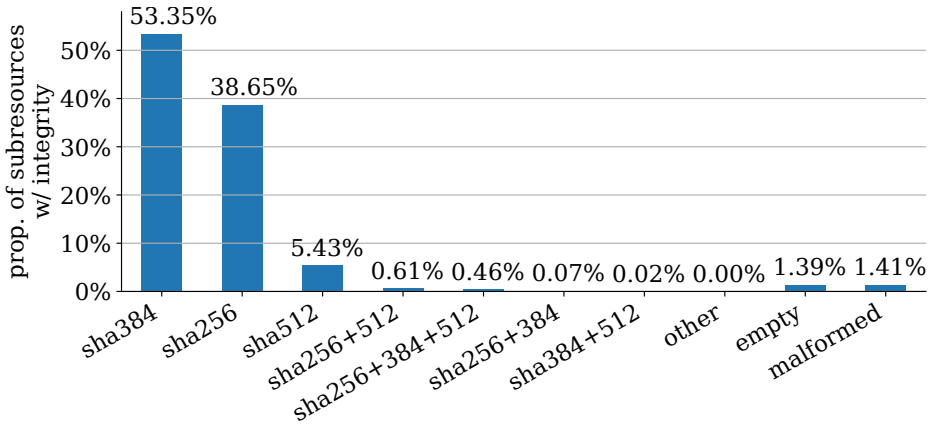


Fig. 6. Distribution of the hash algorithms used in the integrity attributes (in latest snapshot of *CC-SRI*). An integrity attribute can contain more than one digest.

and indeed inserted at the client before the verification is made by the browser. Yet, we manually tested these templates and none of them was properly rendered.

Finally, we looked at the distribution of hash algorithms for two popular libraries for which the snippets of code provided on their websites use different hash algorithms: sha-256 for JQuery and sha-384 for Bootstrap. In the *CC-SRI* dataset, 86.21% of the integrity attributes for the jQuery library (hosted on the jQuery CDN) use sha-256 and 98.14% of the integrity attributes for the Bootstrap library (hosted on the bootstrap CDN) use sha-384. This suggests that the snippets of code are often simply copy-pasted.

Protocols and Paths. Webpages that include subresources can be exposed to different adversaries and associated threats. More specifically there are four possible threats: Alteration of the webpage/subresource on the server/communication channel (see Section 2). Depending on the considered setting and adversary, SRI and TLS (i.e., HTTPS) can offer protection to the security of the webpage. We categorize the different settings using three criteria: whether the *webpage* is served using TLS (i.e., HTTP vs. HTTPS), whether the *subresource* is served using TLS, and whether the path to the subresource is local or external (i.e., "scripts/script.js" vs. "https://www.cdn.com/script.js"). Note that the protocol is sometimes omitted (e.g., "//server.com/script.js"), thus inherited: If TLS is used for serving the webpage, it is used for serving the subresource, otherwise, it is not used either for the subresource. We build the complete path of the subresources (by combining the URL of the webpage with that of the target of the subresource by using `urljoin` from Python's standard

library) and analyze the breakdown between the different aforementioned settings, in the latest snapshot of *CC-SRI*. We also observed that protocol inheritance (i.e., //) is substantially used, especially in webpages served over TLS: 27.61% of the subresources served over TLS and included in webpages served over TLS are specified with protocol inheritance.

We observed that most of the subresources use absolute URLs (i.e., http://, https://, and //) for specifying the path in the target attribute, which are usually used for external subresources. Figure 7 depicts the breakdown in the form of a tree; the levels of the tree correspond to the following criteria: (1) webpage protocol, (2) subresource protocol, and (3) locality. The typical use case (i.e., HTTPS/HTTPS/external) is the most frequent. SRI is particularly meaningful when the webpage is served over TLS as the integrity attribute of its subresources is protected upon transmission. This represents 85.84% of the settings. In this setting, it makes even more sense when subresources are served without TLS, as SRI protects against corruption on the server *and* on the channel (i.e., 0.01% of the settings). Yet, such a practice (i.e., including a subresource served without TLS in a webpage served with TLS) is *not* allowed by browsers (i.e., *mixed content* error) – for valid security reasons – and the subresource will therefore *not* be loaded. Yet, this is marginal. When the webpage is served without TLS (14.16% of the settings), the integrity attributes of the subresources are not protected upon transmission and the security of the webpage is not guaranteed. Yet, assuming that the host and the channel for serving the webpage are not compromised, SRI provides protection against corruption on the server and/or channel serving the subresources.

Although SRI is meant primarily for securing the integrity of external subresources, its use for local subresources still makes sense (i.e., if the subresource is hosted on another server through a reverse proxy or if only some files on the server could be corrupted by the adversary) and should not be interpreted as erroneous or meaningless (Salvador et al. (2018) discuss this point in detail). It could be the result of build tools that automatically compute and insert integrity attributes even for local subresources.

Use of the require-sri-for Directive We compute, in the latest snapshot of *CC-SRI*, the proportion of webpages for which the `require-sri-for` CSP directive is specified in their HTTP headers: It is the case for only 0.02% of the webpages. We manually tested a small sample of webpages with this directive hosted on different domains: 8% of the subresources

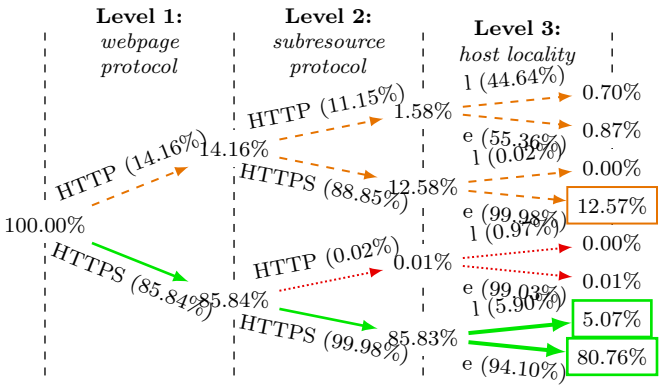


Fig. 7. Categorization of subresources w/ an integrity attribute (in latest snapshot of *CC-SRI*) per webpage protocol (HTTP or HTTPS), subresource protocol (HTTP or HTTPS) and host locality (local, external). Frequent settings are framed. Secure settings are depicted with solid green lines; partially secure settings with dashed orange lines and problematic settings with red dotted lines.

were blocked. It is marginal and its (scheduled) removal from the SRI recommendation will affect only a tiny fraction of the Web. Yet, we believe it should be maintained as it enables system administrators to enforce security policies.

5 Web Developer Experience

Given the results reported in the previous section, we decided to study the level of awareness and understanding of web developers regarding the SRI recommendation. Therefore, we posed the following research questions:

- **RQ1**. Are the web developers aware of the risks associated with (external) subresources?
- **RQ2**. Are web developers aware of SRI?
- **RQ3**. To what extent do web developers understand the implementation of SRI (general and specific behaviors)?
- **RQ4**. What are the current practices of developers when using SRI? (i.e., are they coherent with the recommendation?)

To answer these questions, we conducted an online survey of web developers. We adapted our methodology from similar surveys Acar et al. (2016); Balebako et al. (2014). The panelists were recruited via e-mail then went through quality controls. Panelists that completed the survey participated

in a raffle for USD 100 Amazon vouchers. Next, we describe the method used to design the survey instrument, recruit participants, analyze the data as well as the deployment strategies we adopted to distribute the survey. The study was approved by our IRB.

5.1 Design of the Survey Instrument

The questionnaire contained 32 items, organized into four sections.[2] (i) The first section contained two screening questions to make sure the respondent was comfortable reading and writing in English (the language of the questionnaire) and that the respondent was indeed an active web developer. (ii) The second section focused on the respondent's awareness of the threat model described in this paper (providing data for RQ1) and the SRI recommendation (cf. RQ2). This section had a skip logic: Respondents with no knowledge of SRI were brought to the fourth section. (iii) The third section contained 4 quiz questions designed to assess the respondent's understanding of the SRI recommendation (cf. RQ3) (note that all major browsers – i.e., Chrome/Firefox/Safari/Opera/Edge – strictly follow the recommendation for these quiz questions) and questions to understand how they used SRI in their work (cf. RQ4). (iv) The last section contained questions about the company the respondent works for and their demographic information.

On the last page of the questionnaire, we asked the respondents whether they wanted to receive a summary of the results of the research; 85.4% of respondents opted in to receive a follow-up, thus revealing the general interest in this topic. Furthermore, we clarified the goal of the research and provided a reference to the SRI recommendation, in case respondents were interested to learn more. It took about 10 minutes to complete the questionnaire.

To eliminate possible presentation effects, the answer options of multiple-choice questions were randomized. Before deploying the questionnaire, we conducted four pre-tests that involved individuals at our institution (including some native English speakers). One of the authors sat with the participants and, for each question, asked the participant to re-state, in their own words, what the question asked and how they would answer. Feedback provided at this stage was used to adjust wording and provide additional context.

[2] The full questionnaire is available online: `https://osf.io/yshrx/`.

5.2 Data Reliability and Coding Process

To ensure high data reliability, several quality-assurance (QA) processes were followed when administering the survey instrument: *speeders* and *straightliners* were removed before the analysis. One of the authors went through the open-ended responses and removed respondents that provided answers to open-ended questions that were nonsensical. For open-ended questions, we opted for collaborative coding on the qualitative responses Saldaña (2015). For each question, a codebook was developed iteratively by a lead coder who analyzed an initial set of answers (i.e., ≈100). For the next step, a second coder independently coded the data again using the same codebook. Cohen's kappa (or κ) was used to measure inter-coder agreement to each open-ended question. The average κ value was 0.83 (std 0.18), which was judged sufficient to proceed further with the analysis. Next, the cases of disagreement between the two coders were resolved through discussion MacQueen et al. (1998).

5.3 Deployment Strategies

To reach to the web development community, the questionnaire was disseminated to e-mail addresses obtained as follows:

1. *Wordpress plugin/theme authors.* Wordpress is a web content management system (CMS) based on PHP, JavaScript, HTML and CSS. Any developer can create plugins/themes for Wordpress. The "readme" file of Wordpress plugins/themes often contains the e-mail of the authors. We considered a sample of N≈9500 e-mails from Wordpress.
2. *NPM package authors.* NPM is a repository for JavaScript packages. These packages were developed to be used in web applications or websites. Each package contains a *package.json* file that optionally provides the e-mail of the package developer. We considered a sample of N≈19,000 e-mails from NPM.

Selected web developers were sent an e-mail invitation to fill out the survey. The e-mail contained the following information: the academic research goal of the questionnaire (described as "understanding web development practices" in order to not prime the respondents towards security), the conditions for participation, the incentive, information about data management and anonymity of the responses, contact information of the researchers, and the link to the survey. As our e-mails were unsolicited, we gave them the opportunity to opt out, and we did not send any

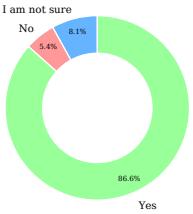
form of reminders. We did not collect any respondent personal identifiable information and did not link their responses to their e-mail. We sent a total of 28,500 e-mails (in 2 batches) and received a total of 477 responses in Sep. 2019. After applying the QA processes described in Section 5.2 and removing incomplete answers, we were left with 227 valid responses. The results reported in the rest of the paper are based only on the valid responses.

5.4 Demographics and General Statistics

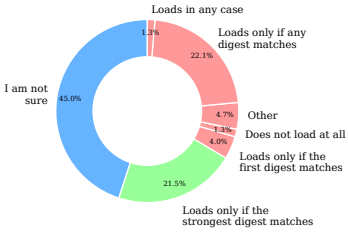
We received answers from professionals in various age ranges: 17.6% of respondents (or 40) were between 18 and 24 years old; the majority, or 42.3% (or 96) were between 25 and 34; 27.8% (or 63) between 35 and 44; and 11.9% (or 27) were older than 44 years. One respondent preferred not to disclose their age. Most respondents were employed full-time (i.e., 141 or 62.1%). 26% of respondents (or 59) were independent contractors, freelancers, or self-employed. The remaining respondents worked part-time (5.7% or 13), were unemployed (2.2% or 5), were retired (0.9% or 2) or had other work arrangements (3.1% or 7). Of the 213 respondents in the workforce, 46% worked for small companies, either startups or individually owned companies (or 98). A fourth worked for SMEs (24.4% or 52) and finally 29.6% (or 63) worked for large corporations. This shows that the sample was well balanced across different types of companies. About half of the sample declared to possess prior education on IT security (44.9% or 102). The remaining respondents did not have any prior education in IT security (48.5% or 110) and 6.6% (15) answered 'Other', which is worrisome given that we advertised among Wordpress and NPM plug-in developers, who produce popular software.

