Next-Generation Web Public-Key Infrastructure Technologies

by

H. M. Udyani Shanika Kumari Herath

BSc (Hons) specialising Computer Science
(University of Peradeniya) – January 2013

Thesis submitted in accordance with the regulations for the Degree of Doctor of Philosophy

School of Electrical Engineering and Computer Science
Science and Engineering Faculty
Queensland University of Technology

2019

Keywords

Public-key Infrastructure, X.509 Certificate, Certificate Transparency, Logging Scheme, DPKIT, Blockchain, Hybrid Signature

Abstract

Public key infrastructures (PKIs) facilitate the safe transfer of information in many applications, including e-commerce and banking on the web, confidential email, and distribution of software. A PKI is used to manage the creation, distribution, and lifecycle of keys and digital certificates which are used in public key encryption and digital signature services when building trustworthy networks. In large PKIs, such as on the web, there are hundreds of certificate authorities who are trusted to honestly distribute certificates binding public keys to identities. With potentially millions of certificates distributed, it can become challenging to ensure all trusted certificate authorities are behaving honestly.

In this work, we investigate methods for ensuring the transparency of operations by trusted parties in PKIs, thereby enabling independent auditing of their actions. Certificate Transparency (CT) is a recently deployed PKI technology that relies on independent loggers to maintain logs of seen certificate, against which clients verify the presence of a certificate during routine operation. We develop an abstract model of a logging scheme to capture the intended security properties of Certificate Transparency, and provide the first formal arguments that CT achieves these security properties.

We subsequently propose a new approach, called Distributed PKI Transparency (DPKIT), which eliminates individual loggers and instead distributes the maintenance of an event log across multiple parties using blockchain technology. We extend the abstract model of logging schemes to cover this distributed case and show that DPKIT meets these goals.

Finally, we look to the future and consider how PKI can transition to support secure communication in a world with quantum computers. As digital signatures play an essential role in PKI, we first examine how the security properties of signature schemes should be defined in the face of semi-quantum or fully-quantum adversaries. As part of a multi-year transition plan, we envision the

use of traditional and quantum-resistant signature schemes simultaneously, so we proceed to consider how to combine two signature schemes with differing security properties, and then investigate how these approaches could be used in widely used security standards such as X.509 certificates, the Transport Layer Security (TLS) protocol, and S/MIME-secured email.

Our research models a currently deployed PKI technology, designs a new technique by extending that model and lastly look in to the future aspects of public key infrastructure. We contribute to the knowledge on PKI trust models and technologies to augment security and productivity of current and future developments.

Contents

List of Figures

List of Tables

List of Acronyms

CA	**C**ertificate **A**uthority
CT	**C**ertificate **T**ransparency
CRL	**C**ertificate **R**evocation **L**ist
ECC	**E**lliptic-**C**urve **C**ryptography
EUFCMA	**E**xistential **U**nforgeability **U**nder **C**hosen **M**essage **A**attack
IETF	**I**nternet **E**ngineering **T**ask **F**orce
MTH	**M**erkle **T**ree **H**ash
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
OCSP	**O**nline **C**ertificate **S**tatus **P**rotocol
PKI	**P**ublic **K**ey **I**nfrastructure
RSA	**R**ivest-**S**hamir-**A**dleman
SCT	**S**igned **C**ertificate **T**imestamp
STH	**S**igned **C**ertificate **H**ead
TLS	**T**ransport **L**ayer **S**ecurity

Declaration

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: QUT Verified Signature **Date:** April 2019

Acknowledgements

There are many people to whom I am truly grateful for supporting me throughout this PhD journey. First of all, I thank my supervisors, Associate Professor Douglas Stebila, for his continued guidance, even after he moved to Canada at the end of my first year, and Dr. Matthew McKague, and Associate Professor Xavier Boyen, for assisting and guiding me in my final year. This thesis would not have been possible, were it not for their expertise, immense support, trust, patience, guidance and friendship. No words quite capture my gratitude.

My research studies have been supported by QUTHDR and QUTPRA scholarships from Queensland University of Technology, for which I am much grateful for. Thank you to Leonie, Ernest, Matthew and Josef for giving me the opportunity to teach and also learn new subject areas along the way. My thanks to my co-authors for their efforts and valuable discussions: Ben, Felix, Nina, Douglas and Matthew.

Many thanks to my colleagues at QUT: Ben, Nick, Raphael, Thomas, Janaka, David, Jack, Qinyi, James, Chris Carr, Chris Djamaludin, Niluka, Mukhtar, Tarun, Shriparen, Iftekhar, Baskar, and Anisur. Thanks to my Sri Lankan friends in Brisbane. All of you made my PhD journey at QUT and life in Australia happy and memorable.

My sincerest thanks to my final seminar panel members, A/Professor Douglas Stebila, A/Professor Xavier Boyen, Dr. Matthew McKague, Dr. Ernest Foo, Professor Colin Fidge, Dr. Vicky Liu, and Dr. Qinyi Li, for their valuable comments and feedback on the thesis.

Finally, I could not have done this without the support and love of my family. Mum and Dad, thank you for the unconditional love and for always encouraging me to do my best. My sisters Vindyani and Amodyani, thanks for always being there for me. Lastly, my husband Chaturanga, whom I am blessed to have. Thank you for always being there and motivating me with love and kindness.

Chapter 1

Introduction

As the Internet develops, more and more individuals and organisations connect their electronic devices—ranging from desktop computers and servers to smart phones and TVs—to the Internet. Unfortunately, the security of the information exchanged between these devices is not guaranteed. It is an important task to ensure the security of this information. An essential step of this task is having digital identities for these devices that can be verified as legitimate. It is impossible to authenticate the remote party and establish a secure communication channel between the two parties without trusted digital identities. Cryptography aims to rectify this issue by enabling schemes that allow two or more parties to communicate over insecure channels. In particular, *asymmetric cryptography* is used in many security protocols. Asymmetric cryptography or public-key cryptography is an encryption scheme that uses two mathematically related, but not identical keys—a public key and a private key. A key pair of public key and private key is generated by the receiving entity, who is responsible for distributing the public key widely (used for encryption) and keeping the private key as a secret (used for decryption).

For two-way communication, the involved entities need to obtain an authentic copy of each other's public key, trust each other and believe that the public key is solely owned by the entity who they claim to be. At present these identities and their public keys are commonly managed by a digital document, called a *certificate*. A certificate contains an identity, a public key and a digital signature created by a trusted party called *Certificate Authority (CA)*. The identity usually is a host

name, an organisation or an individual which is identified by an IP-address or domain name resolved through DNS when communicated over the Internet. One example is email certificates, that can be used to digitally sign and encrypt email messages and attachments.

A certificate is a binding between the public key and a particular distinguished name of an identity. The signature of the CA along with their public key can be used to check the validity of information in the certificate. A public-key infrastructure plays an important role in secure communication as it provides the necessary services for managing digital certificates issued by various CAs. Generally the term "web-public-key infrastructure" is used to identify the set of systems, policies and procedures used to protect communication between web browsers and web content servers. The X.509 standard for public key infrastructure was first introduced in 1988. It defines the format of public key certificates [HPFS08]. Since then, X.509 PKI and digital certificates have become a critical part of security for enterprises, governments and consumers over the world. The initial X.509 system assumed a strict hierarchical arrangement of certificate authorities for issuing the public-key certificates, where trust flows from top to bottom. The Internet Engineering Task Force (IETF)'s public-key infrastructure has adopted the standard to the more flexible organisation of the Internet and includes the flexibility to support other topologies of PKI.

Before issuing any certificates, CAs confirm that the entity who requested the certificate is truthfully related to the specific identity. These certificates are then used in HyperText Transfer Protocol Secure (HTTPS), the secure browsing protocol for world wide web. In HTTPS, the communication protocol is encrypted using the Transport Layer Security (TLS) or formerly, its predecessor, Secure Sockets Layer (SSL) [DR08, DR06, Res00]. It distinguishes the secured traffic from insecure traffic by the use of a different server port. TLS uses X.509 certificates to authenticate the entities that are communicating and to negotiate a symmetric session key between them. When a web browser sets up a TLS connection with a web server, its X.509 certificate is checked. Web browsers decide which certificates to trust based on certificate authorities that come pre-installed in their software.

There are two categories of certificate authorities; root/intermediate CAs and untrusted CAs. An intermediate certificate is a subordinate certificate issued by the trusted root specifically to issue end-entity server certificates. The result is a

certificate chain that begins at the trusted root CA, through the intermediate and ending with the SSL certificate issued to the client. CAs at the top level of trust use the root certificate to add a digital signature to the intermediate certificate that are used at the next level of trust down. This means the web users can see who vouched for the company that vouched for web client's certificate. This procedure indicates how trust flows from top to bottom in the hierarchical model. In a web browser root and intermediate CAs are functionally similar and root CAs are trusted equally. Operating systems and web browsers typically contain a lists of trusted CAs and any certificate issued by these CAs are accepted as valid. IETF request for comment (RFC) document refers to root CAs as trust anchors. A trust anchor specifies the key stores that contain trusted root certificates.

1.1 Alternative concepts for PKI

X.509 is a centralised model for a PKI which mainly relies on CAs. Over the years there have been other proposed models to manage public-keys and certificates to provide secure communication.

1.1.1 Web of trust

The Web of Trust was first introduced by Phil Zimmerman [Zim95]. It is a decentralised system where anyone may sign and therefore affirm to the validity of others' certificates. There is no centralised organisation making decisions. Instead users themselves decide whom to trust based on their personal experiences and knowledge or on opinions of other parties they trust. Web of trust is used in PGP, GnuPG and other OpenPGP systems. Web of trust establishes the authenticity of the binding between a public key and its owner, in which every entity can act as a CA and vouch for anyone. The PKI trust model contrasts with web of trust where every entity can act as a CA and vouch for anyone. It is a network of people who have verified each other's certificates. The key point is you can trust an unknown certificate based on the number of people you trust who have created a relationship with the unknown entity.

1.1.2 Identity-based encryption (IBE)

Identity-based encryption was proposed by Shamir in 1984 [Sha85]. He gave the idea for identity-based signatures and email-address based PKI. The benefit of IBE when compared to PKI is that it enables any two parties to authenticate each other and communicate securely without prior distribution of cryptographic keys and most importantly without having to rely on a online trusted third party. There is a trusted third party, called the private key generator (PKG), which generates a master key pair and publishes the master public key. Any party can compute a public key corresponding to an identity ID by combining the master public key with the identity value to encrypt a message. Some unique information about the identity of the user is used as identity value to generate public key (e.g. a user's name, email address). That information should be readily available to the other party.

The corresponding secret-key is created by the key generation centre (KGC)/PKG which uses the master private-key upon request by the party authorised to use the identity ID. IBE allows parties to encrypt messages or verify signatures with no pre-distribution of keys except for the master public-key. The direct derivation of public-keys eliminates the need for end-entity certificates and some of the related problems.

Since the discovery from Shamir, IBE remained an open problem for many years until 2001 when Boneh et al. [BF01] proposed a pairing based Boneh-Franklin scheme. Moreover Cocks et al.[Coc01] proposed an IBE scheme based on quadratic residues.

The main drawback of IBE lies in the fact that key generation centre/private-key generator must be highly trusted as it is capable of generating any user's private key and hence is able to decrypt or sign any messages without authorisation. This system has inherent key escrow since any private key can be created using the PKG's secret key.

1.1.3 Certificate-less public-key cryptography

First proposed by Al-Riyami and Paterson [AP03], certificate-less public-key cryptography (CL-PKC) overcomes the inherent key escrow of identity-based encryption. There is a trusted CA or KGC to generate keys. The disadvantage with IBE is that in the event of a compromised KGC/PKG, the system will

collapse. In order to prevent that, CL-PKC proposes the idea of splitting the key generation process between the private-key generator and user.

For an entity R, the PKG supplies a partial private key which was generated using an identity value of R and a master key. The PKG must ensure that the partial private keys are delivered securely to the authenticated entity. The entity/receiver R uses some secret information and PKG's public parameters to generate a public/private key pair. The public-key is widely distributed as well as the identity value. To encrypt a message, sender S uses R's identity value and R's public key. The entity R uses the generated private key along with the partial private key supplied by the PKG to decrypt the ciphertext. R performs all cryptographic operations using the complete private key. This concept stands a chance against a compromised certificate authority or key generation center. CL-PKC does not need certificates which leads to benefits such as low storage and communication bandwidth. Moreover there is no need to verify certificates and certificate chains. This system is not identity-based since the public key can not be computed from the identifier alone.
Identity based encryption is useful in cases where pre-distribution of authenticated keys is inconvenient or infeasible. Furthermore there is no heavy dependence on a third party to manage digital certificates. However, compromise of the trusted third party in IBE has higher cost than compromise of a CA in PKI. If the master secret of key generation centre is compromised all the past encrypted messages are exposed and old signatures become worthless. Whereas in a case of compromise in PKI, CA can issue new certificates under a new signing key. Furthermore IBE is unable to provide non-repudiation. Certificate-less public-key cryptography overcomes some of the drawbacks that exist in identity based encryption. However, the drawback of CL-PKC is that it is not purely identity based. Also a secure channel is required to deliver partial private keys. Revocation is also a problem and it lacks the full security of a traditional PKI trust model. When considering these facts on alternative concepts for PKI, it is evident why PKI trust model is the most popular method for authenticating public key information and why it is the best option at present.

However, the PKI trust model is not completely secure. There are several security issues.

1.2 Security issues with PKI trust model

A certificate authority plays a major role in public key infrastructure by assisting with entity authentication needed for secure communication. Unfortunately web PKI relies on hundreds of CAs that are given a higher level of trust. Any trusted CA can issue an acceptable certificate to any domain. If one of these trusted CAs is compromised, attackers gain the ability to perform a man-in-the-middle (MITM) attack on a unsuspecting user, leading to some serious security issues. Since web PKI relies on a large number of CAs, it is possible for an attacker to obtain fraudulent certificates for any domain through a compromised CA. This fraudulent certificate can then be used to impersonate any website and escape detection. These types of attacks are known as *certificate substitution attacks*.

Such scenarios are found throughout the few decades that certificate authorities have been in use. There have been high-profile cases of misissued certificates being used to spoof legitimate websites. For example, in 2011 an intruder managed to issue themselves a valid certificate for the domain `google.com` and its subdomains from the prominent Dutch CA DigiNotar [Fox12]. This certificate was issued in July 2011 and may have been used maliciously for weeks, on large scale man-in-the-middle (MITM) attacks on multiple users in Iran before the detection on August 28, 2011. In another instance, the Comodo group suffered from an attack which resulted in the issuance of nine fraudulent certificates for domains owned by Google, Yahoo!, Skype and others [Com11]. Over the past few years there have been several other incidents involving compromised CA organisations including, GlobalSign and Digicert Malaysia [GSE11, Man12].

A competent PKI depends on CAs being truthful, honest and legitimate. However, there might be situations where CAs fail to retain those properties. A CA might issue the wrong type of certificate by mistake and this could have undesirable consequences. For example, in 2011, a popular Turkish CA TürkTrust accidentally issued two intermediary certificates instead of regular certificates. Intermediate certificates have the authority to issue certificates themselves on behalf of its root CA. One of these certificates was revoked but unfortunately the other was used to issue more fraudulent certificates until it was detected and revoked in late 2012. Though CAs could be blamed of poor security for these breaches, it is evident that state of the art in software engineering and system design are unable to provide complete security against these attacks. These vulnerabilities affect the PKI trust model in turn weakening the reliability and

effectiveness of encrypted communication over the Internet. Moreover, apart from man-in-the-middle attacks, these flaws can enable a wide range of other security attacks, such as website spoofing, and server impersonation.

A web PKI facilitates authenticated communication channels between clients and servers. Apart from entity authentication, integrity and confidentiality of sensitive data exchanged over the Internet are essential properties. Without web PKI, sensitive information can still be encrypted ensuring confidentiality of data and exchanged over the Internet, but entity authentication cannot be achieved. Thus any form of sensitive data exchanged over the Internet is reliant on PKI for security. Although web PKI provides a more secure environment, it is not without drawbacks. The recent incidents discussed above indicate that we cannot rely heavily on certificate authorities. The problem is, these vulnerabilities affect the PKI trust model of secure communication over the Internet. We believe that the problem of fraudulent certificates stems from the inherent centralisation of the current system based on certificate authorities. Current web-public-key infrastructure lacks transparency when it comes to certificate management, making it difficult to detect when fraudulent certificates are issued. Under these circumstances how can we ensure secure communication for web users? How can we achieve it without completely depending on a third party to vouch for the binding between an identity and a public key?

Several alternatives, which we call PKI technologies, have surfaced over the years in an attempt to overcome these issues. These include Sovereign keys [Eck11], Trust Assertions for Certificate Keys (TACK) [Mar13], Attack Resilient Public-Key Infrastructure [BCK+14a], DANE [HS12], and Certificate Transparency (CT) [LKL13, Lau14, Rea13] among others. However, for most of these technologies, a proper security analysis has not yet been performed. Furthermore, not all of these technologies achieve all security properties.

Certificate Transparency aims to detect fraudulently issued certificates by requiring CAs to append certificates being issued to a public log. Since deployed, CT has demonstrated its effectiveness in detecting malicious certificates. Google detected unrequested certificates for two of their sub-domains issued by a Symantec sub-CA Thawte [SE15]. The certificates were issued on September 14, 2015 and detected by September 17, 2015; the certificates were revoked immediately, limiting the exposure of the certificates to just three days. In another case, the Facebook security team discovered an issuance of two certificates on multiple

sub-domains violating Facebook's internal security policies [Hua16]. The incident was investigated and both certificates revoked within hours, even before they were deployed to production systems.

Apart from Certificate Transparency, there have been other log-based schemes proposed to enhance security of PKI trust model. One such proposal is Accountable Key Infrastructure (AKI) [KHP+13], which aims to prevent compromised CAs from impersonating domains. Attack Resilient Public-Key Infrastructure (ARPKI) proposed by Basin et al. [BCK+14a] is a new design of PKI based on AKI's structure that focuses on transparency and accountability of certificate-related operations such as certificate issuance, validation, revocation and update. Although these log-based schemes provide well designed approaches towards better certificate management, their centralised properties could prove undesirable in case of misbehaviour from involved entities. Recently there is a shift towards research on applying decentralisation concepts to current public-key infrastructure model.

After the invention of *blockchain*, an append-only decentralised public ledger, in 2008 by Satoshi Nakamoto [Nak08], a flourish in the use of public ledgers to implement a wide variety of decentralised applications is evident. Applications of blockchain vary mainly from cryptocurrencies like Bitcoin to smart contracts [Woo14], cloud storage, asset tracking and many more. There are some desirable properties of public ledgers that make it a suitable building block for distributed PKIs; tamper-proof, transparent and incentive based mechanism. Blockchain is built over a peer-to-peer network in which participants maintain their own copies of the ledger and can verify the transactions. Although the ledger itself is not distributed (it is replicated), its affirmation mechanism is based on a heavily decentralised consensus mechanism, making it infeasible to alter or delete previously time-stamped records. Over the years many related works have been published, proposing new decentralised schemes for public key infrastructure. *Certcoin* [FVY14], provides a mechanism for operations such as key registration, update, revocation and recovery while focusing on retaining identity retention. In another attempt, *Authcoin* [LCMR16] focuses on validation and authentication of public keys rather than identity retention. Taking a different approach, *Instant Karma PKI* [MR16], seeks to improve log-based PKI by incentivising entities to look for fraudulent certificates and report.

A PKI trust model augmented with features of distributed schemes is a

promising solution for the prevailing issues. So far these issues have evolved around scenarios such as lack of transparency on certificate management, CAs issuing fraudulent certificates for domain either mistakenly or when compromised by some entity with malicious intentions. Issues have not been about some adversary breaking cryptographic algorithms in security protocols used for secure communication over insecure networks like Internet. As of 2018, cryptographic algorithms used in public key infrastructure are considered to be secure under classical computations. Most of these algorithms are public-key algorithms such as RSA digital signature/public-key encryption algorithm, Diffie-Hellman key-exchange algorithm. Their security relies on classic computational hardness of integer factoring problem and discrete logarithm problem. However, with the invention of well known quantum algorithms such as Shor's algorithm [Sho99] and rapid development of technology, the creation of a quantum computer could be not very far ahead in future. The issue is that advances in quantum computing present a grave challenge to widely used cryptographic techniques. Any security protocols that derive security from the above public-key ciphers would be broken as quantum computers, employing quantum algorithms have the ability to quickly factor large composite integers, find discrete logarithms over groups and speed up searches.

1.3 Methodology

Formalisms of provable security are followed throughout our work. Provable security mechanisms are applied to show that a particular model can be mathematically proven to be secure under some generally accepted assumptions, typically a mathematical problem. The Security requirements of the system are clearly defined and state them formally in an adversarial model. Further, clear assumptions are made on adversary's access to the system as well as the computational resources. Security of the system is outlined as a game played between the adversary and a hypothetical challenger, where the security model is defined by responses of the challenger. Security of the system is proved by reducing its security to security of the underlying hard problem.

1.4 Research Objectives and Contributions

The demand for public key infrastructure and digital certificates is getting higher. More and more organisations are looking to install PKI to be more secure and resilient. This high demand makes CAs a more likely target for sophisticated cyber attacks, potentially compromising their customer networks. When comparing with alternatives for PKI (WoT, IBE, CL-PKC), using PKI trust model is the best option since, it provides integrity, authentication and revocation. Further, it is Internet scalable and cost effective. Therefore it is imperative to look for methods to enhance current PKI model for present and future developments than looking to replace it. To achieve this, our overall objective of this research is to model an existing and currently deployed technology, providing better insight, design a new technology, overcoming still existing issues and progress towards a quantum-safe web PKI, for a better and secure future. We contribute to the pool of knowledge on PKI trust models and technologies to enhance security and productivity at the present and for future developments.

We will now summarise the objectives and respective contributions of this thesis.

Analyse Certificate Transparency on its effectiveness in eliminating the flaws in PKI trust model and identify remaining limitations. There have been a few PKI technologies proposed over the years aiming to improve the PKI trust model by either detecting fraudulent certificates, reducing the impact of compromises or improving checks on revoked credentials. But most of the technologies lack a proper security analysis. Furthermore, it is worth questioning whether they achieve the desirable security properties. Thus one of our objectives in this research is to study and analyse a chosen technology on its effectiveness in preventing prevailing issues. We aim to provide more assurance in utilising this technology by providing a security framework and showing that it achieves targeted security properties by following a provable security game-based approach.

To achieve this objective, we study and properly analyse one of the successfully deployed PKI technologies: Certificate Transparency. We present a model for logging schemes, which takes Certificate Transparency in to consideration. We define four security properties of CT using the formalism of provable security. These properties are of two types; security notions which concern a malicious

logger attempting to present different views of the log to different parties or at different points in time, and security notions concerning a malicious monitor who attempts to frame an honest log for failing to include a certificate in the log. We prove that Certificate Transparency satisfies these security properties under various assumptions on Merkle hash trees, all of which reduce to collision resistance of the underlying hash function (and in one case with the additional assumption of unforgeable signatures). We focus on two particular threats in the Certificate Transparency threat model.

We analyse the security of Certificate Transparency and show that it prevents these misbehaving acts by its entities. The usefulness of this chapter's contribution is that the definition of a logging scheme and its security properties contributed has the potential to be applied to any other similar constructions and is not specific to Certificate Transparency.

This work is based on joint work with Benjamin Dowling, Felix Günther, and Douglas Stebila. Part of this work has appeared in the following publication.

- Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure Logging schemes and Certificates Transparency. In *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS 2016), LNCS, vol. 9879. Springer, 2016, pages 140-158.*

Extend, design and construct a new PKI technology to address the limitations identified in Certificate Transparency. Identifying and working on a better design of a technology to manage certificates is our next objective as it is apparent that current PKI scheme has some predominant issues; for example its centralised nature and the lack of transparency in certificate management. Moreover, the limitations of Certificate Transparency, for example heavy reliance of third parties that monitor logs, lack of revocation mechanism and decentralised nature of the protocol itself, motivates a better technology. Thus in this chapter we address our purpose of designing a secure decentralised system to provide stronger transparency to an existing public key infrastructure. A few new approaches have been proposed over the recent years; some log-based and some blockchain-based schemes. But we noticed that these advances have problems, for instance the lack of an external validation mechanism or issues with practical adoption. We propose a decentralised client-based system to manage certificates enforcing full transparency which eliminates single points of failure. Apart from

those design goals, other attributes of the scheme are facilitating multiple certificates per domain, handling multiple domains on a single certificate, being Internet scalable, providing client privacy and a better revocation mechanism. Moreover, the scheme should be backwards-compatible with web PKI and should allow external validation of entities.

We achieve this objective by contributing a new design for auditing and managing issuance and revocation of certificates. We introduce Decentralised PKI Transparency (DPKIT). DPKIT is a decentralised client-based approach to Certificate Transparency based on generic blockchains augmented with efficient data structures. It efficiently leverages an existing blockchain to realise an append-only, distributed associative array, which allows anyone to audit and update the history of all publicly issued certificates and revocations for any domain.

We formally define the notion of distributed append-only ledger as an ideal functionality with which entities can interact honestly. Furthermore, we define three security properties of DPKIT against a malicious entity using the formalism of provable security. Then we prove the security results on DPKIT and show that its instantiation guarantees these three security properties. We show that DPKIT achieves its design goals. This contribution is useful as it provides a strongly immutable record of all issued and revoked certificates as seen by the worldwide community of users, that efficiently supports all modern use cases of certificates in actual web protocols.

This work is based on joint work with Xavier Boyen, Matthew McKague and Douglas Stebila, and has been documented in a technical paper.

Investigate, analyse and plan the transition of PKI to post-quantum cryptography. First two objectives in this research focus on evolution of PKI to a potentially new state in current world. In this section we take a step further to address possibilities of PKI thriving in an era with quantum facilities. Evidently, due to recent advances in the field of quantum computing, threats to the current state-of-art public-key cryptographic systems have increased. As contemporary mechanisms for data protection use cryptographic systems that rely on computational hardness of factoring and discrete-log problems, quantum computers present a serious challenge. These mechanisms include public-key cryptographic algorithms such as RSA signature/encryption, Diffie-Hellman key-exchange and ECDSA (Elliptic Curve Digital Signature Algorithm) signatures. Factoring and discrete-log problems are considered to be intractable to

conventional computers but can be solved by quantum computers in polynomial time. Therefore eventually RSA and cryptography based on discrete-log problems are vulnerable to quantum attacks. In the realm of public key infrastructure there are security concerns as digital certificates are cryptographically signed documents. Unfortunately, the majority of certificates issued contain RSA public keys and are signed with RSA keys [Cam15]. This means when a quantum computer becomes a reality, all the information protected by enabling a PKI will no longer be relevant. According to the European Union's Quantum Manifesto [DTMH+16], the world will see a fully functioning quantum computer by 2035. Looking at how cryptographic techniques have evolved in the past few decades and how slow the world is to transition in to new techniques, we believe the time to start the transition is now. Therefore, our third objective is to analyse the transition of public key infrastructure to post-quantum cryptography.

We achieve our third objective by contributing to the investigation of the use of a composite (hybrid) digital signature scheme in which both a traditional algorithm and one or more post-quantum algorithms are employed. Further, we examine the approaches of supporting hybrid signatures in three real-world standards that involve digital signatures and public-key infrastructure: certificates (X.509), secure channels (TLS), and email (S/MIME). Further, we report on the backward compatibility of these standards when using hybrid signatures. We test this property in popular cryptographic libraries and implementations and propose several approaches to preserve backwards-compatibility.

Finally, we create digital certificates with estimated sizes containing post-quantum signature schemes and include them into X.509 certificates as extensions. Then we test these new certificates on several applications that support TLS connections. It is evident that the software we tested had no problems with certificates or extensions up to certain sizes some software accommodate ideal-lattice schemes and the others accommodate hash-based schemes. Thus, contributions in this chapter are useful for enhancing current PKI model as well as for future developments as it provide practical solutions for transitioning existing quantum vulnerable PKI systems to quantum safe PKI systems. Additionally, this work also opens up new research directions in the context of quantum-safe public key infrastructure.

This work on hybrid signatures and transitioning PKI to a quantum-safe state is a joint work with Nina Bindel, Matthew McKague and Douglas Stebila, and

has appeared as the following publication.

- Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a quantum-resistant public key infrastructure. In *Proceedings of International Workshop on Post-Quantum Cryptography (PQCrypto 2017), LNCS, vol. 10346. Springer, 2017, pages 384-405.*

1.5 Outline

The organisation of this thesis is as follows.

Chapter 2: Background This chapter contains the theoretical and technical background that will be useful as a foundation for the rest of the thesis. First we focus on cryptographic background, assumptions and cryptographic tools which will be used in this thesis. Then we proceed to discuss about public key infrastructure.

Chapter 3: Secure Logging schemes and Certificates Transparency This chapter contains our work on Certificate Transparency and logging schemes. First we introduce Certificate Transparency and its gossiping protocol and threat model. Then we discuss about the logging schemes and its instantiation using Certificate Transparency. Finally we define the security goals and proceed to prove them using formalism of provable security.

Chapter 4: Associative Blockchain for Decentralised PKI Transparency This chapter comprises our work on the new design of PKI technology: Decentralised PKI Transparency. First we discuss the motivation and background behind DPKIT. Thereafter we talk about the desired properties and design goals of DPKIT. Then we continue on to its functionalities and finally to security goals. We prove the security of DPKIT using security reductions and show that we achieve our design and security goals.

Chapter 5: Transitioning to a Quantum-Resistant PKI This chapter carry our work on the use of hybrid digital signature schemes and their backwards compatibility with three popular real-world standards involving digital signatures. First we discuss about security notions for hybrid digital signatures and their

separability. Thereafter we mention several methods for combining signature schemes. Finally we discuss our experiments and results in hybrid signatures in three standards: certificates (X.509), secure channels (TLS), and email (S/MIME).

Chapter 6: Conclusion This chapter summaries the results obtained in this thesis and discusses future research directions.

Chapter 2

Background

In this chapter we describe several background concepts that establish the theoretical foundation for the rest of the thesis. We start by reviewing the cryptographic background; modern and post-quantum and cryptographic assumptions which lay out the basis of our work in this thesis.

Next, we address public key infrastructure, which is the main area of study in our work. We cover the basics of X.509 certificates, the role of certificate authorities in PKI as well as the transparency and security aspects of current PKI. Additionally, we discuss Merkle trees and distributed ledgers, namely blockchains. The Merkle tree is studied thoroughly in subsection 2.4.1, as it is one of the main data structures used in some of our contributions. One of our contributions also employs blockchain as a data structure for a public ledger and is addressed in subsection 2.4.2. Finally, we discuss our approach on methodology.

2.1 Cryptographic Background

Cryptography plays a major role in securing communication: it is concerned with designing and using schemes that enable two or more parties to communicate secretly and authentically in the presence of an eavesdropper who can monitor all communication between them. Generally cryptography is about achieving various aspects of information security such as confidentiality, integrity, and authentication. Confidentiality, which is the prevention of unauthorised disclosure of information, is achieved by *encryption*. Integrity, which is prevention of

unauthorised modification or destruction of information, and authentication which is the verification of a claimed identity or source of information, are achieved by using digital signatures or message authentication codes. Additionally, digital signature schemes also provide the important property of non-repudiation, which refers to a state where the sender being unable to deny having sent a message.

2.1.1 Confidentiality: Symmetric and Asymmetric Encryption

Security of most encryption schemes relies on a secret value known as the *secret key*. In *symmetric* encryption, communicating parties share the same key in advance, unknown to the eavesdropper, and use that key to encrypt and decrypt messages. Confidentiality is achieved as long as the secret-key is only available to the intended parties.

In 1976, Diffie and Hellman [DH76] introduced a new method of distributing cryptographic keys as a solution to the problems of sharing a secret-key in advance. Known as *Diffie-Hellman* key exchange, it allows the two parties to establish a shared secret-key over an insecure channel. This concept gave the initiation for a new class of encryption algorithms: *asymmetric-key algorithms*. In contrast to symmetric-key encryption, in asymmetric or *public-key* encryption, the encryption and decryption algorithms use different keys. In this setting the receiving party generates a pair of keys called *public key* and the *private/secret key*. The public key is widely distributed and is used by the sender to encrypt the message. The private key is used by the receiver to decrypt the resulting ciphertext. The keys are related mathematically, but the secret-key should not be able to be easily derived from the public-key. This idea was implemented by Ron Rivest, Adi Shamir and Leonard Adleman [RSA78] in what is now known as the RSA algorithm.

The Diffie-Hellman and RSA algorithms are the most widely used public-key algorithms. Some others include ElGamal encryption [ElG84], the Cramer–Shoup cryptosystem [CS03], and various elliptic curve techniques.

A public-key encryption scheme is defined by specifying a *message space* $\mathcal{M}$ along with three algorithms $(\text{KeyGen}, \text{Enc}, \text{Dec})$ where:

1. KeyGen: A probabilistic *key-generation algorithm* that outputs $(pk, sk, \mathcal{M}_k)$, pk is the public-key, sk is the secret key and $\mathcal{M}_k$ is the message space asso-

ciated with the pk/sk-pair. Here k is an integer usually called the *security parameter*, which determines the security level.

2. Enc: A probabilistic *encryption algorithm* that takes as input $m \in \mathcal{M}_k$ and a public-key pk, and outputs a ciphertext $c \in \mathcal{C}$.

3. Dec: A deterministic *decryption algorithm* that takes as input a ciphertext $c \in \mathcal{C}$ and a secret key sk, and outputs either a message $m \in \mathcal{M}$ or the error symbol $\perp$.

A public-key encryption scheme must satisfy the correctness property: for all valid key pairs (pk, sk),

$$\forall m \in \mathcal{M}_k, \mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(m)) = m. \tag{2.1}$$

We now review two security notions for public-key encryption schemes: *indistinguishability against chosen ciphertext attacks (IND-CCA)* and *indistinguishability against chosen plaintext attacks (IND-CPA)*, referring to Katz and Lindell's text [KL14].