5.5 Results

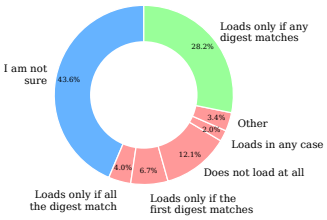
RQ1. Are the web developers aware of risks associated with (external) subresources? In the second section of the survey, we asked respondents to discuss potential threats that could affect a website if some subresources (e.g., scripts, stylesheets, images, videos) were hosted in a server separate from the server where the main website was hosted. In the rest of this section, we will refer to the described configuration as *the scenario*. The majority of respondents could identify the main threats of the scenario we provided, namely malicious code injection, cross-site scripting, etc. (56.8% or 129). A smaller group of participants (13.2% or 30) described secondary effects that could be produced if the main attacks could be



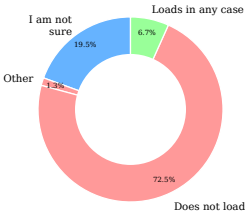
(a) Is it meaningful to use Subresource Integrity (SRI) if the target is served through HTTPS? (Q2)



(b) What happens when an integrity attribute has 2 or more valid hash values created with different algorithms? (Q6)



(c) What happens when an integrity attribute has 2 or more valid hash values generated with the same algorithm? (Q7)



(d) What happens if the digest in an integrity attribute is malformed (i.e., unsupported algorithm, non-base64)? (Q8)

Fig. 8. Response statistics for the questions related to the functioning and implementation of SRI.

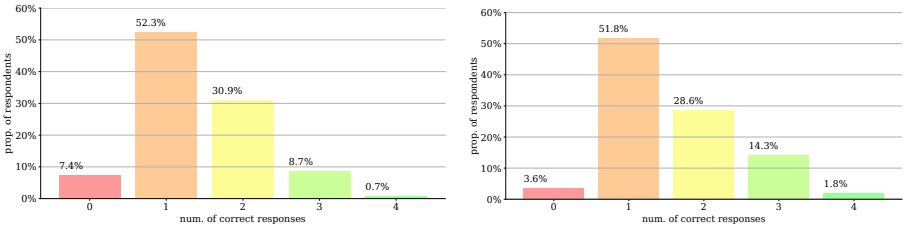
completed: compromised users' privacy, key-logging, redirection to fake websites, DDoS attacks, etc. Finally, about a third of participants (30% or 68) provided generic answers, non-applicable responses, or simply had no idea about possible risks.

RQ2. Are web developers aware of SRI? In a follow-up question, we asked the respondents to list possible solutions to protect the website, in the considered scenario, against the threats they reported. The respondents could list multiple solutions. More than half of the respondents (50.7% or 115) provided answers that were not applicable, or described solutions that would introduce additional problems without providing a definitive solution to the threat (e.g., "*do not use CDNs*"). The other large portion of respondents (30.8%, or 70) provided the right answers, essentially naming SRI or using a descriptive explanation in case they were not familiar with the name of the recommendation (e.g., "*content validation with MD5 checksum*"). Finally, 18.5% (or 42) of the respon-

dents described a technique or technology that would not ensure protection from the attacks (e.g., "*https*", "*two factor authentication*"). When we asked whether respondents had knowledge of the SRI recommendation (i.e., recognition over recall), we found that 41% (or 93) of the respondents had basic knowledge, and 24.6% (or 56) used SRI as part of their web development practices. About a third of the respondents (34.4% or 78) had no knowledge about SRI. Therefore, comparing the results to these two questions, we conclude that although two thirds of respondents declared to know or use SRI, only about 31% of the respondents could match the scenario with the solution provided with SRI. Hence, the difference (i.e., $\approx 30\%$) could be due to respondents who heard the acronym (or saw a snippet of code referring to SRI) but had no concrete idea of its purpose.

RQ3. To what extent do web developers understand the implementation of SRI? To those respondents who reported knowing or using SRI (149 or 65.6%), we asked them to describe in their own words how SRI could be used on a website and for what purpose. Most respondents described correctly the purpose and implementation of SRI (88.6% or 132). However, 11.4% (or 17) respondents could not. Next, we asked respondents 4 questions designed to assess the actual level of understanding of the recommendation. We manually investigated the "Other (please specify)" responses and edited them to the closest option whenever appropriate (otherwise we kept them as "Other" and considered them as incorrect). In total, we edited 3 responses, all for the second question and for the same reason (detailed below). The raw results are depicted in Fig. 8.

The first question looked at whether it was meaningful to use SRI in combination with HTTPS (for subresources). For this question, 86.5% (or 129) answered 'Yes' (i.e., the rest answered 'No' or 'Not Sure'). Then, we asked respondents what would happen if the integrity attribute would contain multiple (i.e., >1) valid hash values generated with *different* algorithms. Unfortunately, only 21.5% (or 32) identified the correct response, specifically that the browser would load the resource only if the digest created with the strongest hashing algorithm matches that of the resource. Three participants selected "Other" and specified that it is in fact the strongest algorithm *supported by the browser*. This is a valid point; but in practice, both responses are equivalent, as all browsers support all hashing algorithms. We edited them to the correct responses. The following question was similar to the previous, but this time the two hash values were created with the *same* algorithm. Slightly more respondents found



(a) all respondents (N=149) (b) respondents who use SRI (N=56)

Fig. 9. Number of correct responses for each of the 4 questions testing the respondents' understanding of SRI.

the correct answer (28.2% or 42), namely that the browser would load the resource only if any digest in the list matches that of the resource. For the following question, we asked respondents what would happen if the digest in an integrity attribute would be malformed. To this question only 6.7% (or 10) of respondents found the correct answer. This shows that the recommendation is somewhat counterintuitive, assuming that the respondents did not know the recommendation on this particular point and tried to answer this question based on common sense. This choice was probably in line with the best-effort strategy implemented in browsers for rendering webpages. We believe that the default behavior should be *to not load* resources with malformed integrity attribute. It should be noted that for the last three questions, the proportion of respondents who were not sure is substantial; it is worrisome to observe that many web developers doubt about the behavior of browsers regarding some security features they use.

Figure 9 presents the number of correct responses identified by the respondents. Only 0.7% (or 1) of respondents correctly answered *all* the 4 questions, thus revealing the small proportion of web developers who have a deep understanding of SRI implementation. We observe that the respondents who reported using SRI (i.e., 24.6% or 56) have a slightly better understanding than the average.

RQ4. What are the current practices of developers when using SRI? We asked respondents to select from a list of options about how they typically include SRI in their development practices. We wrote the list of options based on a number of practices we identified while performing the literature review and speculated from the result of the CommonCrawl analysis. We provided respondents the ability to select multiple practices from the list and to specify additional practices via a free text field. Most respondents reported to copy-paste snippets with examples of code using SRI from official documentations (45%, or 67). This strategy is not opti-

mal because it might work for popular libraries but might not be available for all external subresources (e.g., specialized or custom libraries). Also, it is not automated (i.e., it requires human intervention) and it is thus not scalable and error prone (e.g., miss the last or first digits upon copy-paste as we observed in our *CC* analysis). The second most frequent option reported by the respondents was to configure the build tools to compute and to include the checksums automatically (38.9%, or 58). This is the most secure and scalable approach to implement the SRI recommendation. The third largest group of respondents reported to compute the checksums of the subresources themselves and include these manually in the code (19.5% or 29). This strategy is secure but, again, not automated hence not scalable and error prone. Every time the external subresources are updated, the developers must download the updated version of the files, compute the new digests and update the code on their website. A fourth group of respondents declared to copy-paste snippets from online communities (10.1%, or 15). This is the most dangerous strategy because developers basically trust other random contributors. A community developer could be malicious and publish code crafted to create harm. Finally, the remaining respondents were either not sure (9.4%, or 14) or had not used SRI yet (12.1%, or 18).

To the participants who had knowledge or used SRI, we asked whether they thought SRI should be extended to additional types of subresources (beyond stylesheets and scripts). Majority of developers (66.4% or 99) answered "Yes". In the follow-up questions, the respondents selected the following types from a pre-defined list: images (83.8% or 83), videos (79.8% or 79), sounds (73.7% or 73), and downloads (i.e., `<a>` elements pointing to a file not rendered in the browser)–as suggested by Cherubini et al. (2018)–(67.7% or 67).

6 Related Work

Our work relates to the literature on the use of web security mechanisms, including SRI and its alternatives, and on web developer security practices.

Analysis of the use of security mechanisms on the web. A large body of work focuses on the use of HTTPS on the Web and on the effect on users of browser warnings Modic and Anderson (2014); Reeder et al. (2018); Egelman and Schechter (2013); Egelman et al. (2008); Jenkins et al. (2016); Akhawe and Felt (2013); Felt et al. (2015). Felt et al. (2017) study the adoption of HTTPS from a browser perspective. Lavrenovs and Melón

(2018) study the HTTP security headers of the 1M most popular websites. In particular, they analyze the prevalence of the most important response headers related to web security aspects, such as `Content-Security-Policy`. They notably show that HTTPS websites are more inclined to implement web security policies. Stark et al. (2019) study the adoption of the certificate transparency (CT), which fixes several structural flaws in the TLS certificate system and measure the error rates users experience. They show that CT has been widely adopted with minimal amount of warning displayed to the users. Another body of work focuses on the issues related to the inclusion of third-party subresources and trackers in webpages Roesner et al. (2012); Nikiforakis et al. (2012); Bielova (2013); Somé et al. (2017); Arshad et al. (2016); Musch et al. (2019). Arshad et al. (2018) perform the first large-scale analysis of scriptless CSS injection. They show that around 9% of 10k most popular websites contain at least one vulnerable page, out of which more than one third can be exploited. Anis et al. (2018) argue that many web applications contain vulnerabilities and promote various security mechanisms, including SRI. Van Acker et al. (2017) assess the security of 50k+ login webpages and show that very few of them deploy security measures; e.g., only 98 use SRI.

Closer to our work, Shah and Patil (2017) present existing attacks and describe how SRI improves the current situation. In a follow-up work, they perform a preliminary analysis of the use of SRI in the 1M most popular websites in 2017 Shah and Patil (2018). Their analysis shows that only 7k websites (i.e., 0.7%) implement SRI, and that less than 1% of those enforce SRI on all external subresources. This is consistent with our results. Kumar et al. (2017) also study security issues in the 1M most popular websites and find that less than 1% rely on SRI, which corroborates the findings of Shah and Patil (2018) and ours. Lauinger et al. (2017) conduct a large-scale study on client-side JavaScript library over 133k websites and show that 37% include at least one library with a known vulnerability. They mention SRI as one possible solution but stress the fact that SRI is misaligned with the objective of libraries to be transparently updated by third-party storage providers such as CDNs. Soni et al. (2015) argue (prior to the publication of SRI) that SRI applies only to websites that remain mostly static and evaluate, on the 500 most popular websites, the proportion of those that rely on static or changing scripts. Based on their 3-month longitudinal study, they identify 33k scripts, of which about 2300 change over time, which implies that SRI could be applied to 93% of the scripts without affecting website usability. How-

ever, they also find that only 69 out of 500 websites have *all* their scripts that remain static. They develop a multi-layered solution for whitelisting scripts that can tolerate changes without sacrificing security. Recently, a scanner for monitoring and alerting on accidental/intentional modifications to external subresources has been developed and open-sourced by CISCO Cisco CSIRT (2019).

Unlike in prior studies, in this work, we consider a much *larger* set of webpages (≈3B URLs/snapshot), we study the *evolution* of the use of SRI over time (for more than three years), and we perform an *in-depth* analysis of how SRI is used. In addition, we study the main *factors* behind the development, adoption and usage of SRI.

Another, less related, body of works studies the limitations of integrity verification mechanisms (including SRI) and proposes solutions and alternatives. Salvador et al. (2018) highlight the risks when the server that hosts the webpages is compromised. They address this issue by developing *wraudit*, a tool that transparently monitors the integrity of the published code based on a trusted and user-input baseline. Cap and Leiding (2018) address this issue with openly accessible code reviews of static code files combined with blockchain technology. West (2017) addresses the restriction of SRI to static content by enabling SRI to validate the integrity of subresources based a public key signatures instead of digests, thus enabling the inclusion of changing scripts published by the same (trusted) entity. Yet, this would open the door to downgrade attacks, i.e., replacing a recent version of a library with an old one with known and exploitable vulnerabilities. Kerschbaumer (2016) proposes to enforce content security by default. In the background, he mentions SRI as one important specification for ensuring content security. Cherubini et al. (2018) study the use of checksums for verifying the integrity of web downloads and, via a 40-participant in situ experiment, show that checksums suffer serious usability issues that negatively impact their effectiveness as a security mechanism. They develop a Chrome extension for automatically verifying checksums of downloaded files, whenever available.

Analysis of developer practices on (Web) security mechanisms. Although security practices of (web) developers have been extensively studied, to the best of our knowledge, there are no such studies related to SRI. Krombholz et al. (2019) present a qualitative study of the mental models associated with HTTPS and highlight knowledge gaps that affect experts. By interviewing 18 end-users and 12 experienced administrators, they find that "non-experts" underestimate the protection offered by HTTPS and that even "experts" have very little knowledge of how HTTPS works.

Acar et al. (2016) systematically study how the use of various information sources affects the security of mobile applications. By surveying 295 developers with apps listed in the Google Play Store, they observe that most developers use search engines and Stack Overflow to find write security-related code. Their lab study with 54 Android developers shows that developers who use Stack Overflow are more likely to write functionally correct code, but less likely to come up with a secure solution. In order to better understand the context in which developers produce security-relevant code, Tahaei and Vaniea (2019) survey 49 research papers at the intersection between usable security and software development. They provide an overview of existing works on developer-centered security and show that security is often being ignored because it is considered a secondary requirement. Balebako et al. (2014) study the security-related decision of app developers with a two-phase approach. First, they conduct interviews with 13 developers to better understand what decisions they make and what resource they use to make them. Second, they perform an online survey with 228 developers. They find that many developers lack awareness about security measures.

7 Discussion and Conclusion

In this article, we have provided the first comprehensive study on the use of the SRI recommendation. Our study, based on a longitudinal analysis of the CommonCrawl datasets over the last 3.5 years sheds light on the current adoption and usage of SRI: The adoption rate is modest (currently at ≈3.40%) but growing, and it is influenced by library developers and CDN providers who make code snippets that include integrity attributes available to developers. As pointed out in prior work Soni et al. (2015), the fact that SRI is suited only for subresources that do not change might impede its adoption. Our complementary survey of web developers has shown a good awareness and knowledge of SRI among developers but also some worrisome misunderstandings regarding its functioning in some situations. It has also revealed that the use of SRI by developers is mostly manual hence is not scalable and error prone, thus calling for a better integration of SRI by build tools.

Our study has some limitations, mainly due to our data sources. CommonCrawl might not be representative of the entire web. Also, it gives substantially more weight to multiple-page websites (counted as distinct URLs) compared to single page websites (counted as a single URL) Mikowski and Powell (2013). Finally, the webpages consist of raw

HTML code (i.e., not rendered), before the execution of scripts at the client side. Our questionnaire data is of modest size (i.e., 227 valid answers in total, 149 for the quiz) and the respondents might not be representative of web developers in general (i.e., "power developers", English-speaking, from specific ecosystems–i.e., NPM and WordPress). In addition, our results are based on self-reported behaviors, which might differ from actual (observed) behaviors.

The recent advances in the web development community are quite encouraging: More and more major libraries and CDN providers provide snippets with SRI (Bootstrap, jsdelivr). And major build tools, including Ruby on Rails Peek (2019), Webpack Scheid (2019), and Grunt Hernandez (2019), now integrate SRI, mostly through plug-ins. The *native* integration of SRI in build tools, but also in CMS such as WordPress and Drupal, would substantially increase the adoption of SRI. Note that plug-ins are available for WordPress and the integration of SRI in Drupal (core) is being discussed. Finally, the extension of SRI to signature-based verification (i.e., with a public key as integrity attribute) West (2017) instead of digest-based verification, discussed in the WebAppSec group, would make SRI more suitable for subresources that are transparently updated, thus increasing its adoption.

We intend to improve the awareness and understanding of SRI and to promote its use through a dedicated website that would contain a brief description and illustration of SRI, including its functioning in specific scenarios that are not well understood by developers, as well as the statistics provided in this article, updated periodically based on the CommonCrawl datasets. We intend to investigate the adoption of SRI in different categories of websites (e.g., popular, banking, e-commerce). We also intend to push (or participate in) the revision of the recommandation to extend it to other resources such as images and videos, because this would thwart the risks of media-based web defacement (a majority of developers reported being interested in such an extension). Extending SRI to downloads would also favorably replace checksums displayed in webpages, as they suffer from serious usability issues. Finally, we also intend to extend our survey to a larger population of web developers, but also to further study developers' perception of SRI – through interviews – in order to gain a deeper understanding of their mental models on SRI, as recently done by Krombholz et al. (2019) for HTTPS or by Acar et al. (2016) for the security of mobile apps.

Acknowledgments

The authors are grateful to the anonymous reviewers for their insightful feedback, to Holly Cogliati for her great editing job, and to Romain Artru for his help in the implementation of the preliminary analysis scripts. The work was partially funded with grant #19024 from the Hasler Foundation and with a grant from HEC Lausanne.

Bibliography

Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, San Jose, CA, USA, 289–305. https://doi.org/10.1109/SP.2016.25

Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness.. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, Washington D.C., USA, 257–272.

A. Anis, M. Zulkernine, S. Iqbal, C. Liem, and C. Chambers. 2018. Securing Web Applications with Secure Coding Practices and Integrity Verification. In *Proc. of the IEEE Int'l Conf. on Dependable, Autonomic and Secure Computing, Int'l Conf on Pervasive Intelligence and Computing, 4th Int'l Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. IEEE, Athens, Greece, 618–625. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00112

Sajjad Arshad, Amin Kharraz, and William Robertson. 2016. Include Me Out: In-Browser Detection of Malicious Third-Party Content Inclusions. In *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*, Jens Grossklags and Bart Preneel (Eds.). Springer, Christ Church, Barbados, 441–459. https://doi.org/10.1007/978-3-662-54970-4_26

Sajjad Arshad, Seyed Ali Mirheidari, Tobias Lauinger, Bruno Crispo, Engin Kirda, and William Robertson. 2018. Large-Scale Analysis of Style Injection by Relative Path Overwrite. In *Proc. of the Int'l Conf. on World Wide Web (WWW)*. ACM, Lyon, France, 237–246. https://doi.org/10.1145/3178876.3186090

Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. In *Proc. of the Workshop on Usable Security (USEC)*. Internet Society, San Diego, CA. https://doi.org/10.14722/usec.2014.23006

Nataliia Bielova. 2013. Survey on JavaScript Security Policies and Their Enforcement Mechanisms in a Web Browser. *The Journal of Logic and Algebraic Programming* 82, 8 (2013), 243–262. https://doi.org/10.1016/j.jlap.2013.05.001

Frederik Braun. 2019. Revert 'require-Sri-For' from the SRI Recommendation. https://github.com/w3c/webappsec-subresource-integrity/pull/82, last visited: Oct. 2019.

Clemens H. Cap and Benjamin Leiding. 2018. Ensuring Resource Trust and Integrity in Web Browsers Using Blockchain Technology. In *Advanced Information Systems Engineering Workshops (Lecture Notes in Business Information Processing)*, Raimundas Matulevičius and Remco Dijkman (Eds.). Springer, 115–125.

Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2018. Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Toronto, ON, Canada, 1256–1271. `https://doi.org/10.1145/3243734.3243746`

Cisco CSIRT. 2019. SRI-Monitor: Subresource Integrity Monitor. https://github.com/ciscocsirt/SRI-Monitor, last visited: Oct. 2019.

Common Crawl. 2019a. Common Crawl. https://commoncrawl.org/.

Common Crawl. 2019b. Statistics of Common Crawl Monthly Archives. https://commoncrawl.github.io/cc-crawl-statistics/, last visited: Oct. 2019.

Terence Eden. 2018. Major Sites Running Unauthenticated JavaScript on Their Payment Pages. https://shkspr.mobi/blog/2018/11/major-sites-running-unauthenticated-javascript-on-their-payment-pages/, last visited: Oct. 2019.

Serge Egelman, Lorrie Faith Cranor, and Jason Hong. 2008. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Florence, Italy, 1065–1074. `https://doi.org/10.1145/1357054.1357219`

Serge Egelman and Stuart Schechter. 2013. The Importance of Being Earnest [in Security Warnings]. In *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*. Springer, Okinawa, Japan, 52–59. `https://doi.org/10.1007/978-3-642-39884-1_5`

Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL Warnings: Comprehension and Adherence. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Seoul, Republic of Korea, 2893–2902. `https://doi.org/10.1145/2702123.2702442`

Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, Vancouver, BC, Canada, 16.

Neftaly Hernandez. 2019. Grunt-Sri: SRI Plug-in for Grunt. https://github.com/neftaly/grunt-sri, last visited: Oct. 2019.

Dan Hubbard. 2019. Cisco Umbrella 1 Million. https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/, last visited: Oct 2019.

Internet Archive. 2019. Wayback Machine. https://web.archive.org/.

Jeffrey L. Jenkins, Bonnie Brinton Anderson, Anthony Vance, C. Brock Kirwan, and David Eargle. 2016. More Harm than Good? How Messages That Interrupt Can Make Us Vulnerable. *Information Systems Research* 27, 4 (2016), 880–896. https://doi.org/10.1287/isre.2016.0644

Christoph Kerschbaumer. 2016. Enforcing Content Security by Default within Web Browsers. In *Proc. of the IEEE Conf. on Cybersecurity Development (SecDev)*. IEEE, Boston, MA, USA, 101–106. https://doi.org/10.1109/SecDev.2016.033

K. Krombholz, K. Busse, K. Pfeffer, M. Smith, and E. von Zezschwitz. 2019. "If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 1138–1155. https://doi.org/10.1109/SP.2019.00060

Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J. Alex Halderman, and Michael Bailey. 2017. Security Challenges in an Increasingly Tangled Web. In *Proc. of the Int'l Conf. on World Wide Web (WWW)*. ACM, Perth, Australia, 677–684. https://doi.org/10.1145/3038912.3052686

Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *Proc. of the Network and Distributed System Security Symp. (NDSS)*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2017.23414

A. Lavrenovs and F. J. R. Melón. 2018. HTTP Security Headers Analysis of Top One Million Websites. In *Proc. of the Int'l Conf. on Cyber Conflict (CyCon)*. IEEE, Tallinn, Estonia, 345–370. https://doi.org/10.23919/CYCON.2018.8405025

Kathleen M. MacQueen, Eleanor McLellan, Kelly Kay, and Bobby Milstein. 1998. Codebook Development for Team-Based Qualitative Analysis. *CAM Journal* 10, 2 (1998), 31–36. https://doi.org/10.1177/1525822X980100020301

Michael Mikowski and Josh Powell. 2013. *Single Page Web Applications: JavaScript End-to-End* (1st ed.). Manning Publications Co., Greenwich, CT, USA.

David Modic and Ross Anderson. 2014. Reading This May Harm Your Computer: The Psychology of Malware Warnings. *Computers in Human Behavior* 41 (2014), 71–79. https://doi.org/10.1016/j.chb.2014.09.014

Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. 2019. ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices. In *Proc. of the ACM Asia Conf. on Computer and Communications Security (Asia CCS)*. ACM, 391–402. https://doi.org/10.1145/3321705.3329841

Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You Are What You Include: Large-Scale Evaluation of Remote Javascript Inclusions. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Raleigh, NC, USA, 736–747. https://doi.org/10.1145/2382196.2382274

Joshua Peek. 2019. Sprockets-Rails: SRI Plug-in for Rails. https://github.com/rails/sprockets-rails, last visited: Oct. 2019.

Robert W. Reeder, Adrienne Porter Felt, Sunny Consolvo, Nathan Malkin, Christopher Thompson, and Serge Egelman. 2018. An Experience Sampling Study of User Reactions to Browser Warnings in the Field. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Montréal, Canada, 512:1–512:13. https://doi.org/10.1145/3173574.3174086

Franziska Roesner, Tadayoshi Kohno, and David Wetherall. 2012. Detecting and Defending Against Third-Party Tracking on the Web. In *Proc. of the Symp. on Networked Systems Design and Implementation (NSDI)*. USENIX, San Jose, CA, USA, 14.

Johnny Saldaña. 2015. *The Coding Manual for Qualitative Researchers*. Sage, Arizona State University, USA.

David Salvador, Jordi Cucurull, and Pau Julià. 2018. Wraudit: A Tool to Transparently Monitor Web Resources' Integrity. In *Proc. of the Int'l Conf. on Mining Intelligence and Knowledge Exploration (MIKE)*, Adrian Groza and Rajendra Prasath (Eds.). Springer, Cluj-Napoca, Romania, 239–247. https://doi.org/10.1007/978-3-030-05918-7_21

Julian Scheid. 2019. Webpack-Subresource-Integrity: SRI Plug-in for Webpack. https://github.com/waysact/webpack-subresource-integrity, last visited: Oct. 2019.

Ronak Shah and Kailas Patil. 2018. A Measurement Study of the Subresource Integrity Mechanism on Real-World Applications. *International Journal of Security and Networks* 13, 2 (2018), 129. https://doi.org/10.1504/IJSN.2018.092474

Ronak N Shah and Kailas R Patil. 2017. Securing Third-Party Web Resources Using Subresource Integrity Automation. *International Journal on Emerging Trends in Technology* 4, 2 (2017), 5.