Definition 1 (Indistinguishability against Chosen-Ciphertext Attacks (ind-cca)[KL14])**.** *Let $\mathcal{A}$ be any PPT adversary in the security parameter k, against a public-key encryption scheme $\Pi = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$. The* ind-cca *security experiment for the public-key encryption scheme Π, $\mathrm{Exp}^{\mathsf{ind\text{-}cca}}_{\Pi,\mathcal{A})}(1^k)$, is defined as follows:*

1. $\mathrm{KeyGen}(1^k)$ *is run to obtain* (pk, sk)

2. *Adversary $\mathcal{A}$ is given pk, and access to a decryption oracle $\mathrm{Dec}_{sk}(\cdot)$. It outputs a pair of equal length messages m_0, m_1 in the message space associated with pk.*

3. *A uniform bit $b \in \{0, 1\}$ is chosen and then a ciphertext $c \leftarrow \mathrm{Enc}(pk, m_b)$ is computed and given to $\mathcal{A}$.*

4. *$\mathcal{A}$ continues to interact with the decryption oracle, but may not request a decryption of c itself.*

5. *$\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if $b' = b$, and 0 otherwise*

6. *$\mathcal{A}$ wins if $b' = b$*

The public-key encryption scheme Π *is* ind-cca*-secure, if for every PPT adversary* $\mathcal{A}$ *the advantage of winning the security experiment* $\mathrm{Exp}^{\mathsf{ind\text{-}cca}}_{\Pi}(\mathcal{A})$*:* $\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\Pi}(\mathcal{A})$*, is negligible in the security parameter* k*.*

Definition 2 (Indistinguishability against Chosen-Plaintext Attacks (ind-cpa)[KL14])**.** *Let* $\mathcal{A}$ *be any PPT adversary in the security parameter* k*, against a public-key encryption scheme* $\Pi = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$*. The* ind-cpa *security experiment for the public-key encryption scheme* Π*,* $\mathrm{Exp}^{\mathsf{ind\text{-}cca}}_{\Pi,\mathcal{A})}(1^k)$*, is defined as follows:*

1. $\mathrm{KeyGen}(1^k)$ *is run to obtain* (pk, sk)
2. *Adversary* $\mathcal{A}$ *is given* pk*, and outputs a pair of equal length messages* m_0, m_1 *in the message space (*$|m_0| = |m_1|$*).*
3. *A uniform bit* $b \in \{0, 1\}$ *is chosen and then a ciphertext* $c \leftarrow \mathrm{Enc}(pk, m_b)$ *is computed and given to* $\mathcal{A}$*.* c *is the challenge ciphertext.*
4. $\mathcal{A}$ *outputs a bit* b'*. The output of the experiment is* 1 *if* $b' = b$*, and* 0 *otherwise.*
5. $\mathcal{A}$ *wins if* $b' = b$

The public-key encryption scheme Π *is* ind-cpa*-secure, if for every PPT adversary* $\mathcal{A}$ *the advantage of winning the security experiment* $\mathrm{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\Pi}(1^k)$*:* $\mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{A})$*, is negligible in the security parameter* k*.*

2.1.2 Integrity/Authentication: Digital Signature Schemes

A digital signature is a mathematical technique used to validate the authenticity of a document, digital message or a software. When using asymmetric cryptography, each entity has a unique key pair: the private key which is known only to the owner and the public key. This private key can be used by the owner to form a digital signature for a particular file or message. The corresponding public key can be used by others to verify the authenticity of the digital signature on the message, thus digital signatures are *publicly verifiable*.

One of the main properties provided by digital signatures is authentication of the sender for a particular message. Since only the *signer* knows the private key of an associated public key of a signature, only the signer could have created the signature. It should be computationally infeasible to generate a valid signature

$\underline{\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{A})}$:
1: $(pk, sk) \xleftarrow{\$} \mathrm{KeyGen}()$
2: $M \leftarrow \{\}$
3: $(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathsf{OSign}}(pk)$
4: **return** $\mathrm{Vfy}_{pk}(m, \sigma) \wedge (m \notin M)$

$\underline{\mathsf{OSign}(m)}$:
1: $\sigma \leftarrow \mathrm{Sign}_{sk}(m)$
2: $M \leftarrow M \cup \{m\}$
3: **return** (m, σ)

Figure 2.1: Security experiment for existential unforgeability under chosen message attack of a signature scheme $\mathtt{SIG} = (\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Vfy})$.

for an entity without knowing that entity's private key. For efficiency, signature schemes typically use a hash function to reduce the amount of material processed using asymmetric cryptography. The use of hash functions provide assurance of message integrity since any change in message will result in a different hash value. Another important property provided is non-repudiation which makes digital signatures particularly useful for e-commerce applications. Collision-resistant hash functions along with digital signature schemes are used to achieve non-repudiation.

The most widely used digital signature schemes are RSA, DSA (Digital Signature Algorithm) and ECDSA (Elliptic Curve DSA).

A digital signature scheme $\mathtt{SIG} = (\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Vfy})$ typically consists of 3 algorithms;

1. KeyGen: A probabilistic key generation algorithm that takes as input the security parameter 1^k and outputs a public key pk and secret key sk. This also specifies the message space $\mathcal{M}$ and the signature space $\mathcal{S}$.

2. Sign: A (potentially) probabilistic signing algorithm that takes as input a secret key sk a message $m \in \mathcal{M}$ and outputs a signature $\sigma \in \mathcal{S}$.

3. Vfy: A deterministic signature verification algorithm that takes as input a public key pk, message m, and signature σ, and outputs a boolean value: *true* if σ is a valid signature under pk or *false* otherwise.

It is required that except with negligible probability over (pk, sk) output by $\mathrm{KeyGen}(1^k)$, it holds that $\mathrm{Vfy}(m, \mathrm{Sign}_{sk}(m)) = 1$ for every (legal) message m.

We now describe the security notion called *existential unforgeability under chosen message attacks* (EUFCMA).

Definition 3 (Signature scheme unforgeability)**.** *Let $\mathcal{M}$ be a set. A digital signature scheme is defined as a tuple of algorithms* $\mathtt{SIG} = (\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Vfy})$. $\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{A})$*, is negligible in the security parameter k.*

- $(pk, sk) \xleftarrow{\$} \text{KeyGen}(1^k)$
- $\sigma \xleftarrow{\$} \text{Sign}_{sk}(m)$
- $\{0,1\} \leftarrow \text{Vfy}_{pk}(m, \sigma)$

The security experiment $\text{Exp}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}$ *for* existential unforgeability under chosen-message attacks *is given in Figure 2.1. If* $\mathcal{A}$ *is an algorithm, we define* $\text{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{A}) = \Pr\left[\text{Exp}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{A}) = 1\right]$.

2.1.3 Hash Functions

A hash function is a deterministic algorithm that maps data of arbitary length to a bit string (digest) of fixed length. There are four basic properties of hash functions: fixed length output, one-wayness, collision resistance, and a small change in the message produces a major change in the resulting digest.

Hash functions are utilised as building blocks for other cryptographic primitives, mainly for integrity checks for sent data and to transform the message into a value that can operated on by some cryptographic schemes (e.g. digital signatures). Depending on their context of use, hash functions require different cryptographic properties.

- Collision resistance: Given a hash function, $\text{H} : \{0,1\}^* \to \{0,1\}^\lambda$ where λ is the fixed output length of H, it should be difficult to find two distinct input values m, m' such that $\text{H}(m) = \text{H}(m')$ but $m \neq m'$.
- One-wayness: Given a hash function $\text{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a hash output h generated by sampling an m at random, it should be difficult to find a value m' such that $\text{H}(m') = h$.
- Second pre-image resistance: Given a hash function $\text{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a value m, it should be infeasible to find $m' \neq m$ such that $\text{H}(m') = \text{H}(m)$.

It is evident that the difficulty of these problems are related. Any adversary capable of breaking the second pre-image resistance property of a hash function H can also be used to break the collision resistance property of H. We can declare that the advantage of any adversary $\mathcal{A}$ against the second pre-image resistance of H is bounded by the advantage of $\mathcal{A}$ against the collision resistance of H.

Definition 4 (Hash collision finding)**.** *Let* $\mathcal{M}$ *be a set, let* $\mathrm{H} : \mathcal{M} \to \{0,1\}^{\lambda}$ *be an unkeyed hash function, and let* $\mathcal{A}$ *be an algorithm. We say that* $\mathcal{A}$ finds a collision in H *if* $\mathcal{A}$ *outputs a pair* (m, m') *such that* $m \neq m'$ *and* $\mathrm{H}(m) = \mathrm{H}(m')$.

2.1.4 Post-Quantum Cryptography

Post-quantum cryptography refers to cryptographic algorithms that are thought to be secure against an attack by a quantum computer. However, with advancing research, it is found out that most deployed public-key algorithms have the potential to be efficiently broken by a large scale quantum computer. Public-key cryptography is an essential part for secure communication protocols. If the public-key algorithms fail, all the confidential messages that are sent over public channels become legible for any passive observer, that is secure communication becomes hard to achieve. The security of current popular algorithms are mainly based on two computational hard problems: the integer factorisation problem and the discrete logarithm problem. These problems are considered difficult to be broken by conventional computers. The issue is that all these problems can be solved in polynomial time by a powerful quantum computer which runs the Shor's algorithm [Sho99].

Most symmetric cryptographic algorithms like AES and hash functions are considered to be relatively secure against attacks by quantum computers for appropriate key sizes [Cam15]. However, public-key algorithms like RSA and ECC face a major threat with development of quantum computers. Public-key algorithms are soundly studied and well accepted by the security community and are used worldwide due to their functionalities in security protocols and applications. Thus it is imperative that we prepare for a time when quantum computing becomes a threat. In Chapter 5 we further discuss how these changes can be conducted in some of the popular security protocols and applications, focusing mainly on public key infrastructure.

2.1.5 Digital Certificates

The majority of organisations and business entities highly prefer public-key encryption for secure communication at present since it provides important properties such as authentication and non-repudiations along with confidentiality and integrity. When two parties communicate, they need to obtain an authentic

copy of each other's public key and trust each other and believe that the private key is solely owned by the sender. This trust on the public key can be formed using public-key certificates which are also known as digital certificates. The integrity of a digital certificate is provided using digital signature schemes. Anyone who knows the signer's public key can verify that the message has been originated from the signer and has not been modified in transit. The Signer or the entity that signs a digital certificate vouches for the accuracy of contents in the certificate. This entity is called a Certificate Authority.

Digital certificates bind a public key to an identity. The ownership can be determined by the information included in the certificate such as facts about the key, identity of the owner and the digital signature of an entity. A person can communicate with the certificate's owner if the signature in the certificate is valid and if he trusts the signer. Public key cryptography along with digital certificates provide the organisations confidentiality, access control, integrity, authentication, and non-repudiation they need. The most common format for public key certificates is defined by X.509 standard, specifically as a certain use case such as Public Key Infrastructure defined in RFC 5280 [CSF+08].

2.2 Public-Key Infrastructure

In this section, we discuss the basics of public-key infrastructure. In the above sections we gave an introduction to basic cryptography. Now we focus on an aspect of public-key cryptography, namely managing key pairs. Public-key cryptography uses an infrastructure; which provides essential services for managing digital certificates and encryption keys for people, programs and systems. For these reasons public key infrastructure is established. PKI involves the hardware, software, policies and standards that are required to manage digital certificates. Generally user-names and passwords are used to authenticate people. But by using a PKI, it enables a different authentication than the mere passwords and user-names. PKI supports the distribution and identification of public keys, allowing users to verify each other's identity and then securely exchange data over insecure networks. It enables users of a basically insecure public network to securely and privately exchange data and financial information [LYH03]. This is done by the use of a public and private cryptographic key pair that is obtained and shared through a trusted authority. A PKI consists of:

- A certificate authority; who issues and verifies digital certificates
- A registration authority; who verifies identity of certificate requesters
- A central directory; which stores the certificates and their public keys
- A certificate management system
- A certificate policy

To achieve these functionalities a trust model is used. A trust model is a collection of rules that informs applications on how to decide the legitimacy of a digital certificate. There are two popular trust models; *hierarchical* and *web of trust*. Web of trust uses self-signed certificates and third party attestations of those certificates. In contrast with web of trust, the hierarchical trust model has trusted third parties known as certificate authorities which are at the top most level and trust flows from top to bottom. It is the most popularly used trust model in current public key infrastructure.

In general PKIs are expected to provide security, availability and efficiency when public keys of entities are authenticated by clients. The subject of the certificate is the entity its public key is associated with (i.e. the "owner" of the certificate). Generally the term web public key infrastructure is used to identify the set of systems, policies and procedures used to protect communication between web browsers and web content servers. A legitimate CA should issue a certificate for a given domain name only to the 'owner' of that domain; the subject. The main security property that any web PKI must satisfy is preventing impersonation attacks on domains. That is if a client visits a website and creates the connection with that domain, the entity with which clients communicate must be the legitimate owner of that domain. The other security properties a web PKI should provide can be identified as follows:

- Legitimate initial certificate registration: A domain's certificate should be registered by the infrastructure only if it satisfies the requirements specified by the infrastructure's policy.
- Legitimate certificate updates: A domain's certificate should be updated by the infrastructure only if the new certificate satisfies the requirements specified in the formally registered certificate.

- Visibility of attacks: If an adversary attacks the infrastructure by compromising the entities such as certificate authorities and domains successfully, it should become visible for detection [BCK+14a].

2.2.1 Certificate Authorities and Certificate Issuance

A Certificate Authority is an entity responsible for issuing these certificates. A CA acts as a trusted third party—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. Before issuing any certificates, CAs are expected to confirm the entity who requested the certificate is truthfully related to the specific identity. Moreover, CAs are also expected to confirm that the requesting entity holds the ownership of private key. Then CAs sign and issue the certificate. The CA/Browser forum [For07] defines few classifications for certificates; *"DV" (Domain Validated)*,*"OV" (Organisation Validated)*, *"IV" (Individual Validated)*, and *"EV" (Extended Validation)*. Originally the certificates were of OV type and update to EV state is a recent change. Instructions on how to distinguish between types of certificates are also provided in their specification. All HTTPS server certificates are required to have a domain name which is controlled by the certificate holder. When an entity requests a certificate for a domain name, it is required that the entity proves its control over the specific domain to CA which implicitly assumes that domain names are mapped to the correct web server (IP address). The mapping is realised through DNS thus called domain validated certificates [For07, BvOP+09, CvO13, SHF02, Vac04].

Over time CAs have begun to use a completely automated validation process, leading to a decrease of the quality of validation. Furthermore domain validation suffers from some structural limitations. As a solution the CA/Browser forum [For07] has introduced a type of certificate that has more validation than DV certificates called as extended validation certificates. EV certificates involve a more controlled issuance process than DV certificates. The main goals of EV certificates are to provide users with much confidence concerning the identity of the organisations that control the web sites they visit and to facilitate the exchange of encryption keys between the web site and the user's web browser. Web browsers show the verified legal identity prominently in their user interface, either before, or instead of, the domain name. Yielding extra information of the owning entity is a requirement for obtaining a EV certificates, thus information such as entity/organisation name, domain name, jurisdiction of incorporation, registration

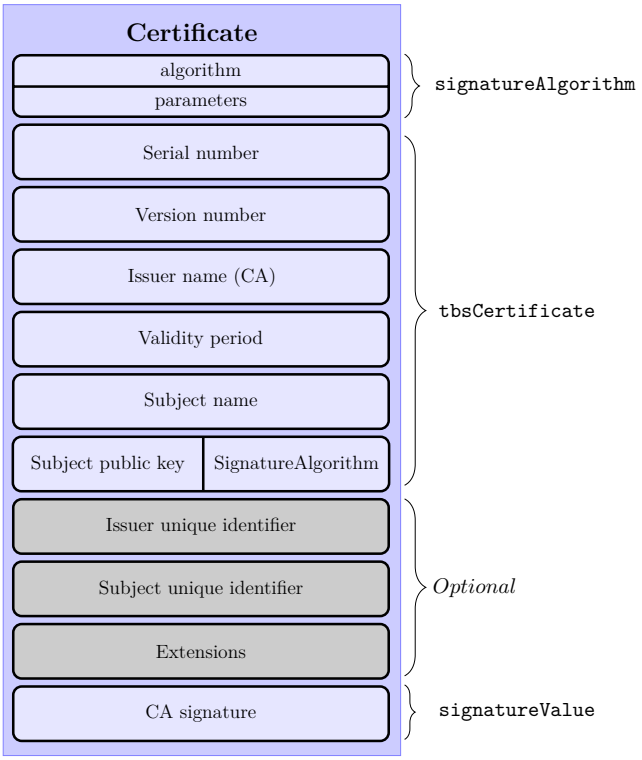


Figure 2.2: Version 3 of the X.509 certificate format, with additional extension fields.

number and address of place of business etc. can be found in the certificate. This extra information is not available with a domain-validated certificate. The EV SSL issuance process is designed to make sure only parties like government entities, private organisations, or business entities having a physical location in the real world can acquire it. EV certificates use the same encryption as organisation-validated certificates and domain-validated certificates: the increase in security is due to the identity validation process, which is indicated within the certificate by the policy identifier.

2.2.2 X.509 Certificates

X.509 is defined by the International Telecommunications Union's Standardisation sector (ITU-T), and is a standard for PKI that specifies a suite of data formats and algorithms which assume a strict hierarchical arrangement of trusted third parties or certificate authorities for issuing public-key certificates. It defines the most common format for digital certificates, which are called *X.509 certificates*

[HPFS08]. The format of the X.509 version 3 certificate is shown in Figure 2.2. Version 3 allows for extensions in the certificate. These extensions can include data such as additional subject identification information, key attribute information, policy information, and certification path constraints. The certificate is a sequence of three required fields; `tbsCertificate`, `signatureAlgorithm` and `signatureValue`.

The `tbsCertificate` field contains the names of the subject and issuer, a public key associated with the subject, a validity period, extensions and other associated information. The `signatureAlgorithm` field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate and `signatureValue` field contains a digital signature computed using the CA's private key upon the ASN.1 DER encoded `tbsCertificate`.

Additionally, the certificate body also allows for optional extensions. Extension fields in X.509 certificates allow additional attributes with users and public-keys to be defined and provide a method to manage associations between CAs. The certificate format also allows private extensions to be defined and can be designated as either critical or non-critical. If a critical extension cannot be recognised or processed, it is rejected by the system. A non-critical extension is processed if it is recognised and may be ignored if it is not. For the purpose of Chapter 5 of this thesis, we specially focus on certificate extensions which allow us to append additional functionality to a certificate.

2.2.3 Certificate Chains

Current certificate authorities can be classified as root CAs, intermediate CAs and untrusted CAs. Root CAs are trusted by browsers, intermediate CAs are trusted by one of the root CAs and those neither trusted by browsers nor any intermediate CA falls under the untrusted category [SS12]. Root CAs are also known as *"trust anchors"*. The term *'Chain of Trust'* is used to describe the trust relationship between identities when using intermediate CAs. This allows the delegation of certificate issuance by intermediate CAs. The chain of trust has to end eventually with a root CA, meaning that intermediate CAs do not necessarily have to be verified by a root CA but can be verified by another intermediate CA, but when followed through all the CAs involved, the final link in the line should be a root CA. In a browser, root CAs are built in and are trusted by the browser. A root CA certificate may be the base to issue multiple intermediate CA

certificates with varying validation requirements. If those root CAs are trusted, their intermediate CAs are automatically accepted and are equally trusted by browsers. There is a long list of trusted root CAs and any of those CAs is capable of issuing a certificate for any website in any country or top level domain. In fact there are no restrictions for a CA which forbid issuing a certificate to a malicious third party.

The major web browsers (Internet Explorer, Firefox, Chrome and Safari) have their own technical standards that specify how they select their trusted CA list. Some browsers maintain their own database while others rely on the list provided by the operating system. Browsers perform a number of checks when a TLS connection is established with a website. The certificate of a certain website is methodically checked evaluating whether it matches the its domain name, it is signed by a CA that user's browser trusts and it has not expired. After the evaluation, if it is satisfied with the authenticity of the website, a TLS connection is established.

Web browsers typically display additional information when it is through, which are known as *security indicators*. Browsers employ security indicators that users should evaluate when connecting to web and when making a security or privacy decision. Browser includes the protocol used, the domain name, the SSL certificate and visual elements in browser window. There are mainly four security indicators that signify the strong security offered by HTTPS [SBvP08, Ste10, SEA+09].

1. URL which begins with HTTPS in the location bar.
2. Correct domain name of the URL in the location bar.
3. Displaying lock icon in the non-content part in the browser.
4. Indicators of extended validation certificate; green background in the location bar and name of the company on its certificate.

Furthermore, browsers also warn their users when faced with expired certificates or unknown CAs before establishing the TLS connection with a website.

2.2.4 Revoking Certificates

All certificates have a specific life span that is coded as a validity period. Certificate expiration occurs when it is no longer valid because the end date has arrived.

The invalidation of a certificate before the end of its expected life cycle is known as certificate revocation. A certificate can be revoked by the CA who issued the certificate after a request from an authorised person, such as the domain owner. There may be several reasons for a certificate to be revoked:

- Improper issuance of a certificate by a CA.
- Compromise of the CA.
- Failure of an entity to comply with the policy requirements specified by the CA.
- Failure of an entity to hold the sole possession of the private key (lost or stolen private-key).
- Unspecified or changed affiliation of an entity of the certificate.
- Weaknesses in the algorithms used in the certificate (e.g. MD5, SHA-1, RSA-1024).

There are four main revocation procedures that are been used so far [Mye98]: *Certificate Revocation List (CRL)*, *Online Certificate Status Protocol (OCSP)*, *short-lived certificates*, and *trusted directories*. The CA/Browser model uses first two and thus we have limited our study to only CRLs and OCSP.

A list of certificates that are no longer considered valid is known as a certificate revocation list [SHF02]. The list is uploaded to a public repository, such as an FTP directory. CRL indicates that entities which represent those revoked certificates should no longer be trusted by the users/clients. CRLs are generally issued by CAs which issue the corresponding certificates and usually carry digital signature associated with the CA to prevent any attacks like spoofing. When a client wants to check if a certificate is revoked, a recent CRL is obtained and checked to see if the serial number of the certificate is in the list. Users have to manually download and check this list.

An alternative to CRLs is Online Certificate Status Protocol [GSM+13], which can be used to obtain the revocation status of a particular certificate. Although CRLs prevent clients from creating a TLS connection with domains with revoked certificates, the drawback is that clients need to be able to access current CRLs. Furthermore some CRLs can be of large size as there could be many revoked certificates. To resolve this online validation requirement, OCSP was proposed.

When queried, a server called an OCSP responder answers with an OCSP response. This response is a time-stamped data structure signed by the CA, which contains the current revocation status of a certificate. OCSP responses contain less information than a CRL therefore the burden on network and client resources is lesser in comparison. Unfortunately it requires CAs to respond to every client of a given certificate in real time which involves a considerable cost for the CAs and breach of clients' privacy. *OCSP Stapling* [Eas11] reduces those issues by attaching or "stapling" a time-stamped OCSP response signed by the CA in the initial TLS handshake. Although it resolves the massive privacy concern with OCSP, an attacker could easily intervene and disable OCSP stapling in the TLS connection.

Though it is vitally important to deploy HTTPS, the current revocation system is not efficient or reliable to handle a large scale attack. Consequently a trustworthy revocation mechanism is a necessity for the current PKI.

2.2.5 Transparency and Security Issues

Public-key infrastructure depends on CAs being truthful, honest and legitimate. Thus, if an attacker is able to gain control of a CA, he/she then can use it to issue fraudulent certificates for any subject. A certificate substitution attack can happen where those fraudulent certificates are used to masquerade as the real website to a user, intercepting the communication between user and web server. If this compromised CA was accepted previously by a user's browser as valid, then the user will not be notified of the attack due to a lack of warnings from browser security indicators. Over the past few years there have been many incidents involving compromised CA organisations. These include, DigiNotar, GlobalSign, Comodo and Digicert Malaysia [GSE11, Man12].

Domain owners may find it useful to monitor certificate issuance for their domain and use that to detect misissued certificates. However, in the current PKI model, there is no efficient way to get a comprehensive list of certificates issued to a certain domain. Thus, there is the possibility of a certificate being issued without the knowledge of the domain owner. Furthermore, there are limited methods (e.g. Public key pinning [EP15]) for any entity to determine whether certificates have been mistakenly or maliciously issued. If left unchecked, these transparency issues can facilitate a wide range of security attacks, such as website spoofing, server impersonation, and man-in-the-middle attacks.

There are a few forms of MITM attacks. Two popular one are remote and local attacks. For instance in an online banking scenario, remote attacks can be considered as attacks done using techniques such as phishing; the fraudulent attempt to obtain sensitive information such as usernames, passwords, and credit card details, by impersonating as a trustworthy entity. In this instance, attackers lure the banking customer to a rogue website that looks like the real website, using malicious certificates. A local attack is carried out using malicious software installed on the customer's computer, also called spy-ware or crime-ware — usually infecting a computer through downloads or e-mail attachments.

2.2.5.1 Attacks due to Expired/Revoked Certificates

The intention of certificate revocation is to notify a complete withdrawal of trust in the certificate and to protect web users from fraud, theft, and eavesdropping. Nevertheless, certificate revocation is not handled properly by some browsers. As a result, if a certain certificate for a domain is revoked, most frequent users of the site and even its administrators can be unaware of it and continue using the certificate. If an adversary obtains of a certain certificate's private-key and if it is not revoked immediately he/she has the possibility to perform a MITM attack by presenting the certificate to unsuspecting users whose browsers will not give out any warnings.

From what we have discussed above, we can say that many of the failures in current PKI have occurred due to lack of transparency between involved entities. The basic web PKI includes several types of entities, performing different tasks: certificate authorities, web servers, browser vendors and web browsers. In its current standard scheme, unwarranted trust in CAs has lead to many organisations being attacked and compromised. These issues over the recent years have significantly motivated researchers to look for mechanisms to enhance PKI from its current state.

2.3 Security Frameworks

In order to formalise our understanding of the security protocols, it is necessary to define both the exact security goals and adversarial threat model that the protocol is measured against. In this research, we aim to use *provable security* security framework.

2.3.1 Provable Security: Game-based

In general, provable security is any type or level of security that can be proven rigorously. It aims to give a mathematical guarantee that a scheme is secure; usually given as mathematical proofs in cryptography. Such a proof considers the capabilities of attackers as an adversarial model. It should be proven that a specific property of the scheme cannot be broken by a class of attackers assuming that they have access to the system as well as enough computational resources [Ste14]. Apart from assumptions that distinguish models from real world systems, for such a proof we need:

- a description of the scheme/protocol
- a description of security goals
- an adversarial model (attacking environment, how adversary interacts with the scheme and winning condition)
- a proof that no adversary can achieve the winning condition in the defined adversarial model

Provable security framework has several approaches. The main two approaches are *Information theoretic security* and *Computational security*. Computational security, uses complexity theoretic techniques to define security against a class of attackers who have restricted computational power.

- *Formal methods*: Computer-verified security of scheme/protocol. Typically assumes underlying cryptography is perfect.
- *Reductionist proof*: Manual proof of security of scheme/protocol by reducing security of the scheme to the security of an underlying hard problem. Introduced by Goldwasser and Micali [SM84].
  - Game-based: Security is outlined as a game played between an adversary and a hypothetical challenger. The challenger's responses define the security model.

We use game-based proof techniques to relate the difficulty of breaking the overall protocol to the difficulty of breaking the individual primitives. These primitives are typically formalised as a game played between a well defined challenger and

an adversary, which is defined as a *security experiment*. The game is played between two algorithms: a *challenger* and an *attacker*. The challenger typically does three things:

1. *Setup.* Sets up the experiment. Eg. Generating any parameters.
2. *Execution.* Executes the adversary. Eg. Provide inputs to adversary, let it run, interact with it (optional) and receive output from adversary.
3. *Winning condition.* Decides whether the adversary has completed the task or not.

The difficulty of breaking the primitives (or completing the task) is expressed as the advantage of an adversary at meeting winning conditions evaluated by the challenger. We define a successful adversary as one that has a non-negligible advantage (or probability) of winning the game. It should also be noted that proofs from provable security only promise security within the assumptions of the model, which may not include implementation details of the scheme, and could only be about a mathematical abstraction of a protocol, not the protocol as used in practice.

2.4 Data Structures

2.4.1 Merkle Trees

Now we review a well known data structure, Merkle trees, which will also be discussed further in later chapters of this thesis. Merkle trees are also known as Merkle hash trees or binary hash trees, and was first proposed by Merkle [Mer79, Mer90] for authenticating large amounts of data. Since then, Merkle trees have been used in many areas of cryptography and computer science, including in the construction of public-key signatures from hash functions. Most uses of Merkle trees concern a static dataset. However, for the purpose of this thesis, we are concerned with a dynamic dataset, and in particular append-only nature of the dataset. A Merkle tree is constructed by placing the values of a set of records/entries in leaves of a binary tree and hashing each record. Each intermediate or parent node is built by combining the hashes of its child nodes. Typically, Merkle trees have a branching factor of 2, meaning that each node has up to 2 children. The root of the tree acts as a fingerprint for all the values

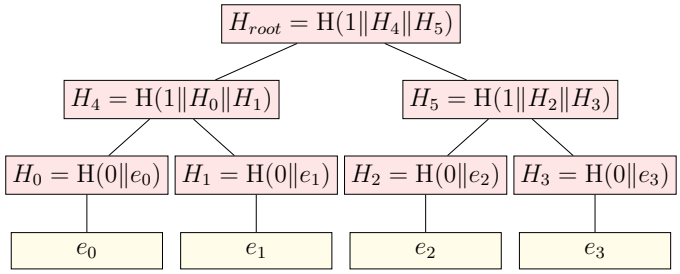


Figure 2.3: Merkle hash tree with an input of entries labeled e_0 though e_3. Each of these entries are hashed using some hash function. Then each pair of nodes are recursively hashed until the root node, Merkle Hash is reached, which is a hash of all nodes below it.

contained in tree, known as the *Merkle root hash*. One can prove that a certain record exists in a tree with n nodes by providing the $\mathcal{O}(log\ n)$ hashes needed to reconstruct the root hash.

Figure 2.3 shows an example of a Merkle tree hash calculation. Note the use of prefixes 0 and 1 in hash function calculations provides "domain separation" between hash calculations for leaves ($\mathrm{H}(0\|\dots)$) and intermediate nodes ($\mathrm{H}(1\|\dots)$); preventing an attacker from gluing part of a tree into a leaf or vice versa. This is called a length extension attack where attacker can use $\mathrm{H}(m_1)$ and the length of m_1 to calculate $\mathrm{H}(m_1\|m_2)$ for an attacker-controlled m_2. This allows the attacker to include extra information at the end of the message and produce a valid hash without knowing the m_1. Algorithms like MD5, SHA-1, and SHA-2 that are based on the Merkle–Damgård construction [Mer79] are susceptible to length extension attack.

Now we formally define a Merkle tree hash system as follows.

Definition 5 (Merkle tree hash)**.** *Let* $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *be a hash function and let* $\vec{E}$ *be a list of entries. The* Merkle tree hash *system consists of the following algorithms:*

- $\mathtt{MTH}_\mathrm{H}(\vec{E}) \to h$*: A deterministic tree hashing algorithm that takes as input a list of entries* $\vec{E}$ *each of which is in* $\mathcal{M}$ *and outputs a hash value* $h \in \{0,1\}^\lambda$ *representing the complete list of entries as defined in Figure 2.6.*

- $\mathtt{Path}_\mathrm{H}(m, \vec{E}) \to \vec{A}$*: A deterministic authentication path generation algorithm that takes as input an index* $m \in \{0, \dots, |\vec{E}| - 1\}$ *and a list of entries* $\vec{E}$ *and outputs an authentication path* $\vec{A}$ *as defined in Figure 2.6.*

- $\texttt{CheckPath}_{\mathrm{H}}(e, H, n, \vec{A}, m) \to \{0,1\}$*: A deterministic authentication path verification algorithm that takes as input an entry* $e \in \mathcal{M}$*, a hash* $H \in \{0,1\}^{\lambda}$ *representing a list of* n *entries, a claimed authentication path* $\vec{A}$*, and an index* $m \in \{0, \ldots, n-1\}$*, and outputs 0 or 1 as defined in Figure 2.6. The goal is to output 1 if and only if* $\vec{A}$ *is an authentication path demonstrating that* e *is the* m*-th leaf of the* n*-leaf tree represented by hash value* H*.*

- $\texttt{ConsProof}_{\mathrm{H}}(m, n, \vec{E}) \to \vec{C}$*: A deterministic consistency proof generation algorithm that takes as input a list of entries* $\vec{E}$*, and two indices* $0 \leq m \leq n \leq |\vec{E}|$*, and outputs a consistency proof* $\vec{C}$ *as defined in Figure 2.6.* $\vec{C}$ *consists of intermediate nodes required to simultaneously construct the roots of the trees* $\vec{E}[0:m]$ *and* $\vec{E}[0:n]$*.*

- $\texttt{CheckConsProof}_{\mathrm{H}}(n_0, H_0, n_1, H_1, \vec{C}) \to \{0,1\}$*: A deterministic consistency proof verification algorithm that takes as input two indices* $0 \leq n_0 \leq n_1$*, two hash values* $H_0, H_1 \in \{0,1\}^*$*, and a claimed consistency proof* $\vec{C}$*, and outputs 0 or 1 as defined in Figure 2.6. The goal is to output 1 if and only if the* n_0 *entries represented by* H_0 *are a prefix of the* n_1 *entries represented by* H_1*.*

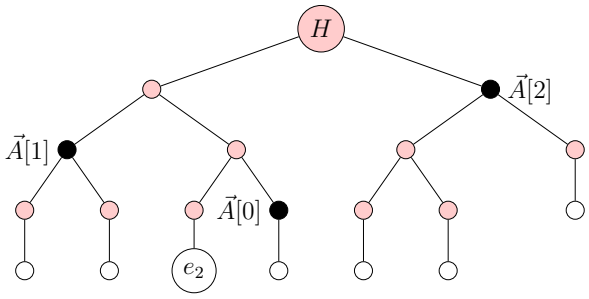


Figure 2.4: Merkle tree authentication path $\vec{A}$ from leaf e_2 to root H. ● denotes nodes corresponding to authentication path values.

Merkle trees are used for efficient data verification. An *authentication path* is used to demonstrate that a piece of data is in a leaf of a tree corresponding to a particular root. For example, in Figure 2.4, the authentication path $\vec{A} = (\vec{A}[0], \vec{A}[1], \vec{A}[2])$ shows that e_2 is the third leaf in the tree corresponding to root H, and this can be verified by computing $h_0 = \mathrm{H}(0\|e_2)$, then $h_1 = \mathrm{H}(1\|h_0\|\vec{A}[0])$, then $h_2 = \mathrm{H}(1\|\vec{A}[1]\|h_1)$, then $H' = \mathrm{H}(1\|h_2\|\vec{A}[2])$ and comparing H' with H. The index of the leaf is an essential part of verifying

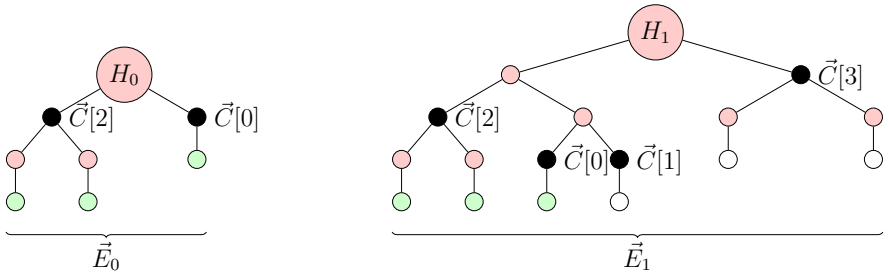


Figure 2.5: Merkle tree consistency proof $\vec{C}$ between roots H_0 (for a tree of size 3) and H_1 (for a tree of size 6). ● denotes nodes corresponding to consistency proof values.

an authentication path. Figure 2.6 describes the authentication path generation algorithm $\texttt{Path}_{\mathrm{H}}(m, \vec{E})$ and verification algorithm $\texttt{CheckPath}_{\mathrm{H}}(e, H, n, \vec{A}, m)$. Further, Figure 2.6 describes the other algorithms in Definition 5.