Dolière Francis Somé, Nataliia Bielova, and Tamara Rezk. 2017. Control What You Include! Server-Side Protection against Third Party Web Tracking. In *Proc. of the Int'l Symp. Engineering Secure Software and Systems (ESSoS)*, Vol. 10379. Springer, 115–132. https://doi.org/10.1007/978-3-319-62105-0_8

Pratik Soni, Enrico Budianto, and Prateek Saxena. 2015. The SICILIAN Defense: Signature-Based Whitelisting of Web JavaScript. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Denver, CO, USA, 1542–1557. https://doi.org/10.1145/2810103.2813710

E. Stark, R. Sleevi, R. Muminovic, D. O'Brien, E. Messeri, A. Felt, B. McMillion, and P. Tabriz. 2019. Does Certificate Transparency Break the Web? Measuring Adoption and Error Rate. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, Los Alamitos, CA, USA, 463–478. https://doi.org/10.1109/SP.2019.00027

Mohammad Tahaei and Kami Vaniea. 2019. A Survey on Developer-Centred Security. In *Proc. of the IEEE European Symp. on Security and Privacy Workshops (EuroS&PW)*. IEEE, Stockholm, Sweden, 129–138. https://doi.org/10.1109/EuroSPW.2019.00021

Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. 2017. Measuring Login Webpage Security. In *Proc. of the ACM Symp. on Applied Computing (SAC)*. ACM, Marrakech, Morocco, 1753–1760. https://doi.org/10.1145/3019612.3019798

W3C. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/.

Mike West. 2017. Signed Content Should Include Resource Names. https://github.com/mikewest/signature-based-sri/issues/5, last visited: Oct. 2019.

A Survey about Security on the Web

Introductory Text Block
The ISPLab (https://www.isplab.unil.ch) at the University of Lausanne (Switzerland) is currently researching web development practices. By filling this survey, you will contribute to the research and can enter a raffle draw to win a $100 Amazon voucher. This survey takes approximately 5-10 minutes to complete. The answers you provide will be treated anonymously and will not be linked to your identity. The data we collect will be used solely for academic research (non-profit). If you have questions about this research, feel free to e-mail us at isplab-survey@unil.ch. The survey will run until 15th September 2019, and the questionnaire is best viewed with Mozilla Firefox and Google Chrome.
Note:

1. The survey is meant for web developers that are 18 years old or older.
2. Participants younger than 18 years old or who do not disclose their age group will not be able to participate in the raffle draw.
3. Data from participants younger than 18 years or who do not disclose their age will be deleted and not analyzed.

Coding rules (not visible to respondents):

- Questionnaire does not allow to go back to previous questions
- Unless specified, one question per page
- All questions mandatory, unless specified as optional

A.1 Screening Block

1. The questionnaire is in English. Are you comfortable reading/writing in English?
   - ○ Yes, I am comfortable taking the questionnaire in English
   - ○ No, I can read, but I am not comfortable writing in English [terminate]
   - ○ No, I am not comfortable taking the questionnaire in English [terminate]
2. Are you an active web developer?
   - ○ Yes, I do web development on my free time
   - ○ Yes, I work as a web developer
   - ○ No, I am not directly involved with web development [terminate]

A.2 Awareness of SRI

Introductory text.
For the following question let's imagine there are some resources (such as stylesheets, scripts, etc.) that are used by a website and that are stored in a server separate from the server where the main site is hosted.

1. Please describe possible security threats caused by external resources included in a website, given the hypothetical situation described above. [free text]
2. If you described security threats in the previous question, please explain possible ways in which the website in the hypothetical scenario could be protected against these threats. [free text]
3. Please rate how much you know about the subresource integrity (SRI) recommendation of the W3C
   Example:
   ```
   <script src="https://cdn.com/js/page.js"
   integrity="sha384-ggOyR0i..."> </script>
   ```
   - ○ No knowledge [skip to demographic section]
   - ○ Little knowledge (I heard the term)
   - ○ Some knowledge (I know the basics)
   - ○ Moderate knowledge (I used SRI)
   - ○ Extensive knowledge (I use it often)

A.3 Knowledge of SRI

1. Please describe in your own words how you can use subresource integrity (SRI) on a website as well as the purpose of SRI. [free text]
2. In your opinion, is it meaningful to use subresource integrity (SRI) if the target is served through HTTPS?
   Example:
   Source: https://www.website.com
   Target: https://cdn.com/js/page.js
   ```
   <script src="https://cdn.com/js/page.js"
   integrity="sha384-ggOyR0i...> </script>
   ```
   - ○ Yes
   - ○ No
   - ○ I am not sure
3. Please explain why you answered Yes/No to this question. [subquestion appears only if Yes/No is selected in previous question]

4. In your opinion, is it meaningful to use subresource integrity (SRI) if the website uses HTTP?

   Example:
   Source: http://www.website.com
   Target: https://cdn.com/js/page.js

   ```
   <script src="https://cdn.com/js/page.js"
   integrity="sha384-ggOyR0i..."></script>
   ```

   - ○ Yes
   - ○ No
   - ○ I am not sure
5. Please explain why you answered Yes/No to this question. [subquestion appears only if "Yes"/"No" is selected in previous question]
6. In your opinion, what happens when an integrity attribute has 2 or more valid hash values (i.e., digests) created with different algorithms?

   Example:

   ```
   <script src="https://cdn.com/js/page.js"
   integrity="sha384-ggOyR0i... sha256-ivzZrY..."> </script>
   ```

   - ○ The user-agent loads the resource only if the digest created with the strongest hashing algorithm matches that of the resource
   - ○ The user-agent loads the resource only if the first digest in the list matches that of the resource
   - ○ The user-agent loads the resource only if any digest in the list matches that of the resource
   - ○ The user-agent does not load the resource at all
   - ○ The user-agent loads the resource in any case
   - ○ I am not sure
   - ○ Other (please specify)
7. In your opinion, what happens when an integrity attribute has 2 or more valid digests generated with the same algorithm?

   Example:

   ```
   <script src="https://cdn.com/js/page.js"
   integrity="sha384-ggOyR0i... sha384-eYWSGB..."> </script>
   ```

   - ○ The user-agent loads the resource only if the first digest in the list matches that of the resource
   - ○ The user-agent loads the resource only if any digest in the list matches that of the resource
   - ○ The user-agent does not load the resource at all
   - ○ The user-agent loads the resource in any case
   - ○ I am not sure

- ○ Other (please specify)

8. In your opinion, what happens if the digest in an integrity attribute is malformed (i.e., generated with an unsupported algorithm, composed of non-base64 characters, etc.)? *Example*:
   ```
   <script src="https://cdn.com/js/page.js"
   integrity="malformed!"> </script>
   ```
   - ○ The user-agent does not load the resource at all
   - ○ The user-agent loads the resource in any case
   - ○ I am not sure
   - ○ Other (please specify)
9. How do you typically include Subresource Integrity (SRI) in development? (Select all that apply)
   - □ Copy-paste snippets from the official documentation
   - □ Copy-paste snippets from online communities
   - □ Compute the checksums of the subresources and include them myself
   - □ Configure my build tool to compute and include the checksums automatically
   - □ I am not sure
   - □ Other (please specify)
10. Do you think the SRI recommendation should be extended to subresources other than the stylesheets `<link>` and the scripts `<script>`?
    - ○ Yes
    - ○ No
    - ○ I am not sure
11. Which of the following subresources would you like to see the SRI recommendation extended to? (select all that apply)
    [subquestion appears only if Yes is selected in previous question]
    - □ Images `<img>`
    - □ Videos `<video>`
    - □ Sounds `<audio>`
    - □ Downloads `<a>`
    - □ Other (please specify)

A.4 Demographics

1. What is your age group?
   - ○ Under 18 [terminate and delete data]
   - ○ 18 - 24
   - ○ 25 - 34
   - ○ 35 - 44

- ○ 45 - 54
- ○ 55 - 64
- ○ 65+
- ○ Prefer not to disclose

2. Which statement best describes your current employment status?
   - ○ Employed full-time
   - ○ Employed part-time
   - ○ Independent contractor, freelancer, or self-employed
   - ○ Not employed, but looking for work
   - ○ Not employed, and not looking for work
   - ○ Retired
   - ○ Other (please specify):
3. What is the size of your company where you work (or own)?
   [only showed if respondent is working]
   - ○ Individually owned company
   - ○ Startup
   - ○ Small and medium-sized enterprises (SME)
   - ○ Large corporation
   - ○ Other (please specify)
4. Do you have an IT security background?
   - ○ Yes, I have a diploma in IT security
   - ○ Yes, I studied IT security in a college/university without earning a degree
   - ○ Yes, I took some courses in IT security but did not specialise in it
   - ○ Yes, I have a university degree or equivalent in IT security
   - ○ Yes, I had some training sponsored by the company where I work
   - ○ No, I do not have any IT security background
   - ○ Other (please specify)

A.5 Communication

1. Do you wish to participate in the raffle draw for a $100 Amazon gift card?
   - ○ Yes
   - ○ No
2. Do you wish to receive the cumulative results of the survey?
   - ○ Yes
   - ○ No
3. Please provide your e-mail address. [free text]
   [only showed if answered “Yes” to one of the two questions above]

A.6 Last page

Your answers have been recorded. Thanks for your help.
This questionnaire focused on Subresource Integrity, a standard defined by W3C that *defines a mechanism by which user agents may verify that a fetched resource has been delivered without unexpected manipulation.* If you want to know more, you can visit this page: `https://www.w3.org/TR/SRI/`. If you have questions about this research feel free to email us at isplab-survey@unil.ch.

5. Measurement and Analysis of Automated Certificate Reissuance

The subsequent paper has been published as follows:

Olamide Omolola, Richard Roberts, Md. Ishtiaq Ashiq, Taejoong Chung, Dave Levin and Alan Mislove. "Measurement and Analysis of Automated Certificate Reissuance". In: Lecture Notes in Computer Science 12671 (2021), pp. 161–174

In this paper, we measure and analyse the adoption of automation in certificate reissuance since the advent of free, automated CAs like Let's Encrypt. We use publicly available data from Certificate Transparency (CT) logs for this measurement.

Abstract. The Transport Layer Security (TLS) Public Key Infrastructure (PKI) is essential to the security and privacy of users on the Internet. Despite its importance, prior work from the mid-2010s has shown that mismanagement of the TLS PKI often led to weakened security guarantees, such as compromised certificates going unrevoked and many internet devices generating self-signed certificates. Many of these problems can be traced to manual processes that were the only option at the time. However, in the intervening years, the TLS PKI has undergone several changes: once-expensive TLS certificates are now freely available, and they can be obtained and reissued via automated programs.
In this paper, we examine whether these changes to the TLS PKI have led to improvements in the PKI's management. We collect data on *all* certificates issued by Let's Encrypt (now the largest certificate authority by far) over the past four years. Our analysis focuses on two key questions: First, *are administrators making proper use of the automation that modern CAs provide for certificate reissuance?* We find that for certificates with a sufficiently long history of being reissued, 80% of them did reissue their certificates on a predictable schedule, suggesting that the remaining 20% may use manual processes to reissue, despite numerous automated tools for doing so. Second, *do administrators that use automated CAs react to large-scale compromises more responsibly?* To answer this, we use a recent Let's Encrypt misissuance bug as a natural experiment, and find that a significantly larger fraction of administrators reissued their certificates in a timely fashion compared to previous bugs.

1 Introduction

The Transport Layer Security (TLS) public key infrastructure (PKI) is an essential component of the modern Internet: it allows users to communicate over the Internet in a trusted and confidential manner. However, previous work [8,21,13,2,3] has demonstrated that despite its importance, the *management* of the TLS PKI is often not compliant with recommended security practices. For example, systems administrators often fail to revoke or even reissue certificates when private keys are compromised [20], many internet-of-things devices generate self-signed certificates (sometimes even with identical keys) [13], and domains sometimes share private keys with third parties due to limitations in the PKI itself [2].

Many of these management issues can be traced to inadequate tools for system administrators. For example, in the wake of the Heartbleed [11] bug in 2014, a significant fraction of web servers potentially had their private keys exposed; as a result, administrators should have revoked their old certificates and reissue new ones. At the time, doing so was a largely manual process: because certificates were typically valid for up to 5 years, many administrators presumably eschewed automating the

infrequent process of obtaining and installing new certificates. As a result, it took over a *week* before even 10% of the vulnerable web servers had reissued their certificates [21]. Similarly, in the DNSSEC PKI, it has been observed that inadequate tools—in the case of DNSSEC, a manual process of uploading DS records—has lead to poor adoption of secure protocols [4].