In Figure 2.5, we show how a *consistency proof* can be used to demonstrate that the data corresponding to one root is a subset (prefix) of the data corresponding to another root. This supports the concept of tamper-evident history trees discussed by Crosby et.al. [CW09, Cro09]. To verify the consistency between two trees, we need to verify that the set of entries represented by the old Merkle tree hash is a subset of the set of entries represented by the new Merkle tree hash. Then we need to verify that the new Merkle tree hash is the concatenation of the old Merkle tree hash plus all the intermediate node hashes of the newly appended entries. The consistency proof is the minimum set of intermediate node hashes needed to compute these two things. The consistency proof $\vec{C}$ in Figure 2.5 shows that the data corresponding to root H_0 is a prefix of the data corresponding to root H_1. Using the consistency proof we can reconstruct each of the two roots from relevant parts of the proof and compare them against the actual roots; the size of the two trees is essential in verifying a consistency proof. The size of the trees are crucial to prove that no invalid entries are added to the tree. These constructions are shown in Figure 2.6 algorithms $\texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$ and $\texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$. The minimum set of intermediate node hashes given in the proof can be considered as an authentication path from the particular leaf node we want to prove existence to the root (i.e., an authentication path from $\mathrm{H}(e_2) = \vec{C}[0]$ to root H_1 in the right side of Figure 2.5)

$\underline{\texttt{MTH}_{\mathrm{H}}(\vec{E}) \to H\text{:}}$
1: $n \leftarrow |\vec{E}|$
2: **if** $n = 1$, **return** $\mathrm{H}(0\|\vec{E}[0])$
3: **else** $(n > 1)$
4: $\quad k \leftarrow 2^{\lceil \log_2(n/2) \rceil}$
5: $\quad$ **return** $\mathrm{H}(1\|\texttt{MTH}_{\mathrm{H}}(\vec{E}[0:k])$
6: $\qquad\qquad\qquad \|\texttt{MTH}_{\mathrm{H}}(\vec{E}[k:n]))$

$\underline{\texttt{Path}_{\mathrm{H}}(m, \vec{E}) \to \vec{A}\text{:}}$
1: $n \leftarrow |\vec{E}|$
2: **if** $n = 1$, **return** $()$
3: **else** $(n > 1)$
4: $\quad k \leftarrow 2^{\lceil \log_2(n/2) \rceil}$
5: $\quad$ **if** $m < k$
6: $\quad\quad$ **return** $\texttt{Path}_{\mathrm{H}}(m, \vec{E}[0:k])$
7: $\qquad\qquad \|\texttt{MTH}_{\mathrm{H}}(\vec{E}[k:n])$
8: $\quad$ **else** $(m \geq k)$
9: $\quad\quad$ **return** $\texttt{Path}_{\mathrm{H}}(m-k, \vec{E}[k:n])$
10: $\qquad\qquad \|\texttt{MTH}_{\mathrm{H}}(\vec{E}[0:k])$

$\underline{\texttt{CheckPath}_{\mathrm{H}}(e, H, n, \vec{A}, m) \to \{0,1\}\text{:}}$
1: $H' \leftarrow \texttt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m)$
2: **return** $(H = H')$

$\underline{\texttt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m) \to H\text{:}}$
1: **if** $n = 1$, **return** $\mathrm{H}(0\|e)$
2: $k \leftarrow 2^{\lceil \log_2(n/2) \rceil}$
3: **if** $m < k$
4: $\quad \ell \leftarrow \texttt{RootFromPath}_{\mathrm{H}}(e, k, \vec{A}[0:|\vec{A}|-1], m)$
5: $\quad r \leftarrow \vec{A}[|\vec{A}|-1]$
6: **else** $(m \geq k)$
7: $\quad \ell \leftarrow \vec{A}[|\vec{A}|-1]$
8: $\quad r \leftarrow \texttt{RootFromPath}_{\mathrm{H}}(e, n-k,$
9: $\qquad\qquad \vec{A}[0:|\vec{A}|-1], m-k)$
10: **return** $\mathrm{H}(1\|\ell\|r)$

$\underline{\texttt{ConsProof}_{\mathrm{H}}(m, n, \vec{E}) \to \vec{C}\text{:}}$
1: *// require:* $0 \leq m \leq n \leq |\vec{E}|$
2: **if** $m = n$
3: $\quad$ **return** $()$
4: **else** $(m < n)$
5: $\quad$ **return** $\texttt{ConsProofSub}_{\mathrm{H}}(m, \vec{E}[0:n], \text{true})$

$\underline{\texttt{ConsProofSub}_{\mathrm{H}}(m, \vec{E}, b) \to \vec{C}\text{:}}$
1: $n \leftarrow |\vec{E}|$
2: **if** $(m = n) \wedge (b = \text{false})$
3: $\quad$ **return** $\texttt{MTH}_{\mathrm{H}}(\vec{E}[0:m])$
4: **else**
5: $\quad k \leftarrow 2^{\lceil \log_2(n)/2 \rceil}$
6: $\quad$ **if** $m \leq k$
7: $\quad\quad$ **return** $\texttt{ConsProofSub}_{\mathrm{H}}(m, \vec{E}[0:k], b)$
8: $\qquad\qquad \|\texttt{MTH}_{\mathrm{H}}(\vec{E}[k:n])$
9: $\quad$ **else** $(m > k)$
10: $\quad\quad$ **return** $\texttt{ConsProofSub}_{\mathrm{H}}(m-k, \vec{E}[k:n], \text{false})$
11: $\qquad\qquad \|\texttt{MTH}_{\mathrm{H}}(\vec{E}[0:k])$

$\underline{\texttt{CheckConsProof}_{\mathrm{H}}(n_0, H_0, n_1, H_1, \vec{C}) \to b\text{:}}$
1: **if** n_0 is a power of two, $\vec{C} \leftarrow H_0\|\vec{C}$
2: $H_0' \leftarrow \texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$
3: $H_1' \leftarrow \texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$
4: **return** $((H_0 = H_0') \wedge (H_1 = H_1'))$

$\underline{\texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) \to H\text{:}}$
1: $k \leftarrow 2^{\lceil \log_2(n_1)/2 \rceil}$
2: **if** $n_0 < k$
3: $\quad$ **return** $\texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1], n_0, k)$
4: **elsif** $n_0 = k$, **return** $\vec{C}[|\vec{C}|-2]$
5: **else**
6: $\quad \ell \leftarrow \vec{C}[|\vec{C}|-1]$
7: $\quad r \leftarrow \texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1],$
8: $\qquad\qquad n_0-k, n_1-k)$
9: $\quad$ **return** $\mathrm{H}(1\|\ell\|r)$

$\underline{\texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) \to H\text{:}}$
1: **if** $|\vec{C}| = 2$, **return** $\mathrm{H}(1\|\vec{C}[0]\|\vec{C}[1])$
2: $k \leftarrow 2^{\lceil \log_2(n_1)/2 \rceil}$
3: **if** $n_0 < k$
4: $\quad \ell \leftarrow \texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1], n_0, k)$
5: $\quad r \leftarrow \vec{C}[|\vec{C}|-1]$
6: **else**
7: $\quad \ell \leftarrow \vec{C}[|\vec{C}|-1]$
8: $\quad r \leftarrow \texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1],$
9: $\qquad\qquad n_0-k, n_1-k)$
10: **return** $\mathrm{H}(1\|\ell\|r)$

Figure 2.6: Merkle tree algorithms

2.4.1.1 Merkle Tree Security Properties

We now note some well-known facts about the collision resistance of Merkle tree hashing and the security of authentication paths in Merkle trees [Mer79, Mer90]. If the hash function used in Merkle tree is proved to be collision resistant then the hashes in Merkle tree are also collision resistant (see Lemma 1 below). Furthermore, if the hash function used in Merkle tree, consistent authentication paths are also guaranteed. In other words, a path with respect to Merkle tree hashing cannot be generated for an entry that does not exist in the Merkle Tree (see Lemma 2). Now we proceed to prove the Merkle tree security properties.

Lemma 1 (Collision Resistance of Merkle Trees)**.** *If* H *is collision-resistant, then Merkle-tree hashing using* H *is also collision-resistant. More precisely, if* $\mathcal{A}$ *finds a collision in* $\mathtt{MTH}_{\mathrm{H}}$*, then there exists algorithm* $\mathcal{B}_1^{\mathcal{A}}$ *that finds a collision in* H*. Moreover, the runtime of* $\mathcal{B}_1^{\mathcal{A}}$ *consists of the runtime of* $\mathcal{A}$*, plus at most a quadratic (in the size of the larger list) number of hash evaluations.*

Proof of Lemma 1. Suppose $\mathcal{A}$ outputs sets $\vec{E}_0$ and $\vec{E}_1$ such that $\mathtt{MTH}_{\mathrm{H}}(\vec{E}_0) = \mathtt{MTH}_{\mathrm{H}}(\vec{E}_1)$ but $\vec{E}_0 \neq \vec{E}_1$. The algorithm $\mathcal{B}_1$ in Figure 2.7 then runs $\mathcal{R}_1$ on $\vec{E}_0$, $\vec{E}_1$. $\mathcal{R}_1$ is a recursive algorithm which takes as input two lists that are distinct but hash to the same value under $\mathtt{MTH}_{\mathrm{H}}$. Somewhere in the recursive computation of $\mathtt{MTH}_{\mathrm{H}}(\vec{E}_0)$ and $\mathtt{MTH}_{\mathrm{H}}(\vec{E}_1)$, a collision occurs in H, and $\mathcal{R}_1$ recurses until it finds that collision. □

Lemma 2 (Authentication Paths Consistency)**.** *If* H *is collision-resistant, then no* $\mathtt{CheckPath}_{\mathrm{H}}$ *authentication path* $\vec{A}$ *can be generated with respect to Merkle-tree hashing* $\mathtt{MTH}_{\mathrm{H}}$ *for an entry* e *not contained in the Merkle tree. More precisely, if* $\mathcal{A}$ *outputs* $(e, \vec{E}, \vec{A}, m)$ *such that* $\mathtt{CheckPath}_{\mathrm{H}}(e, \mathtt{MTH}_{\mathrm{H}}(\vec{E}), |\vec{E}|, \vec{A}, m) = 1$ *and* $e \notin \vec{E}$*, then there exists algorithm* $\mathcal{B}_2^{\mathcal{A}}$ *that finds a collision in* H*. Moreover, the runtime of* $\mathcal{B}_2^{\mathcal{A}}$ *consists of the runtime of* $\mathcal{A}$*, plus at most a quadratic (in* $|\vec{E}|$*) number of hash evaluations.*

Proof of Lemma 2. Suppose the algorithm $\mathcal{R}_2$ in Figure 2.8 receives as input values $e, \vec{D}, \vec{A}, m$ such that

$$\mathtt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m) = \mathtt{MTH}_{\mathrm{H}}(\vec{D}) \tag{2.2}$$

(where $n = |\vec{D}|$) but

$$e \notin \vec{D} \ . \tag{2.3}$$

We claim that $\mathcal{R}_2$ outputs a collision for H when run with values satisfying (2.2) and (2.3). The proof of this claim proceeds by induction on n.

First suppose $n = 1$. Then $\mathtt{MTH}_{\mathrm{H}}(\vec{D}) = \mathrm{H}(0\|\vec{D}[0])$ and $\mathtt{RootFromPath}_{\mathrm{H}}(e, 1, \vec{A}, m) = \mathrm{H}(0\|e)$, and these are equal by assumption on $\mathcal{R}_2$'s inputs. Since $e \notin \vec{D}$, we have that $e \neq \vec{D}[0]$ and thus $0\|\vec{D}[0] \neq 0\|e$, but $\mathrm{H}(0\|\vec{D}[0]) = \mathrm{H}(0\|e)$, which is a collision for H. This is what is output by $\mathcal{R}_2$ on line 3.

Now suppose $n > 1$. Let $k = 2^{\lceil \log_2(n)/2 \rceil}$.

Suppose $m < k$, and let ℓ, r, ℓ', and r' be as on lines 5, 6, 8, and 9 of $\mathcal{R}_2$ in Figure 2.8. Then $\mathtt{MTH}_{\mathrm{H}}(\vec{D}) = \mathrm{H}(1\|\ell\|r)$ and $\mathtt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m) =$

$\underline{\mathcal{B}_1^{\mathcal{A}}()}$:
1: $(\vec{E}_0, \vec{E}_1) \leftarrow \mathcal{A}()$
2: *// assume* $\mathtt{MTH}_{\mathrm{H}}(\vec{E}_0) = \mathtt{MTH}_{\mathrm{H}}(\vec{E}_1)$ *but* $\vec{E}_0 \neq \vec{E}_1$
3: **return** $\mathcal{R}_1(\vec{E}_0, \vec{E}_1)$

$\underline{\mathcal{R}_1(\vec{D}_0, \vec{D}_1)}$:
1: *// require:* $\vec{D}_0 \neq \vec{D}_1$ *but* $\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0) = \mathtt{MTH}_{\mathrm{H}}(\vec{D}_1)$
2: $n_0 \leftarrow |\vec{D}_0|$, $n_1 \leftarrow |\vec{D}_1|$
3: $k_0 \leftarrow 2^{\lceil \log_2(n_0)/2 \rceil}$, $k_1 \leftarrow 2^{\lceil \log_2(n_1)/2 \rceil}$
4: **if** $n_0 = n_1 = 1$, **return** $(0\|\vec{D}_0[0], 0\|\vec{D}_1[0])$
5: **elseif** $(n_0 = 1) \wedge (n_1 > 1)$
6: **return** $(0\|\vec{D}_0[0], 1\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k_1])\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[k_1:n_1]))$.
7: **elseif** $(n_0 > 1) \wedge (n_1 = 1)$
8: **return** $(1\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[0:k_0])\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[k_0:n_0]), 0\|\vec{D}_1[0])$
9: **else** $(n_0, n_1 > 1)$
10: **if** $(\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[0:k_0]) \neq \mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k_1]))$
11: $\vee$ $(\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[k_0:n_0]) \neq \mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[k_1:n_1]))$
12: **return** $(1\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[0:k_0])\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_0[k_0:n_0]),$
13: $1\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k_1])\|\mathtt{MTH}_{\mathrm{H}}(\vec{D}_1[k_1:n_1]))$
14: **elseif** $\vec{D}_0[0:k_0] \neq \vec{D}_1[0:k_1]$
15: **return** $\mathcal{R}_1(\vec{D}_0[0:k_0],\ \vec{D}_1[0:k_1])$
16: **elseif** $\vec{D}_0[k_0:n_0] \neq \vec{D}_1[k_1:n_1]$
17: **return** $\mathcal{R}_1(\vec{D}_0[k_0:n_0],\ \vec{D}_1[k_1:n_1])$

Figure 2.7: Algorithm $\mathcal{B}_1$ for Lemma 1.

$\mathrm{H}(1\|\ell\|r)$, and these are equal by assumption on $\mathcal{R}_2$'s inputs. If $\ell \neq \ell'$ or $r \neq r'$, then we immediately have a collision for H: $(1\|\ell\|r, 1\|\ell'\|r')$, which is what $\mathcal{R}_2$ outputs on line 10. If these are all equal, then $e, \vec{D}[0:k], \vec{A}[|\vec{A}|-1], m$ satisfy (2.2) and (2.3). By induction, this recursive call to $\mathcal{R}_2$ outputs a collision.

Suppose $m \geq k$, and let ℓ, r, ℓ', and r' be as on lines 5, 6, 13, and 14 of $\mathcal{R}_2$ in Figure 2.8. Then $\mathtt{MTH}_{\mathrm{H}}(\vec{D}) = \mathrm{H}(1\|\ell\|r)$ and $\mathtt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m) = \mathrm{H}(1\|\ell\|r)$, and these are equal by assumption on $\mathcal{R}_2$'s inputs. If $\ell \neq \ell'$ or $r \neq r'$, then we immediately have a collision for H: $(1\|\ell\|r, 1\|\ell'\|r')$, which is what $\mathcal{R}_2$ outputs on line 15. If these are all equal, then $e, \vec{D}[k:n], \vec{A}[|\vec{A}|-1], m$ satisfy (2.2) and (2.3). By induction, this recursive call to $\mathcal{R}_2$ outputs a collision.

□

2.4.2 Decentralised Public Ledger: Blockchain

We now discuss another particular security protocol: decentralised ledger. A decentralised ledger is a database that is consensually shared and synchronised across network geographically spread across multiple sites, countries or institutions. The most interesting part here is the decentralised consensus mechanism for

$\underline{\mathcal{B}_2^{\mathcal{A}}():}$
1: $(e, \vec{E}, \vec{A}, m) \leftarrow \mathcal{A}()$
2: *// assume* $\texttt{RootFromPath}_{\mathrm{H}}(e, n, \vec{A}, m) = \texttt{MTH}_{\mathrm{H}}(\vec{E})$ *but* $e \notin \vec{E}$
3: **return** $\mathcal{R}_2(e, \vec{E}, \vec{A}, m)$

$\underline{\mathcal{R}_2(e, \vec{D}, \vec{A}, m):}$
1: *// require:* $\texttt{RootFromPath}_{\mathrm{H}}(e, |\vec{D}|, \vec{A}, m) = \texttt{MTH}_{\mathrm{H}}(\vec{D})$
2: $n \leftarrow |\vec{D}|$
3: **if** $n = 1$, **return** $(0\|\vec{D}[0], 0\|e)$
4: $k \leftarrow 2^{\lceil \log_2(n)/2 \rceil}$
5: $\ell \leftarrow \texttt{MTH}_{\mathrm{H}}(\vec{D}[0:k])$
6: $r \leftarrow \texttt{MTH}_{\mathrm{H}}(\vec{D}[k:n])$
7: **if** $m < k$
8: $\quad \ell' \leftarrow \texttt{RootFromPath}_{\mathrm{H}}(e, k, \vec{A}[0:|\vec{A}|-1], m)$
9: $\quad r' \leftarrow A[|A|-1]$
10: $\quad$ **if** $(\ell \neq \ell') \vee (r \neq r')$, **return** $(1\|\ell\|r, 1\|\ell'\|r')$
11: $\quad$ **else return** $\mathcal{R}_2(e, \vec{D}[0:k], \vec{A}[0:|\vec{A}|-1], m)$
12: **else** $(m \geq k)$
13: $\quad \ell' \leftarrow \vec{A}[|\vec{A}|-1]$
14: $\quad r' \leftarrow \texttt{RootFromPath}(e, n-k, \vec{A}[0:|\vec{A}|-1], m-k)$
15: $\quad$ **if** $(\ell \neq \ell') \vee (r \neq r')$, **return** $(1\|\ell\|r, 1\|\ell'\|r')$
16: $\quad$ **else return** $\mathcal{R}_2(e, \vec{D}[k:n], \vec{A}[0:|\vec{A}|-1], m-k)$

Figure 2.8: Algorithm $\mathcal{B}_2$ for Lemma 2.

agreeing on what data is in the database. There is no central authority controlling the operations nor centralised data storage. These decentralised ledgers allow to have public witnesses reducing the opportunity for manipulation. The participants involved in the network are called nodes and a peer-to-peer network is required to make the connections between nodes in the network. Each of the nodes in the network can access the records shared across the network and has their own copy of the data. One form of widely popularised and successful distributed ledger design is the *blockchain* system.

Blockchain is a continuously growing public ledger of records called blocks, which are linked and secured using cryptography. It was originally designed for storing financial transactions for the Bitcoin cryptocurrency by Satoshi Nakamoto [Nak08]. Blockchain is managed by a large peer-to-peer network collectively working and adopting a certain protocol to validate blocks. A consensus among the entities solve the problem of double spending for digital money without a central authority, which is replaced by an open, dynamic and decentralised network. Transactions are broadcast on this network and recorded on blockchain where each node in the network keeps its own copy.

In blockchain, each block contains a cryptographic hash of the previous block, a timestamp and transaction data. The transaction data is linked to the block

header through the root of a Merkle tree with transactions as leaves. Each block header also contains a hash of the previous block. This process links all the blocks together, creating an ever growing chain. Transactions are collected and included in a block which is appended to the chain by a node in the network selected according to a consensus algorithm. When a new block is created, it is distributed in the network and all participating nodes update their local copy of the chain. Transactions are confirmed by consecutively adding blocks and a fork could be created if there are two blocks added at the same time. Thus a consensus algorithm is used which not only keeps powerful adversaries from hindering the system and successfully forking the chain but also ensures each new block added to the chain is the one and only legitimate block.

There are several consensus protocols proposed for distributed ledgers. Two of the most popular protocols widely used today are *Proof of Work (PoW)* [Bac02, Nak08] and *Proof of Stake (PoS)* [KN12]. In proof of work, there are machines working on solving an extremely difficult cryptographic puzzle. These machines are called miners in the network. PoW is basically a piece of data which is difficult (costly or time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Miners must complete PoW containing all of the data in the block, in order for that block to be accepted by network peers.

In Bitcoin, this cryptographic puzzle is to find a solution such that, for a difficulty target t

$$\mathrm{H}(\mathrm{H}(\mathit{blockheader} \| \mathit{solution})) \leq t.$$

Difficulty target is adjusted every 2 weeks so that on average a new block is appended to the blockchain every 10 minutes. The *longest chain* in Bitcoin's PoW is considered to be the blockchain with the most accumulated difficulty. When a new block is created, there is a *block reward* and also a transaction cost collected by all the transactions included in that block. This amount is paid to the address found in the fist transaction in the block, called the *coinbase transaction*, which is the address of the miner who created the block. This reward is the incentive for miners to keep adding new blocks to the blockchain and maintains the public ledger. However, with PoW based consensus, adding a new block to the blockchain is economically expensive as it requires expensive hardware and large amount of electricity. Even with adequate resources, finding the solution to a new block is sometimes unlikely for the individual miners, thus the common

practice now is for miners work together in pools, called *mining pools*. If any miner in the pool finds the solution to the puzzle, the whole pool shares the reward.

2.4.3 Blockchain Security

One of the benefits of blockchain is its resilience against adversaries. As mentioned above, since each block is connected to all the blocks before and after it, it makes difficult to tamper with a single record. To succeed tampering a block, an adversary must create two conflicting transactions by forking the main chain and then make the network accept the fork. Bitcoin and other cryptocurrencies based on Proof of Work protects against this kind of attack by making it computationally expensive to produce a new block thus fork the chain. Being unable to interfere with a previous record prevents *double-spending attacks*. Double spending on Bitcoin or any other cryptocurrencies is where an adversary tries to double spend a coin, that is trying to buy goods twice using the same coin.

However, if an adversary, or a colluding group of adversaries have the power of a mining pool that controls the majority of the hashing power, a double spending attack is possible to execute. Dubbed as a *51% attack*, it occurs if a single entity or set of individual entities together can contribute to the network's mining *hashrate* they would have the full dominance, and would be able to manipulate the Blockchain. Among other undesirable acts, a majority of network power could enable double spending and have the ability to prevent transaction confirmations, thus making them invalid. However, this attack on blockchain is still hypothetical.

Without the power of 51% hashrate, to succeed in an event of double spending, an adversary $\mathcal{A}$ must create two conflicting transactions by forking the main chain and then make the peers of the network accept the fork. Nevertheless, blockchain is built to prevent such type of attacks. As mentioned above, a blockchain based on a proof of work mechanism protects against these attacks by making it computationally expensive to produce a new block. The decision of the majority of nodes is represented by the longest chain which has the greatest proof of work effort invested in. If some adversary wishes to override this decision, they will have to put an effort more than all the honest nodes combined in the network. Thus, an adversary may find it is more beneficial to be an honest user than trying to surpass the honest chain as more effort im terms of resources and time needed to exceed the honest chain.

For the purpose of this thesis, we use blockchain purely for its properties of transparency, decentralisation and immutability. The security of blockchain has been shown in the success of the Bitcoin cryptocurrency. Thus, we do not consider any formal definitions of blockchain security properties and proving them using a security framework is out of scope of this thesis.

Chapter 3

Secure Logging Schemes and Certificate Transparency

Public-key infrastructure plays a huge role in achieving secure communication over the Internet. Thus, over the recent years, there has been a trend of many organisations implementing an effective public-key infrastructure and using the services of digital certificates. This demand puts a spotlight on Certificate Authorities, making them likely targets for sophisticated cyber attacks. However, replacing PKI with alternatives is not a practical solution. Therefore, the aim should be to enhance the current scheme rather than replacing it. Researchers have proposed several solutions to avoid existing flaws in the Certificate Authority ecosystem. Sovereign keys [Eck11], TACK [Mar13], ARPKI [BCK+14a], DANE [HS12], and Certificate Transparency [LKL13, Lau14, Rea13] could be named as few. Nevertheless, for most of these mitigation approaches, no proper security analysis has been done. In this chapter, we choose Certificate Transparency and define notions of security and prove its effectiveness in eliminating the flaws in PKI trust model and identify remaining limitations.

3.1 Introduction

The security of web communication via the TLS protocol relies on the safe distribution of public-keys in the from of X.509 certificates. In creating these digital certificates, CAs act as agents of trust in a PKI. As long as users trust

a CA and its business policies for issuing and managing certificates, they can trust certificates issued by the CA. However, over the past years shortcomings of the security procedures of CAs has threatened the trust in entire PKI on which the Internet depends. Especially, any attack targeting CAs causes PKI to be vulnerable.

In recent years there have been several high-profile cases of certificate mis-issuances. For instance, in 2011 a hacker managed to issue themselves a valid certificate for `google.com` and its sub-domains from the popular certificate authority DigiNotar [Fox12]. This was issued in July 2011 and may have been used maliciously for weeks before the large-scale MITM attack in August 2011 was detected and the certificate was revoked. In another instance, the Comodo Group suffered from an attack which resulted in the issuance of nine fraudulent certificates for domains owned by Google, Yahoo!, Skype, and others [Com11].

Therefore, over the years there have been many proposed models to manage public-keys and certificates to provide secure communication and alleviate the heavy reliance on a third party. When developing enhancements or alternatives to the current PKI, it is evident that the attention has shifted mostly to log-based certificate management. One such proposed project is Certificate Transparency, which is currently being implemented. Thus, it is important to verify that it achieves its security goals. In this work, we define four security properties of logging schemes such as CT that can be assured via *cryptographic means*, and show that CT does achieve these security properties. We consider two classes of security goals regarding the entities involved in CT. Furthermore, we show that CT satisfies these security properties under various assumptions on Merkle trees all of which reduce to collision resistance of the underlying hash function (and in one case with the additional assumption of unforgeable signatures).

We begin with a thorough analysis of Certificate Transparency and its properties in Section 3.2. In Sections 3.3 and 3.5 we present our contributions in brief and lay out the cryptographic foundation needed for the rest of the chapter. In Section 3.6 we specify the algorithms that constitute a logging scheme and its instantiation. We focus on security properties of Certificate Transparency in Sections 3.7 and 3.8 and we conclude the chapter with a discussion in Section 3.9. Much of the following has been reproduced from our work **Secure Logging Schemes and Certificate Transparency**, published in *Proceedings of the 21st European Symposium on Research in Computer Security* (ESORICS

2016) [DGHS16].
[1]

3.2 Certificate Transparency

Certificate Transparency [LKL13, Lau14, Rea13] is an experimental protocol originally proposed by Google and standardised by the Internet Engineering Task Force (IETF). The main idea behind Certificate Transparency is a public, verifiable, append-only log which mitigates the threat of fraudulently issued certificates. The open auditing and monitoring system in CT allows domain owners to examine and verify the authenticity of certificates issued for their domains. Additionally it provides assurance that no certificate has been issued fraudulently for their domain. The end goal of CT is that web clients would only accept digital certificates that are publicly logged and that a CA should not be able to issue a certificate for any domain without it being publicly logged and visible. Web clients benefit from the scheme by receiving log-authenticated "promise" values that can be used to determine if a given log contains the paired certificate, providing confidence to the client that the certificate (if fraudulent) will be audited and revoked if necessary.

CT includes several types of entities which perform different tasks.

- *Loggers* or *log servers* maintain publicly accessible logs of the certificates. A logger stores all the entries of certificates in an append-only Merkle hash tree [Mer79]

- *Submitters* submit certificates to a log server. Domain owners or CAs are expected to be submitters and they receive a promise to log the certificate within a certain amount of time upon submitting to a log; the promise is called a *signed certificate timestamp (SCT).*

- *Monitors* are public or private services that observe the logs for any misbehaving activities or suspicious certificates. They keep track of all the new entries, keep copies of the entire log and verify the consistency between published revisions of the log.

[1] Some sections of the paper (Merkle tree security properties and proofs) are not included in this chapter as they are the work of our co-authors.

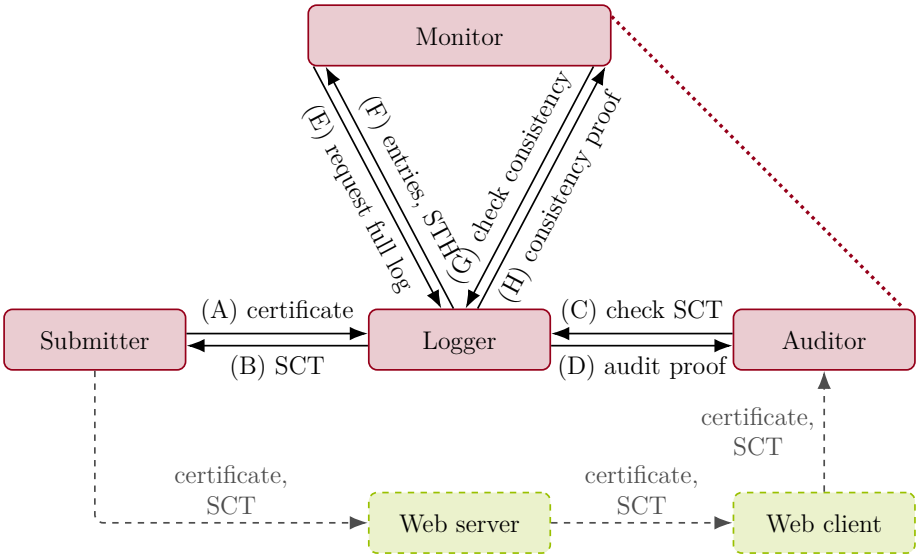


Figure 3.1: Interaction between entities in Certificate Transparency

- *Auditors* are also involved in the verification procedure. They make sure the certificates that are promised to log are in fact included to the log.

In CT, the original entities from the web PKI also have some additional tasks:

- CAs should act as submitters above.
- Web servers should include their SCT along with the certificate when communicating with clients. Web servers may choose to submit their certificate to a log server if their CA does not do so for them.
- Web clients, upon receiving an SCT from a web server, after a certain time period has elapsed, may choose to verify that the log named in the SCT actually has publicly logged the certificate (thereby taking on the role of an auditor as above).
- Browser vendors may push updates that remove CAs or revoke certificates based on claims from monitors and web servers about misbehaving CAs.

Figure 3.1 provides an overview of the interaction between entities in CT. The labelling in the figure does not indicate a particular order of requests. Upon the submission of a new entry (step A), the logger returns a signed certificate timestamp (SCT) (step B). The SCT is considered as a promise by a logger to log the certificate within a certain time period called a *maximum merge delay*

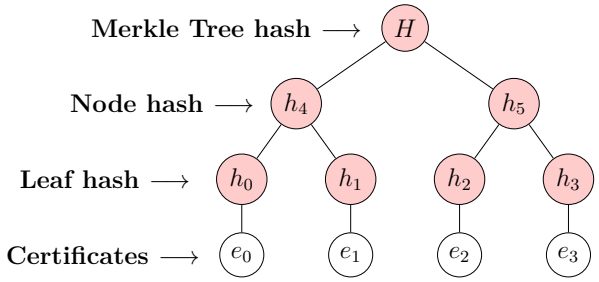


Figure 3.2: Merkle hash tree; a simple binary tree consisting of hashed leaves and nodes

(MMD). MMD is a parameter that is published by every log and indicates the maximum time period between issuing a timestamp and when they promise to include the certificate into the log.

In CT, an append-only Merkle tree [Mer79, Mer90], is used to store the entries. Recalling from Section 2.4.1, data is placed at the leaves of a binary tree and each intermediate node is the hash of its two child nodes—the root of the trees acts as a fingerprint of all included data (see Figure 3.2). This special cryptographic mechanism facilitates efficient public auditing of certificates and logs. When the log server signs the Merkle root hash along with other information, it is known as *Signed Tree Head (STH)*. In CT, the STH represents all the data enclosed in the tree. These fingerprints are retrieved and observed by all entities in the system through a protocol called gossiping [NGR15], described in Section 3.2.1.

Regularly, the logger appends all of its newly gained certificates to the log within MMD period. Appending is done by creating a new separate tree with the most recently submitted set of certificates and then combining the new Merkle tree hash with old tree hash, generating a new Merkle tree hash. The new root hash is then signed to create a new STH and then published. The process continues over and over, growing the tree with newly added entries. Because of the way the tree is structured, it allows a logger to prove two things efficiently—consistent append-onliness of all certificates and the presence of a particular certificate. The two types of cryptographic proofs are called audit proof and consistency proof.

A *consistency proof* verifies that any two versions of the log are consistent. The proof allows an auditor or monitor to verify that the log is append-only, that any two versions of the log are consistent (steps G and H). In particular the log represented by a fingerprint at one point in time t_0 is a prefix of the log represented by a fingerprint at a later point in time $t_1 > t_0$. In CT, a consistency

proof is a subset of intermediate nodes in the Merkle tree needed to connect the two root hashes.

An *audit proof* verifies that a specific certificate has been included in a log. The proof shows that a particular certificate/SCT that a logger has promised to include is actually included in the log represented by a fingerprint, as shown in steps C and D. The Merkle audit proof is an authentication path, namely the missing node hashes required to compute all of the nodes between the leaf and the tree root. If it is possible to compute a root hash using the audit path that is similar to the root hash currently advertised for the log, then without a doubt one can accept that the particular leaf that is the certificate exists in the tree. Steps E and F show the request for the full set of entries represented by a fingerprint.