However, the TLS PKI has changed dramatically since 2014. While previously expensive, TLS certificates are now free with the advent of certificate authorities such as Let's Encrypt [14] (which is now, by far, the most popular CA [16]). More importantly, these free CAs often have much shorter certificate lifetimes (90 days for Let's Encrypt), encouraging the automation of the process of certificate reissuance and installation (as it happens every three months, rather than every five years). Open-source protocols (e.g., ACME) and tools (e.g., `certbot`, `acme.sh`, cPanel) now allow administrators to automate the entire process.

In this paper, we examine whether the presence of these tools and services has led to better TLS certificate reissuance. To understand the effects of automated tools in certificate reissuance, we focus on certificates issued by Let's Encrypt. We chose Let's Encrypt as it is by far the largest ACME-based CA [16], and it has the longest history of operation (and hence, the highest likelihood of having domain sets that have a long history of reissues). We use Certificate Transparency (CT) [12] logs to obtain a list of *all* 1.03B certificates Let's Encrypt issued over the past four years. We group certificates in this list by the set of domains they contain (similar to prior work [21], we refer to this as a *domain set*), enabling us to measure how often certificates are reissued.

We also use a recent bug discovered by Let's Encrypt as a natural experiment. In brief, in early 2020, Let's Encrypt discovered that over 3M certificates had been issued improperly, as they had failed to check for Certificate Authority Authorization (CAA) [19] records properly before issuance [5]. Because they were improperly issued, Let's Encrypt announced that they planned to revoke the certificates one week later, informing all system administrators that they needed to reissue their certificates. This serves as a natural experiment, as we can examine whether administrators took the necessarily manual action of reissuing their certificates, rather than simply relying on their automated reissuance.

Our paper makes two contributions: *First*, we examine the behavior of system administrators reissuing TLS certificates with the advent of free CAs such as Let's Encrypt. We find that approximately 80% of domain sets with a sufficiently long history of being reissued, did reissue their certificates on a predictable schedule. In addition, 60% of all domain sets

show a median reissuance period of 60 days (the default recommended by Let's Encrypt [14] and used by many ACME tools [23,6] for automated certificate reissuance).

Second, we use the Let's Encrypt bug mentioned above to explore whether system administrators now respond more quickly and completely when manual intervention *is* required. We focus on the subset of the 2M domain sets with a misissued certificate, and identify 98,652 domain sets that show a regular period of reissuance with at least one new certificate issued after the bug was discovered on February 29, 2020.[1] We demonstrate that, of these domain sets, at least 28% appear to have taken the manual steps necessary to reissue their certificates within a week, suggesting that, indeed, system administrators are better able to reissue certificates securely today when compared to previous incidents requiring certificate reissuance.

2 Background

We begin with an overview of the TLS certificate ecosystem and related work.

2.1 Certificates

TLS is based on certificates, which are bindings between identities (typically domain names) and public keys. Certificates are signed by *certificate authorities* (CAs), who verify the identity of the requestor. Certificates have a well-defined validity period, which is expressed as `NotBefore` and `NotAfter` fields in the certificate; clients will refuse to accept certificates outside of their validity period. As a result, certificate owners have to periodically *reissue* their certificate by contacting their CA (or another CA) and obtaining a new certificate.

While certificates originally only contained a single identity (domain name), this often made the administration difficult for web servers that served multiple domains. Today, certificates can carry multiple identities (domain names) via a `Subject Alternate Names` list. In essence, the owner of the certificate's public key has been verified by the CA to control *all* of the identities (domains).

Finally, domain owners may wish to limit the set of CAs who are authorized to issue certificates for a given domain. They can now do so by

[1] Because of the way the bug manifested itself, the misissued certificates are *not* a random sample of all certificates. We explore this in Section 3.

publishing Certificate Authority Authorization (CAA) records, which are DNS records that specify a list of CAs that are/are not allowed to issue certificates (if no such record exist, all CAs are implicitly authorized). CAs today are required to check for the CAA records for domains before issuing certificates.

2.2 Let's Encrypt

For a long time, TLS certificates were relatively expensive to obtain (typically $50 or more) and were valid for multiple years (typically 3–5) [13]. The cost and extended validity ended up having two effects: the overall adoption of HTTPS was relatively low (as administrators had to spend significant money to obtain the necessary certificates), and the system administrators who did purchase certificates were not incentivized to automate the infrequent reissuance process. Additionally, the certificate issuance and renewal processes were manual, administratively burdensome, and technically cumbersome.

In 2015, Let's Encrypt disrupted the TLS certificate business model by offering *free* certificates that were valid for 90 days. Other free CAs have also been created such as ZeroSSL[2] and Buypass[3], and the TLS ecosystem has since changed dramatically: the fraction of web connections using HTTPS has increased from ∼27% in early 2014 to ∼85% in 2020 [16], and Let's Encrypt is now the largest CA, with over 1B certificates issued and over 35% of the Alexa top 1M sites using Let's Encrypt certificates [1]. Importantly, while prior CAs often required certificates to be requested/reissued via web forms, Let's Encrypt is entirely automated via the ACME protocol; several popular ACME clients exist, including `certbot`, `acme4j`, and `acme.sh`.

In February 2020, Let's Encrypt announced that they discovered a bug in the Boulder software they used to issue certificates [5]. Specifically, the software failed to properly check for CAA records in requested certificates if (a) a certificate was requested for multiple domains, and (b) Let's Encrypt had previously checked the domain control validations (DCV) for these domains in the preceeding 30 days. While Let's Encrypt was supposed to re-check the CAA record for all domain names included in the certificate within 8 hours of issuing the certificate, under these circumstances, it only picked one domain name among the multiple domains in the certificate and ran the CAA check n times (equivalent to the

[2] `https://zerossl.com/features/certificates/`
[3] `https://www.buypass.com/ssl/products/acme`

number of domains in the certificate). Let's Encrypt originally announced on February 29, 2020 that it planned to revoke all these certificates on March 5, 2020, and it emailed all affected domain administrators. On March 5, 2020, Let's Encrypt reversed their decision and decided to not revoke en-masse [15].

2.3 Related work

Improvements in the ability to scan the Internet [10] in 2013 have led to a better understanding of the entire TLS ecosystem [9]. Researchers have unfortunately found that TLS clients and servers are often incorrectly managed [13], leading to reduced security for internet users. In the aftermath of the Heartbleed bug, it became evident that manual revocation and reissuance of certificates is a major security problem: most administrators failed to revoke or even reissue, and those that did sometimes reissued using the same key pair [21,8]. Similar behavior had been observed years prior when a bug in Debian caused many domains the need to reissue certificates [20]. Some domains have chosen to outsource certificate management to third-parties such as content delivery networks (CDNs); while this improves certificate management, it often requires sharing private keys [2].

To the best of our knowledge, there has not been significant study of automated certificate reissuance in the TLS PKI. Previous work by Matsumoto et al. proposed a decentralized audit-based system: Instant Karma PKI (IPK) to promote automation among HTTPS domains [18]. The recent development of CAA records also provides a useful tool for automation as the domain name holders or DNS operators can use CAA records to control which CAs that they would like to get a certificate from [19].

3 Methodology

We now describe the datasets we collected and our methodology to determine a set of certificates that have been reissued.

3.1 Certificates

Our goal is to see how certificates have been (re)issued by the system administrators. We focus on Let's Encrypt as it is the largest free CA, and it has the longest history of operation. To this end, we obtain *all*

certificates issued by Let's Encrypt by leveraging the Certificate Transparency (CT) logs; when issuing a certificate, Let's Encrypt publishes the certificate to one of the CT logs managed by Google.[4] Thus, to obtain a nearly complete view of the certificates issued by Let's Encrypt, we first fetch all certificates from all of the CT log servers managed by Google,[5] obtaining 5.3B certificates in total from September 9, 2014 to May 18, 2020. We then identify the certificates issued by Let's Encrypt according to their `Issuer` field, which leaves us with 1.03B certificates.[6]

3.2 Let's Encrypt CAA bug list

On February 29th, 2020, Let's Encrypt announced the CAA issuance bug in their certificate issuance process (see § 2.2). Let's Encrypt publicly released a list of the certificates impacted by this bug [5] containing serial numbers of 3,048,289 certificates, some of which were potentially misissued (i.e., the CAA records for some of domains in the certificate may have not permitted Let's Encrypt to issue a certificate, even though they did). We use this list to study how the impacted certificates have been *reissued* by administrators.

3.3 Defining Certificate Reissuances

While it is easy to identify when certificates are issued, there is a bit of subtlety to determining when they are *reissued*. In particular, we face two challenges: *First*, CT logs do not contain any identifier of the client such as IP address that sent a Certificate Signing Request (CSR), thus making it hard to identify if the certificate has been reissued from the same client; thus, we first link the certificates that share the same `Subject Alternate Name` (SAN) list.[7] We refer to this set of domains in the SAN list as the *domain set*. *Second*, we do not know when the client has replaced the old certificate with the new one; thus, we use the logging timestamp on the CT log server as a proxy.

[4] In order for a certificate to be "CT qualified" in modern browsers such as Chrome, it has to be logged on multiple CT log servers and one of them has to be from a Google log [7].

[5] `aviator`, `icarus`, `argon2018`∼`2023`, `xenon2019`∼`2023`, `pilot`, `rocketeer`, `skydiver`

[6] We intentionally exclude pre-certificates from the analysis (which Let's Encrypt has published as well since 2018 [17]) as they do not guarantee the issuance of their actual (final) certificates.

[7] Thus, if the same client adds or removes one domain, it changes the SAN list. Therefore, ACME processes it as a separate certificate request, not a reissuance, thereby supporting our methodology of grouping by domain sets

In summary, we group certificates by their domain set and order them based on their timestamp on the CT logs; we refer to any certificates other than the first as reissued certificates. Using this, methodology we obtain 188M unique domain sets and 1.03B corresponding certificates issued during our measurement period. Out of the 188M domain sets, we find that 67M (35.7%) domain sets have no reissued certificates, 23M (12.2%) domain sets have reissued once and, 14M (7.8%) domain sets have reissued twice. One limitation of relying on CT logs alone worth noting is that we are unable to quantify how domain sets change, as we would need a way to "link" domain sets which is unavailable to us [2]. In these cases, the modified domain set would be considered a separate domain set in our analysis.

4 Results

We analyze the reissuance behaviors of certificates issued by Let's Encrypt. We aim to understand reissuance behavior of two types: reissuance that is likely done *automatically* (e.g., via a `cron` job) and reissuance that is likely done *manually* (e.g., directly invoked by a system administrator). We begin by describing how we distinguish these two cases.

4.1 Automated Reissuance

One of Let's Encrypt's key principles is that it makes it possible to automate obtaining and reissuing certificates. A new user of Let's Encrypt need only set up the first certificate issuing process with any ACME client of choice, then they can create a `cron` job to continually check if the certificate is still valid and request a new certificate once the current certificate nears expiry.

We first need to identify when we believe a certificate has been reissued via an automated process. As discussed previously we are not privy to Let's Encrypt's internal logs, so we can only rely on publicly available data from the CT logs. To do so, we group all Let's Encrypt certificates by the domain set present in them, and then sort these lists by the time in the CT log timestamp. We then examine the amount of time that passes between each pair of successive reissues.

In Figure 1, we plot the cumulative distribution of the median of these reissue time lists in the line labeled "all LE certs." We immediately observe a large "spike" around 60 days, and observe that over 55% of domain sets have a median reissue time between 55 and 65 days. This

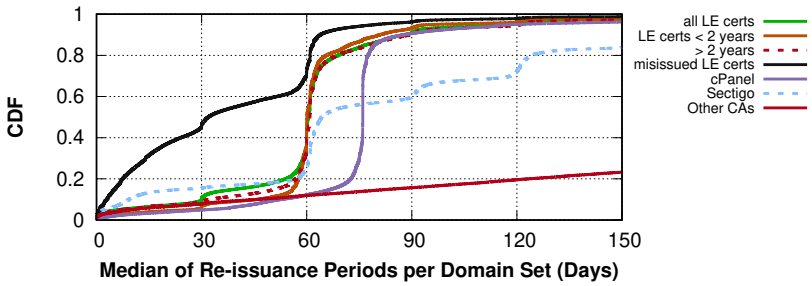


Fig. 1. Distribution of median reissuance period per domain set for all Let's Encrypt certificates with or without lifetimes and misissued certificates. For comparison, we also include the median reissuance period per domain set for a few other CAs: Sectigo, cPanel, and other top 10 CAs (we plot cPanel and Sectigo separately as they show different behavior than the others).

lines up with the reissuance policy recommended by Let's Encrypt, which recommends reissuing certificates that are within 30 days of their expiry (i.e., are at least 60 days old) [14]. Moreover, this timing lines up well as the default policies of many ACME clients: `certbot` [6] and `acme.sh` [23] both default to renewing within 30 days of expiry. We also observe that the "spike" does not happen *entirely* at the 60 day mark; this is likely because the renewal occurs the first time the `cron` job runs after reaching the mark. Finally, we observe a much smaller spike around 30 days, which is likely the behavior of a different ACME client or a system administrator who manually changed their client's behavior.

Next, we examine whether this median reissue period of 60 days is only present in domain sets that have a long history of being reissued (i.e., that have been around a long time) or if it is also present in newer domain sets. To do so, we divide the "all LE certs" line into those first issued greater than two years ago, and those first issued within the past two years; these are both plotted in Figure 1. We can observe the shapes of these curves are quite similar, suggesting that the behavior is relatively consistent between these two groups.

We also discover that roughly 10% of Let's Encrypt domain sets in all categories had a median re-issuance period of *greater than* 90 days, meaning the certificates were more often than not renewed after expiry. This behavior could occur if the administrator did not set up a `cron` job, incorrectly set up a `cron` job to run very infrequently, or if the system was not always online. We leave a deeper exploration of these domain sets to future work.

Finally, we also briefly compare the Let's Encrypt domain set behavior to that of other CAs. To do so, we extract the domain sets in the same manner from the CT logs for the top 10 CAs (other than Let's Encrypt), and compute the median reissuance periods in the same manner per CA. We plot these as well in Figure 1 under the lines "cPanel", "Sectigo", and "Other CAs"; we separate out cPanel and Sectigo as they show different behavior than the others. In brief, we see that most of the other top CAs show very long median reissuance periods, while cPanel shows a "spike" at 75 days and Sectigo at 60, 90, and 120 days.

Coefficient of Variation (CoV) While the median of the reissue time periods being so clearly at 60 days is suggestive that the administrators use automated software to reissue their certificates, it is not entirely definitive. Thus, we look for further evidence of automation by looking at how *similar* the reissuance periods of a given domain set are to each other. In other words, if a given domain set was using an automated process to reissue certificates, we would expect that the period between reissues would be highly consistent.

To do so, we calculate the *coefficient of variation* (CoV)—which is simply the standard deviation of a distribution over its mean—of the amounts of time between each successive reissuance. Automated reissuance would often lead to a consistent period between reissues, meaning that the CoV would be low i.e., 0.1 or smaller. We choose the CoV threshold of 0.1 as a cut-off as would allow, for example, a domain set with a mean reissuance time of 60 days to be classified as automated if the variance is less than 6 days (roughly one week). For this analysis, we only keep the domain sets where we have a sufficient reissue history of at least five reissues. Figure 2 plots the distribution of CoVs for the reissue time periods for each domain set under the "all LE certs" line. We can observe that many domain sets do show evidence of automation: 30.3% of domain sets have a CoV of less than 0.1.

We were concerned that particular domains with unusual patterns of reissuance may end up artificially shaping this curve, as our analysis is at the *domain set* level, rather than at the *system administrator* level. Thus, we additionally perform an aggregation to the second-level domain to see whether particular domains are skewing the results.

We aggregate domain sets into second-level domain through a weighted average: for each second-level domain S, we compute the average CoV for all domain sets that have at least one domain name from S. For domain sets that include domains from multiple second-level domains, we simply

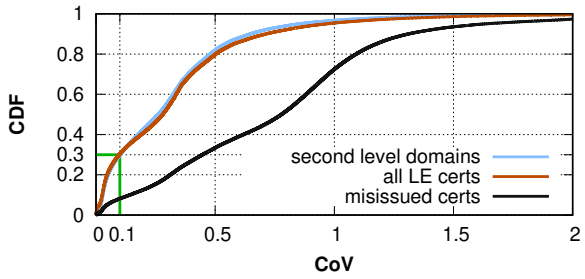


Fig. 2. Distribution of coefficient of variation (CoV) for all Let's Encrypt domain sets, second level domains, and the misissued certificates.

weigh the domain set's CoV by the fraction of domains that belong to S. The resulting cumulative distribution is also shown in Figure 2, and we can observe that the distribution is quite similar to the analysis at the domain set level. Thus, we have some confidence that the (potentially odd) behavior of a small number of second-level domain sets is not dramatically altering the results.

Noticing that many domain sets tend to have a high CoVs, we next examine how well the CoV methodology identifies domain sets with regular reissuance patterns. We do so by dividing up domain sets by their CoV, and plotting the cumulative distribution of their median reissuance time in Figure 3. We can immediately observed that the median reissuance time of certificates varies dramatically by CoV: we find that the median reissuance period of domain sets with a very low CoV (0.1 or smaller) is 60 days, while domain sets with a CoV greater than 1 are much less predictable. Further, Figure 3 reveals that over 88% of domain sets with highly automated reissuance (CoV < 0.001) have a median reissue period of between 59 and 61 days (consistent with the reissue occurring during the first `cron` job to run after the 60 day period).

Initial renewal setup Moving on, we hypothesize that the initial setup and use of ACME clients may result in multiple, irregular requests, which would affect our CoV calculation. To understand the effects, we focus on certificates that have at least five reissues, and make the assumption that most administrators would be comfortable with operating ACME clients after a year. Out of 188M unique domain sets, only 60M unique domain certificates have at least five reissues; these form the basis of the following analysis.

Roughly 48.2% of domain sets with at least five reissues have a CoV less than 0.1. However, if we also look at *subsequences* of reissues, ignor-

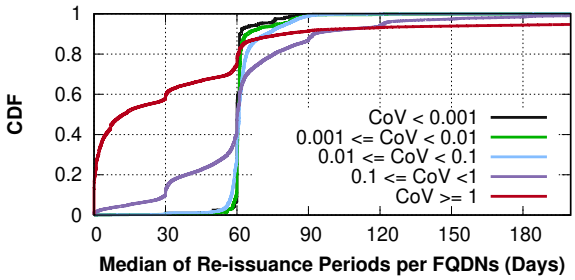


Fig. 3. Distribution of all Let's Encrypt domain set reissuances, divided across different CoV groups. We can see the groups with lower CoV tend to have a median reissuance period of 60 days.

ing the first set of reissues as long as at least five reissues remain, we can identify an *additional* 29.9% domain sets that have a subsequence of reissues with a CoV less than 0.1. In other words, 78% of domain sets with a subsequence of at least five reissues have a regular reissue cycle that begins at some point in their lifetimes. Thus, we have identified a limitation of the CoV metric, as it may be too conservative in cases where administrators have an irregular initial reissunce cycle before fully debugging their ACME client setup.

4.2 Manual Reissuance

Having a good understanding on domain sets with likely automated reissuance infrastructure, we now turn to examine what happens for these domains when manual intervention is required. To do so, we use the Let's Encrypt misissuance bug as a natural experiment: because all of these certificates need to be reissued, we have a collection of domain sets where we can study whether the system administrator did, in fact, reissue their certificate.

We first need to examine the set of certificates affected by the bug, which was announced on February 29, 2020. Let's Encrypt reported that over 3M certificates were affected; we collected all of these certificates and plotted their issue time in Figure 4. We can see that these certificates went as far back as December 2, 2019, which would be expected given Let's Encrypt 90-day certificate lifetime. Importantly, the certificates appear to have been issued uniformly throughout the prior 90 days.

However, there are multiple reasons why these misissued certificates are *not* a random sample of all Let's Encrypt certificates. *First*, the bug only affected certificates with multiple domains in them, meaning any

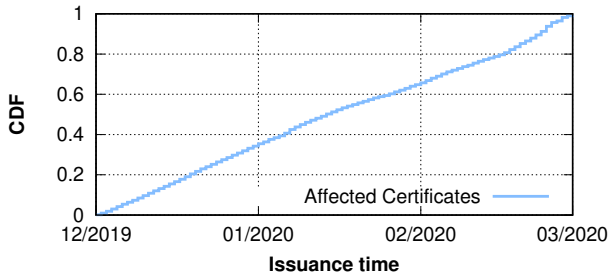


Fig. 4. Distribution of when the misissued certificates were issued.

certificates with a single domain were not misissued. *Second*, and more importantly, it only affected domains *where the CAA record had been verified within the past 30 days*. As we observed previously, most certificates are reissued after *60* days, this means that the only certificates that were affected were ones that were either (a) not on a regular schedule to begin with, or (b) were on a regular schedule, but happened to be reissued in late 2019/early 2020 for another reason. This observation explains why the misissued certificates behave quite differently from all Let's Encrypt certificates in Figures 1 and 2: due to the nature of the bug, domain sets that had regular, 60-day reissue periods were much less likely affected. In fact, such domain sets would only have been affected if one of the domains in the domain set happened to be in *another* domain set whose certificate was reissued in the previous 30-day time period, or where the administrator had manually reissued that domain set during that period.

Nevertheless, we need to identify when we believe a certificate was manually reissued from among the misissued certificates. Recall that we do not have access to Let's Encrypt's logs, so we can only rely on the timestamps public CT logs. We want to see how certificates affected by the bug were automatically reissued before the bug, but manually issued a new certificate in response to the bug. We therefore focus on those domain sets that (a) were affected by the Let's Encrypt bug, (b) were on a regular cycle prior to February 29, 2020, and (c) had at least one new certificate issued after February 29, 2020 (to see if the regular cycle continued). To see if a domain set was on a regular reissue cycle prior to February 29, 2020, we see if the five certificate reissues prior to the bug date had a CoV less than 0.1. In total, 98,652 domain sets satisfy these three criteria.

Next, we calculate the CoV of the five reissues before the bug date *and* the first reissue after the bug date. If the CoV including the new certificate

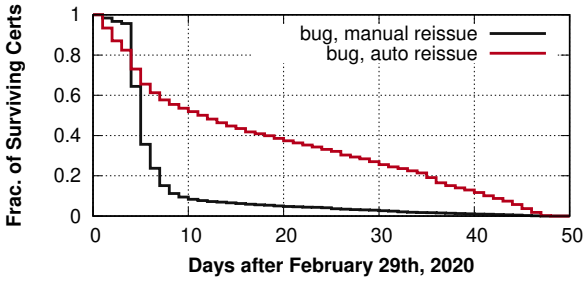


Fig. 5. Graph showing how long certificates "survived" after Let's Encrypt bug was announced. We plot (a) the 33,099 certificates that we inferred were manually reissued, and (b) the 66,553 certificates that we inferred were automatically reissued. We can see the manually reissued certificates we largely reissued quickly after the bug announcement.

is high, then the first certificate after the Let's Encrypt bug could not have been automatically reissued; some form of manual intervention disrupted the issue cycle and caused the previously low CoV to increase. If the CoV including the new certificate remains low (<0.1), then the new certificate was likely issued on its expected regular schedule. It is also possible, though unlikely, that a new certificate was manually issued at the same time we would expect the next automatically reissued certificate. Of the 98,652 domain sets, 33,099 saw a significant CoV increase (i.e., likely had manual intervention) in the first reissue after the Let's Encrypt bug, and 65,553 likely did not.

We now examine *how quickly* these 33,099 certificates were manually reissued after Let's Encrypt announced the bug, and emailed all administrators to tell them to reissue their certificates manually. Figure 5 plots the number of these certificates that survive in the line labeled "bug, manual reissue". We can observe that most certificates that are manually reissued are reissued quite quickly: within a week, over 84% of all certificates that we believe are manually reissued have been reissued. For comparison, we plot the same graph for the 66,553 certificates that were reissued close to their next reissue in the line labeled "bug, auto reissue". This group shows less-prompt reissuing than the manual reissues, as only 42% of likely-automatic reissues occurred in the 7 days following the bug announcement.

Recall from Section 2 that Let's Encrypt rescinded its decision to revoke certificates on March 5, 2020 (five days after the initial email stating they would be revoking certificates on March 5, 2020). Thus, there may be system administrators who intended to reissue but who delayed

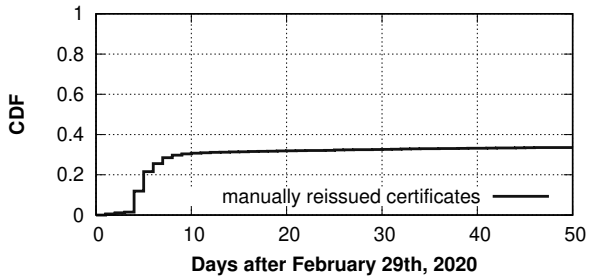


Fig. 6. Cumulative distribution of the manual reissues (CoV less than 0.1) after the announcement of the Let's Encrypt bug.

reissuing their certificates, only to decide it was no longer necessary after receiving the second message. While we cannot measure how large this group is, we believe it is likely small as Let's Encrypt decided sent out the second message on the day they originally announced as the deadline to reissue. Regardless, our results still serve as a lower bound on the number of system administrators who did take action.