A recent incident proves the effectiveness of CT, by detecting fraudulent certificates. Google employees detected unrequested and unauthorised EV pre-certificates issued for two of their sub-domains by a Symantec intermediate CA Thawte [SE15]. These certificates were issued on September 14, 2015 and detected by September 17, 2015 via CT logs. They were revoked immediately by Google, limiting the damage.

3.2.1 Gossiping Protocol in CT

Certificate Transparency includes an additional mechanism called *gossiping* [NGR15], which allow monitors, auditors and web clients to share information they receive from log servers. The goal of this protocol is to collectively detect misbehaviour of log servers while limiting the damage to user privacy. The parties who hold the same fingerprint of a log are cryptographically assured that they have the same view of the tree as everyone else.

The specification describes three streams of communication between parties.

1. *SCT feedback*: web clients record the SCT values they receive from web servers during TLS handshake. Then on later visits (when client connects to a server for that particular domain), they replay these values to the domain owners that it supposedly received the SCT values in the first place. The underlying assumption here is that an MITM attacker sending fraudulent SCT values to a web client will eventually cease, and the web server will eventually receive an SCT value for past fraudulent certificates it did not

request. This also preserves the privacy of the web client, as it does not communicate SCT values to a third party monitor, thereby not informing third-parties of their browsing habits.

2. *STH pollination*: monitors, auditors and web clients exchange STH (fingerprints) values that they have seen for any logger. Basically a monitor will have been communicated many previous STH values for a given logger, and thus can perform consistency checks for any STH values that a logger has produced.

3. *Trusted auditor*: web clients record SCTs, certificates and STHs and communicate them with a "trusted" monitor. This is essentially a combination of the previous two streams, but with the drawback that it does not keep the browsing habits of the end-user private from the monitor, and thus web clients are advised to only set up a trusted auditor stream if it is reasonable from a privacy perspective.

In SCT feedback stream, it should be assured that the client does not send SCT values to any other server than one serving the domain, which is also referred in the certificate signed by that SCT. Failing to do so would lead to two types of privacy leaks. First is that the sever receiving irrelevant SCTs will learn about the other web sites visited by the client. Secondly, auditors and monitors receiving SCTs from the server will learn about other servers visited by its clients. If the client has configuration settings in place for not sending cookies to third parties then the SCT values must be treated as cookies. This prevents third party tracking through the use of SCTs. Out of the three gossiping streams, SCT feedback is the most privacy-preserving gossip mechanism, as it does not directly expose any links between an end user and the sites they've visited to any third party.

The overall goal of gossiping in CT is that interested parties such as monitors and auditors will eventually agree on their view of a log's behaviour. In our model, we assume that gossiping works as intended, and assume that monitors and auditors share all STH values among them.

3.2.2 Threat Model

The main goal of CT is to allow domain owners to learn when and if fraudulent certificates are issued for their domain. We could expect that to achieve this goal,

each and every CA should submit their certificates to log servers and that the log servers must be trusted. Though having CAs being honest and submitting certificates to CT would definitely improve the security, CT does not entirely depend on honest CAs or trusted log servers. Instead, to achieve the main goal of the CT framework, trust is decentralised among the various entities: loggers, monitors and auditors. These entities collectively watch each other for any suspicious activity. This approach reduces the requirement of a trusted third party. Although it has benefits, one major drawback is having to deal with the threat of one or more entities being malicious. For example, a malicious log server might try to present different views to other parties or could try to remove certificates at a later date. Conversely, a malicious monitor could accuse an honest logger of not having included a particular promised certificate with the goal of undermining trust in that log server and having browser vendors remove it.

Such potential threats are described in the informational IETF draft "Attack Model for Certificate Transparency" [Ken15, Ken18]. This draft contains potential attack scenarios when Certificate Transparency is used in the context of web public key infrastructure. The top level differentiation of this analysis is whether a mis-issued certificate has been issued by a malicious CA or not. Based on those two scenarios, it is further differentiated based on mainly whether the logs and monitors are malicious or benign. The certificate's subject, i.e. a domain owner, and browser operations are also considered in the analysis.

The IETF draft defines the term certificate mis-issuance to capture violations of both semantic and syntactic constraints. The most significant semantic constraint for a certificate is that it should be issued to the entity that genuinely represents the *subject* or *subject alternative* named in the certificate. Any certificate that does not adhere to this constraint is regarded as a mis-issued/fake/fraudulent certificate. A certificate is considered to be syntactically mis-issued when it breaches the syntax constraints related to the type of certificate it aims to represents.

Here we focus mainly on the threats in the draft which are associated with the various interacting parties in CT:

1. Misbehaving log server

   (L1) Creating an entry for a fake certificate

(L2) Presenting different log entry views to different entities

(L3) Not performing syntactic checks on certificate entries

(L4) Issuing SCTs for fake certificates

(L5) Reporting syntactic errors for a syntactically valid certificate

2. Misbehaving monitor

(M1) Not notifying the lack of SCT in the certificate received from CA

(M2) Not informing the targeted domain owner about fake certificates

(M3) Issuing false warnings to a targeted domain owner

(M4) Not reporting syntactic errors of certificates when noted

(M5) Reporting syntactic errors for a syntactically valid certificate

3. Malicious submitter/certificate authority

(S1) Failing to log a certificate

(S2) Issuing an erroneous certificate and causing the log to not perform checks

(S3) Refusing to revoke/delay revoking once mis-issuance is detected and reported

4. Web client

(C1) Not rejecting certificates that do not have SCTs

The IETF draft carefully captures each scenario under semantic and syntactic mis-issuance of certificates when the each entity in CT is being benign or malicious. The facts presented above are based on the instances where entities can maliciously behave and cause havoc in web PKI context.

We view CT as an instantiation of logging schemes in general and model the security properties that can be cryptographically assured in such logging schemes. We will not require log entries to be certificates nor assume a PKI. Thus, some of the threats above will not be considered. For example, some threats (L3, L5, M4, M5, and S2) relate to the validity or syntax of certificates, which we will exclude from our model of general-purpose logging schemes.

Threats L1, L2, and L4 concern a logger that misbehaves by logging a fake entry and try to cover up their misbehaviour by presenting different views to different

entities. Of these threats, we capture L2 directly as a collection of cryptographic security notions which prevent loggers from splitting the view presented to different parties, removing certificates, or falsely claiming the inclusion of entries. Indirectly, this also protects against threats L1 and L4 as any fake certificate included or promised for inclusion in Certificate Transparency will become known to the monitoring parties.

Threat M3 involves a malicious monitor framing an honest log for bad behaviour. We capture this in our model by showing how CT cryptographically prevents a monitor from falsely framing a logger for not including a promised certificate.

Threats M1 and M2 concern the failure of a monitor in notifying affected parties of misbehaviour. While an important threat, this is best captured via contractual obligations between monitors and other entities, rather than cryptographic means. It is similar for S3.

Threats S1 and C1 go hand-in-hand with each other and with the gossiping protocol. No theoretical measure can force a CA to log a certificate or force a client to reject something. However, if clients do reject certificates that lack an SCT, then submitters who violate S1 will find their certificates rejected by clients.

From this categorization of the threats against CT, it is evident that the primary threats that can be analysed cryptographically are L2 and M3. Therefore, our model of general-purpose logging schemes will focus on these threats and the entities involved in them, specifically loggers and monitors/auditors.

Threats change over time and new classes of adversaries may rise. Motivations and capabilities of adversaries may change. In this chapter of work we rely on Kent's threat model for Certificate Transparency [Ken15, Ken18] as it documents perceived threats against the system. This threat model provides a complete set of threats in web PKI context and helps in understanding the attacks against CT.

3.3 Our Contribution

Given the practical significance of Certificate Transparency, it is important to have a formal understanding of the security goals of CT and analyse whether CT achieves those goals. The objective of our work is to define security goals of logging schemes using the formalism of provable security, and attempt to prove that CT satisfies these security goals under suitable cryptographic assumptions.

Our model of logging schemes does not assume a PKI context, so we do not assume that log entries must have a particular syntax, and thus we leave the threats involving validity or syntax of log entries to existing analyses on certificate validity. Similarly, we omit consideration of threats where an entity *fails* to act.

As noted above, we will focus on two particular threats in the CT threat model: whether a misbehaving log server can present different views of the log and whether a misbehaving monitor can frame an honest log server for bad behaviour. Thus, our model will focus on two entities: the logger and the monitor/auditor.

Definition of logging schemes In Section 3.6, we formally define logging schemes, naming operations that each entity can perform. This model does not attach any semantic meaning to the entries being logged; in particular, we do not assume that log entries are certificates. Afterwards, we describe the operations of Certificate Transparency as a specific instantiation of the logging scheme framework.

Security definitions Next, we introduce cryptographic security properties for logging schemes in Section 3.7 that are inspired by the CT threat model but can also be applied logging schemes. More specifically, we treat two types of properties. First, we define security notions which concern a malicious logger:

- entry-coll: can a malicious logger present an audit proof that claims a particular entry corresponds to a fingerprint as well as a set of entries corresponding to the same fingerprint such that the particular entry is not actually in the list of entries?
- proof-coll: can a malicious logger present an audit proof that claims a single fingerprint represents both a particular entry as well as a set of entries such that the particular entry is not actually in the list of entries?
- entry-cons: can a malicious logger present two fingerprints connected by a valid consistency proof and two sets of entries such that the entries corresponding to the first fingerprint are not a prefix of the entries corresponding to the second fingerprint?
- entry-cons-multi: can a malicious logger present a sequence of fingerprints all connected by valid consistency proofs and a sequence of sets of entries

such that the entries corresponding to the first fingerprint are not a prefix of the entries corresponding to the last fingerprint?

Next, we define a security notion concerning a malicious monitor:

- promise-incl: can a malicious monitor frame an honest logger for not including a promised entry when it actually has?

Security of Certificate Transparency Finally, we analyse the security of Certificate Transparency in Section 3.8 and show that CT both prevents logger misbehaviour (i.e., CT satisfies the entry-coll, proof-coll, and entry-cons security properties) as well as protection from framing of honest loggers by misbehaving monitors (i.e., CT satisfies the promise-incl property.) All of these proofs are based on properties of Merkle tree hashing and audit/consistency proofs all of which are derived from the collision resistance of the hash function. The last property, promise-incl, also depends on the unforgeability of the signature scheme used by loggers.

3.4 Related Work

Transparency logs. In recent years a few more approaches have emerged around the concept of *transparency logs*. Because CT itself does not cover revocation of certificates, Laurie and Kasper [LK12] propose Revocation Transparency as a mechanism to handle these aspects. Ryan [Rya14] introduces *Enhanced Certificate Transparency* which is an alternative mechanism for handling revocation and has applied it specifically to end-to-end encrypted email. Apart from revocation, more advanced features such as limitations on certificate issuance, validation, and update have been incorporated in some proposals, such as Attack Resilient Public-Key Infrastructure (ARPKI) [BCK+14b] and Accountable Key Infrastructure (AKI) [KHP+13]. AKI aims to prevent compromised CAs from impersonating domains. AKI operates with three entities; a certification authority, Integrity log server (ILS) and validators. The certification authority authenticates domains and issues certificates. The ILS maintains an 'Integrity tree' that logs certificates and ILS operations along with misbehaviours such as sudden disappearances of certificates are monitored by validators along with CAs.

The AKI concept has a few similarities to Certificate Transparency. In CT, append only logs are used to make CA issued certificates publicly visible. Likewise,

AKI maintains an integrity tree that logs certificates. Similar to monitors and auditors in CT, AKI has validators who monitor integrity trees. Furthermore Merkle tree structure is used to implement both CT log server and AKI integrity tree. However, security details are not proven and AKI fails to prevent clients from accepting a compromised AKI certificate. The Electronic Frontier Foundation's *Sovereign Keys Project* [Ele] constitutes a system where certificates are cross-signed by sovereign keys to be considered valid. Sovereigns keys are then published in a semi-centralized, append-only data structure called "timeline servers" which differ from Certificate Transparency in particular by not using Merkle trees.

Melara et al. [MBB+15] present *CONIKS*, a system that builds on transparency logs using Merkle trees similar to Certificate Transparency, but in contrast focuses on transparency of *user* keys in end-to-end encryption/secure messaging scenarios. CONIKS eliminates the need for global third party monitors and aims at additional privacy properties for identity–key bindings, however without providing a formal security model or cryptographic proofs.

Other fields beyond PKI have also embraced notions of transparency logs. Secure Untrusted Data Repository *SUNDR* was introduced by Li et al. [LKMS04], which considers securing data against unauthorized user modification. Like CT, SUNDR is concerned with detection as opposed to prevention of attacks. It ensures users have the same view of the modification history when the data is stored on an untrusted server. In addition, SUNDR achieves so-called *fork consistency* against untrusted servers, a similar property to our notions of entry-coll, which allow users to detect differences in user views of modification history if they can communicate between themselves. These examples provide evidence of the potential of our security model to be used in analysis of protocols other than CT.

3.5 Cryptographic Notation

In this section we review cryptographic building blocks involved in Certificate Transparency.

Notation. We denote by $\vec{E}$ an ordered list of entries, where () denotes the empty list. Indexing is 0-based: $\vec{E} = (e_0, \ldots, e_{n-1})$, and we write $\vec{E}[i]$ to denote e_i and $\vec{E}[i : j]$ to denote the sublist $(e_i, \ldots, e_{j-1})$. We adopt the convention that $\vec{E}[-1] = ()$. We write $e \in \vec{E}$ to indicate that an entry e is contained in the list $\vec{E}$. We let $\vec{E} \| \vec{E}'$ denote the concatenation of two entry lists and write

$\vec{E} \prec \vec{E}'$ if $\vec{E}$ is a prefix of $\vec{E}'$. If we define $P \leftarrow (t, e, \sigma)$, then we can later access fields of P using "object-oriented" notation: $P.t$, $P.e$, $P.\sigma$. Moreover, if $\vec{P}$ is a list $(P_0, \ldots, P_{n-1})$, then the notation $\vec{P}.e$ means the list $(P_0.e, \ldots, P_{n-1}.e)$. The expression $k \leftarrow 2^{\lceil \log_2(n/2) \rceil}$ corresponds to setting k to be the largest power of two less than n, i.e., $\frac{n}{2} \leq k = 2^i < n$.

We formally defined the signature scheme unforgeability and security experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}$ for *existential unforgeability under chosen-message attacks* in Section 2.1.2 Figure 2.1.

As mentioned in Section 2.4.1, the Merkle tree is a hash based data structure which can be used to authenticate large amounts of data. Let $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function. A list $\vec{E}$ is stored in the tree such that, the entries of $\vec{E}$ are placed at the leaves of the Merkle tree and each non-leaf or intermediate node is the hash of its two child nodes. A fingerprint is the output of the algorithm $\mathtt{MTH}_{\mathrm{H}}(\vec{E})$ in Figure 2.6 in Section 2.4.1.

The consistency proof generation algorithm $\mathtt{ConsProof}_{\mathrm{H}}(m, n, \vec{E})$ and verification algorithm $\mathtt{CheckConsProof}_{\mathrm{H}}(n_0, H_0, n_1, H_1, \vec{C})$ are shown in Figure 2.6 in Section 2.4.1. We have reformulated these from how they appear in the RFC [LKL13]: ours use a top-down recursive approach, whereas the RFC versions are bottom-up looping algorithms; the two are equivalent, but our versions are more helpful in proving our theorems.

In Section 2.4.1 we have formally defined the Merkle tree hash system algorithms and proved the collision resistance properties of Merkle tree hashing.

3.6 Logging Schemes

Now we specify the algorithms that comprise a logging scheme and formulate CT as a logging scheme. Our definition of a logging scheme is based around functionalities of Certificate Transparency. However, our model of the logging scheme has the potential to be used for general purposes of recording objects. To achieve it, we use non-CT specific language such as "fingerprint" instead of the CT-specific "signed tree head". Certificate Transparency is about recording digital certificates, but our logging scheme is not only for certificates—any type of object can be logged.

In this context of work, our logging scheme is intended to be an abstract model for CT. However, it should be noted our security model does not completely

model CT functionalities. We only capture the properties that can be assured cryptographically, that is our model represents important aspects for our defined security properties. As mentioned earlier, the IETF attack model for Certificate Transparency, captures both semantic and syntactic scenarios for potential threats. Our security model excludes all syntactic checks and further it apprehends and models semantic constraints that can be proven via cryptographic measures.

3.6.1 Definition of Logging Schemes

Definition 6 (Logging Scheme)**.** *A* logging scheme LS *consists of the following algorithms, some of which are run by a logger and some of which are run by a monitor/auditor.*

The following algorithm is used by a logger to initialise its log:

- KeyGen() $\xrightarrow{\$}$ (st, pk, sk)*: A probabilistic algorithm that returns a state* st *and a public-key/secret-key pair* (pk, sk)*.*

The following algorithms are used by a logger to add entries to its log, using a two-step process of promising to add an entry to the log and then a batch update actually adding the entries:

- PromiseEntry$(e, t, sk) \xrightarrow{\$} P$*: A probabilistic algorithm that takes as input a log entry* e*, a time* t*, and the secret-key* sk *and outputs a promise* P*; the promise contains the entry and time as sub-fields* $P.e$ *and* $P.t$*.*

- UpdateLog$(st, \vec{P}, t, sk) \xrightarrow{\$} (st', F)$*: A probabilistic algorithm that takes as input a state* st*, a potentially empty ordered list of promises* $\vec{P}$ *to add to the log, a time* t *and the secret-key* sk *and returns an updated state* st' *and a fingerprint* F *(where the latter includes the indicated time, denoted as* $F.t$*)*

The following algorithms are used by a logger to demonstrate various properties to monitors/auditors:

- PresentEntries$(st, F) \rightarrow \vec{E}$ *or* $\perp$*: A deterministic algorithm that takes as input a state* st *and a fingerprint* F *and outputs an ordered list of log entries* $\vec{E}$*, or an error symbol* $\perp$*.*

- ProveMembership$(st, e, F) \xrightarrow{\$} \vec{M}$ *or* $\perp$*: A probabilistic algorithm*[2] *that*

[2]In CT, ProveMembership and ProveConsistency are deterministic, though in principle these could be probabilistic in a logging scheme.

takes as input a state st, a log entry e, and a fingerprint F and outputs a membership proof $\vec{M}$, or an error symbol $\perp$.

- ProveConsistency$(st, F_0, F_1) \xrightarrow{\$} \vec{C}$ *or* $\perp$*: A probabilistic algorithm*[2] *that takes as input a state st and two fingerprints F_0 and F_1 and outputs a consistency proof $\vec{C}$, or an error symbol $\perp$.*

The following algorithms are used by monitors/auditors to check a log:

- CheckPromise$(P, pk) \to \{0,1\}$*: A deterministic algorithm that takes as input a promise P (which includes an entry $P.e$) and a public-key pk and outputs a bit $b \in \{0,1\}$.*
- CheckFingerprint$(F, pk) \to \{0,1\}$*: A deterministic algorithm that takes as input a fingerprint F and a public-key pk and outputs a bit $b \in \{0,1\}$.*
- CheckEntries$(\vec{E}, F, pk) \to \{0,1\}$*: A deterministic algorithm that takes as input an ordered list of log entries $\vec{E}$, a fingerprint F, and a public-key pk and outputs a bit $b \in \{0,1\}$.*
- CheckMembership$(F, e, \vec{M}, pk) \to \{0,1\}$*: A deterministic algorithm that takes as input a fingerprint F, an entry e, a membership proof $\vec{M}$, and a public-key pk and outputs a bit $b \in \{0,1\}$.*
- CheckConsistency$(F_0, F_1, \vec{C}, pk) \to \{0,1\}$*: A deterministic algorithm that takes as input two fingerprints F_0 and F_1, a consistency proof $\vec{C}$, and a public-key pk and outputs a bit $b \in \{0,1\}$.*

Definition 7 (Correctness of a Logging Scheme)**.** *We say that a logging scheme* LS *is* correct *if for all $(st_0, pk, sk) \xleftarrow{\$}$ KeyGen(), all $n \in \mathbb{N}$, all $m_1, \ldots, m_n \in \mathbb{N}$, all ordered lists of entries $\vec{E}_1 = (e_{1,1}, \ldots, e_{1,m_1}), \ldots, \vec{E}_n = (e_{n,1}, \ldots, e_{n,m_n})$, all ordered lists of promised entries $\vec{P}_1 = (P_{1,1}, \ldots, P_{1,m_1}), \ldots, \vec{P}_n = (P_{n,1}, \ldots, P_{n,m_n})$, all timestamps $t_1 \leq \cdots \leq t_n$, all states $st_1, \ldots, st_n$, and all fingerprints $F_1, \ldots, F_n$ such that $P_{i,j} \xleftarrow{\$}$ PromiseEntry$(e_{i,j}, t_{i,j}, sk)$ for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m_i\}$, and any timestamps $t_{i,j}$ and $(st_1, F_1) \xleftarrow{\$}$ UpdateLog$(st_0, \vec{P}_1, t_1, sk), \ldots, (st_n, F_n) \xleftarrow{\$}$ UpdateLog$(st_{n-1}, \vec{P}_n, t_n, sk)$ the following holds:*

1. PresentEntries$(st_n, F_n) = \vec{E}_1 \| \ldots \| \vec{E}_n$.
2. CheckEntries$(\vec{E}_1 \| \ldots \| \vec{E}_n, F_n, pk) = 1$.

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{KeyGen}() \to (st, pk, sk):}$
1: $\vec{E} \leftarrow ()$
2: $st \leftarrow (\vec{E})$
3: $(pk, sk) \xleftarrow{\$} \mathtt{SIG}.\mathrm{KeyGen}()$
4: **return** (st, pk, sk)

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{PromiseEntry}(e, t, sk) \to P:}$
1: $\sigma \leftarrow \mathtt{SIG}.\mathrm{Sign}_{sk}(t \| e)$
2: **return** $P \leftarrow (t, e, \sigma)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{UpdateLog}(st, \vec{P}, t, sk) \to (st', F):}$
1: **for each** $P \in \vec{P}$ **do**
2: **if** $\mathrm{CheckPromise}(P, pk) = 0$, **return** $(st, \perp)$
3: $st.\vec{E} \leftarrow st.\vec{E} \| \vec{P}.e$
4: $n \leftarrow |st.\vec{E}|$
5: $H \leftarrow \mathtt{MTH}_{\mathrm{H}}(st.\vec{E})$
6: $\sigma \leftarrow \mathtt{SIG}.\mathrm{Sign}_{sk}(t \| n \| H)$
7: **return** $F \leftarrow (t, n, H, \sigma)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{PresentEntries}(st, F) \to \vec{E}:}$
1: **if** $\mathrm{CheckFingerprint}(F, pk) = 0$, **return** $\perp$
2: **return** $st.\vec{E}[0 : F.n]$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{ProveMembership}(st, e, F) \to \vec{M}:}$
1: **if** $\mathrm{CheckFingerprint}(F, pk) = 0$, **return** $\perp$
2: **find** $m < F.n$ such that $e = st.\vec{E}[m]$
3: **if** no such m exists, **return** $\perp$
4: $\vec{A} \leftarrow \mathtt{Path}_{\mathrm{H}}(m, \vec{E}[0 : F.n])$
5: **return** $\vec{M} \leftarrow (\vec{A}, m)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{ProveConsistency}(st, F_0, F_1) \to C:}$
1: **if** $\mathrm{CheckFingerprint}(F_0, pk) = 0$, **return** $\perp$
2: **if** $\mathrm{CheckFingerprint}(F_1, pk) = 0$, **return** $\perp$
3: **return** $\vec{C} \leftarrow \mathtt{ConsProof}_{\mathrm{H}}(F_0.n, F_1.n, st.\vec{E})$

Figure 3.3: Certificate Transparency: algorithms run by loggers.

3. $\mathrm{CheckFingerprint}(F_i, pk) = 1$ *for all* $1 \leq i \leq n$.

4. $\mathrm{CheckConsistency}(F_i, F_j, \vec{C}, pk) = 1$ *for all* $1 \leq i < j \leq n$ *and* $\vec{C} \xleftarrow{\$} \mathrm{ProveConsistency}(st_n, F_i, F_j)$.

5. $\mathrm{CheckMembership}(F_n, e, \vec{M}, pk) = 1$ *for all* $e \in \vec{E}_1 \| \ldots \| \vec{E}_n$ *and* $\vec{M} \xleftarrow{\$} \mathrm{ProveMembership}(st_n, e, F_n)$.

6. $\mathrm{CheckPromise}(e_{i,j}, P_{i,j}, pk) = 1$ *for all* $i \in \{1, \ldots, n\}$ *and* $j \in \{1, \ldots, m_i\}$.

3.6.2 Instantiation of Certificate Transparency as a Logging Scheme

Following our Definition 6, we formulate Certificate Transparency using H and SIG as a logging scheme $\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}$ as shown in Figures 3.3 and 3.4. A log entry in CT is a chain of X.509 certificates: the certificate (or partially completed pre-certificate) itself, and each intermediate CA's certificate leading to the root CA's cert. We treat entries in our formalisation of logging schemes as opaque bit strings: our formulation hence omits any syntactical checks for the entries it manages. Doing these syntactical checks as described in threat model is independent of the logging properties. The promise P is called a *signed certificate timestamp* (SCT). The fingerprint F is called the *signed tree head* (STH).

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{CheckPromise}(P, pk) \to b:}$
1: **return** $\mathtt{SIG}.\mathrm{Vfy}_{pk}(P.t \| P.e, P.\sigma)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{CheckFingerprint}(F, pk) \to b:}$
1: **return** $\mathtt{SIG}.\mathrm{Vfy}_{pk}(F.t \| F.n \| F.H, F.\sigma)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{CheckEntries}(\vec{E}, F, pk) \to b:}$
1: **if** $\mathrm{CheckFingerprint}(F, pk) = 0$, **return** 0
2: $H' \leftarrow \mathtt{MTH}_{\mathrm{H}}(\vec{E})$
3: **return** $(|\vec{E}| = F.n) \wedge (H' = F.H)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{CheckMembership}(F, e, \vec{M}, pk) \to b:}$
1: **if** $\mathrm{CheckFingerprint}(F, pk) = 0$, **return** 0
2: **return** $\mathtt{CheckPath}_{\mathrm{H}}(e, F.H, F.n, \vec{M}.\vec{A}, \vec{M}.m)$

$\underline{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}.\mathrm{CheckConsistency}(F_0, F_1, \vec{C}, pk) \to b:}$
1: **if** $\mathrm{CheckFingerprint}(F_0, pk) = 0$, **return** 0
2: **if** $\mathrm{CheckFingerprint}(F_1, pk) = 0$, **return** 0
3: **return** $\mathtt{CheckConsProof}(F_0.n, F_0.H, F_1.n, F_1.H, \vec{C})$

Figure 3.4: Certificate Transparency: algorithms run by monitors/auditors.

$\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}$ employs a cryptographic hash function $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a signature scheme SIG which we formally defined in Section 3.5. The hash function H is used to store the entries in a Merkle hash tree. The root node represents the entries included in the tree at a certain time and is used in within the signed tree head. Certificate Transparency utilises authentication paths to prove the existence of a certain entry in the tree and consistency proofs to connect subsequently published root nodes.

A log entry in CT is a chain of X.509 certificates: the certificate itself, and each intermediate CA's certificate leading to the root CA's cert. The initial certificate may be a pre-certificate, which is a partial X.509 data structure which has not yet been signed. As already mentioned, we treat entries in our formalisation of logging schemes as opaque bit strings.

Now we describe the algorithms used to formulate Certificate Transparency as a logging scheme.

KeyGen is used by a logger to initialise the scheme. In CT, the logger generates a signature public/secret-key pair (pk, sk) and initialises the state st with an empty list of entries $\vec{E}$ contained in the log.

PromiseEntry is used by a logger to output a promise to log a particular certificate/pre-certificate chain as requested by a submitter. In CT, the promise P is called a *signed certificate timestamp* (SCT), and consists of the current timestamp t, the certificate/pre-certificate chain e, and the logger's signature σ over these values.

UpdateLog is used by a logger to incorporate previously promised entries into the log and output a new fingerprint of the log. In CT, the fingerprint F is called the *signed tree head* (STH), and consists of a timestamp t, the number of entries n currently in the log, the root H of the Merkle hash tree of all the entries $\vec{E}$ in the log, and the logger's signature σ over these values.

PresentEntries is used by a logger to output all entries associated with a given fingerprint. In CT, the signed tree head (fingerprint) includes the number n of entries comprising that fingerprint, so this algorithm simply outputs the first n entries of the log.

ProveMembership is used by a logger to prove that a particular entry is in the log. In CT, the logger constructs an authentication path in the Merkle tree from the leaf node containing the entry to the root of the tree as described in the signed tree head (note that the tree may have subsequently grown since this particular STH was issued).

ProveConsistency is used by a logger to output a proof of consistency between two fingerprints. In CT, this uses the algorithm `ConsProof` (cf. Section 2.4.1) to construct a consistency proof between the roots in two signed tree heads: the proof consists of the intermediate nodes required to construct both roots simultaneously.

CheckPromise is used by monitors/auditors to confirm that a promise was indeed issued by a specific logger. In CT, this is done by verifying the signature of the signed certificate timestamp using the logger's public-key.

CheckFingerprint is used by monitors/auditors to confirm that a fingerprint was indeed issued by a specific logger. In CT, this is done by verifying the signature of the signed tree head using the logger's public-key.

CheckEntries is used by monitors/auditors to confirm that a list of entries corresponds to a given fingerprint. In CT, this is done by reconstructing the Merkle tree hash of the list of entries and comparing that with the root in the signed tree head.

CheckMembership is used by monitors/auditors to confirm that a membership proof indeed proves that an entry is in the log. In CT, this is done by using the provided authentication path to construct an alleged Merkle tree root, and comparing this with the root in the signed tree head.

CheckConsistency is used by monitors/auditors to confirm the "append-onliness" of the log has been adhered to – that the list of entries represented by one fingerprint is a prefix of the list of entries represented by another fingerprint. In CT, this is done by using the `CheckConsProof` algorithm to verify a consistency proof between the root values in two signed tree heads by reconstructing both roots from intermediate nodes.

3.7 Security Goals

In Section 3.2.2, we have summarised the threats against CT in the context of the web PKI as considered in the informational IETF draft [Ken15] which defines potential threats for Certificate Transparency in web public key infrastructure. Threats are considered regarding each of the interacting entities: malicious log servers, malicious monitors, malicious CAs, and malicious web clients. We then clustered the threats into three categories: those involving validity or syntax of log entries, those involving failures to notify an affected party of misbehaviour, and those involving attempts to present false information. For example, a malicious log server might try to present different views of its log to different parties, claiming inclusion of a certificate to some parties (e.g., a web client) while trying to present a log that excludes that certificate to other parties (e.g., monitors and domain owners). A malicious logger might also try to remove logged certificates at a later point in time. Our model of logging schemes does not assume a PKI context, so we do not assume that log entries must have a particular syntax, and thus we leave the first class of threats to existing analyses on certificate validity. Likewise, we do not consider threats where an entity fails to act upon.

Thus, in our model we focus on threats where an entity attempts to present false information. There are two perspectives: threat (L2), which is that a malicious logger may attempt to present different views of the log to different entities, and threat (M3), where a malicious monitor may attempt to issue false warnings to a domain owner about fake certificates, thereby framing an honest logger or certificate authority for dishonest behaviour.

For the security properties of logging schemes that can be proved cryptographically, our security definitions follow a provable security game-based approach. We consider three properties involving security against a malicious logger, in which the experiment acts as an honest monitor/auditor which the logger is trying to fool. We also consider one security property involving security against a malicious monitor/auditor, in which the experiment acts as an honest logger which the monitor/auditor is trying to frame for bad behaviour.

3.7.1 Security Against a Malicious Logger

Since the fingerprint is used to concisely represent the contents of the log, the first two cryptographic security properties against a malicious logger, shown in

$\underline{\mathrm{Exp}_{\mathrm{LS}}^{\mathsf{entry\text{-}coll}}(\mathcal{A})}$:
1: $(\vec{E}_0, \vec{E}_1, F, pk) \xleftarrow{\$} \mathcal{A}()$
2: **return** 1 iff $(\mathrm{CheckEntries}(\vec{E}_0, F, pk) = 1) \ \wedge \ (\mathrm{CheckEntries}(\vec{E}_1, F, pk) = 1) \ \wedge \ (\vec{E}_0 \neq \vec{E}_1)$

$\underline{\mathrm{Exp}_{\mathrm{LS}}^{\mathsf{proof\text{-}coll}}(\mathcal{A})}$:
1: $(e, \vec{E}, F, \vec{M}, pk) \xleftarrow{\$} \mathcal{A}()$
2: **return** 1 iff $(\mathrm{CheckEntries}(\vec{E}, F, pk) = 1) \ \wedge \ (\mathrm{CheckMembership}(e, F, \vec{M}, pk) = 1) \ \wedge \ (e \notin \vec{E})$

$\underline{\mathrm{Exp}_{\mathrm{LS}}^{\mathsf{entry\text{-}cons}}(\mathcal{A})}$:
1: $(\vec{E}_0, \vec{E}_1, F_0, F_1, \vec{C}, pk) \xleftarrow{\$} \mathcal{A}()$
2: **return** 1 iff $(\mathrm{CheckConsistency}(F_0, F_1, \vec{C}, pk) = 1) \ \wedge \ (\mathrm{CheckEntries}(\vec{E}_0, F_0, pk) = 1)$
$\wedge \ (\mathrm{CheckEntries}(\vec{E}_1, F_1, pk) = 1) \ \wedge \ (\vec{E}_0 \not\prec \vec{E}_1)$

Figure 3.5: Security properties of a logging scheme LS against a malicious logger.

Figure 3.5, concern the ability of the logger to make the fingerprint represent different, conflicting information. *Collision resistance of entries*, defined in the experiment entry-coll, requires that it is hard for a malicious logger to come up with a single fingerprint representing two different sets of entries. *Collision resistance of proofs*, formalised in the experiment proof-coll, is about the difficulty for a malicious logger to create a proof that an entry is represented by a fingerprint while simultaneously claiming that the set of entries represented by that fingerprint does not include that particular entry. A scheme that satisfies both of these ensures that a malicious logger cannot make parties who use the same fingerprint believe different things about the log entries represented by that fingerprint.

Logs are updated over time, but are meant to be append-only. However, since logs are only represented by fingerprints, consistency proofs are used to connect two fingerprints and are meant to prove that the set of entries represented by one fingerprint is a subset of the set of entries represented by a second fingerprint—in other words, that the fingerprints are representative of an append-only log. The final security property in Figure 3.5 captures the *consistency of entries*, i.e., the difficulty for a malicious logger to remove an entry from a log. The experiment entry-cons is concerned with two fingerprints connected by a single consistency proof.