Finally, we plot the same data as in Figure 5, but do so as a fraction of *all* misissued domain sets with a CoV less than 0.1 before the bug date. This graph is presented in Figure 6, and it shows that among all the domain sets with a CoV less than 0.1 (those on a regular schedule before Feb. 29, 2020), at least 28% had reissued their certificate manually within a week of the bug announcement. This result is a significant improvement over prior incidents; with the Heartbleed bug, after a week, barely 10% of affected certificates had been reissued (and even fewer revoked) [22]. Even though circumstances between the two bugs differ significantly (such as notification of revocation), they both provide opportunities for natural experiments to see how the PKI is evolving over time, and the comparison suggests that system administrators may now be better managing the PKI.

5 Concluding discussion

Over the past five years, the TLS PKI ecosystem has changed dramatically: largely due to new CAs such as Let's Encrypt, we have moved from primarily expensive, long-lived certificates to primarily free, short-lived certificates. In this paper, we examined whether this change in the nature of the certificate ecosystem has also improved the *management* of the TLS PKI, as it has been previously been observed that system adminis-

trators often fail to properly manage their certificates. Though we find significant evidence that most clients of Let's Encrypt have indeed set up automated processes for reissuing and installing their certificates using over four years of CT logs, a surprising fraction (20%) of clients with a sufficiently long history of being reissued still appear to use manual processes. Moreover, we find evidence that even when manual intervention is required, system administrators are more prompt in doing so when compared to studies from the 2014 Heartbleed bug and the 2009 Debian PRNG bug. Taken together, our results underscore the importance of reducing the burden of management of the TLS PKI, and how changes in the infrastructure and tools available to system administrators can lead to significant management improvements.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Cecilia Testart, for their helpful comments. This research was supported in part by NSF grants CNS-1900879 and CNS-1901325.

References

1. J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen, and a. B. Warren. Let's Encrypt: An Automated Certificate: Authority to Encrypt the Entire Web. *CCS*, 2019.
2. F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. *CCS*, 2016.
3. T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measuring and Applying Invalid SSL Certificates: The Silent Majority. *IMC*, 2016.
4. T. Chung, R. van Rijswijk-Deij, D. Choffnes, A. Mislove, C. Wilson, D. Levin, and B. M. Maggs. Understanding the Role of Registrars in DNSSEC Deployment. *IMC*, 2017.
5. CAA Rechecking Bug. `https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591`.
6. Certbot User Guide. `https://certbot.eff.org/docs/using.html`.
7. Certificate Transparency in Chrome. 2019. `https://github.com/chromium/ct-policy/blob/master/ct_policy.md`.
8. Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The Matter of Heartbleed. *IMC*, 2014.
9. Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. *IMC*, 2013.
10. Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. *USENIX Security*, 2013.

11. Heartbleed Bug. http://heartbleed.com.
12. B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013. http://www.ietf.org/rfc/rfc6962.txt.
13. Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. *IMC*, 2015.
14. Let's Encrypt. https://letsencrypt.org.
15. Let's Encrypt Community Support: 2020.02.29 CAA Rechecking Bug. https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591/3.
16. Let's Encrypt Stats. https://letsencrypt.org/stats/.
17. LetsEncrypt: Submit final certs to CT logs (#3640). https://github.com/letsencrypt/boulder/commit/1271a15be79b9717ee5b98e707b76e7ac86a9a0e.
18. S. Matsumoto and R. M. Reischuk. IKP: Turning a PKI Around with Decentralized Automated Incentives. *IEEE S&P*, 2017.
19. Q. Scheitle, T. Chung, J. Hiller, O. Gasser, J. Naab, R. van Rijswijk-Deij, O. Hohlfeld, R. Holz, D. Choffnes, A. Mislove, and G. Carle. A First Look at Certification Authority Authorization (CAA). *CCR*, 48(2), 2018.
20. S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability. *IMC*, 2009.
21. L. Zhang, D. Choffnes, T. Dumitraş, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. *IMC*, 2014.
22. L. Zhang, D. Choffnes, T. Dumitraş, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed. *CACM*, 61(3), https://cacm.acm.org/magazines/2018/3/225489-analysis-of-ssl-certificate-reissues-and-revocations-in-the-wake-of-heartbleed/fulltext, 2018.
23. acme.sh. https://github.com/acmesh-official/acme.sh.

6. Conclusion

In this thesis, we focused on analyzing security protocols and features on the internet. A significant part of the analysis in our work focused on how stakeholders, i.e. end-users, developers e.t.c utilized these protocols and features. Each paper in this thesis addressed one or more of the research questions outlined in Chapter 1. We discuss the answers to the research questions from our work below.

6.1. RQ1 - How does the technical specification of protocols and complexity of usage influence their adoption and security?

Protocols or security features are usually specified in documents such as Request For Comments (RFCs) to guide their implementation and usage. Regardless of how secure a protocol or feature is in theory, if the technical specification is unclear or ambiguous, it could lead to misinterpretation. Misinterpretation, in this case, could affect the implementation quality of the tool. Misinterpretation of the specification could also result in the usage of the tool in a manner that defeats its purpose.

Section 4 specifically answers this question. SRI was a recent security feature introduced to prevent the manipulation of external subresources included in web pages. SRI ensures that a subresource is only loaded if its hash matches the hash included by the website developer. In the work of Section 4, we performed an empirical study of Subresource Integrity. Our study included a longitudinal study as well as a developer study, and it revealed that the SRI recommendation was ambiguous on specific points, causing the developers to misinterpret its behaviour. This misinterpretation could lead to the wrong usage, leaving the developers' webpage open to attacks. Furthermore, we discovered that more developers adopted SRI in environments where platforms automated its use.

Our work in Section 5 tried to understand how recent changes to the TLS-PKI have influenced its adoption. The TLS-PKI has been known to be very complex. A significant component of the TLS-PKI is the certificate. Certificates are a signed attestation from a third party. Due to the complexity of the TLS-PKI and the effort involved in managing certificates, developers have been known to adopt insecure practices such as using a certificate past its due date, reusing a revoked certificate, or even the continual usage of insecure certificates. Recently, there has been a move towards the automation of certificate management led by Let's Encrypt. Let's Encrypt provides free certificates and also an automated system for managing these certificates without the intervention of system administrators or users. In this research, we explicitly studied the adoption

of automated certificate reissuance and discovered that over 80% of users had adopted automated certificate reissuance.

From our work on SRI [14] and the TLS-PKI [55], we discovered that the clarity of the technical specification and the availability of automated tools ease the adoption of complex security protocols or features.

6.2. RQ2 Do protocols enhancements fulfill their security guarantees in practice without causing other vulnerabilities?

Security protocols or features are often introduced to resolve vulnerabilities or to combat specific threats. However, sometimes these new protocols or features present security threats of their own or weaken certain security guarantees such as privacy, while improving others.

Certificate Transparency is an example of the latter. The creators of Certificate Transparency weakened the privacy guarantee of the entire TLS-PKI while combating the threat of certificate misissuance. Certificate Transparency is a system for logging all certificates. The goal is to mandate all CAs to log all the certificates they issue to a public append-only log for auditing by users and browsers. For browsers to verify that this certificate is logged, they have to validate information, i.e., a Signed Certificate Timestamp (SCT) provided by the entity presenting the certificate. The process of validating this information reveals the users' browsing habits, thereby weakening their privacy on the internet. Our work in Section 2 proposed a novel solution using Private Information Retrieval (PIR), Multi-tier Merkle trees and Distributed Point Functions (DPFs) to protect users' privacy without losing the benefit of public certificate audits.

Additionally, we analyzed a novel PKI implementation called PB-PKI in Section 3 of this thesis. PB-PKI is a novel PKI implementation based on the blockchain. Its goal is to create a privacy-aware PKI implementation with all the stakeholders on the blockchain. We analyzed the PB-PKI and found that the PB-PKI proposal introduced multiple security weaknesses that would render it unusable. We then proposed improvements to this protocol and implemented a proof-of-concept to demonstrate its usage.

From our work on Certificate Transparency [37] and blockchain PKI [56], we discovered that the creation of a protocol should be iterative with multiple layers of feedback to catch the possible introduction of new security weaknesses.

6.3. Future Work

This thesis focused on the HTTPS ecosystem and its components. In the future, we wish to analyze more complex protocols and provide solutions to the weaknesses we find. Further, we aim to raise more awareness in the professional community about automation and how it aids in adopting security protocols. Finally, our goal is to understand how the complexity of email protocols affects their usage.

Bibliography

[1] *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017.

[2] Andreas Abraham, Felix Hörandner, Olamide Omolola and Sebastian Ramacher. "Privacy-Preserving eID Derivation for Self-Sovereign Identity Systems". In: *ICICS*. Vol. 11999. Lecture Notes in Computer Science. Springer, 2019, pp. 307–323.

[3] Maarten Aertsen, Maciej Korczynski, Giovane C. M. Moura, Samaneh Tajalizadehkhoob and Jan van den Berg. "No domain left behind: is Let's Encrypt democratizing encryption?" In: *CoRR* abs/1612.03005 (2016). arXiv: 1612.03005. URL: http://arxiv.org/abs/1612.03005.

[4] Eric Allman, Jon Callas, Mark Delany, Miles Libbey, Jim Fenton and Michael Thomas. "DomainKeys Identified Mail (DKIM) Signatures". In: *RFC* 4871 (2007), pp. 1–71.

[5] Waqar Aqeel, Zachary Hanif, James Larisch, Olamide Omolola, Taejoong Chung, Dave Levin, Bruce Maggs, Alan Mislove, Bryan Parno and Christo Wilson. "Assertion-Carrying Certificates". English. In: *Workshop on Foundations of Computer Security 2020*. June 2020.

[6] Roy Arends, Rob Austein, Matt Larson, Dan Massey and Scott Rose. "DNS Security Introduction and Requirements". In: *RFC* 4033 (2005), pp. 1–21. DOI: 10.17487/RFC4033. URL: https://doi.org/10.17487/RFC4033.

[7] Vijay Atluri and Günther Pernul, eds. *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*. Vol. 8566. Lecture Notes in Computer Science. Springer, 2014.

[8] Louise Axon and Michael Goldsmith. "PB-PKI: A Privacy-aware Blockchain-based PKI". In: *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications - Volume 6: SECRYPT, (ICETE 2017)*. INSTICC. SciTePress, 2017, pp. 311–318. ISBN: 978-989-758-259-2. DOI: 10.5220/0006419203110318.

[9] Daniel Barbará, Rajni Goel and Sushil Jajodia. "Using Checksums to Detect Data Corruption". In: *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*. Ed. by Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl and Torsten Grust. Vol. 1777. Lecture Notes in Computer Science. Springer, 2000, pp. 136–149. DOI: 10.1007/3-540-46439-5_9. URL: https://doi.org/10.1007/3-540-46439-5_9.

[10] Steffen Bickel and Tobias Scheffer. "Dirichlet-Enhanced Spam Filtering based on Biased Samples". In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. Ed. by Bernhard Schölkopf, John C. Platt and Thomas Hofmann. MIT Press, 2006, pp. 161–168. URL: http://papers.nips.cc/paper/3158-dirichlet-enhanced-spam-filtering-based-on-biased-samples.

[11] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford and Prateek Mittal. "Bamboozling Certificate Authorities with BGP". In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 833–849. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/birge-lee.

[12] Dan Boneh. "Twenty Years of Attacks on the RSA Cryptosystem". In: 1998.

[13] Frank Cangialosi, Taejoong Chung, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove and Christo Wilson. "Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers and Shai Halevi. ACM, 2016, pp. 628–640. DOI: 10.1145/2976749.2978301. URL: https://doi.org/10.1145/2976749.2978301.

[14] Bertil Chapuis, Olamide Omolola, Mauro Cherubini, Mathias Humbert and Kévin Huguenin. "An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources". In: *WWW*. ACM / IW3C2, 2020, pp. 34–45.

[15] Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic and Kévin Huguenin. "Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes and XiaoFeng Wang. ACM, 2018, pp. 1256–1271. DOI: 10.1145/3243734.3243746. URL: https://doi.org/10.1145/3243734.3243746.

[16] Taejoong Chung, Yabing Liu, David R. Choffnes, Dave Levin, Bruce MacDowell Maggs, Alan Mislove and Christo Wilson. "Measuring and Applying Invalid SSL Certificates: The Silent Majority". In: *Internet Measurement Conference*. ACM, 2016, pp. 527–541.

[17] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John P. Rula, Nick Sullivan and Christo Wilson. "Is the Web Ready for OCSP Must-Staple?" In: *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*. ACM, 2018, pp. 105–118. URL: https://dl.acm.org/citation.cfm?id=3278543.

[18] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove and Christo Wilson. "A Longitudinal, End-to-End View of the DNSSEC Ecosystem". In: *USENIX Security Symposium*. USENIX Association, 2017, pp. 1307–1322.

[19] Taejoong Chung, Roland van Rijswijk-Deij, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove and Christo Wilson. "Understanding the role of registrars in DNSSEC deployment". In: *Internet Measurement Conference*. ACM, 2017, pp. 369–383.

[20] William W. Cohen. "Learning Rules that Classify E-Mail". In: 1996.

[21] Sophia Yakoubov Conner Fromknecht Dragos Velicanu. *CertCoin: A NameCoin Based Decentralized Authentication System*. Massachusetts Institute of Technology, 2014.

[22] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley and W. Timothy Polk. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile". In: *RFC* 5280 (2008), pp. 1–151.

[23] David Derler, Christian Hanser and Daniel Slamanig. "Privacy-Enhancing Proxy Signatures from Non-interactive Anonymous Credentials". In: *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*. Ed. by Vijay Atluri and Günther Pernul. Vol. 8566. Lecture Notes in Computer Science. Springer, 2014, pp. 49–65. DOI: 10.1007/978-3-662-43936-4_4. URL: https://doi.org/10.1007/978-3-662-43936-4_4.

[24] David Derler, Christian Hanser and Daniel Slamanig. "Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives". In: *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*. Ed. by Kaisa Nyberg. Vol. 9048. Lecture Notes in Computer Science. Springer, 2015, pp. 127–144. DOI: 10.1007/978-3-319-16715-2_7. URL: https://doi.org/10.1007/978-3-319-16715-2_7.

[25] John DeTreville. "Making Certificates Programmable". In: *First Annual PKI Workshop*. 2002.

[26] Melvin Diale, Turgay Çelik and Christiaan Van Der Walt. "Unsupervised feature learning for spam email filtering". In: *Computers & Electrical Engineering* 74 (2019), pp. 89–104. DOI: 10.1016/j.compeleceng.2019.01.004. URL: https://doi.org/10.1016/j.compeleceng.2019.01.004.

[27] Benjamin Dowling, Felix Günther, Udyani Herath and Douglas Stebila. "Secure Logging Schemes and Certificate Transparency". In: *ESORICS (2)*. Vol. 9879. Lecture Notes in Computer Science. Springer, 2016, pp. 140–158.

[28] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer and Vern Paxson. "The Matter Of Heartbleed". In: *ACM Internet Measurement Conference*. 2014.

[29] Cynthia Dwork and Christina Ilvento. *SmartCert: Redesigning Digital Certificates with Smart Contracts*. `https://arxiv.org/pdf/2003.13259.pdf`. 2020.

[30] Terence Eden. *Major Sites Running Unauthenticated JavaScript on Their Payment Pages*. en-GB. https://shkspr.mobi/blog/2018/11/major-sites-running-unauthenticated-javascript-on-their-payment-pages/, last visited: Oct. 2019. 2018.

[31] William Enck and Adrienne Porter Felt, eds. *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 2018.

[32] Let's Encrypt. *Let's Encrypt*. `https://letsencrypt.org/`.

[33] Let's Encrypt. *Let's Encrypt Growth*. `https://letsencrypt.org/stats/`.

[34] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach and Tim Berners-Lee. "Hypertext Transfer Protocol - HTTP/1.1". In: *RFC* 2616 (1999), pp. 1–176. DOI: `10.17487/RFC2616`. URL: `https://doi.org/10.17487/RFC2616`.

[35] Niv Gilboa and Yuval Ishai. "Distributed Point Functions and Their Applications". In: *EUROCRYPT*. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 640–658.

[36] Heartbleed. *The Heartbleed Bug*. `https://heartbleed.com/`.

[37] Daniel Kales, Olamide Omolola and Sebastian Ramacher. "Revisiting User Privacy for Certificate Transparency". In: *4th IEEE European Symposium on Security and Privacy*. 2019.

[38] Kristián Kozák, Bum Jun Kwon, Doowon Kim, Christopher Gates and Tudor Dumitras. "Issued for Abuse: Measuring the Underground Trade in Code Signing Certificate". In: *CoRR* abs/1803.02931 (2018). arXiv: `1803.02931`. URL: `http://arxiv.org/abs/1803.02931`.

[39] Murray S. Kucherawy and Elizabeth D. Zwicky. "Domain-based Message Authentication, Reporting, and Conformance (DMARC)". In: *RFC* 7489 (2015), pp. 1–73.

[40] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J. Alex Halderman and Michael Bailey. "Security Challenges in an Increasingly Tangled Web". en. In: *Proc. of the Int'l Conf. on World Wide Web (WWW)*. Perth, Australia: ACM, 2017, pp. 677–684. ISBN: 978-1-4503-4913-0. DOI: `10.1145/3038912.3052686`.

[41] James Larisch, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove and Christo Wilson. "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 539–556. DOI: 10.1109/SP.2017.17. URL: https://doi.org/10.1109/SP.2017.17.

[42] Ben Laurie. "Certificate Transparency". In: *ACM Queue* 12.8 (2014), pp. 10–19.

[43] Ben Laurie, Adam Langley and Emilia Käsper. "Certificate Transparency". In: *RFC* 6962 (2013), pp. 1–27.

[44] David Lie, Mohammad Mannan, Michael Backes and XiaoFeng Wang, eds. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018.

[45] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman and Christo Wilson. "An End-to-end Measurement Of Certificate Revocation In The Web's PKI". In: *ACM Internet Measurement Conference*. 2015.

[46] Kirk Lougheed and Yakov Rekhter. "Border Gateway Protocol 3 (BGP-3)". In: *RFC* 1267 (1991), pp. 1–35. DOI: 10.17487/RFC1267. URL: https://doi.org/10.17487/RFC1267.

[47] Kirk Lougheed and Yakov Rekhter. "Border Gateway Protocol (BGP)". In: *RFC* 1163 (1990), pp. 1–29. DOI: 10.17487/RFC1163. URL: https://doi.org/10.17487/RFC1163.

[48] Wouter Lueks and Ian Goldberg. "Sublinear Scaling for Multi-Client Private Information Retrieval". In: *Financial Cryptography*. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 168–186.

[49] Ralph C. Merkle. "A Certified Digital Signature". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 218–238. DOI: 10.1007/0-387-34805-0_21. URL: https://doi.org/10.1007/0-387-34805-0_21.

[50] Paul V. Mockapetris. "Domain names - concepts and facilities". In: *RFC* 1034 (1987), pp. 1–55. DOI: 10.17487/RFC1034. URL: https://doi.org/10.17487/RFC1034.

[51] John G. Myers and Marshall T. Rose. "Post Office Protocol - Version 3". In: *RFC* 1939 (1996), pp. 1–23.

[52] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[53] Cormac O'Brien and Carl Vogel. "Spam filters: bayes vs. chi-squared; letters vs. words". In: *Proceedings of the 1st Intenational Symposium on Information and Communication Technologies, Dublin, Ireland, September 24-26, 2003*. Vol. 49. ACM International Conference Proceeding Series. Trinity College Dublin, 2003, pp. 291–296. URL: https://dl.acm.org/citation.cfm?id=963658.

[54] Olamide Omolola, Stefan Josef More, Edona Fasllija, Georg Wagner and Lukas Alber. "Policy-based Access Control for the IoT and Smart Cities". In: *Open Identity Summit*. Mar. 2019.

[55] Olamide Omolola, Richard Roberts, Md. Ishtiaq Ashiq, Taejoong Chung, Dave Levin and Alan Mislove. "Measurement and Analysis of Automated Certificate Reissuance". In: Lecture Notes in Computer Science 12671 (2021), pp. 161–174.

[56] Paul Plessing and Olamide Omolola. "Revisiting Privacy-aware Blockchain Public Key Infrastructure". In: *ICISSP*. SCITEPRESS, 2020, pp. 415–423.

[57] Jon Postel and Joyce K. Reynolds. "Telnet Protocol Specification". In: *RFC* 854 (1983), pp. 1–15. DOI: 10.17487/RFC0854. URL: https://doi.org/10.17487/RFC0854.

[58] *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*. ACM, 2018.

[59] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160.

[60] Ronald L. Rivest, Adi Shamir and Yael Tauman. "How to Leak a Secret: Theory and Applications of Ring Signatures". In: *Theoretical Computer Science, Essays in Memory of Shimon Even*. Ed. by Oded Goldreich, Arnold L. Rosenberg and Alan L. Selman. Vol. 3895. Lecture Notes in Computer Science. Springer, 2006, pp. 164–186. DOI: 10.1007/11685654_7. URL: https://doi.org/10.1007/11685654_7.

[61] Quirin Scheitle, Taejoong Chung, Jens Hiller, Oliver Gasser, Johannes Naab, Roland van Rijswijk-Deij, Oliver Hohlfeld, Ralph Holz, David R. Choffnes, Alan Mislove and Georg Carle. "A First Look at Certification Authority Authorization (CAA)". In: *Computer Communication Review* 48.2 (2018), pp. 10–23.

[62] Jay Schiavo. "Code signing for end-user peace of mind". In: *Network Security* 2010.7 (2010), pp. 11–13. DOI: 10.1016/S1353-4858(10)70093-3. URL: https://doi.org/10.1016/S1353-4858(10)70093-3.

[63] Ronak Shah and Kailas Patil. "A Measurement Study of the Subresource Integrity Mechanism on Real-World Applications". en. In: *International Journal of Security and Networks* 13.2 (2018), p. 129. ISSN: 1747-8405, 1747-8413. DOI: 10.1504/IJSN.2018.092474.

[64] Nick Sullivan. *Keyless SSL: The Nitty Gritty Technical Details*. 2014-09-19. URL: https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/ (visited on 16/05/2019).

[65] W3C. *Subresource Integrity*. en. https://www.w3.org/TR/SRI/. 2016.

[66] Georg Wagner, Olamide Omolola and Stefan Josef More. "Harmonizing Delegation Data Formats". In: *Lecture Notes in Informatics*. Vol. 2017. Germany: Gesellschaft für Informatik, Oct. 2017. ISBN: 978-3-88579-671-8. DOI: 20.500.12116/3578.

[67] Stephanie Weinhardt and Olamide Omolola. "Usability of Policy Authoring tools: A layered approach". In: *International Conference on Information Systems Security and Privacy*. Feb. 2019.

[68] Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers and Shai Halevi, eds. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 2016.

[69] Tatu Ylönen and Chris Lonvick. "The Secure Shell (SSH) Transport Layer Protocol". In: *RFC* 4253 (2006), pp. 1–32. DOI: 10.17487/RFC4253. URL: https://doi.org/10.17487/RFC4253.

[70] Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl and Torsten Grust, eds. *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*. Vol. 1777. Lecture Notes in Computer Science. Springer, 2000.

[71] Liang Zhang, David Choffnes, Tudor Dumitras, Dave Levin, Alan Mislove, Aaron Schulman and Christo Wilson. "Analysis Of SSL Certificate Reissues And Revocations In The Wake Of Heartbleed". In: *ACM Internet Measurement Conference*. 2014.

[72] Liang Zhang, David R. Choffnes, Tudor Dumitras, Dave Levin, Alan Mislove, Aaron Schulman and Christo Wilson. "Analysis of SSL certificate reissues and revocations in the wake of heartbleed". In: *Commun. ACM* 61.3 (2018), pp. 109–116. DOI: 10.1145/3176244. URL: https://doi.org/10.1145/3176244.

[73] Yan Zhang, PengFei Liu and JingTao Yao. "Three-way Email Spam Filtering with Game-theoretic Rough Sets". In: *International Conference on Computing, Networking and Communications, ICNC 2019, Honolulu, HI, USA, February 18-21, 2019*. IEEE, 2019, pp. 552–556. DOI: 10.1109/ICCNC.2019.8685642. URL: https://doi.org/10.1109/ICCNC.2019.8685642.

[74] Philip Zimmermann. *PGP User's Guide, Volume I: Essential Topics*. 1994. URL: https://web.pa.msu.edu/reference/pgpdoc1.html (visited on 04/11/2018).

A. Appendix

We include other activities that we were involved in during this PhD below

LIGHTest Project The author of this thesis led a workpackage in the successful LIGHTest project. LIGHTest is an EU funded project that aims to enable cross-border verification of electronic transactions. The author was responsible for the timely deployment of the project software components. He also made the following presentations in the context of the project.

1. **Cross-Border Verification in LIGHTest** presented at *OIX Economics of Identity, London* (8th November 2018).
2. **Cross-Border Verification in LIGHTest** presented at *Consumer Identity World, Singapore* (22nd November 2018).
3. **Workpackage Updates** presented at multiple LIGHTest meeting over the course of 3 years.

Research Activities The author has presented multiple papers at different conferences. He also had research internships at University of Lausanne, Switzerland and Northeastern University.

Supervised Student Theses

1. Paul Plessing: *Privacy Aware PKI on Blockchain* (Bsc/Completed)
2. Roman Markus Holler: *PKI on Blockchain* (BSc/Completed)