The "multi-hop" version for the security property *consistency of entries*, is shown in the experiment entry-cons-multi in Figure 3.6. This concerns a chain of fingerprints connected by consistency proofs. It is evident that "multi-hop" property follows directly from "single-hop" version. The chain of fingerprints connected by consistency proofs show that each set of entries represented by a certain fingerprint F_i is a subset of the set of entries represented by fingerprint

$\underline{\text{Exp}_{\text{LS}}^{\text{entry-cons-multi}}(\mathcal{A})}$:
1: $(\vec{E}_0, \ldots, \vec{E}_n, F_n, \vec{C}_1, \vec{C}_n pk) \xleftarrow{\$} \mathcal{A}()$
2: **return** 1 iff $\bigwedge_{i=0}^{n-1}$ (CheckConsistency$(F_i, F_{i+1}, \vec{C}_i, pk) = 1) \ \wedge \ \bigwedge_{i=0}^{n}$(CheckEntries$(\vec{E}_i, F_i, pk) = 1)$
$\wedge \ (\vec{E}_i \not\preceq \vec{E}_{i+1})$ for some $0 \ \leq \ i \leq \ n$

Figure 3.6: "Multi-hop" version of the security property *Consistency of entries* against a malicious logger.

F_{i+1} in chain.

3.7.2 Security Against a Malicious Monitor/Auditor

Most of the security properties we discuss here are cryptographic, meaning that under some computational assumptions it is not possible for a malicious logger to perform certain actions. However, there are some security goals of CT that are not cryptographic. For example, cryptographic techniques cannot prevent a log that chooses not to include an entry it has promised to do so. Should a log issue a fingerprint after the time by which it has promised to log an entry but the log does not contain an entry, that constitutes evidence of the log's misbehaviour.

Another aspect with threats regarding entities is framing honest parties for malicious activities. It should not be possible to set up an honest logger for misbehaviour that did not actually happen. This particular security property is formalised as *inclusion of promises* in experiment promise-incl in Figure 3.7. Here the experiment plays the role of an honest logger against a malicious monitor/auditor. The experiment allows the adversary (the malicious monitor/auditor) to interact with experiment oracles that carry out the actions of an honest log, such as adding entries or proving membership. The experiment includes a global time which advances at the adversary's command, and is parameterised by a *maximum merge delay* $\mathsf{MMD} > 0$, within which an honest log is expected to include a promised entry. The list $\vec{E}_{promised}$ tracks entries that the log has promised to include; in calls to OTick the experiment (acting as the honest log) automatically adds the list of promised entries by the end of the maximum merge delay window.

3.8 Security of Certificate Transparency

In this section we prove the security properties we have declared on Certificate Transparency. We show that CT's instantiation $\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}$ within our logging scheme

$\underline{\mathrm{Exp}^{\mathsf{promise\text{-}incl}}_{\mathrm{LS,MMD}}(\mathcal{A})}$:
1: $T \leftarrow 0$
2: $\vec{E}_{promised} \leftarrow ()$
3: $(st, pk, sk) \xleftarrow{\$} \mathrm{KeyGen}()$
4: $(F, P, \vec{E}) \xleftarrow{\$} \mathcal{A}^{\mathsf{OTick},\mathsf{OPromiseEntry},\mathsf{OUpdateLog},\mathsf{OProveConsistency},\mathsf{OProveMembership}}(pk)$
5: **return** 1 iff $(\mathrm{CheckFingerprint}(F, pk) = 1) \ \wedge \ (\mathrm{CheckPromise}(P.e, P, pk) = 1)$
$\wedge \ (\mathrm{CheckEntries}(\vec{E}, F, pk) = 1) \wedge \ (P.e \notin \vec{E}) \ \wedge \ (P.t + \mathsf{MMD} \leq F.t)$

$\underline{\mathsf{OTick}()}$:
1: $T \leftarrow T + 1$
2: $\vec{P} \leftarrow \{P \in \vec{E}_{promised} : P.t + \mathsf{MMD} \geq T\}$
3: **if** $\vec{P} \neq ()$,
4: $\quad F \xleftarrow{\$} \mathsf{OUpdateLog}(\vec{P})$
5: $\quad \vec{E}_{promised} \leftarrow \vec{E}_{promised} \setminus \vec{P}$
6: $\quad$ **return** (T, F)
7: **else return** T

$\underline{\mathsf{OPromiseEntry}(e)}$:
1: $(st, P) \xleftarrow{\$} \mathrm{PromiseEntry}(st, e, T, sk)$
2: $\vec{E}_{promised} \leftarrow \vec{E}_{promised} \ || \ \{P\}$
3: **return** P

$\underline{\mathsf{OUpdateLog}(\vec{P})}$:
1: $(st, F) \xleftarrow{\$} \mathrm{UpdateLog}(st, \vec{P}, T, sk)$
2: **return** F

$\underline{\mathsf{OProveConsistency}(F_0, F_1)}$:
1: $(st, \vec{C}) \xleftarrow{\$} \mathrm{ProveConsistency}(st, F_0, F_1)$
2: **return** $\vec{C}$

$\underline{\mathsf{OProveMembership}(e, F)}$:
1: $(st, \vec{M}) \xleftarrow{\$} \mathrm{ProveMembership}(st, e, F)$
2: **return** $\vec{M}$

Figure 3.7: Security properties of a logging scheme LS against a malicious monitor/auditor framing a log for failing to include a promised entry.

frameworks guarantees collision resistance of entities and proofs, consistency of entries, and inclusion of promises.

It is evident that Theorems 1 and 2 below connect rather immediately with the security properties of the underlying Merkle tree hash. Lemmas 1 and 2 then connect the Merkle tree hash properties to finding a collision in H, which is infeasible if H is collision-resistant. CT relies upon the collision resistance of Merkle trees and does not employ some other properties hash functions provide; for instance, second-preimage resistance [DOTV08] or pseudorandomness [BDE+11] by XORing (pseudo)random values into intermediate node computations. Thus we also depend on collision resistance of Merkle tree hashing to produce our security results.

In Theorem 1 we prove that it is hard for a malicious logger to produce a single fingerprint which represent two contrasting sets of entries.

Theorem 1 (Collision resistance of entries)**.** *If hash function* H *is collision-resistant, then, in Certificate Transparency (with hash function* H*), no malicious logger can present different log entries for the same fingerprint. More precisely, if* $\mathcal{A}$ *wins* $\mathrm{Exp}^{\mathsf{entry\text{-}coll}}_{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}}$*, then algorithm* $\mathcal{B}^{\mathcal{A}}$*, which runs* $\mathcal{A}$ *and then returns the first two components of* $\mathcal{A}$*'s output, finds a collision in* $\mathtt{MTH}_{\mathrm{H}}$*. Moreover, the runtime of* $\mathcal{B}^{\mathcal{A}}$ *is the same as that of* $\mathcal{A}$*.*

By Lemma 1, a collision in $\texttt{MTH}_\text{H}$ leads to a collision in H, which is infeasible if H is collision-resistant.

Proof. We show that a successful adversary $\mathcal{A}$ effectively outputs two colliding entry sets under the same root of the Merkle tree used in Certificate Transparency, which, by Lemma 1, leads to a hash collision.

Suppose $\mathcal{A}$ wins $\text{Exp}^{\text{entry-coll}}_{\texttt{CT}_{\text{H,SIG}}}$. Then $\mathcal{A}$ will output $(\vec{E}_0, \vec{E}_1, F, pk)$ such that

$$\text{CheckEntries}(\vec{E}_0, F, pk) = \text{CheckEntries}(\vec{E}_1, F, pk) = 1$$

but $\vec{E}_0 \neq \vec{E}_1$. Based on the definition of CheckEntries for $\texttt{CT}_{\text{H,SIG}}$ (cf. Section 3.6.2), this implies that $\texttt{MTH}_\text{H}(\vec{E}_0) = \texttt{MTH}_\text{H}(\vec{E}_1)$ but $\vec{E}_0 \neq \vec{E}_1$. This is immediately a collision in $\texttt{MTH}_\text{H}$, which is what $\mathcal{B}$ outputs.

□

In Theorem 2 we prove that it is hard for a malicious logger to produce an audit proof that represents an entry by a fingerprint while simultaneously claiming that the set of entries represented by that fingerprint does not include that particular entry.

Theorem 2 (Collision resistance of proofs)**.** *If hash function* H *is collision-resistant then, in Certificate Transparency (with hash function* H*) no malicious logger can present a list of log entries under some fingerprint and a membership proof under the same fingerprint for an entry not contained in this list. More precisely, if* $\mathcal{A}$ *wins* $\text{Exp}^{\text{proof-coll}}_{\texttt{CT}_{\text{H,SIG}}}$ *by outputting* $(e, \vec{E}, F, \vec{M}, pk)$*, then algorithm* $\mathcal{B}^{\mathcal{A}}$*, which runs* $\mathcal{A}$ *and then returns* $(e, \vec{E}, \vec{M}.\vec{A}, \vec{M}.m)$*, breaks authentication path consistency in the sense of Lemma 2. Moreover, the runtime of* $\mathcal{B}^{\mathcal{A}}$ *is the same as that of* $\mathcal{A}$*.*

By Lemma 2, a break of authentication path consistency in $\texttt{MTH}_\text{H}$ leads to a collision in H, which is infeasible if H is collision-resistant.

Proof. Suppose $\mathcal{A}$ wins $\text{Exp}^{\text{proof-coll}}_{\texttt{CT}_{\text{H,SIG}}}$ by outputting $(e, \vec{E}, F, \vec{M}, pk)$ such that $\text{CheckEntries}(\vec{E}, F, pk) = 1$ and $\text{CheckMembership}(F, e, \vec{M}, pk) = 1$, but $e \notin \vec{E}$. Based on the definition of CheckEntries and CheckMembership for $\texttt{CT}_{\text{H,SIG}}$ (cf. Section 3.6.2), this implies that $|\vec{E}| = F.n$, $\texttt{MTH}_\text{H}(\vec{E}) = F.H$, and $\texttt{CheckPath}_\text{H}(e, F.H, F.n, \vec{M}.\vec{A}, \vec{M}.m) = 1$. Outputting the values $(e, \vec{E}, \vec{M}.\vec{A}, \vec{M}.m)$ breaks authentication path consistency in the sense of Lemma 2.

□

In Theorem 3 we prove that it is hard for a malicious logger to remove an entry once it is logged in the tree. We define and employ Algorithm $\mathcal{B}_3$ in Figure 3.8 which recursively finds a collision in Merkle Hash tree.

Theorem 3 (Consistency of entries)**.** *If hash function* H *is collision-resistant, then, in Certificate Transparency (with hash function* H*), no malicious logger can present two lists of entries, two fingerprints, and a consistency proof such that each list corresponds to the fingerprint, and the fingerprints are connected via the consistency proof, but the first list of entries is not a prefix of the second list of entries. More precisely, if* $\mathcal{A}$ *wins* $\mathrm{Exp}^{\mathsf{entry\text{-}cons}}_{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}}$*, then algorithm* $\mathcal{B}_3^{\mathcal{A}}$ *given in Figure 3.8 find a collision in* H*. Moreover, the runtime of* $\mathcal{B}_3^{\mathcal{A}}$ *consists of the runtime of* $\mathcal{A}$*, plus at most a quadratic (in the size of the second list) number of hash evaluations.*

In Section 2.4.1 Figure 2.6 we defined the Merkle Tree algorithms which we use for the following proof. Using the consistency proof $\vec{C}$ we can reconstruct each of the two roots from relevant parts of the proof and compare them against the actual roots. These constructions are done using the algorithms $\mathtt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$ and $\mathtt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$. The algorithm $\mathcal{R}_3$ works recursively to find a collision when given two entry sets and a consistency proof.

Proof. Suppose $\mathcal{R}_3$ receives as input values $\vec{D}_0, \vec{D}_1, \vec{C}$ such that

$$\mathtt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) = \mathtt{MTH}_{\mathrm{H}}(\vec{D}_0) \tag{3.1}$$

$$\mathtt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) = \mathtt{MTH}_{\mathrm{H}}(\vec{D}_1) \tag{3.2}$$

(where $n_0 = |\vec{D}_0|$ and $n_1 = |\vec{D}_1|$) but

$$\vec{D}_0 \not\preccurlyeq \vec{D}_1. \tag{3.3}$$

We claim that $\mathcal{R}_3$ outputs a collision for H when run with values satisfying (3.1)–(3.3). The proof of the claim proceeds by induction on the size of $\vec{C}$. $\mathcal{R}_3$ will recursively descend through the trees induced by $\vec{D}_0$ and $\vec{D}_1$, as well as the reconstructions of the two roots from the consistency path. At some point, $\mathcal{R}_3$ will find a collision. The three main cases are represented in Figures 3.9(a)–3.9(c).

Case 1. First suppose $|\vec{C}| = 2$. Assume that $n_0 = 2^{\lceil \log_2(n_1)/2 \rceil}$. (See Figure 3.9(a).) (This is true in honestly constructed consistency proofs; if lengths

$\underline{\mathcal{B}_3^{\mathcal{A}}()}$:

1: $(\vec{E}_0, \vec{E}_1, F_0, F_1, \vec{C}, pk) \leftarrow \mathcal{A}()$
2: **if** $F_0.n$ is a power of two, $\vec{C} \leftarrow F_0.H \| \vec{C}$
3: *// assume* $\vec{E}_0 \not\prec \vec{E}_1$
and $\texttt{MTH}_{\text{H}}(\vec{E}_0) = \texttt{Root0FromConsProof}_{\text{H}}(\vec{C}, F_0.n, F_1.n)$
and $\texttt{MTH}_{\text{H}}(\vec{E}_1) = \texttt{Root1FromConsProof}_{\text{H}}(\vec{C}, F_0.n, F_1.n)$
4: **return** $\mathcal{R}_3(\vec{E}_0, \vec{E}_1, \vec{C})$

$\underline{\mathcal{R}_3(\vec{D}_0, \vec{D}_1, \vec{C})}$:

1: $n_0 \leftarrow |\vec{D}_0|$
2: $n_1 \leftarrow |\vec{D}_1|$
3: *// require:* $\texttt{Root0FromConsProof}_{\text{H}}(\vec{C}, n_0, n_1) = \texttt{MTH}_{\text{H}}(\vec{D}_0)$
4: *// require:* $\texttt{Root1FromConsProof}_{\text{H}}(\vec{C}, n_0, n_1) = \texttt{MTH}_{\text{H}}(\vec{D}_1)$
5: $k \leftarrow 2^{\lceil \log_2(n_1)/2 \rceil}$
6: $\vec{C}' \leftarrow \vec{C}[0 : |\vec{C}| - 1]$
7: **if** $n_0 < k$
8: $\quad \ell_1 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_1[0 : k])$
9: $\quad \ell_1' \leftarrow \texttt{Root1FromConsProof}_{\text{H}}(\vec{C}', n_0, k)$
10: $\quad r_1 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_1[k : n_1])$
11: $\quad r_1' \leftarrow \vec{C}[|\vec{C}| - 1]$
12: $\quad$ **if** $(\ell_1 \neq \ell_1') \vee (r_1 \neq r_1')$
13: $\quad\quad$ **return** $(1 \| \ell_1 \| r_1, 1 \| \ell_1' \| r_1')$
14: $\quad$ **else return** $\mathcal{R}_3(\vec{D}_0, \vec{D}_1[0 : k], \vec{C}')$
15: **elsif** $n_0 = k$
16: $\quad \ell_0 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_0)$
17: $\quad \ell_1 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_1[0 : k])$
18: $\quad$ **if** $\ell_0 = \ell_1$ **return** $\mathcal{R}_1(\vec{D}_0, \vec{D}_1[0 : k])$
19: $\quad$ **elsif** $\ell_0 \neq \vec{C}[|\vec{C}| - 2]$
20: $\quad\quad$ *// will not occur due to line 2 of* $\mathcal{B}_3$
21: $\quad$ **else** $(\ell_1 \neq C[|C| - 2])$
22: $\quad\quad$ **return** $(1 \| \ell_1 \| \texttt{MTH}_{\text{H}}(\vec{D}_1[k : n_1],$
$\quad\quad\quad\quad 1 \| \vec{C}[|\vec{C}| - 2] \| \vec{C}[|\vec{C}| - 1]))$

$\underline{\mathcal{R}_3 \text{ continued:}}$

23: **else** $(n_0 > k)$
24: $\quad$ **if** $\vec{D}_0[0 : k] = \vec{D}_1[0 : k]$
25: $\quad\quad \ell \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_0[0 : k])$
26: $\quad\quad$ **if** $\ell = \vec{C}[|\vec{C}| - 1]$
27: $\quad\quad\quad r_0 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_0[k : n_0])$
28: $\quad\quad\quad r_0' \leftarrow \texttt{Root0FromConsProof}(\vec{C}', n_0 - k, n_1 - k)$
29: $\quad\quad\quad$ **if** $r_0 \neq r_0'$, **return** $(1 \| \ell \| r_0, 1 \| \ell \| r_0')$
30: $\quad\quad\quad r_1 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_1[k : n_1])$
31: $\quad\quad\quad r_1' \leftarrow \texttt{Root1FromConsProof}(\vec{C}', n_0 - k, n_1 - k)$
32: $\quad\quad\quad$ **if** $r_1 \neq r_1'$, **return** $(1 \| \ell \| r_1, 1 \| \ell \| r_1')$
33: $\quad\quad\quad$ **return** $\mathcal{R}_3(\vec{D}_0[k : n_0], \vec{D}_1[k : n_1], \vec{C}')$
34: $\quad\quad$ **else** $(\ell \neq \vec{C}[|\vec{C}| - 1])$
35: $\quad\quad\quad$ **return** $(1 \| \ell \| \texttt{MTH}_{\text{H}}(\vec{D}_0[k : n_0]),$
$\quad\quad\quad\quad\quad 1 \| \vec{C}[|\vec{C}| - 1]$
$\quad\quad\quad\quad\quad \| \texttt{Root0FromConsProof}_{\text{H}}(\vec{C}', n_0 - k, n_1 - k))$
36: $\quad$ **else** $(\vec{D}_0[0 : k] \neq \vec{D}_1[0 : k])$
37: $\quad\quad \ell_0 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_0[0 : k])$
38: $\quad\quad \ell_1 \leftarrow \texttt{MTH}_{\text{H}}(\vec{D}_1[0 : k])$
39: $\quad\quad$ **if** $\ell_0 = \ell_1$, **return** $\mathcal{R}_1(\vec{D}_0[0 : k], \vec{D}_1[0 : k])$
40: $\quad\quad$ **else** $(\ell_0 \neq \ell_1)$
41: $\quad\quad\quad$ **if** $\ell_0 \neq \vec{C}[|\vec{C}| - 1]$
42: $\quad\quad\quad\quad$ **return** $(1 \| \ell_0 \| \texttt{MTH}_{\text{H}}(\vec{D}_0[k : n_0]),$
$\quad\quad\quad\quad\quad\quad 1 \| \vec{C}[|\vec{C}| - 1]$
$\quad\quad\quad\quad\quad\quad \| \texttt{Root0FromConsProof}_{\text{H}}(\vec{C}', n_0 - k, n_1 - k))$
43: $\quad\quad\quad$ **else** $(\ell_1 \neq \vec{C}[|\vec{C}| - 1])$
44: $\quad\quad\quad\quad$ **return** $(1 \| \ell_1 \| \texttt{MTH}_{\text{H}}(\vec{D}_1[k : n_1]),$
$\quad\quad\quad\quad\quad\quad 1 \| \vec{C}[|\vec{C}| - 1]$
$\quad\quad\quad\quad\quad\quad \| \texttt{Root1FromConsProof}_{\text{H}}(\vec{C}', n_0 - k, n_1 - k))$

Figure 3.8: Algorithm $\mathcal{B}_3$ for Theorem 3.

mismatch in malicious proofs, the algorithms in Figure 2.6 in Section 2.4.1 will access invalid memory addresses and are assumed to abort.)

Now,

$$\texttt{Root1FromConsProof}_{\text{H}}(\vec{C}, n_0, n_1) = \text{H}(1 \| \vec{C}[0] \| \vec{C}[1])$$

by `Root1FromConsProof` line 1 in Figure 2.6 in Section 2.4.1. By definition, we have

$$\texttt{MTH}_{\text{H}}(\vec{D}_1) = \text{H}(1 \| \texttt{MTH}_{\text{H}}(\vec{D}_1[0 : k]) \| \texttt{MTH}_{\text{H}}(\vec{D}_1[k : n_1]))$$

By (3.2),

$$\texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) = \texttt{MTH}_{\mathrm{H}}(\vec{D}_1)$$

If $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k]) \neq \vec{C}[0]$ or $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[k:n_1]) \neq \vec{C}[1]$, then we have a collision in H: this is what is output by $\mathcal{R}_3$ on line 22.

So we now assume that $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k]) = \vec{C}[0]$ and $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[k:n_1]) = \vec{C}[1]$. By line 4 of `Root0FromConsProof` in Figure 2.6, we have that, $\texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) = \vec{C}[0]$. By (3.1), this is equal to $\texttt{MTH}_{\mathrm{H}}(\vec{D}_0)$. Furthermore, by the assumption at the start of this paragraph this is also equal to $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k])$. Since $\vec{D}_0 \not\prec \vec{D}_1$ and $|\vec{D}_0| = k$, we must have that $\vec{D}_0 \neq \vec{D}_1[0:k]$. Thus, $(\vec{D}_0, \vec{D}_1[0:k])$ constitutes a collision for $\texttt{MTH}_{\mathrm{H}}$, and by Lemma 1, this leads to a collision in H: this is what is output by $\mathcal{R}_3$ in its call to $\mathcal{R}_1$ on line 18.

Case 2. Now suppose $|\vec{C}| > 2$.

Case 2(a). First, suppose $n_0 < k$. (See Figure 3.9(b).) Use the definitions of $\ell_1, \ell_1', r_1, r_1'$ on lines 8–11 of $\mathcal{R}_3$. By lines 4, 5, and 8 of `Root1FromConsProof`, $\mathrm{H}(1\|\ell_1'\|r_1')$ is the value returned by $\texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1)$. By definition of `MTH`, $\mathrm{H}(1\|\ell_1\|r_1)$ is the value returned by $\texttt{MTH}_{\mathrm{H}}(\vec{D}_1)$. By equation (3.2), these hashes are equal. If $\ell_1 \neq \ell_1'$ or $r_1 \neq r_1'$, we have a collision in H: this is what is output by $\mathcal{R}_3$ on line 13. Suppose $\ell_1 = \ell_1'$ and $r_1 = r_1'$. Then in particular

$$\texttt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k]) = \texttt{Root1FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1])$$

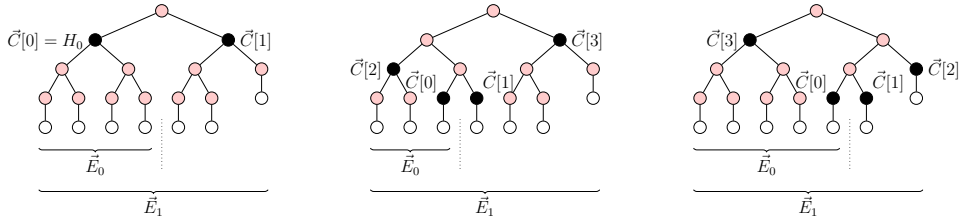
By line 3 of `Root0FromConsProof` and equation (3.1), we have that

$$\begin{aligned} \texttt{MTH}_{\mathrm{H}}(\vec{D}_0) &= \texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) \\ &= \texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}[0:|\vec{C}|-1], n_0, k) \end{aligned} \tag{3.4}$$

Moreover, $\vec{D}_0 \not\prec \vec{D}_1[0:k]$. Thus, $\vec{D}_0, \vec{D}_1[0:k, \vec{C}[0:|\vec{C}|-1]$ satisfy conditions (3.1)–(3.3). By induction, $\mathcal{R}_3$'s recursive call on line 14 will yield a collision in H.

Case 2(b). Second, suppose $n_0 > k$. (See Figure 3.9(c).) While we know that $\vec{D}_0 \neq \vec{D}_1[0:n_0]$, there are two possibilities: either $\vec{D}_0[0:k] = \vec{D}_1[0:k]$ but $\vec{D}_0[k:n_0] \neq \vec{D}_1[k:n_0]$, or $\vec{D}_0[0:k] \neq \vec{D}_1[0:k]$.

Suppose $\vec{D}_0[0:k] = \vec{D}_1[0:k]$ but $\vec{D}_0[k:n_0] \neq \vec{D}_1[k:n_0]$. This means that $\texttt{MTH}_{\mathrm{H}}(\vec{D}_0[0:k]) = \texttt{MTH}_{\mathrm{H}}(\vec{D}_1[0:k])$, which is ℓ on line 25 of $\mathcal{R}_3$. Now, either



(a) Case 1: $|\vec{C}| = 2$ (i.e., $n_0 = k$).

(b) Case 2(a): $|\vec{C}| > 2$, $n_0 < k$.

(c) Case 2(b): $|\vec{C}| > 2$, $n_0 > k$.

Figure 3.9: Tree and consistency path for cases in proof of Theorem 3. $|\vec{E}_0| = n_0$, $|\vec{E}_1| = n_1$, $k = 2^{\lceil \log_2(n_1)/2 \rceil}$. ○ denotes leaf nodes, ◎ denotes inner nodes, and ● denotes nodes corresponding to consistency proof values.

$\ell = \vec{C}[|\vec{C}| - 1]$ or not. If $\ell \neq \vec{C}[|\vec{C}| - 1]$, then we already have a collision in equation (3.1), and this is output by line 35. If $\ell = \vec{C}[|\vec{C}| - 1]$, then—intuitively—we have to look on the right side of the trees for the collision. If r_0 (the right side of the $\vec{D}_0$ tree) is not equal to r'_0 (the right side of the reconstructed path to root zero), then we have a collision:

$$\mathrm{H}(1\|\ell\|r_0) = MTH_{\mathrm{H}}(\vec{D}_0) = \texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}, n_0, n_1) = \mathrm{H}(1\|\ell\|r'_0)$$

and this is what $\mathcal{R}_3$ outputs on line 29. Similarly if r_1 (the right side of the $\vec{D}_1$ tree) is not equal to r'_1 (the right side of the reconstructed path to root one): this is what $\mathcal{R}_3$ outputs on line 32. Otherwise, $r_0 = r'_0$ and $r_1 = r'_1$, and thus $\vec{D}_0[k : n_0], \vec{D}_1[k : n_1], \vec{C}'$ satisfy conditions (3.1)–(3.3): by induction, the recursive call to $\mathcal{B}_3$ on line 33 will return a collision.

Finally, suppose $\vec{D}_0[0 : k] \neq \vec{D}_1[0 : k]$. Either these two lists hash to the same value under $\texttt{MTH}_{\mathrm{H}}$ or they do not. If they do, then this constitutes a collision in $\texttt{MTH}_{\mathrm{H}}$, and by Lemma 1, this leads to a collision in H: this is what is output by $\mathcal{R}_3$ in its call to $\mathcal{R}_1$ on line 40. If these two lists hash to different values ℓ_0, ℓ_1 under $\texttt{MTH}_{\mathrm{H}}$, then at least one of ℓ_0, ℓ_1 must be different from $\vec{C}[|\vec{C}| - 1]$. Suppose $\ell_0 \neq \vec{C}[|\vec{C}| - 1]$. By condition (3.1), the definition of MTH, and lines 6–8 of Root0FromConsProof in Figure 2.6, $1\|\ell_0\|\texttt{MTH}_{\mathrm{H}}(\vec{D}_0[k : n_0])$ and $1\|\vec{C}[|\vec{C}| - 1]\|\texttt{Root0FromConsProof}_{\mathrm{H}}(\vec{C}', n_0 - k, n_1 - k)$ hash to the same value but are different strings, constituting a collision for H: this is what $\mathcal{R}_3$ outputs on line 42. Similarly, if $\ell_1 \neq \vec{C}[|\vec{C}| - 1]$, we obtain a collision related to condition (3.2), which is what $\mathcal{R}_3$ outputs on line 44.

□

Now we prove the security property against a malicious monitor or an auditor. In Theorem 4 we show that it is difficult to frame an honest logger for not including a promised entry within certain time period (MMD).

Theorem 4 (Inclusion of promises)**.** *If hash function* H *is collision-resistant and signature scheme* SIG *is existentially unforgeable under chosen-message attacks, then, in Certificate Transparency (with hash function* H *and signature scheme* SIG*), no malicious monitor/auditor can frame an honest logger for not including a promised entry within the maximum merge delay. More precisely, if algorithm* $\mathcal{A}$ *wins* $\mathrm{Exp}^{\mathsf{promise\text{-}incl}}_{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}}}$*, then there exist algorithms* $\mathcal{B}^{\mathcal{A}}$ *and* $\mathcal{C}^{\mathcal{A}}$*, described in the proof, that find a collision in* $\mathtt{MTH}_{\mathrm{H}}$ *or a forgery in* SIG*, respectively. Moreover, the runtimes of* $\mathcal{B}^{\mathcal{A}}$ *and* $\mathcal{C}^{\mathcal{A}}$ *are approximately the same as that of* $\mathcal{A}$*.*

Proof. By definition of OTick (cf. Figure 3.7), the simulated honest logger will keep track of any promise P issued through OPromiseEntry and will include the P through OUpdateLog by time $T = P.t + \mathsf{MMD}$. As $\mathsf{MMD} > 0$, this ensures that any fingerprint issued by the honest logger at time $T' \geq T$ will include the promised entry $P.e$.

Assume $\mathcal{A}$ wins by outputting $(F, P, \vec{E})$, i.e., F is a valid fingerprint representing entries $\vec{E}$ and P is a promise for an entry $e \notin \vec{E}$ although $P.t + \mathsf{MMD} \leq F.t$. This means either one of the promise P or the fingerprint F (or both) were not issued by the simulated honest logger through an invocation of OPromiseEntry or OUpdateLog, or that $\mathcal{A}$ repeated an honest F that matches an entry list $\vec{E}$ different from the entry list $\vec{E}'$ hold by the honest logger when creating the fingerprint.

The second case constitutes a Merkle-tree hash collision (as $\mathtt{MTH}_{\mathrm{H}}(\vec{E}) = \mathtt{MTH}_{\mathrm{H}}(\vec{E}')$, but $\vec{E} \neq \vec{E}'$). Hence $\mathcal{A}$'s advantage in winning through this case can be bound by the advantage of an algorithm $\mathcal{B}$ (that simulates the oracles and simply outputs the colliding $\vec{E}$ and $\vec{E}'$) against the collision resistance of $\mathtt{MTH}_{\mathrm{H}}$. (Applying Lemma 1 leads to a collision in H.)

For the first case, we show how this allows constructing a signature forgery attacker $\mathcal{C}$ against the euf-cma security of SIG, which works as follows. First of all, $\mathcal{C}$ creates an initial state with an empty list of entries. It then simulates experiment $\mathrm{Exp}^{\mathsf{promise\text{-}incl}}_{\mathtt{CT}_{\mathrm{H},\mathtt{SIG}},\mathsf{MMD}}$ for $\mathcal{A}$, providing the public-key pk from its euf-cma game as input for $\mathcal{A}$. It furthermore uses its euf-cma signing oracle OSign when required to generate a signature in the simulations of the OPromiseEntry and OUpdateLog oracles and keeps a list of all the values queried to the signing oracle.

If $\mathcal{A}$ halts (outputting $(F, P, \vec{E})$) and wins, as argued above, at least one of P or F was not output through $\mathcal{C}$'s simulation of OPromiseEntry and OUpdateLog (as we excluded the case of a Merkle-tree hash collision). Hence, in particular, the according value was not queried to the euf-cma signing oracle, so $\mathcal{C}$ checks which of the two values is not contained in its list of queries and outputs this as its valid signature forgery. □

3.9 Discussion

One of the prevailing issues with web PKI is a lack of transparency when it comes to certificate management, making it difficult to detect when fraudulent certificates are issued. Certificate Transparency aspires to provide this much needed transparency by having CAs post certificates to publicly accessible qualified CT logs. It makes CA certificate issuance open to auditing and monitoring, which can be used to detect fraudulently issued certificates. In our work, we have analysed CT and introduced a generic model for logging schemes. In our model we have captured CT as one specific instance. We have formalised security notions regarding different entities involved in CT. Based on these formalisation, we have analysed the cryptographic aspects of CT and show how its cryptographic mechanisms prevent both undetected misbehaviour of log servers as well as false accusations of honest loggers.

The IETF attack model for CT presents potential threats in the context of web PKI. Our security properties attribute some of these potential threats, as the model covers both semantic and syntactic constraints and we have focused solely on semantic constraints. In CT, cryptography plays an important role to establish trust in its public auditable logging scheme. However, not all of these semantic aspects in CT can be captured in a cryptographic model. Some are difficult or even impossible to capture in such a security model as ours and are out of our scope and control. For instance, the performance of CT's gossiping protocol may play a huge part in how and when misbehaviour of entities will be detected. There may be various conditions on adversary control of the network with various patterns of honest entity behaviour. Nonetheless success will depend on how fast SCTs and STHs can be propagated between participating entities.

The CT gossiping protocol addresses some aspects of preserving privacy of the end users, but the threat model does not focus on it. Thus, our security model

also does not specifically address privacy concerns of the web clients. We present an abstract model for Certificate Transparency which can also be applied to other similar logging schemes. However, it is not without limitations. For example, the cryptographically assured security proofs may not be used to guarantee complete assurance as our abstract model does not represent every aspect of CT specification. Therefore, the real world implementation would involve many more constraints that we capture in our abstract model.

Since its first announcement in 2013, Certificate Transparency is implemented and widely adopted by two major browsers, Google Chrome and Mozilla Firefox. Google announced that certificates issued after April 2018 that are not CT qualified will not be trusted in Chrome. Users are shown certificate error messages for certificates that are not CT compliant. Google encourages all CAs to write the certificates they issue to publicly verifiable, append-only, tamper-proof logs. As in March 2019 Google transparency report [Goo19] mentions that in future Chrome and other browsers may not accept certificates that have not been written to such logs. As a result, most major certificate authorities are rallying up to get their certificates equipped with CT. Prior to CT, there was not an efficient way to get a comprehensive list of certificates issued to a particular domain. Certificate Transparency has positively impacted by reducing vulnerabilities in web PKI trust model. As one of the first abstract models designed for Certificate Transparency as well as being a logging scheme for objects other than certificates, we believe our proposed security model has made a good contribution to web PKI research field.

Chapter 4

Associative Blockchain for Decentralised PKI Transparency

Widely used Internet-based applications require high levels of security. The security of these web communications rely on public-key infrastructure supporting safe distribution and identification of public keys, allowing users to verify each other and securely exchange data over insecure networks. This is primarily obtained by communicating via the TLS protocol [DR06] with the use of X.509 certificates [CSF+08]. More organisations are focusing on managing keys and certificates through a web PKI to establish and maintain a trustworthy networking environment. This demand for public-key certificates makes Certificate Authorities likely targets for cyber attacks. However, it is not a practical solution to replace the web PKI with proposed alternatives like web of trust or certificate-less public-key cryptography as they are not cost effective or Internet scalable. As a result, the research community has been paying attention to the existing flaws in the web public-key infrastructure and proposing mitigation approaches.

We have studied quite a few proposals of new schemes intending to either enhance or replace current public-key infrastructure [BCK+14a, HS12, Eck11, Mar13]. Although some of these schemes have the potential to be successful, they are not without limitations. Furthermore, for most of these proposed approaches, a proper security analysis has not been done. In the previous chapter we chose Certificate Transparency [LKL13, Lau14, Rea13] and proposed a generic cryptographic model for secure logging schemes. We also formalised security

notions against malicious entities in Certificate Transparency. Our careful analysis of Certificate Transparency made us realise some of its limitations and some flaws in web PKI trust model that are still unsolved. This chapter focuses on our effort on extending, designing and constructing a new PKI technology to address the limitations identified in Certificate Transparency and provide stronger transparency to the existing public-key infrastructure. We present Decentralised PKI Transparency; a decentralised client-based approach to enforcing transparency in certificate issuance and revocation while eliminating single points of failure. We make use of an existing blockchain to realise an append-only distributed associative array. Further, we define three security properties and prove the results using provable security mechanisms and show that DPKIT achieves our initial design goals.

4.1 Introduction

The conventional hierarchical model for PKI, unfortunately suffers from an excess of trust placed in certificate authorities. As of 2018, there have been over 100 million certificates issued, all rooted in a few hundred certificate authorities, that are trusted by operating-system and browser vendors. Unfortunately, the web PKI trust model introduces many single points of failure, whereby any one of the hundreds of browser-trusted CAs is able to mount a man-in-the-middle impersonation attack against any domain, using their ability to issue valid but unauthorised certificates for the intended target. Attackers gain the upper hand when targeting an unsuspecting user or a group of users as these attacks could remain undetected for a long time, especially if performed periodically. These threats are not idle speculation: a couple of high-profile cases of mis-issued certificates have indeed been used to spoof legitimate websites, eventually resulting in the eviction of the offending CAs from most popular web browsers [Fox12, Com11, Duc13, SE15].

Though these cases were popularised, there might be many that were off the radar. As a domain can be linked to several digital certificates, it is challenging for clients to differentiate between a legitimate and a forged certificate, leading the possibility of a man-in-the-middle attack. Another issue with the current PKI is inefficiency of the revocation mechanism, especially with the sudden growth of HTTPS certificates (which has been partly due to “Let’s Encrypt” [Eck14]—a

prolific CA initiative that offers free certificate that are easy to obtain). Thus, traditional revocation mechanisms, off-line CRL [SHF02], to OCSP and OCSP stapling [Eas11, GSM+13] are simply not efficient or reliable enough to handle a large number of revoked certificates at once; and while identity-based-encryption approaches may theoretically fare better in that regard [BF01], they would be a big step backward in terms of decentralisation of trust.

Mitigation strategies against those issues fall into two categories: incompatible PKI redesigns, and compatible PKI add-ons. Our objective in this chapter falls in the latter category. We seek to create a client-driven, fully decentralised *certificate transparency* mechanism, that any interested user can privately run in their own browser, in order to catch and deter CA trust abuse. Compared with prior proposals, ours features a better combination of decentralisation, privacy, and compatibility with existing infrastructure and reluctant participants.

4.2 Our Contribution

We propose *Decentralised PKI Transparency (DPKIT)*, a fully decentralised approach to "certificate transparency", that seeks to rectify PKI trust issues on a user-driven voluntary basis without requiring any change to the underlying hierarchical PKI. There are two reasons for this: (1) compatibility with existing systems; and (2) the recognition that traditional CAs play a valuable role in vetting and vouching for the identity of domain owners in the real world.

To achieve this, DPKIT provides a strongly immutable record of all issued and revoked certificates *as seen by* the worldwide community of users, that efficiently supports all modern use cases of certificates in actual web protocols (including revocations, multiple certificates per domain, and multiple domains per certificates). The reporting function of DPKIT allows participating users to record any and all *valid* certificates or revocations that is presented to them, in order to provide a global audit. By design, the audit function of DPKIT also provides revocation checking mechanism with much greater privacy than the industry-standard OCSP online certificate validation.

Abstractly, DPKIT can be thought of as an *associative distributed ledger*, resulting from the subornation of a distributed key-value store to a *secure ledger* in the blockchain sense (which we model as a low-volume append-only record-keeping oracle). Our distributed ledger is a composition of a collection of (key, value) pairs

and each possible key appears at most once in the collection. Thus DPKIT can be described as an associative distributed ledger. Concretely, the key-value store component is efficiently instantiated as the fusion of Merkle and binary search trees, anchored to a blockchain for immutability; see Figure 4.1. Our reliance on a blockchain is merely for its decentralised append-only record-keeping functionality, irrespective of "currency" considerations such as preventing double-spending.

4.3 Background and Related Work

Now we examine some of the proposed blockchain-based PKI schemes and discuss the motivation behind our design.

4.3.1 PKI Transparency

In most cases involving attacks on PKI schemes, compromising the PKI scheme itself is not the end goal of the attackers; rather they are typically motivated to obtain other desirable information like credit card details, trade secrets or gain control over a particular organisation in order to achieve other targets. However, in most instances, compromising PKI is the first step to gain credentials required to access desired data. Thus it could be said that the main security property that any PKI must satisfy is preventing impersonation attacks, as it is one of the most compelling reasons for initiating an attack. To prevent such incidents, it is imperative to enhance these properties.

In terms of enhancing current PKI, achieving proper transparency has been the main focus. If somebody manages to compromise a CA or otherwise obtains a certificate that they aren't supposed to have, there is no proper method to know it has happened unless it is detected and widely reported. Many failures have occurred due to the lack of transparency between involved entities in the scheme with managing certificates. Basic web PKI consists several types of entities, performing different tasks: certificate authorities, web servers, browser vendors and web browsers. In its current standard scheme, overly trusting CAs has lead to many organisations being attacked and compromised. These issues over the recent years have significantly motivated researchers to look for alternatives or enhancements to current PKI.

One proposed alternative to achieve the goal of transparency is log-based PKI, which allows domain owners to learn when fraudulent certificates are issued

for their domains. Log-based schemes are founded on public logs that wield append-only databases of X.509 certificates. These logs facilitate public auditing of certificates and provide efficient proofs of presence of a specific entry in the log. Over the years several projects have been carried out based on public logs.

The *SSL Observatory* [EB10] by the Electronic Frontier Foundation (EFF) is perhaps the first large-scale deployment of this concept, whereby a browser extension enables users to report previously unseen certificates to the EFF, which published anonymised versions of those logs. While the integrity of the EFF is widely acknowledged, this approach still remains based on a central repository. Another popular and currently in use log-based scheme is *Certificate Transparency*. In Chapter 3, we discuss Certificate Transparency in length.

4.3.1.1 Certificate Transparency and its Limitations

When we analysed Certificate Transparency with the intention of defining a formal model for its security goals, we could see that CT has a few limitations that prevents it from achieving complete transparency as a PKI scheme. One of the end goals of CT is to ensure that it should be impossible for a CA to issue certificates for a domain without it being public knowledge. Another goal is that web clients should only accept certificates that are publicly logged and monitored. CT introduces several new entities to the existing web PKI: *Loggers*, *Submitters*, *Monitors* and *Auditors*. Loggers maintain the publicly accessible append-only logs of certificates which are submitted to be logged by submitters. Monitors are public or private services that periodically monitor the log for any suspicious behaviour. Auditors may be standalone entities or integrated in to web clients who verify the correct behaviour of a log.

In CT, trust is decentralised among the loggers, monitors and auditors whose main task is to watch each other and report misbehaviours. While this eliminates the requirement of a single trusted third party, and CT has indeed helped with detecting fraudulent certificates in some recent cases, it still raises concerns for a potential situation with one or more entities involved being malicious. Since a log server is the main party that maintains the publicly accessible logs of certificates, a misbehaving one could create critical issues [DGHS16]. Another problem is the heavy reliance on the third parties that monitor logs and the lack of consistency if a particular domain decides to have multiple certificates. Furthermore, there have been proposals [LK12] to address the revocation issue since CT does not

have a mechanism built in for revoking certificates. Furthermore, there is a lack of incentive for third parties to monitor the log might affect the efficiency of detecting misbehaviour of involved entities. Although trust is decentralised among its entities, the protocol itself is not as decentralised as desired.

Several other similar approaches to log-based certificate management are AKI [KHP+13], ARPKI [BCK+14a] and DTKI [YCR16], with varying levels of centralisation. However, the majority of projects on decentralised PKIs published over the recent years are based on Blockchain structure.

4.3.2 Blockchain-based PKI

A large proportion of recent projects seeking to rectify PKI trust issues are based on full redesigns, almost always involving a blockchain structure for greater decentralisation. Unlike log-based approaches, these are often incompatible with existing infrastructure.

Blockchain was first introduced as the public ledger for all transactions for the Bitcoin cryptocurrency [Nak08]. Unique combinations of properties make Blockchain suitable for variety of applications. For instance, while the ledger itself is *not* distributed (it is replicated), its affirmation mechanism is based on a heavily decentralised consensus mechanism that makes it increasingly infeasible to alter or delete previously time-stamped records.

In principle, blockchains would provide ideal environments for decentralised PKIs—except that the *replicated* nature of a blockchain can make it impractical for large data sets. This is not expected to be a huge issue for DPKIT as only the Merkle hash values are stored in the blockchain. Cryptocurrencies such as Bitcoin have two distinct extremely costly aspects—computational and storage cost—which brings on this impracticality. However, we see quite a few proposed redesigned PKI schemes based on blockchain.

Certcoin by Fromknecht et al. [FVY14] is a decentralised PKI that employs the consistency guarantees provided by cryptocurrencies such as Namecoin to build a PKI that offers strong identity retention and is founded on the concept of PGP web of trust (WoT). Certcoin proposes a completely decentralised PKI with no CAs involved. The protocol provides mechanisms for key registration, update and revocation, key recovery, verification and lookup built on top of Namecoin properties. Namecoin is a cryptocurrency that uses blockchain as a storage medium for digital identities. However, Certcoin does not focus on fighting

malicious users or verification and authentication of users. Furthermore, the lack of external validation for identities might be problematic when implemented in the real world.

Leiding et al. [LCMR16] introduce *Authcoin* which focuses on validation and authentication of public keys rather than identity retention. Authcoin aims to achieve a fault tolerant, hard to manipulate, replicated and transparent system that is based on blockchain structure making it difficult for adversaries to introduce malicious certificates into the system. Authcoin shares some basic ideas with Certcoin for a decentralised Blockchain based WoT. It differs from Certcoin by employing a flexible partially automated challenge response validation and authentication scheme for public keys. However, its interactive steps could be costly in performance and the lack of credibility of the party which carry out validation and authentication could be an issue.

Fredriksson's Master's thesis [Fre17] proposes a novel Proof-of-Stake protocol based on Merkle proofs to build a decentralised PKI which includes mappings between public keys and domains stored in Merkle tree. Integrity is achieved by including the Merkle tree hash in block header and certificates being cross-signed by domain owner's public key. There is an *Account tree* (stored in a Merkle tree) which contains a Blockchain id, unique hash and signing key used for certificates and CA proof. Nevertheless, the requirement of having two extensions added to X.509 certificate to achieve end goals could result in adaptation problems.

CeCoin proposed by Qin et al. [QHW+17] is a distributive Blockchain-based PKI which entirely removes the use of a trusted third party. Leveraging Merkle Patricia tree to implement a certificate library which stores every certificate, the scheme is completely decentralised with no CAs included. They aim to provide MITM attack prevention, multi-certificates and identity assignment services. However, CeCoin lacks a mechanism for external validation of identities.

Instant Karma PKI (IKP) proposed by Matsumoto and Reischuk [MR16] attempts to improve log-based PKI by incentivising the entities to look for fraudulent certificates and report. By decentralising and offering rewards to those who help detect misbehaviours IKP aims to automate the process of certificate handling. IKP extends the standard TLS architecture leveraging Ethereum smart contracts. These contracts handle various functions of the systems like CA and domain registration, policy negotiation, certificate purchase and reporting misbehaviour. An IKP contract takes a certificate as input, provided by a monitor,

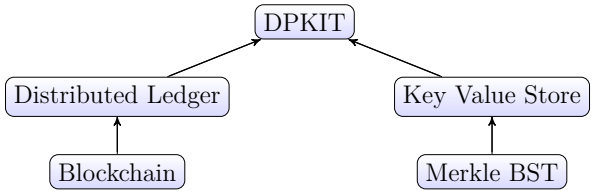


Figure 4.1: Decentralised PKI Transparency as an Associative Blockchain

and checks this certificate against a Domain Certificate Policy which specifies a list of CAs allowed to issue certificates for this particular domain. If the certificate is issued by a CA not present in this list, a Reaction Policy is executed which performs the escrow operation transferring Ethereum coins from the misbehaving CA to the affected user and the monitor who reported the violation. However, when implementing a possible issue with IKP is that not every CA will necessarily agree with the procedure as it has potential drawbacks for them. Among other things, none of these proposed systems allow for validation against real world identities; our DPKIT does.

4.4 Preliminaries

Notation. We denote by $\vec{E}$ an ordered list of entries, where () denotes the empty list. Indexing is 0-based: $\vec{E} = (e_0, \ldots, e_{n-1})$, and we write $\vec{E}[i]$ to denote e_i and $\vec{E}[i:j]$ to denote the sublist $(e_i, \ldots, e_{j-1})$. We adopt the convention that $\vec{E}[-1] = ()$. We write $e \in \vec{E}$ to indicate that an entry e is contained in the list $\vec{E}$. Each entry e is a key-value pair (k, v), and for such $e_j \in \vec{E}$ we write $e_j = (k_j, v_j)$.

4.4.1 Merkle Hash Tree and Binary Search Tree

As discussed in Chapter 2 Section 2.4.1, the Merkle tree is a useful cryptographic primitive to prove the existence of a record within a list $\vec{E}$. Let $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function. The tree is constructed by placing the values of $\vec{E}$ in the leaves of a binary tree and hashing each record. Each intermediate or parent node is built by hashing the hashes of its two child nodes. The root of the tree or *root hash* acts as a fingerprint for all the set $\vec{E}$. One can prove that a certain record exists in a tree with n nodes by providing the $\mathcal{O}(log\ n)$ hashes needed to reconstruct the root hash.

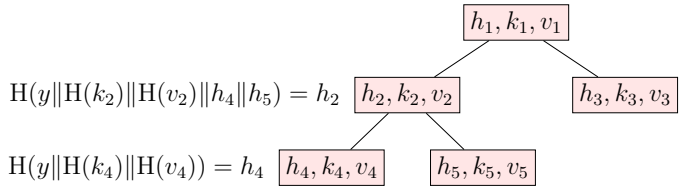


Figure 4.2: DPKIT Merkle Binary Search Tree. In each node, k_i is the search key (the domain name), v_i is a pointer to a secondary Merkle-BST (for storing the domain's certificates), and h_i is the root hash of the same, computed as shown where y is a "domain separation" label based on the node's position. (see § 4.4.1)

A *Binary Search tree* (BST), also known as ordered binary tree, is a type of data structure that stores items sorted and ordered by their keys and allows for fast lookups, insertion and removal of items [Cor09]. When looking for a particular key in the tree, one must traverse from root to leaf comparing the keys stored in nodes and continuing on left or right subtree based on the comparison of keys. Hence each operation takes on average time complexity of $\mathcal{O}(log\ n)$ if the keys were inserted in random order, or the tree is explicitly balanced.

4.4.1.1 Merkle-BST and Security Properties

Ideally, a tree which offers fast insertion and removal along with short proofs should be used for storing ordered sets of data. Thus to utilise properties of both Merkle hash tree and Binary search tree, we introduce the use of Merkle tree on top of a Binary Search tree. Merkle trees are useful in distributed, peer-to-peer systems where the same data should exist in multiple places. Constructing a Merkle tree on top of a BST provides the properties of an associative data structure implemented with the collision resistance of Merkle tree hashing. We introduce the Merkle-BST hybrid as a data structure which offers fast insertion and removal along with short proofs of existence, for storing ordered sets of data. The Merkle-BST data structure is useful as it provides a deterministic and remotely verifiable way of efficiently replicating large indexed sets of $(key, value)$ bindings, across many peers in a distributed system.

Nodes of a Merkle-BST have up to two children, and every node stores a (k, v) key-value entry along with a hash $\mathrm{H}(n)$. The hash is calculated by hashing the hashes of the children together with the node's entry, along with a special "domain-separation" flag $y \in \{1, 2, 3, 4\}$ describing the node's local topology (1 for leaves, 2 and 3 for nodes with a left or right child, and 4 for nodes with two

children; unlike the Merkle hash tree, we allow our Merkle-BSTs to be unbalanced and have single-child nodes). For an interior node n, let $LC(n)$ and $RC(n)$ denote its left and right child respectively and key k and v be node n's key and value, respectively. H denotes the use of a standard hashing algorithm. Its hash is calculated as (see Figure 4.2 for the diagram and Figure 4.6 for the algorithm),

$$\begin{aligned}
\mathrm{H}(n) &= \mathrm{H}(1\|\mathrm{H}(k)\|\mathrm{H}(v)) \\
\mathrm{H}(n) &= \mathrm{H}(2\|\mathrm{H}(k)\|\mathrm{H}(v)\|\mathrm{H}(RC(n))) \\
\mathrm{H}(n) &= \mathrm{H}(3\|\mathrm{H}(k)\|\mathrm{H}(v)\|\mathrm{H}(LC(n))) \\
\mathrm{H}(n) &= \mathrm{H}(4\|\mathrm{H}(k)\|\mathrm{H}(v)\|\mathrm{H}(LC(n))\|\mathrm{H}(RC(n)))
\end{aligned}$$

Lemma 3 states the proposition that if the hash function used in Merkle tree is proved to be collision resistant then the hashes in Merkle tree are also collision resistant with the use of the

Lemma 3 (Collision Resistance of Merkle-BST)**.** *If* H *is collision-resistant, then Merkle-BST hashing using* H *is also collision-resistant. More precisely, if there exists an* $\mathcal{A}$ *that finds a collision in Merkle-BST with probability* ε_{col}*, then there is an efficient* $\mathcal{B}_1^{\mathcal{A}}$ *that finds a collision in* H *with the same probability* ε_{col}*.*

The algorithm $\mathcal{B}_1$ is shown in Figure 2.7. The proof for the Merkle-BST hashing would be similar to the proof we have in Section 2.4.1 for Merkle binary tree hashing proof. Thus we do not elaborate it here.

4.4.2 Distributed Ledger and Blockchain

A *blockchain*, as known as distributed ledger, is a mechanism for decentrally maintaining an append-only ledger, replicated among peers. One of the key elements is its immutability. Any record added will be reserved forever, and can't be changed unless a group of miners can take control of the majority of the computational power in the network. This particular attack known as 51% is specific to blockchains with proof of work mechanism. Another important benefit is resilience against malicious adversaries. Since each block is connected to all the blocks before and after, it is difficult for attackers to tamper with a single record. Trying to change one record and also avoid detection would require changing the block containing that record as well as those after it. To achieve this, an attacker needs to assemble more CPU power or stake than all the honest parties.

Evidently it is more profitable to play by the rules as it costs more in terms of resources and time to break the rules and surpass the honest chain. Another inherent characteristic which ensures security through cryptography is use of digital signatures on each transaction. Moreover, blockchains are decentralised and distributed across peer-to-peer networks that are continuously updated.

The original and most famous example is the Bitcoin blockchain [Nak08], secured without any central authority by "*miners*" competing for rewards using proofs of work; see, e.g. [BBSU12]. The appeal of work-secured blockchains is their ability to resist Sybil attacks (wherein a user clones themselves to take control) without requiring an enrolment authority; see [ES14].

We take into account the robustness of blockchain security properties that have been proved by the success of cryptocurrencies. Our application does not require the full power of cryptocurrency blockchains, which also serve to (permanently) resolve mutually exclusive transactions. Rather, we view blockchains as black-box oracles that let users append small and infrequent amounts of data on a commonly shared (and replicated) timeline.

4.4.2.1 Ideal Functionality of Distributed Ledger

An ideal functionality is a protocol in which a trusted party that can communicate over perfectly secure channels with all protocol participants computes the desired protocol outcome [Can00]. A secure task can be formulated as an ideal functionality and translate some realistic schemes into an ideal protocol so that it can securely emulate the behaviour of the trusted party for honest users. In order to have solid foundation of security properties to be used, we define an ideal functionality framework for the append-onliness of a distributed ledger, in Figure 4.3. In our case, ideal functionality merely defines a set of oracles, or interface functions, with which participants can interact securely. Access to these functionalities are denoted by $\mathcal{L}$ and are used later on to show that desired properties of the system can be securely realised by its logging scheme when provided an ideal append only ledger functionality.

4.5 Decentralised PKI Transparency

Traditional PKI approaches heavily rely on trusted third parties. Although it has major drawbacks, getting rid of the services of certificate authorities could cause

Functionality F_{AOL}

Init() : Set $\vec{L} \leftarrow [\,]$ (ledger initialisation)

AddEntry(e) : Set $\vec{L} \leftarrow \vec{L}||e$ and return $|\vec{L}| - 1$ (add the entry to the ledger)

Handle() : return $|\vec{L}| - 1$ (get a handle representing the current state of the ledger)

GetBlock(l) : return $\vec{L}[l]$ or $\perp$ if $l > |\vec{L}|$ (get one entry from the ledger)

GetAll() : return $\vec{L}$ (get all entries in the ledger)

Figure 4.3: Ideal functionality of an append-only ledger $\mathcal{L}$.

more issues, such as the lack of a mechanism for external validation of identities. Another potential issue is cybersquatting (also known as domain squatting) in case of a key compromise. Therefore, in contrast to other approaches to a decentralised PKI, we do not eliminate certificate authorities entirely. Although with quite a few drawbacks, they are very effective regarding identity verification. We suggest more of a hybrid approach of utilising CAs for verifying identities and a distributed system for recording the certificates for a well managed decentralised PKI. Thus, at this stage in DPKIT, having some trust in the CAs is a necessity. Additionally, the fact that any compromise should be notified at the earliest possibility will help to handle any possible attacks.

4.5.1 Properties and Design Goals

Our purpose is to design a system that is secure and distributed which provides maximum transparency to an existing public-key infrastructure. Conventional schemes provide security, availability and efficiency when domains are authenticated by clients. Basic properties should include legitimate registration, update and responsible revocation of certificates and visibility of attacks if an entity is compromised. In addition, providing transparency and consistency for a domain with multiple certificates is essential. Apart from these basic functionalities, now we describe the desired properties of DPKIT:

- *Transparency*: Provide a publicly auditable system that enables anyone who connects to verify certificates or detect misbehaviour.
- *Multiple certificates*: Allow an entity to register multiple certificates per domain, mapping one identity to multiple public keys.

- *Multiple domain names*: Handle multiple domain names on a single certificate.
- *No single point of failure*: No centralisation or federation of any kind.
- *Scalability*: Remain efficient as more and more identities are recorded.
- *Efficiency*: Have low impact on TLS servers and minimal client storage/processing.
- *Client privacy*: Keep end-user browsing habits maximally private from observers and third parties.
- *Revocation*: Provide a reliable revocation system.

DPKIT should achieve these design goals in order to overcome the limitations that exist in Certificate Transparency and to provide a prominent transparent environment for management of digital certificates. These design goals were chosen to address the shortcomings of other certificate logging systems while maintaining functionality necessary for modern server architectures. We extend our design goals further to capture security aspects of DPKIT as a certificate management system by considering three more properties.

- *Non-removable entry*: Once a certificate entry is logged in DPKIT through honest functionalities, it is difficult for an adversary to remove it.
- *Proof consistency*: If a particular entry is invalid or not yet logged, it is difficult for an adversary to prove otherwise.
- *Revocation reveal*: Once revocation information is logged for a certificate, it is difficult for an adversary to hide/ remove it.

By presenting these cryptographically assured properties, we prove that it is hard for an adversary to tamper with the DPKIT chain and achieve its goals. Thus, we are motivated to achieve these security goals as well as our above mentioned design goals.

4.5.2 Entities, Operations and Functionalities

Basic entities of DPKIT includes CAs, domain owners, servers, clients and verifiers. The impact on current PKI is lesser than some proposed work we discussed in

Section 4.3.2 as existing entities are kept to handle basic functionalities of the system and only one new entity—*peers*—is added. Below we summarise various interactions of the entities.

1. Certificate authorities: issue certificates for domains.
2. Domain owners: obtain certificates from CAs, monitor the DPKIT for any suspicious activity.
3. Servers: allow clients to connect, and (optionally, if DPKIT-aware) request proofs of certificate membership from DPKIT peers.
4. Clients: initiate TLS connections with servers, accept or reject server certificates based on DPKIT information provided by peers and servers.
5. Peers/Verifiers: maintain the DPKIT data structure, record certificates, supply servers with proofs of certificate membership, provide auditing functionality

The main functionality, which begins when a certificate is issued and ends with a TLS connection between a client and server, consists of these steps (see Figure 4.4):

1. A domain owner requests a certificate from a CA. The CA applies its vetting policies and issues a certificate to the domain owner (steps A and B).
2. Optionally, the domain owner or the CA sends the certificate to a DPKIT peer, who adds the certificate to the DPKIT data structure. The domain owner supplies the certificate to the domain TLS server (steps C and D).
3. A client requests a connection to the server. Optionally, the server requests a proof of certificate membership for its certificate from the DPKIT peer, which the peer provides. The server sends its certificate and optionally the proof to the client (steps E, F, G, H, I and J).
4. The client requests the recent root hash from a DPKIT peer and uses it to check the proof of certificate membership (It also obtains the proof itself from the peer if not provided by the server). Depending on its security policy, the client may then complete the connection to the server (steps K, L and M).

Additionally, DPKIT provides the following functionalities:

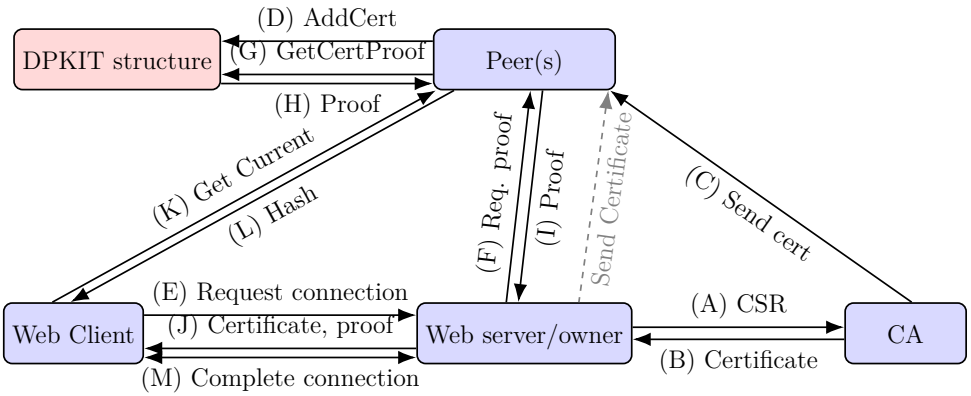


Figure 4.4: Overview over the interaction between entities involved in DPKIT scenario. See Section 4.5.2.

- Revocation certificates may also be submitted to DPKIT peers. The revocation is recorded, and its presence is noted in proofs of certificate membership.
- Any entity may submit a certificate to a DPKIT peer, and peers should add onto the DPKIT any missing valid certificates for which a proof is requested. This means that clients may contribute certificates that they have found which are missing from the DPKIT, increasing the probability that rare certificates are discovered and recorded.
- DPKIT peers also supply auditing functionality, including enumerations over all certificates for a domain, or over all domains, and indications whether certificates have been revoked.

Now we define a certificate scheme and its algorithms.

Definition 8 (Certificate scheme)**.** *A certificate scheme consists of these algorithms, wherein c represents either a certificate or a revocation "certificate":*

- SubjectNames$(c) \to l$*: returns a list* l *of subject names for which the certificate* c *applies.*
- FP$(c) \to \{0,1\}^\lambda$*: returns a fingerprint of the certificate.*
- IsRevocation$(c) \to \{0,1\}$*: returns a bit indicating whether or not* c *is a revocation certificate.*

- RevokedCert$(c) \to \{0,1\}^{\lambda}$*: for any revocation certificate that exists, returns the fingerprint of that certificate being revoked.*
- ValidateCert$(c) \to \{0,1\}$*: returns a bit indicating whether or not this is a proper (e.g., X.509) certificate.*

We do not explicitly define the API for certificate creation or revocation since these functions are not used directly by DPKIT. We assume that these functions are available to the CAs via an appropriate mechanism. For the remainder of this study, we assume that the function used to generate fingerprints of certificates is collision-resistant. For X.509 v.3 certificates, used for TLS on the web, SubjectNames() would return the domain name from "subject common name" as well as any additional domain names stated in the "subject alternative names" extension; FP() would return the hash of the certificate under a fixed cryptographic hash function. There are additional functions to actually obtain data such as public keys from a certificate but it is not needed for our formulation. Below we define what is a scheme for decentralised public-key infrastructure transparency.

Definition 9 (Decentralised PKI Transparency)**.** *A decentralised PKI transparency scheme (DPKIT) θ for a certificate scheme consist of these algorithms, where d is a data structure representing the overall state of the system:*

- $\text{Init}_D() \to d$*: deterministic algorithm that outputs an initial data structure d.*
- $\text{AddCert}_D(d, c) \to d'$ *or* $\perp$*: deterministic algorithm taking a certificate c and a data structure d as input and outputs an updated data structure d' or $\perp$ if not valid.*
- $\text{Search}_D(d, n) \to s$ *or* $\perp$*: deterministic algorithm that takes a data structure d and a subject name n and outputs a data structure representing all certificates for n or $\perp$ if name is not present.*
- $\text{SearchCert}_D(s, f) \to (c, r)$ *or* $\perp$*: deterministic algorithm that takes a data structure s and a certificate fingerprint f and outputs a certificate and revocation tuple (c, r), either of which may be $\perp$, or $\perp$ if both not present.*
- $\text{GetCertProof}_D(d, c) \to P$ *or* $\perp$*: deterministic algorithm that takes a certificate c and a data structure d and outputs a membership proof P or $\perp$.*

$\underline{\text{Exp}^{\text{non-removable}}_{\theta,\mathcal{L}}(\mathcal{A})}$:
1: $\text{Init}_{\mathcal{L}}()$
2: $d \leftarrow \text{Init}^{\mathcal{L}}_{\theta}()$
3: $(c, st) \xleftarrow{\$} \mathcal{A}_1()^{\theta*(d,\cdot),\mathcal{L}}$
4: $d \leftarrow \text{AddCert}^{\mathcal{L}}_{\theta}(d, c)$
5: **if** $d = \perp$, **return** 0
6: $\mathcal{A}_2(st)^{\theta*(d,\cdot),\mathcal{L}}$
7: **if** $c \notin \text{GetAllCerts}^{\mathcal{L}}_{\theta}(d)$, **return** 1
8: **for** N **in** SubjectNames(c);
9: $s \leftarrow \text{Search}^{\mathcal{L}}_{\theta}(d, N)$
10: **if** $s = \emptyset$, **return** 1
11: $(c', r') \leftarrow \text{SearchCert}^{\mathcal{L}}_{\theta}(s, \text{FP}(c))$
12: **if** IsRevocation(c) = 0
13: **if** $c' \neq c$, **return** 1
14: **else**
15: **if** $r' \neq c$, **return** 1
16: **return** 0

$\underline{\text{Exp}^{\text{show-revoke}}_{\theta,\mathcal{L}}(\mathcal{A})}$:
1: $\text{Init}_{\mathcal{L}}()$
2: $d \leftarrow \text{Init}^{\mathcal{L}}_{\theta}()$
3: $(c, r) \xleftarrow{\$} \mathcal{A}_1()^{\theta*(d,\cdot),\mathcal{L}}$
4: **if** RevokedCert(r) $\neq FP(c)$, **return** 0
5: $d \leftarrow \text{AddCert}^{\mathcal{L}}_{\theta}(d, r)$
6: $P \leftarrow \mathcal{A}_2()^{\theta*(d,\cdot),\mathcal{L}}$
7: $F \leftarrow \text{GetFingerprint}^{\mathcal{L}}_{\theta}(d)$
8: **if** $\text{VerifyCertProof}^{\mathcal{L}}_{\theta}(P, F, c) = 0$ **return** 0
9: **if** $\text{TestRevoke}^{\mathcal{L}}_{\theta}(P) = 1$ **return** 0
10: **return** 1

$\underline{\text{Exp}^{\text{entry-proof}}_{\theta,\mathcal{L}}(\mathcal{A})}$:
1: $\text{Init}_{\mathcal{L}}()$
2: $d \leftarrow \text{Init}^{\mathcal{L}}_{\theta}()$
3: $(P, c) \xleftarrow{\$} \mathcal{A}()^{\theta*(d,\cdot),\mathcal{L}}$
4: $F \leftarrow \text{GetFingerprint}^{\mathcal{L}}_{\theta}(d)$
5: **if** $\text{VerifyCertProof}_{\theta}(P, F, c) = 0$ **return** 0
6: **for** N **in** SubjectNames(c);
7: $s \leftarrow \text{Search}^{\mathcal{L}}_{\theta}(d, N)$
8: **if** $s = \emptyset$, **return** 1
9: $(c', r') \leftarrow \text{SearchCert}^{\mathcal{L}}_{\theta}(s, \text{FP}(c))$
10: **if** IsRevocation(c) = 0
11: **if** $c' \neq c$, **return** 1
12: **else if** IsRevocation(c) = 1
13: **if** $r' \neq c$, **return** 1
14: **return** 0

Figure 4.5: Security properties of a DPKIT scheme θ, using appending ledger $\mathcal{L}$, against a malicious $\mathcal{A}$.

- $\text{GetAllCerts}_D(d) \rightarrow \vec{E}$ *or* $\perp$*: deterministic algorithm that takes a data structure* d *and outputs an ordered list of entries or* $\perp$*.*
- $\text{GetFingerprint}_D(d) \rightarrow F$*: deterministic algorithm that takes a data structure* d *and returns a fingerprint* F *that captures all entries in the structure.*
- $\text{VerifyCertProof}_D(P, F, c) \rightarrow \{0, 1\}$*: deterministic algorithm that, given a certificate* c*, fingerprint* F *and membership proof* P*, outputs a bit* $b \in \{0, 1\}$*.*
- $\text{TestRevoke}_D(P) \rightarrow \{0, 1\}$*: A deterministic algorithm that takes as input a membership proof* P *and outputs a bit* $b \in \{0, 1\}$*.*

Algorithms defined in above Definitions 8 and 9 can be used to carry out all the interactions between peers/clients and DPKIT structure as described in Figure 4.4 which includes inserting and auditing certificates in the system.

4.5.3 Security Goals

In DPKIT, the main entity we consider when analysing threats is a "verifier", which can be a client or a peer in our terminology. Thus, we focus on threats where a verifier is presented false information in an attempt to disrupt normal operations. We consider security in an "ideal-ledger model", wherein the append-only ledger is modelled using the oracles of the ideal ledger functionality. We follow a provable-security game-based approach to define and prove DPKIT security properties. We consider three security properties against a malicious entity, which "plays" in a security experiment that acts as a challenger which the entity is trying to deceive. The three security properties formally specified in Figure 4.5 capture three ways in which a malicious entity may be able to pass conflicting information. We name and describe the three properties as follows:

Non-removable entry, per Experiment non-removable, demands that it is hard for an adversary to remove an entry once logged through honest functionalities.

Proof consistency, per Experiment entry-proof, entails that it is hard for an adversary to provide a proof for an entry that is invalid or not yet logged.

Revocation reveal, per Experiment show-revoke, demands that it is hard for an adversary to hide an entry's revocation information if it exists in the log.

A scheme that satisfies the above properties ensures that a malicious verifier cannot make peers interpret different things about entries covered by a root hash or a proof.

In experiments, θ represents the (possibly restricted) tuple of DPKIT algorithms that the adversary is given access to. $\mathcal{L}$ stands for an instantiation of the ideal append-only ledger functionality defined in Figure 4.3. We write $\theta * (d, \cdot)$ to indicate that $\mathcal{A}$ has oracle access to all or specific function(s) in θ, invoked with the data structure d as fixed parameter. For example,

$$\theta * (d, \cdot) = (\text{AddCert}, \text{SearchValue}, ..)$$

The parameter d indicates that these are the functions that the adversary can use to manipulate the global system state d, where it is implied that these manipulations may have side-effects, as the global state can be updated. (To fix ideas, since d is anchored to an underlying blockchain, and since in our model it

is not possible to remove entries from a blockchain, the adversary is not given access to such hypothetical functions.)

With these experiments in place we can now define security and correctness of a DPKIT scheme.

Definition 10. *Let $\mathbb{A}$ be set of algorithms. We say that a DPKIT scheme θ is* ϵ-secure *with respect to $\mathbb{A}$ under the ideal-ledger assumption if, for all $\mathcal{A} \in \mathbb{A}$,*

$$\Pr[\mathrm{Exp}^{\mathsf{non\text{-}removable}}_{\theta,\mathcal{L}}(\mathcal{A}) = 1] \leq \epsilon \tag{4.1}$$

$$\Pr[\mathrm{Exp}^{\mathsf{entry\text{-}proof}}_{\theta,\mathcal{L}}(\mathcal{A}) = 1] \leq \epsilon \tag{4.2}$$

$$\Pr[\mathrm{Exp}^{\mathsf{show\text{-}revoke}}_{\theta,\mathcal{L}}(\mathcal{A}) = 1] \leq \epsilon \tag{4.3}$$

where $\mathcal{L}$ is an ideal-ledger oracle functionality as defined in Figure 4.3.

Definition 11. *A DPKIT scheme θ is* correct *provided that:*

1. *If* $\mathrm{AddCert}_{\theta,\mathcal{L}}(d, c)$ *has not been called on certificate c, then* $\forall n \in \mathrm{SubjectNames}(c)$

$$\mathrm{SearchCert}_{\theta,\mathcal{L}}(\mathrm{Search}_{\theta,\mathcal{L}}(d, n), FP(c)) \neq (c, \cdot) \tag{4.4}$$

$$c \notin \mathrm{GetAllCerts}_{\theta,\mathcal{L}}(d) \tag{4.5}$$

2. *If* $\mathrm{AddCert}_{\theta,\mathcal{L}}(d, c)$ *has been called on a certificate c, then we always have* $\mathrm{VerifyCertProof}_{\theta,\mathcal{L}}(\mathrm{GetCertProof}_{\theta,\mathcal{L}}(d, c), \mathrm{GetFingerprint}_{\theta,\mathcal{L}}(d), c) = 1$.

3. *For a certificate c, if* $\mathrm{AddCert}_{\theta,\mathcal{L}}(d, r)$ *has never been called on any revocation r bearing on to c, that is, s.t.* $\mathrm{RevokedCert}(r) = FP(c)$*, then we always have* $\mathrm{TestRevoke}_{\theta,\mathcal{L}}((\mathrm{GetCertProof}_{\theta,\mathcal{L}}(d, c)), (\mathrm{GetFingerprint}_{\theta,\mathcal{L}}(d)), c) = 0$.

4.5.4 Key-Value Store with Proofs of Membership

To achieve the desired functionalities of DPKIT, we leverage an associative data structure comprised of an append only distributed ledger anchoring a key value store containing the bulk of the data. Blockchain is utilised as the distributed ledger and Merkle binary search tree as the key value store (see Figure 4.1). We now describe the main components of DPKIT that is instantiated using an append only ledger and a key value store. For the ledger, we use the ideal functionality of

$\underline{\text{Init}(\text{FP}_t, \text{H}) \to t\text{:}}$
1: $t.root = \emptyset$
2: $t.hash = \text{H}$
3: **return** t

$\underline{\text{CreateNode}_M(k, v, p) \to n\text{:}}$
1: $n \leftarrow ()$
2: $n.k = k, n.v = v, n.parent = p$
3: $n.subtree = \emptyset, n.left = \emptyset$
4: $n.right = \emptyset, n.hash = \text{NodeHash}_M(n)$
5: **return** n

$\underline{\text{InsertEntry}_M(t, k, v) \to t'\text{:}}$
1: **if** $(t.root = \emptyset)$
2: $\quad t.root \leftarrow \text{CreateNode}_M(k, v, \emptyset)$
3: **if** $(k = t.root.k)$, **return** $\perp$
4: **if** $(k < t.root.k)$
5: $\quad$ **if** $(t.root.left = \emptyset)$
$\quad\quad t.root.left \leftarrow \text{CreateNode}_M(k, v, t.root)$
6: $\quad$ **return** $\text{InsertEntry}_M(t.root.left, k, v)$
7: **else**
8: $\quad$ **if** $(t.root.right = \emptyset)$
$\quad\quad t.root.right \leftarrow \text{CreateNode}_M(k, v, t.root)$
9: $\quad$ **return** $\text{InsertEntry}_M(t.root.right, k, v)$
10: **return** t

$\underline{\text{SearchValue}_M(t, k) \to v\text{:}}$
1: **if** $(t.root = \emptyset)$ **return** $\perp$
2: $n \leftarrow \text{SearchNode}_M(t.root, k)$
3: $v = n.v$
4: **return** v

$\underline{\text{NodeHash}_M(n) \to h\text{:}}$
1: $h_1 = n.right.hash, h_2 = n.left.hash$
2: **if** $(n.right = \emptyset) \wedge (n.left = \emptyset)$
3: $\quad h = \text{H}(1 \| \text{H}(n.k) \| \text{H}(n.v))$
4: **if** $(n.left = \emptyset)$
5: $\quad h = \text{H}(2 \| \text{H}(n.k) \| \text{H}(n.v) \| h_1)$
6: **if** $(n.right = \emptyset)$
7: $\quad h = \text{H}(3 \| \text{H}(n.k) \| \text{H}(n.v) \| h_2)$
8: **else** $h = \text{H}(4 \| \text{H}(n.k) \| \text{H}(n.v) \| h_2 \| h_1)$
9: **return** h

$\underline{\text{SearchNode}_M(t.root, k) \to n\text{:}}$
1: **if** $(t.root = \emptyset)$ **return** $\perp$
2: **else if** $(k = t.root.k)$
3: $\quad n = t.root$ **return** n
4: **else if** $(k > t.root.k)$
$\quad\quad$ **return** $\text{SearchNode}_M(t.root.right, k)$
5: **else return** $\text{SearchNode}_M(t.root.left, k)$

$\underline{\text{GetFingerprint}_M(t) \to F\text{:}}$
1: $F \leftarrow t.root.hash$
2: **return** F

$\underline{\text{GetAll}_M(t) \to \vec{E}\text{:}}$
1: $\vec{E} \leftarrow [\,]$
2: **if** $(t.root = \emptyset)$, **return** $\perp$
3: $\text{GetAll}_M(t.root.left)$
4: $\vec{E}.append(t.root.left.k)$
5: $\text{GetAll}_M(t.root.right)$
6: $\vec{E}.append(t.root.right.k)$
7: **return** $\vec{E}$

Figure 4.6: Construction of KVSMP using Merkle BST

Figure 4.3. For the key-value store with membership proofs of membership, we use our notion of Merkle binary search tree (Merkle-BST), for which we define the following algorithms.

Definition 12 (Key-Value Store with Membership Proofs)**.** *A* key-value store with membership proofs *(*KVSMP*) consists of the following algorithms, where t is a data structure representing the overall state of the system. The first few algorithms are used to manage entries in the key-value store.*

- $\text{Init}_M(\text{FP}, \text{H}) \to t$*: deterministic algorithm that given a value fingerprinting hash function* FP *and a hash function* H*, outputs an initial data structure t.*
- $\text{InsertEntry}_M(t, k, v) \to t'$ *or* $\perp$*: deterministic algorithm that takes a key-value entry (k, v), a data structure t, and returns an updated data structure t' or $\perp$ if entry is already inserted.*

- $\text{SearchNode}_M(t, k) \to n$ *or* $\perp$*: A deterministic algorithm that takes an entry key* k *and a root node* t*, and outputs a node* n *or* $\perp$*.*
- $\text{SearchValue}_M(t, k) \to v$ *or* $\perp$*: A deterministic algorithm that takes an entry key* k *and a data structure* t*, and outputs a value* v *or* $\perp$*.*
- $\text{NodeHash}_M(n) \to h$*: A deterministic algorithm that takes a node as an input, and outputs its hash value* h*.*

The next algorithms are used to test various properties of the key-value store:

- $\text{GetFingerprint}_M(t) \to F$*: deterministic algorithm that takes a data structure* t *and outputs a fingerprint* F *representing the complete list of entries.*
- $\text{GetAll}_M(t) \to (\vec{E})$*: deterministic algorithm that takes a data structure* t *and outputs an ordered list of entries* $\vec{E}$*.*

The last few algorithms generate and verify membership proofs:

- $\text{GetProof}_M(t, k) \to P$ *or* $\perp$*: deterministic algorithm that takes as inputs an entry key* k *and a data structure* t *and outputs a membership proof* P *or* $\perp$ *if the key is not present.*
- $\text{VerifyProof}_M(P, F, k, v) \to \{0, 1\}$*: A deterministic algorithm that takes as inputs an entry key* k*, a value fingerprint* f*, a data-structure fingerprint* F*, and a membership proof* P*, and outputs a bit* $b \in \{0, 1\}$*.*

Now we define the correctness properties of our key-value store with membership proofs.

Definition 13 (Correctness of a KVSMP)**.** *A* KVSMP *scheme is* correct *if,*

1. *If* $\text{InsertEntry}_M(t, k, v)$ *is called, then any subsequent call* $\text{GetAll}_M(t)$ *will return a list that contains the pair* (k, v)*.*
2. *Any call* $\text{SearchValue}_M(t, k)$ *will return the value* v *corresponding to the most recent prior call* $\text{InsertEntry}_M(t, k, v)$ *with matching key* k*, or an error symbol* $\perp$ *if no such call had been made.*
3. *If* $\text{InsertEntry}_M(t, k, v)$ *has been called on an entry* (k, v)*, then it is always the case that* $\text{VerifyProof}_M(\text{GetProof}(t, k), \text{GetFingerprint}_M(t)) = 1$*.*

$\text{GetProof}_M(t, k) \rightarrow P$:

1: $P \leftarrow ()$, $fn \leftarrow ()$, $j = 0$
2: $F \leftarrow t.root.hash$
3: $u \leftarrow \text{SearchNode}_M(t, k)$
4: **if** $u = \perp$ **return** $\perp$
5: **if** $((u.l = \emptyset) \wedge (u.r = \emptyset))$
6: $fn_j.re = fn_j.le = 0$ `/* no child nodes */`
7: **if** $(u.r = \emptyset)$
8: $fn_j.re = 0$, $fn_j.le = 1$ `/* no right child node */`
9: $fn_j.lh = u.l.hash$
10: **if** $(u.l = \emptyset)$
11: $fn_j.re = 1$, $fn_j.le = 0$ `/* no left child node */`
12: $fn_j.rh = u.r.hash$
13: **else**
14: $fn_j.r = 1$, $fn_j.l = 1$
15: $fn_j.rh = u.r.hash$,$fn_j.lh = u.l.hash$
16: $P_j.first = fn_j$
17: **while** $(u.hash \neq F)$
18: **if** $(u.subtree = r)$
19: $P_j.subtree = r$, $P_j.level = j + 1$
20: **if** $(u.parent.l = \emptyset)$ `/* no sibling node for node u */`
21: $P_j.sibling = 0$, $P_j.pkv = \text{H}(\text{H}(u.parent.k) \| \text{H}(u.parent.v))$
22: **else**
23: $P_j.sibling = 1$, $P_j.childh = u.parent.l.hash$,
$P_j.pkv = \text{H}(\text{H}(u.parent.k) \| \text{H}(u.parent.v))$
24: **if** $(u.subtree = l)$
25: $P_j.subtree = l$, $P_j.level = j + 1$
26: **if** $(u.parent.r = \emptyset)$
27: $P_j.sibling = 0$, $P_j.pkv = \text{H}(\text{H}(u.parent.k) \| \text{H}(u.parent.v))$
28: **else**
29: $P_j.sibling = 1$, $P_j.childh = u.parent.r.hash$,
$P_j.pkv = \text{H}(\text{H}(u.parent.k) \| \text{H}(u.parent.v))$
30: $u = u.parent$, $j{+}{+}$
31: **endwhile**
32: **return** P

Figure 4.7: Algorithm GetProof of KVSMP using Merkle BST. For presentation purposes r, l, re, le, rh, and lh represent the notions right, left, right child exists, left child exists, right child hash and left child hash respectively.

A membership proof in a Merkle-BST consists of the authentication path from a node to root and any change cascading through the path up to the root. For our construction, we instantiate a KVSMP from a Merkle-BST data structure, a cryptographic hash function $\text{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$, and a separate function $\text{FP}(c) \rightarrow \{0,1\}^\lambda$ giving fingerprints of values as specified in Figures 4.6, 4.7 and 4.8.

VerifyProof$_M(P, F, k, v) \rightarrow b$:

1: $i = 0$
/* Calculating hash value of the node based on its child nodes */
2: **if** $((P_i.fn.le = 1) \wedge (P_i.fn.re = 1))$
3: $hashv = \mathrm{H}(4 \| \mathrm{H}(k) \| \mathrm{H}(v) \| P_j.first.lh \| P_j.first.rh)$
4: **if** $((P_i.fn.le = 1) \wedge (P_i.fn.re = 0))$
5: $hashv = \mathrm{H}(3 \| \mathrm{H}(k) \| \mathrm{H}(v) \| P_j.first.lh)$
6: **if** $((P_i.fn.le = 0) \wedge (P_i.fn.re = 1))$
7: $hashv = \mathrm{H}(2 \| \mathrm{H}(k) \| \mathrm{H}(v) \| P_j.first.rh)$
8: **if** $((P_i.fn.le = 0) \wedge (P_i.fn.re = 0))$
9: $hashv = \mathrm{H}(1 \| \mathrm{H}(k) \| \mathrm{H}(v))$
10: **if** $(P_i.subtree = r)$
/* Calculating intermediate parent node from 'hashv' */
11: **if** $(P_i.sibling = 1)$
12: $tempRootH_i = \mathrm{H}(4 \| P_i.pkv \| P_i.childh \| hashv)$
13: **else** $tempRootH_i = \mathrm{H}(2 \| P_i.pkv \| hashv)$
14: **if** $(P_i.subtree = l)$
15: **if** $(P_i.sibling = 1)$
16: $tempRootH_i = \mathrm{H}(4 \| P_i.pkv \| hashv \| P_i.childh)$
17: **else** $tempRootH_i = \mathrm{H}(3 \| P_i.pkv \| hashv)$
18: **while** $(P_{i+1}.level \neq 0)$
19: **if** $(P_{i+1}.subtree = r)$
20: **if** $(P_{i+1}.sibling = 1)$
21: $tempRootH_{i+1} = \mathrm{H}(4 \| P_{i+1}.pkv \| P_{i+1}.childh \| tempRootH_i)$
22: **else** $tempRootH_{i+1} = \mathrm{H}(2 \| P_{i+1}.pkv \| tempRootH_i)$
23: **if** $(P_{i+1}.subtree = l)$
24: **if** $(P_{i+1}.sibling = 1)$
25: $tempRootH_{i+1} = \mathrm{H}(4 \| P_{i+1}.pkv \| P_{i+1}.childh \| tempRootH_i)$
26: **else** $tempRootH_{i+1} = \mathrm{H}(3 \| P_{i+1}.pkv \| tempRootH_i)$
27: $newRootH = tempRootH_{i+1}$, $i++$
28: **endwhile**
29: **if** $newRootH = F$, **return** 1
30: **return** 0

Figure 4.8: Algorithm VerifyProof of KVSMP using Merkle BST. For presentation purposes r, l, re, le, rh, and lh represent the notions right, left, right child exists, left child exists, right child hash and left child hash respectively.

4.5.5 Instantiation of DPKIT as a Logging Scheme

A significant semantic constraint for a certificate management environment is that certificates should be issued to the entity that genuinely represents the subject or subject alternative named in the certificate. As for the web clients, they have the right to be updated on the current status of the certificate and decline the TLS connection if the certificate is found to be invalid or revoked. Our abstract model for DPKIT mainly focus on constraints that can be assured cryptographically. We make use of the notions of validity and revocation of a certificate, but we leave the assessing of certificates on such measures to the certificate scheme (see Definition 8).

$\underline{\text{Init}() \to d}$:
1: $d \leftarrow ()$
2: $d.tree \leftarrow \text{Init}_M(\text{FP}_d, \text{H})$
3: **return** d

$\underline{\text{Search}_D(d, N) \to s}$:
1: $s \leftarrow \text{SearchValue}_M(d.tree, N)$
2: **return** s

$\underline{\text{SearchCert}_D(s, \text{FP}(c)) \to (c, r)}$:
1: $(c', r') \leftarrow \text{SearchValue}_M(s, \text{FP}(c))$
2: **return** (c', r')

$\underline{\text{GetCertProof}_D(d, c) \to P}$:
1: $N \leftarrow \text{SubjectNames}(c)[0]$
2: $s \leftarrow \text{Search}_D(d.tree, N)$
3: **if** $s = \perp$ **return** $\perp$
4: $(c', r) \leftarrow \text{SearchCert}_D(s, \text{FP}(c))$
5: **if** $c \neq c'$ **return** $\perp$
6: $p_1 \leftarrow \text{GetProof}_M(d.tree, N)$
7: $p_2 \leftarrow \text{GetProof}_M(s, \text{FP}(c)))$
8: **if** $r \neq \perp$,$f_r \leftarrow h(r)$
9: **else** $f_r \leftarrow \emptyset$
10: $f_c \leftarrow \text{FP}(c)$
11: $f_s \leftarrow \text{GetFingerprint}_M(s)$
12: **return** $P \leftarrow (p_1, p_2, f_r, f_c, f_s)$

$\underline{\text{GetAllCerts}_D(d) \to \vec{E}}$:
1: $\vec{E} \leftarrow [\,]$
2: $L \leftarrow \text{GetAll}_{\mathcal{L}}(d.ledger)$
3: **for** $n = 0$ **to** $Length(L) - 1$
4: $(F, c) \leftarrow L[n]$
5: $\vec{E} \leftarrow \vec{E}||c$
6: **return** $\vec{E}$

$\underline{\text{GetFingerprint}_D(d) \to F}$:
1: $(F, c) \leftarrow \text{GetBlock}_{\mathcal{L}}(\text{Handle}_{\mathcal{L}}())$
2: **return** F

$\underline{\text{AddCert}_D(d, c) \to d'}$:
1: **if** $\text{ValidateCert}(c) = 0$ **return** $\perp$
2: $d' \leftarrow d$
3: **for** N **in** $\text{SubjectNames}(c)$
4: $s \leftarrow \text{Search}_D(d.tree, N)$
5: **if** $s = \emptyset$
6: $s \leftarrow \text{Init}(\text{FP}, \text{H})$
7: **if** $\text{IsRevocation}(c) = 0$
8: $s \leftarrow \text{InsertEntry}_M(s, \text{FP}(c), (c, \emptyset))$
9: **else**
10: $s \leftarrow \text{InsertEntry}_M(s, \text{FP}(c), (\emptyset, c))$
11: **else**
12: $(c', r') \leftarrow \text{SearchCert}_D(s, \text{FP}(c))$
13: **if** $\text{IsRevocation}(c) = 0$
14: **if** $c' \neq c$ **return** $\perp$
15: **else**
16: $s \leftarrow \text{InsertEntry}_M(s, \text{FP}(c), (c', c))$
17: $d'.tree \leftarrow \text{InsertEntry}_M(d'.tree, N, s)$
18: $L \leftarrow \text{AddEntry}_{\mathcal{L}}(\text{GetFingerprint}_D(d'.tree), c)$
19: **return** d'

$\underline{\text{VerifyCertProof}_D(P, F, c) \to b}$:
1: $n \leftarrow \text{SubjectNames}(c)[0]$
2: **if** $\text{VerifyProof}_M(P.p_1, F, n, P.f_s) = 0$
3: **return** 0
4: $f_{cr} \leftarrow \text{H}(P.f_c || P.f_r)$
5: **if** $\text{VerifyProof}_M(P.p_2, P.f_s, \text{FP}(c), f_{cr}) = 0$
6: **return** 0
7: **return** 1

$\underline{\text{TestRevoke}_D(P) \to b}$:
1: **if** $P.f_r \neq \emptyset$ **return** 1
2: **return** 0

Figure 4.9: Pseudo-code of our DPKIT construction.

Now that we have the required key-value store and append-only ledger, we can construct our associative distributed ledger for PKI transparency, DPKIT, as specified in Figure 4.9.

The DPKIT of Figure 4.9 stores domain names as keys and certificates as values in a Merkle-BST, accessed through the KVSMP functions defined in Figure 4.6. The hash value of the root node F represents the entries of every node contained in tree at a certain time. Every node (including the root node) contains a specific key-value entry (k, v).

In our application, the search key k is a domain name, and the value v is a pointer to its very own data structure; a secondary Merkle-BST (as illustrated in

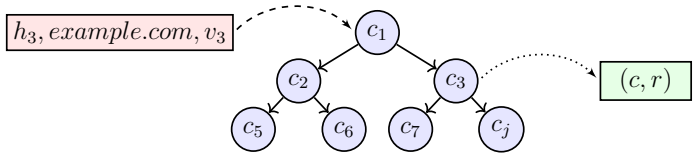


Figure 4.10: The Secondary Merkle-BST rooted at c_1 is attached to node $(h_3, k_3 = example.com, v_3)$ in the primary tree, and providing access to every certificate c_j for the domain *example.com*. The certificates and their revocations (if any) are stored in a certificate container tuple accessible from nodes from one or more secondary Merkle-BST(s)

blue in Figure 4.10). This certificate tree is unique to every node, that is every node has a certificate tree on its own and stores all the certificates that pertain to that particular domain name k. The secondary Merkle-BST, however, does *not* directly store the certificates, as those may be *shared* with other domains. Each node of the secondary Merkle-BST points to a separate tuple or container storing the updates on that specific certificate: the issuance certificate, and a revocation certificate if there is one. Other secondary Merkle-BSTs, pertaining to other domain names, will then be able to point to the same certificate container, in case the certificate happens to be shared. All these certificate updates contribute to the hash of the node in the secondary Merkle-BST, and thus its root hash, and from there the hash of the domain-name node in the primary Merkle-BST, and in turn the primary root hash. In other words, the root hash of the secondary tree is used as the value in the primary tree.

Some certificates allow for a primary domain name and a few more additional subject alternative names (SANs) in a single certificate. For example, there could be `www.gmail.com`, `www.mail.google.com` etc. for *Gmail*. These various domain names will be represented by separate nodes in the main tree. If they share a certificate, their secondary tree will all have a node pointing to the same tuple.

Adding new certificates to DPKIT is treated as transactions on the distributed ledger. These transactions are then assessed by peers using the latest block header which contains the most recently updated Merke BST root hash. If verification fails, for example it is a duplicate transaction or a fraudulently issued certificate for a domain, the peer can refuse to add the requested certificates as transactions, in turn preventing it been added to DPKIT structure. The root hash provided by the Merkle BST in key value store is then broadcast on the distributed ledger. The root hash is a part of the block header. After a new block is added, peers

of the network could then verify the new block and accept it as current or else discard the new block and continue with the previous block. To verify the certificates in the block peers could update their local copy of the MBST with newly added certificates/revocations, compute the new root hash, check to make sure that it agrees with the root hash on the new block, and check that the new certificates/revocations are all valid. Every peer should be in agreement with the transactions on the block. Generally, all root certificates are self-signed. To agree on same trust anchors for certificate validation these root certificates can be stored on the log with a special identifier in addition to the domain name. Furthermore, all transaction information needs to be in its original state, without alterations or corrupted information. This agreement on the current root hash representing every entry in the underlying tree is achieved by using a consensus algorithm. The consensus algorithm (for example Bitcoin's *proof of work (PoW)*) ensures the validity of a block and its transactions, that the next block in a blockchain is the one and only version of the truth.

4.5.6 Maintaining the decentralised ledger

Blockchain is typically built to keep adding new blocks onto old blocks. To prevent tampering previous blocks and to promote working only on extending the chain with new blocks, there should be an incentive mechanism. Moreover, all blockchain implementations currently enforce a strict time ordering, which presents mutually incompatible transactions. Therefore, using an incentive is recommended to solve the issue of reaching agreement within a decentralised network and to deter attacks. It should be noted that incentives do not have to have a monetary value. Good reputation could be an incentive for involved parties for not deviating from protocol. However, incentive determines whether people want to participate in the consensus mechanism or not.

Generally, a consensus algorithm like Bitcoin's proof-of work does two things: it keeps powerful adversaries from hindering the system and successfully forking the chain, and also ensures each new block added to the chain is the one and only legitimate block. The consensus algorithm could either be proof of work or *proof of stake (PoS)* depending on the blockchain chosen for the distributed ledger. A proof of work is a piece of data which is difficult (costly or time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Miners must complete proof-of-work containing all of the data in block, in order

for that block to be accepted by network peers and rewarded. In proof of stake, mining power for miners is attributed to the proportion of coins held by each miner.

In DPKIT, we do not differentiate between PoW and PoS as either could be applied on the ledger. If the blockchain utilising PKI transparency has a PoW based consensus then the miners of the blockchain would be verifiers of any entity discussed above. The stakeholders or miners verify the legitimacy of transactions in each block. DPKIT could piggyback a current blockchain or smart contracts from Ethereum. A valuable possibility of smart contracts is that they could be used to ensure that only root hashes for valid data structures are stored on the chain. It will essentially remove duplicated efforts when peers check each root hash for validity as they search for the most recent acceptable root hash. Another alternative is to make use of a private blockchain implemented just for the use of DPKIT structure.

In order for our system to be secure, the costs of maintaining the ledger (via proof of work or otherwise) must be covered — miners must be paid. In the basic case, where a CA adds certificates, the cost of adding certificates could be rolled into the cost of issuing a certificate. Another plausible option is offering bounties for certificates from particular CAs or domains. For example, Google considering their large number of domain certificates in use and also the number of trusted CAs listed in browser, can set up a bounty for any certificate signed by a CA who's root certificate is installed in Chrome. In that way clients who add those certificates first receive the intended bounty, encouraging clients to report new certificates, while the service providers gain valuable information about the security of their services. In our model, every valid certificate can be added, so some form of DoS protection is necessary — in the above case this is accomplished by the fee paid to miners for adding certificates. Other incentive structures are possible, but this method allows for a very open system.

4.6 Security Analysis and Evaluation

First we define the collision resistance of hash function.

Definition 14 (Collision resistance of Hash function)**.** *Let $\mathcal{M}$ be a set, let $\mathrm{H} : \mathcal{M} \to \{0,1\}^{\lambda}$ be an unkeyed hash function, and let $\mathbb{A}$ be a set of algorithms. We say that no $\mathcal{A} \in \mathbb{A}$ finds a pair (m, m') such that $m \neq m'$ and $\mathrm{H}(m) = \mathrm{H}(m')$*

in time t with probability higher than ϵ.

For practical purposes, ϵ can be set to 2^{-128}.

Lemma 4 (Correctness of DPKIT scheme)**.** *A DPKIT construction θ, given Figure 4.9, is correct according to definition 11 if,*

- $\text{AddCert}_{\theta,\mathcal{L}}(d, c) \implies c \in \text{GetAllCerts}_{\theta,\mathcal{L}}(d)$.
- $\text{AddCert}_{\theta,\mathcal{L}}(d, c) \implies$
  $\text{VerifyCertProof}_{\theta,\mathcal{L}}(\text{GetCertProof}_{\theta,\mathcal{L}}(d, c), \text{GetFingerprint}_{\theta,\mathcal{L}}(d), c) = 1$.
- $\text{AddCert}_{\theta,\mathcal{L}}(d, r) \wedge \text{RevokedCert}(r) = FP(c) \implies$
  $\text{TestRevoke}_{\theta,\mathcal{L}}((\text{GetCertProof}_{\theta,\mathcal{L}}(d, c)), (\text{GetFingerprint}_{\theta,\mathcal{L}}(d)), c) = 1$.

The proposed DPKIT system is correct according to Definition 11 and Lemma 4 follows straightforwardly from the definition of the protocol and the properties of MBST's. We now prove the security results on Decentralised PKI Transparency, namely that its instantiation with our associative data structure guarantees the 3 properties of entry non-re-movability, revocation reveal and proof consistency from Figure 4.5

The theorem 5 shows that once a certificate entry is logged in DPKIT through honest functionalities, it is difficult for an adversary to remove it from the system.

Theorem 5 (Non-removable entry)**.** *If hash function* H *is collision-resistant, then in DPKIT scheme θ (with hash function H) in ideal ledger model $\mathcal{L}$, no malicious entity can present two different entries for the same fingerprint. More precisely, no $\mathcal{A}$ wins* $\text{Exp}^{\text{non-removable}}_{\theta,\mathcal{L}}(\mathcal{A})$*, with probability higher than ϵ. That is,*

$$Pr[\text{Exp}^{\text{non-removable}}_{\theta,\mathcal{L}}(\mathcal{A}) = 1] \leq \epsilon.$$

Proof. The adversary can only manipulate the data structure through θ and $\mathcal{L}$. The challenger simulates all functions in the oracle for the adversary. We show that a successful adversary $\mathcal{A}$ effectively removes an entry after it was added to the Merkle BST, which leads to a contradiction in append only ledger $\mathcal{L}$ or by Lemma 3, leads to a hash collision.

Suppose $\mathcal{A}$ wins $\text{Exp}^{\text{non-removable}}_{\theta,\mathcal{L}}(\mathcal{A})$ from the case in line 7 in the experiment. In line 4, c is added to ds and L. This means, $\mathcal{A}$ has managed to remove the entry c, and when challenger calls $\text{GetAllCerts}_D(d)$ which outputs a list of all entries

logged, in which c is not included. In line 17 of $\text{AddCert}_D(d, c)$ (in Figure 4.9) the c is inserted in to $\mathcal{L}$. This implies a contradiction of the definition of ideal ledger functionality which is append only, since $\mathcal{A}$ has no means to manipulate the functionalities outside oracle access. Thus $\mathcal{A}$ can never win in this case at all.

Suppose $\mathcal{A}$ wins $\text{Exp}_{\theta,\mathcal{L}}^{\text{non-removable}}(\mathcal{A})$ from the case in line 10. By correctness property of $DPKIT$ in Definition 11, we claim that when AddCert is called, the entry will be added and subsequent Search will return a value of the entry. But line 10 implies an empty value. This is a contradiction on oracle functionalities, meaning that no $\mathcal{A}$ will win here either.

Finally, suppose $\mathcal{A}$ wins $\text{Exp}_{\theta,\mathcal{L}}^{\text{non-removable}}(\mathcal{A})$ from the case in lines 13 or 15. This implies that an entry is included for the domain name but it is neither c' nor r'. This immediately gives a collision for the certificate fingerprint hash function. By assumption, no $\mathcal{A}$ wins in this case with probability higher than ϵ. □

The theorem 6 shows that if a particular entry is invalid or not yet logged, it is difficult for an adversary to prove otherwise.

Theorem 6 (Proof consistency)**.** *If hash function* H *is collision-resistant, then in DPKIT scheme θ (with hash function H) in ideal ledger model $\mathcal{L}$, no malicious entity can present a proof for an entry which is invalid or not logged. More precisely, no $\mathcal{A}$ wins* $\text{Exp}_{\theta,\mathcal{L}}^{\text{entry-proof}}(\mathcal{A})$*, with probability higher than ϵ. That is,*

$$Pr[\text{Exp}_{\theta,\mathcal{L}}^{\text{entry-proof}}(\mathcal{A}) = 1] \leq \epsilon.$$

Proof. We show that a successful adversary $\mathcal{A}$ effectively produces a valid proof for an entry, which does not exist in the log, which, by Lemma 3, leads to a hash collision.

The provided proof P must be valid for winning conditions. Suppose $\mathcal{A}$ wins $\text{Exp}_{\theta,\mathcal{L}}^{\text{entry-proof}}(\mathcal{A})$ from condition in line 8 in the experiment. This implies that the entry does not exist in current log. But by Definition 13, SearchValue() returns an updated state of the log, which should return the entry value. This implies a contradiction in definition of append only ideal ledger functionality. Thus, no $\mathcal{A}$ wins in this case.

In lines 10 and 12, c is checked to see its status of revocation. Suppose $\mathcal{A}$ wins $\text{Exp}_{\theta,\mathcal{L}}^{\text{entry-proof}}(\mathcal{A})$ in line 11 or 12. Lines 11 and 12 suggest that, included entry is not the same entry which the proof stands for whether it is revoked or not.

This immediately gives a collision for the certificate fingerprint hash function. Precisely, no $\mathcal{A}$ wins in this case with probability higher than ϵ. □

The theorem 7 shows that once revocation information is logged for a certificate, it is difficult for an adversary to hide or remove it from the system.

Theorem 7 (Revocation reveal)**.** *If hash function* H *is collision-resistant, then in DPKIT scheme θ (with hash function H) in ideal ledger model $\mathcal{L}$, when a client requests a proof for an entry, revocation information should be included if there exists any corresponding revocation certificate. In other words, no malicious entity should be able to hide revocation certificate of a particular entry. More precisely, no $\mathcal{A}$ wins* $\mathrm{Exp}^{\mathsf{show\text{-}revoke}}_{\theta,\mathcal{L}}(\mathcal{A})$*, with probability higher than ϵ.*

$$Pr[\mathrm{Exp}^{\mathsf{show\text{-}revoke}}_{\theta,\mathcal{L}}(\mathcal{A}) = 1] \leq \epsilon.$$

Proof. The challenger simulates all functions in the oracle for the adversary as it can only interact with the data structure through θ. We show that a successful adversary $\mathcal{A}$ effectively conceal a revocation record for an entry in its proof when an actual revocation record is stored in the data structure, which by Lemma 3, leads to a hash collision.

Adversary $\mathcal{A}$ first outputs a valid entry and its corresponding revocation information. In line 4 of $\mathrm{Exp}^{\mathsf{show\text{-}revoke}}_{\theta,\mathcal{L}}(\mathcal{A})$, the adversary loses if the fingerprint of the revocation entry is not equal to that of certificate output. A revocation entry is then added to the system. The adversary then outputs a proof P for this particular entry. Line 8 retrieves the current fingerprint of data structure. Suppose $\mathcal{A}$ wins $\mathrm{Exp}^{\mathsf{show\text{-}revoke}}_{\theta,\mathcal{L}}(\mathcal{A})$ in line 9. This implies that the $\mathcal{A}$ has provided a valid proof for the already logged entry.

After winning line 9, adversary has to win line 10 as well to win the security game. Suppose $\mathcal{A}$ wins $\mathrm{Exp}^{\mathsf{non\text{-}removable}}_{\theta,\mathcal{L}}(\mathcal{A})$ in line 10. This implies that the challenger could not find a revocation represented by the given proof. This immediately leads to a hash collision by Lemma 3 as it infers that the $\mathcal{A}$ has managed to provide a valid proof for the entry c but the entry logged in corresponding position is not the same entry indicated by the proof. That is, no $\mathcal{A}$ wins in this case with probability higher than ϵ. □

4.7 Discussion

We have proposed, modelled, and instantiated a new decentralised approach to certificate transparency based on generic blockchains augmented with efficient data structures. It is compatible with existing PKI as customary entities are kept to manage basic functionalities. DPKIT is expected to be highly effective at catching and deterring trust abuses by malicious certificate authorities. Most of the proposed blockchain-based PKI systems allow for validation against real world entities. In DPKIT, validation against real world identities is done as in traditional PKI, by certificate authorities as part of the process of issuing certificates. Furthermore, in DPKIT certificates and revocation certificates are issued by certificate authorities exactly as is currently done in traditional PKI. DPKIT does not by itself create or revoke certificates, but logs the existence of creation and revocation certificates.

Now we evaluate DPKIT based on design goals set in Section 4.5.1.

- **Transparency**: Once a certificate is logged as a transaction on DPKIT distributed ledger, it is open to viewing by anyone who connects and enables verification.
- **Multiple certificates**: DPKIT achieves this by having secondary a Merkle BST storing every available and visible certificate issued for a certain domain.
- **Multiple domain names**: DPKIT provides the means to handle a certificate with multiple domain names by having separate nodes representing each of those names and then by making their secondary trees point to the same tuple that store the particular certificate.
- **No single point of failure**: There is no central authority that grants access or verification for the certificates logged, so there is no single point of failure in this system. Data is shared across a peer-to-peer network and is continually reconciled.
- **Scalability**: DPKIT provides short proofs. Clients need the root hash to verify the proof. Since a MBST with n nodes needs to provide only $\mathcal{O}(log\ n)$ hashes to reconstruct the root hash, the large number of certificate logged does not affect the efficiency of DPKIT, thus it can be efficiently scaled to support Internet users.

- **Efficiency**: DPKIT keeps basic PKI entities and adds one extra, a peer. Servers have to interact with peers to obtain a proof of certificate membership for its certificate, thus having a low impact on TLS servers.
- **Client privacy**: In a client-server connection, a server sends the certificate along with the proof of membership to the client. The client then has to fetch only the recent root hash from a DPKIT peer to verify the certificate. This procedure keeps clients' browsing habits maximally private from observers and third parties.
- **Revocation**: DPKIT records revocation certificates. Once recorded, its presence is noted in the proof of certificate membership and will notify the clients when they verify the certificate.

Based on statistics collected by VeriSign, the first quarter of 2018 ended with approximately 333.8 million domain name registrations across all top-level domains. There is an increase of approximately 1.4 million domain name registrations, compared to the fourth quarter of 2017 [VER18]. Domains are typically registered and renewed on a yearly basis. If we assume that each registered domain name is either renewed or revoked once a year, DPKIT will process nearly 334 million transactions per year, or 6360 transactions during a 10 minute time slot.

As of July 2018, Bitcoin blockchain has nearly 328 million of transactions stored. When the total size of all block headers and transactions are considered, it has a size of nearly 171 GB [BL18]. Based on these facts, we can assume that DPKIT will require similar storage requirements and would increase over time when more domains are registered. However, the number of certificates issued/revoked per day is far less than the number of Bitcoin transactions occurring in a day. DPKIT is expected to be more efficient than Bitcoin blockchain operations. Furthermore, a tree with n nodes needs to provide only $\mathcal{O}(log\ n)$ hashes to reconstruct the root hash and only Merkle root hash is stored in the blockchain headers. Thus, DPKIT structure is not expected to heavily contribute to scalability issues of blockchain.

DPKIT provides a strongly immutable record of all issued and revoked certificates that are seen by a community of users and are requested to be added. For a particular domain name, multiple valid certificates are recorded. Whenever a new certificate is issued or revocation update for an existing certificate is recorded,

the root hash of the primary Merkle BST changes, which changes the validity proof held by peers and clients. This is a requirement in DPKIT to avoid any misbehaviours from peers and users. To reduce overhead, web servers could request the proof in time intervals and keep it for a certain period like 24 hours. Thus, clients could use the cached data of DPKIT and keep updating every 24 hours. Between the daily updates, if a client come by an updated/different root hash to their own copy, they can cross-check the authenticity among few peers and start using the most recent root hash and continue.

DPKIT is proposed as a fully decentralised approach to Certificate Transparency. When comparing relative advantages between CT and DPKIT, it is evident that DPKIT has achieved its design goals and a step further ahead from CT. Unlike CT, DPKIT does not rely on third parties to monitor logs. Once a certificate is recorded on the blockchain it is open for anyone to view thus allowing public auditing. Any malicious entity wanting to record a fraudulent certificate will have a hard time as it is difficult to get accepted by the majority in order to be recorded as a transaction. We believe DPKIT handle multiple certificates for a domain much better than CT does as they are recorded under one Merkle BST. Although it has been proposed [LK12], CT does not have any revocation mechanism built in yet, wherein DPKIT we have mechanisms for handling revoked certificates.

Public-key infrastructures enable users to look up and verify each other's public keys based on identities. An identity, to the most possible extent, should only be issued, updated or revoked after the permission from owner of that identity. The current approaches to PKIs are vulnerable in terms of identity retention as they do not effectively prevent one user from registering another's already registered public key as their own. Lately, various approaches [FVY14, QHW+17] have been proposed to guarantee strong identity retention properties which do not rely on a trusted third party. *Identity retention* is the inability of a user to impersonate an identity already registered to someone else. As for now, we have not incorporated an identity retention mechanism in DPKIT. A prospective method to obtain this property is having certificate updates (issuance and/revocation) cross signed by the domain owner's signature under their private key.

Additionally, we suggest having a domain policy similar to [MR16, Dec16], which allows a holder of a domain to specify which CAs are trusted. The domain policy of a domain can be easily verified when identity retention is enabled and it

provides an extra measure to validate the certificates. Having a domain policy supports detecting attacks. Basically, an attack on the system is when a fraudulent certificate is added to an already registered identity as an unauthorised update. This is possible when the authorised CA for that domain is compromised and issuing a second certificate for a different public key or a rogue CA is trying to issue a new certificate for the domain under a new public key. If the domain owners have the choice to declare the CAs authorised to issue certificates for their domain, an attempt of a rogue CA would be unsuccessful and their fraudulent certificate will be rejected. Dealing with a situation where an authorised CA is compromised and issuing a second certificate for a domain could be an issue, especially when multiple certificates for a single domain are allowed. It can be prevented by proper validation and by implementing identity retention. Requiring the owner's signature for any update will alleviate such incidents since an adversary has to take control over the CA and access to domain owner's private key at the same time.

Constructing a blockchain-based PKI is a feasible alternative method for the mainstream issues related to conventional and log based approaches. We present DPKIT as a platform that offers more transparency and decentralisation than existing PKI. Based on the security notions we formalised, we have shown that our scheme prevents some misbehaviour of entities involved. Having certificate revocation built in our system is a benefit that is not provided by current PKI. As revocation information is included in the proof of certificate itself, it mitigates the need for a centrally administered OCSP servers run by CAs.

We present an abstract model for DPKIT which can be applied similar blockchain-based logging schemes. However, our model is not without limitations. For instance, we provide cryptographically assured security properties. Thus, these properties should be carefully evaluated for a real world implementation. Moreover, our model of DPKIT does not represent some functionalities of the involved entities (CAs, domain owners, web servers and web clients). As a result, a real world implementation would involve more constraints than our abstract model of DPKIT. We have omitted a thorough analysis of security properties of a blockchain as we only view it as a black-box oracle that lets users append, share and replicate small and infrequent amounts of data. Hence, the practical aspects of blockchains have to be considered if our abstract model is used for a real world implementation.

Chapter 5

Transitioning to a Quantum-Resistant Public-Key Infrastructure

Public-Key Infrastructure play a huge role in secure communication over insecure channels, allowing the users of Internet-based applications to authenticate each other and exchange data. This is mainly obtained by communicating via the TLS protocol [DR06] with the use of X.509 certificates [CSF+08]. Obtaining these X.509 digital certificates, that is transitioning from HTTP to HTTPS, used to be a complex and expensive task. But thanks to Certificate Authorities like Let's Encrypt and Comodo, offering automated, domain-validated certificates at no cost to end users, having digital certificates is not an obstacle any more. However, this demand for public-key certificates attracts more sophisticated cyber attacks. Further, it is not cost effective or even practical, due to scalability issues, to replace web PKI with proposed alternatives like web of trust or certificate-less public-key cryptography. As for now, the best option left is to enhance the current web PKI and overcome its limitations.

In the last two chapters, we have discussed the evolution of PKI to a potentially new state in current world; from log-based to blockchain-based. At present, blockchain-based PKI technologies seems a promising improvement to the existing issues in web PKI trust model. But what about the future? More specifically, how would the PKI trust model in its current state face in the world of quantum

computing? In this chapter, we take a step further to address possibilities of PKI thriving in an era with quantum computing facilities. According to the European Union's Quantum Manifesto [DTMH+16], the world will see a fully functioning quantum computer by 2035. Looking at recent evolutions in history of cryptography, it is not too soon to start the transition of PKI to quantum era. It is evident that the technical world takes a long time to adopt to new systems. A fine example of this is the deprecation of SHA1 in 2017, since the first theoretical attack on it in 2005 [WYY05]. In this chapter, we investigate a hybrid digital signature scheme which employs both traditional and post-quantum algorithms. Then we examine the use of this signature scheme in three real-world standards that involve digital signatures and public-key infrastructure. Finally, we analyse the backwards-compatibility of these standards and present the results.

5.1 Introduction

Until the late 20th century, cryptography was mostly considered an art. Constructing good codes, or breaking new ones, relied on creativity. This helped to develop a sense of how codes work but there was little theory to rely on. Beginning in end of 1970s and early 1980s this picture of cryptography radically changed with the introduction of modern symmetric and public-key cryptography. Since then there have been very few changes in the use of algorithms, or rather transitions from one widely deployed algorithm to another. These include adaptation from DES and Triple-DES to AES; from MD5 and SHA-1 to the SHA-2 family; from RSA key transport and finite field Diffie–Hellman to elliptic curve Diffie–Hellman key exchange; and from RSA and DSA certificates to ECDSA certificates. Some of these transitions have been successful. For example, AES is used worldwide as DES is considered insecure and has been withdrawn as a standard by National Institute of Standards and Technology (NIST). Moreover some modern communication protocols now use ECDH key exchange. Though this seems promising in terms of cryptography in general, some applications have not done so well with switching to secure schemes. For instance, public-key infrastructure has a more mixed record. It is evident that browser vendors and CAs have had a long transition period from SHA-1 to SHA-2 in certificates, with repeated delays of deadlines, although it is long been known that SHA-1 is at risk [WYY05]. In addition, the transition to elliptic curve certificates has been

even slower, and still today a vast majority of certificates issued for the web use RSA key algorithm.

Well organised timely transitions ensure secure communication over the Internet. However, with the prospect of quantum computers, another inevitable transition is to *post-quantum* public-key cryptography. Post-quantum cryptography is crucial as most of the public-key cryptography that is used on the Internet today is based on algorithms that are vulnerable to quantum attacks. These include public-key algorithms such as RSA, ECC, Diffie-Hellman and DSA, which are considered easily broken by Shor's algorithm and are regarded to be insecure against quantum computers. In order to preserve secure communication that is reliant on cryptography, it is mandatory to identify new mathematical techniques on which cryptography can be built. Some aspects of the post-quantum transition will be straightforward. For instance in TLS, a variety of key exchange protocols based on post-quantum primitives have been tested and demonstrated to be successful [BCNS15, Bra16]. Implementation results show that the performance of key-exchange is quite competitive and can be adopted piecewise. However, this is not the case with most other technologies. The migrations will be harder, especially when it is difficult for old and new configurations to operate simultaneously. A recent whitepaper [Cam15] discusses some of these issues at a high level.

The transition to post-quantum cryptography is further complicated by the relative immaturity of some of the underlying mathematical assumptions in current candidates. Since these new techniques have not been studied and tested for very long, there is a risk that they might be insecure against quantum attacks. This motivates *hybrid* approaches, in which both a traditional algorithm and one or more post-quantum algorithms are used in parallel. When implemented securely, it would allow early adopters to have the potential of quantum-safe cryptographic techniques without having to abandon the security provided by existing mechanisms. The benefit in this method is that as long as one of the schemes remains unbroken, confidentiality or authenticity would be ensured. These speculations lead us to three research questions:

1. What are the appropriate security properties for hybrid digital signatures?
2. How should we combine signature schemes to construct hybrid signatures?
3. How can hybrid signatures be realised in popular standards and software, ideally in a backwards-compatible way?

In this chapter, we mainly focus on first and third questions. We introduce hybrid signatures and define the security notions depending on how quantum the adversary is. Much of the following has been reproduced from our work **Transitioning to a Quantum-Resistant Public-Key Infrastructure**, published in *Proceedings of International Workshop on Post-Quantum Cryptography* (PQCrypto 2017) [BHMS17]. Some sections of the published paper (quantum computation proof and signature combiner proofs) are not fully documented in this chapter as they are the work of our co-authors. As we investigate hybrid schemes constructed using signature combiners, they are briefly mentioned for completeness. Our goal is to provide a guide for how to transition to hybrid post-quantum digital signatures in various standards and software. We consider three popular standards in our experiments based on their rate of applicability in present secure communication technologies using public-key cryptography.

X.509 for certificates [SHF02]: The majority of certificates issued by commercial CAs contain RSA public-keys and only a few are starting to use elliptic curve public-keys for their certificates. There are currently no CAs issuing certificates for quantum-safe public-keys or signing certificates with a quantum-safe signature scheme.

Transport Layer Security protocol [DR08]: The handshake sub-protocol, which authenticates and establishes shared secret keys mostly involve public-key operations. Since X.509 certificates containing RSA keys are used to authenticate the majority of servers, authentication in TLS is vulnerable to quantum attacks.

Cryptographic Message Syntax (CMS) and S/MIME [Hou09]: Sending secure email messages using CMS as part of Secure/Multipurpose Internet Mail Extensions (S/MIME) [RT10] requires the use of a digital signature or a X.509 certificate. Algorithms used for digital signatures such as DSA or RSA provides inadequate security in presence of a quantum computer.

It can be a challenge to modify an established standard against quantum attacks because not only the security aspect should be considered but also the non-security issues like adoption rates and backwards compatibility. Observing these security protocols, it is apparent that some of them require fundamental messaging and data structure changes to protect them from quantum threats. As it might take

a long time to fully convert into quantum-safe techniques, we believe a hybrid approach will make a smooth transition. For each standard, we ask:

3.a) How can hybrid / multiple signature schemes be used in the standard?

3.b) Is this approach backwards-compatible with old software?

3.c) Are there potential problems involving varying public-key or signature sizes?

We identify promising techniques for hybrid X.509 certificates, and hybrid S/MIME signed messages; using multiple signature algorithms in TLS does not seem immediately possible, though one mechanism in the current draft of TLS 1.3 seems to allow multiple client authentications and a recent proposal could allow multiple server authentications. In Section 5.2 we discuss the security notions for hybrid signatures. We talk about signature combiners in Section 5.3 and in Section 5.4 we address post-quantum signature schemes. Then in Section 5.5, we discuss hybrid signatures in standard and in Section 5.6 we conclude the chapter.

5.2 Security notions for Hybrid Digital Signatures

Digital signatures are a standard element of most cryptographic protocol suites and are used commonly in many cases when authentication, integrity and non-repudiation is needed. Most early signature schemes involve the use of a trapdoor permutation such as the RSA function. When used directly this type of scheme is found to be susceptible to a key-only existential forgery attack. Therefore in practice to avoid such attacks, the message to be signed is first hashed to produce a short digest which is then signed. If a forgery attack is attempted on such a message, it would only produce a hash function output that corresponds to the signature, but not the actual message that was used to create that hash output in the first place. Thus, the widely accepted security notion for digital signatures is *unforgeability under chosen message attack* (EUF-CMA): the adversary interacts with a signing oracle to obtain signatures on any desired messages, and then must output a forgery on a new message. When constructing hybrid signatures for post-quantum cryptosystems they should retain this particular property where the signature is existentially unforgeable, even against a chosen-plaintext

attack. We formally defined the signature scheme unforgeability and security experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathsf{SIG}}$ for *existential unforgeability under chosen-message attacks* in Section 2.1.2 Figure 2.1.

In a typical post-quantum context, the adversary has a quantum computer at their disposal, but only gets classical access to the cryptographic scheme. However, if adversaries have a quantum computer and also get quantum access—that is for instance, gain the ability to encrypt or authenticate in superposition—then they can easily break many cryptosystems currently believed to be safe from quantum attacks. Against such settings, Boneh and Zhandry [BZ13] initiated the study of quantum-secure digital signatures and quantum chosen ciphertext security. They have proposed security notions for digital signature schemes against quantum adversaries, and given a quantum analogue of existential unforgeability under chosen message attacks (EUF-CMA). This model allows the adversary to issue quantum chosen message queries and for superposition of messages, the adversary receives a superposition of signatures on those messages. Basically a quantum adversary is allowed to interact with a quantum signing oracle and is able to receive signatures on quantum states of its choosing.

As we consider a smooth transition to post-quantum digital signatures, Boneh and Zhandry's definition might be excessively strong. For example, we might choose a signature scheme to be used for the next five years and we are confident that no adversary would gain quantum powers during this period to attack our system. Moreover, we will definitely not sign any messages in superposition. But later, the adversary might gain access to a quantum computer and might have the ability to interact with our chosen scheme in superposition. Thus, we need to consider several security notions depending on how quantum the adversary is. We use the notation $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ to denote the adversary's type with respect to three options:

- X: whether the adversary is classical ($\mathsf{X} = \mathsf{C}$) or quantum ($\mathsf{X} = \mathsf{Q}$) *during* the period in which it can interact with the signing oracle;
- y: whether the adversary can interact with the signing oracle classically ($\mathsf{y} = \mathsf{c}$) or quantumly ($\mathsf{y} = \mathsf{q}$); and
- Z: whether the adversary is classical ($\mathsf{Z} = \mathsf{C}$) or quantum ($\mathsf{Z} = \mathsf{Q}$) *after* the period in which it can interact with the signing oracle.

We use the notion $A \implies B$, in which if a security scheme satisfies property

A, it also satisfies property B. Security notions we have defined here, form a natural hierarchy, ($\mathsf{Q^qQ} \implies \mathsf{Q^cQ} \implies \mathsf{C^cQ} \implies \mathsf{C^cC}$) with separations between each level of the hierarchy. We present it as a hierarchy of intermediate notions, differentiating how the honest parties and adversary act in this environment. Would honest parties sign classical or quantum messages? Does the adversary have access to a classical or quantum oracle? Is the adversary classical or quantum during the period it has access to the signing oracle? Are we concerned about a quantum adversary only in the future or also now? We distinguish between these conceptual outlines through the hierarchy of security notions as follows:

1. *Fully classical* ($\mathsf{C^cC}$): The adversary at all times has access only to a classical computer, and obtains signatures from a classical signing oracle and possibly hash values from a classical random oracle [BR93]. This corresponds to the traditional EUFCMA notion.

2. *Future quantum* ($\mathsf{C^cQ}$): The adversary uses only a classical computer during the period of time it is interacting with the (classical) signing oracle. At a later point in time, the adversary may have access to a quantum computer, but no longer has access to a signing oracle. However, the adversary has quantum access to a hash oracle in the random oracle mode [BR93].

3. *Quantum adversary, classical queries* ($\mathsf{Q^cQ}$) : The adversary at all times can use a quantum computer. However, the honest participants in the system only ever use their signing keys on classical computers, so the signing oracle is always accessed classically.

4. *Fully quantum* ($\mathsf{Q^qQ}$): The adversary at all times can use a quantum computer, obtains signatures from a quantum signing oracle, and obtains hash values from a quantum random oracle. This corresponds to Boneh and Zhandry's notion.

$\mathsf{C^cQ}$ above models the scenario of a system which is in use today: parties use classical computers to sign documents. The adversary also has only classical power. However, over time when parties eventually stop signing new documents, the documents signed with the system need to remain unforgeable for a long time, even after quantum computers become available.

When considering the $\mathsf{X^yZ}$ notions, combinatorially, there are $2^3 = 8$ possibilities for various environments and positions the adversary could interact with the

$\underline{\text{Expt}_{\Sigma}^{\mathsf{X^yZ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2)}$:

1 $q_S \leftarrow 0$
2 $(sk, vk) \leftarrow\!\!\$\ \Sigma.\mathsf{KeyGen}()$
3 $st \leftarrow\!\!\$\ \mathcal{A}_1^{\mathcal{O}_S(\cdot)}(vk)$
4 $((m_1^*, \sigma_1^*), \ldots, (m_{q_S+1}^*, \sigma_{q_S+1}^*)) \leftarrow\!\!\$\ \mathcal{A}_2(st)$
5 If $(\Sigma.\mathsf{Verify}(vk, m_i^*, \sigma_i^*) = 1\ \forall\ i \in [1, q_S + 1])$
$\wedge\ (m_i^* \neq m_j^*\ \forall\ i \neq j)$:
6 Return 1
7 Else return 0

$\underline{\text{Classical signing oracle } \mathcal{O}_S(m)}$:

8 $q_S \leftarrow q_S + 1$
9 $\sigma \leftarrow\!\!\$\ \Sigma.\mathsf{Sign}(sk, m)$
10 Return σ to $\mathcal{A}$

$\underline{\text{Quantum signing oracle } \mathcal{O}_S(\sum_{m,t,z} \psi_{m,t,z}|m, t, z\rangle)}$:

11 $q_S \leftarrow q_S + 1$
12 $r \leftarrow\!\!\$\ \mathcal{R}_{\Sigma}$
13 Return state $\sum_{m,t,z} \psi_{m,t,z}|m, t \oplus \Sigma.\mathsf{Sign}(sk, m; r), z\rangle$ to $\mathcal{A}$

Figure 5.1: Unified security experiment for $\mathsf{X^yZ\text{-}eufcma}$ in the standard model: existential unforgeability under chosen-message attack of a signature scheme Σ for a two-stage adversary $\mathcal{A}_1$ (of type X), $\mathcal{A}_2$ (of type Z) with signing oracle of type y; if $\mathsf{y} = \mathsf{c}$ then $\mathcal{A}_1$ has classical access to the signing orcale, otherwise quantum access.

signing oracle. However, some of the combinations do not make sense such as $\mathsf{C^qZ}$ or $\mathsf{Q^yC}$. Therefore, we consider the family of notions $\mathsf{C^cC}$-, $\mathsf{C^cQ}$-, $\mathsf{Q^cQ}$-, $\mathsf{Q^qQ}$-eufcma which form a natural hierarchy.

Figure 5.1 shows our unified definition for EUF-CMA parameterised for any of the four types of adversaries in the standard model. In the standard model the adversary has no access to a random (or hash) oracle. It follows the EUF-CMA formulation of Boneh and Zhandry [BZ13] but separates out the adversary to be a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ (of type X) interacts with either a signing oracle (of type y) and outputs an intermediate state st (of type X), which $\mathcal{A}_2$ (of type Z) then processes. The input to $\mathcal{A}_1$ and the output of $\mathcal{A}_2$ are always classical.

Figure 5.2 shows how the experiment is altered in the classical or quantum random oracle model: at the start of the experiment, a random function H is sampled uniformly from the space of all such functions $\mathcal{H}_{\Sigma}$. In the classical setting, it is common to formulate the random oracle using lazy sampling, but such a formulation does not work in the quantum setting. For simplicity, we assume the adversary has quantum random oracle access whenever it is quantum.

We define advantage as $\text{Adv}_{\Sigma}^{\mathsf{X^yZ\text{-}eufcma}}(\mathcal{A}) = \Pr\left[\text{Expt}_{\Sigma}^{\mathsf{X^yZ\text{-}eufcma}}(\mathcal{A}) = 1\right]$.

Figure 5.3 shows the implications and separations between these notions.

$\underline{\mathrm{Expt}^{\mathsf{X^yZ\text{-}eufcma}}_{\Sigma}(\mathcal{A}_1,\mathcal{A}_2)}$:

0 $H \leftarrow\!\!\$\ \mathcal{H}_\Sigma$
1 $q_H \leftarrow 0,\ q_S \leftarrow 0$
2 $(sk, vk) \leftarrow\!\!\$\ \Sigma.\mathsf{KeyGen}()$
3 $st \leftarrow\!\!\$\ \mathcal{A}_1^{\mathcal{O}_S(\cdot),\mathcal{O}_H(\cdot)}(vk)$
4 $((m_1^*,\sigma_1^*),\ldots,(m_{q_S+1}^*,\sigma_{q_S+1}^*)) \leftarrow\!\!\$\ \mathcal{A}_2^{\mathcal{O}_H(\cdot)}(st)$
5 *// continues as in Figure 5.1*

$\underline{\text{Classical random oracle } \mathcal{O}_H(x)}$:

14 $q_H \leftarrow q_H + 1$
15 Return $H(x)$

$\underline{\text{Quantum random oracle } \mathcal{O}_H(\sum_{x,t,z}\psi_{x,t,z}|x,t,z\rangle)}$:

16 $q_H \leftarrow q_H + 1$
17 Return state $\sum_{x,t,z}\psi_{x,t,z}|x,t\oplus H(x),z\rangle$

Figure 5.2: $\mathsf{X^yZ\text{-}eufcma}$ experiment in the classical and quantum random oracle models; if $\mathsf{X} = \mathsf{C}$, then $\mathcal{A}_1$ has classical access to the random oracle, otherwise quantum access; similarly for Z and $\mathcal{A}_2$.

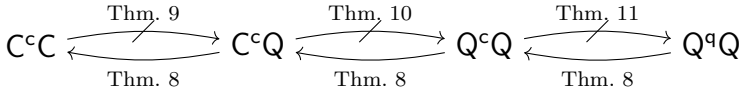


Figure 5.3: Implications and separations between unforgeability notions ($\mathsf{X^yZ\text{-}eufcma}$) for signature schemes. $B \to A$ denotes an implication: every B-secure scheme is also A-secure. $A \not\to B$ denotes a separation: there exist A-secure schemes that are not B-secure.

These implications generate an ordering on security notions, so we sometimes write $\mathsf{C^cC} \le \mathsf{C^cQ}$ etc. The stronger the security notion the smaller the advantage of an adversary $\mathcal{A}$ breaking a signature scheme of corresponding security. For example, let the signature scheme Σ be $\mathsf{C^cQ}$-secure. $\mathsf{C^cQ}$ is stronger than $\mathsf{C^cC}$, i.e., $\mathsf{C^cQ} \ge \mathsf{C^cC}$. Hence, $\mathrm{Adv}^{\mathsf{C^cQ\text{-}eufcma}}_{\Sigma}(\mathcal{A}) \ge \mathrm{Adv}^{\mathsf{C^cC\text{-}eufcma}}_{\Sigma}(\mathcal{A})$. Similarly we use $\max\{\cdot,\cdot\}$ based on this ordering.

The implications in Figure 5.3 are straightforward. Each of the separations $A \not\Longrightarrow B$ follows from a common technique: from an A-secure scheme Σ, construct a (degenerate) A-secure scheme Σ' that is not B-secure, because the additional powers available to a B-adversary allow it to recover the secret signing key of Σ that was cleverly embedded somewhere in Σ'.

- $\mathsf{C^cC\text{-}eufcma} \not\Longrightarrow \mathsf{C^cQ\text{-}eufcma}$: In the public-key for Σ', include a copy of the signing secret key encrypted using an RSA-based public-key encryption scheme. Assuming breaking RSA is classically hard, the encrypted signing key is useless to a $\mathsf{C^cC}$-adversary, but a $\mathsf{C^cQ}$-adversary will be able to break the public-key encryption, recover the signing key, and forge signatures.

- $\mathsf{C^cQ\text{-}eufcma} \not\Longrightarrow \mathsf{Q^cQ\text{-}eufcma}$: In the public-key for Σ', include an RSA-

encrypted random challenge string, and redefine the $\Sigma'.\mathsf{Sign}$ so that, if the adversary queries the signing oracle on the random challenge string, the signing key is returned. Assuming breaking RSA is hard for a classical algorithm, a $\mathsf{C^cQ}$-adversary will not be able to recover the challenge while it has access to the signing oracle, and thus cannot make use of the degeneracy to recover the signing key; a $\mathsf{Q^cQ}$ adversary can.

- $\mathsf{Q^cQ\text{-}eufcma} \not\Longrightarrow \mathsf{Q^qQ\text{-}eufcma}$: Here we hide the secret using a query-complexity problem that can be solved with just a few queries by a quantum algorithm making queries in superposition, but takes exponential queries when asking classical queries. The specific problem we use is a variant of the *hidden linear structure* problem by Beaudrap et al. [dBCW02b].

In the following we state and prove $\mathsf{C^cC\text{-}eufcma} \not\Longrightarrow \mathsf{C^cQ\text{-}eufcma}$ and $\mathsf{C^cQ\text{-}eufcma} \not\Longrightarrow \mathsf{Q^cQ\text{-}eufcma}$ statements formally. The $\mathsf{Q^cQ\text{-}eufcma} \not\Longrightarrow \mathsf{Q^qQ\text{-}eufcma}$ statement is proved formally in Bindel et al.[BHMS17] and we state the theorem here for reference.

Theorem 8 ($\mathsf{Q^qQ} \Longrightarrow \mathsf{Q^cQ} \Longrightarrow \mathsf{C^cQ} \Longrightarrow \mathsf{C^cC}$)**.** *If Σ is a $\mathsf{Q^qQ\text{-}eufcma}$-secure signature scheme, then Σ is also $\mathsf{Q^cQ\text{-}eufcma}$-secure. If Σ is a $\mathsf{Q^cQ\text{-}eufcma}$-secure signature scheme, then Σ is also $\mathsf{C^cQ\text{-}eufcma}$-secure. If Σ is a $\mathsf{C^cQ\text{-}eufcma}$-secure signature scheme, then Σ is also $\mathsf{C^cC\text{-}eufcma}$-secure.*

Proof. Suppose $(\mathcal{A}_1, \mathcal{A}_2)$ is an adversary against $\mathsf{C^cC\text{-}eufcma}$: both $\mathcal{A}_1$ and $\mathcal{A}_2$ are classical. Every classical algorithm is also a quantum algorithm, and thus they win the $\mathsf{C^cQ\text{-}eufcma}$ experiment with at least the same probability as they win the $\mathsf{C^cC\text{-}eufcma}$ experiment: $\mathrm{Adv}_{\Sigma}^{\mathsf{C^cQ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2) \geq \mathrm{Adv}_{\Sigma}^{\mathsf{C^cC\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2)$. Similarly, $\mathrm{Adv}_{\Sigma}^{\mathsf{Q^cQ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2) \geq \mathrm{Adv}_{\Sigma}^{\mathsf{C^cQ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2)$.

Finally, an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ against $\mathsf{Q^cQ\text{-}eufcma}$ is also an adversary against $\mathsf{Q^qQ\text{-}eufcma}$ that simply does not query its oracle in superposition, and thus $\mathrm{Adv}_{\Sigma}^{\mathsf{Q^qQ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2) \geq \mathrm{Adv}_{\Sigma}^{\mathsf{Q^cQ\text{-}eufcma}}(\mathcal{A}_1, \mathcal{A}_2)$

□

In Theorem 9, we show that if there exists a signature scheme that is hard for a classical adversary, it is $\mathsf{C^cC\text{-}eufcma}$-secure. But when the adversary gains quantum abilities, a scheme that is $\mathsf{C^cC\text{-}eufcma}$-secure will not necessarily be $\mathsf{C^cQ\text{-}eufcma}$-secure.

Theorem 9 ($\mathsf{C^cC} \not\Rightarrow \mathsf{C^cQ}$)**.** *If the RSA problem is hard for classical computers and there exists a signature scheme Σ that is* $\mathsf{C^cC}$-eufcma*-secure, then there exists a signature scheme Σ' that is* $\mathsf{C^cC}$-eufcma*-secure but not* $\mathsf{C^cQ}$-eufcma*-secure.*

Proof. Let Π be a public-key encryption scheme that is IND-CPA-secure against classical adversaries and whose security relies on the hardness of the RSA problem, e.g., [FS00] or OAEP [BR95]. However, a quantum adversary could use Shor's algorithm to factor the modulus and decrypt ciphertexts encrypted using Π. We construct a scheme Σ' that is based on Σ, but the public-key of Σ' includes a Π-encrypted copy of the Σ secret key:

- $\Sigma'.\mathsf{KeyGen}()$: $(sk, vk) \leftarrow_\$ \Sigma.\mathsf{KeyGen}()$. $(dk, ek) \leftarrow_\$ \Pi.\mathsf{KeyGen}()$. $c \leftarrow_\$ \Pi.\mathsf{Enc}(ek, sk)$. $vk' \leftarrow (vk, ek, c)$. Return (sk, vk').
- $\Sigma'.\mathsf{Sign}(sk, m)$: Return $\Sigma.\mathsf{Sign}(sk, m)$.
- $\Sigma'.\mathsf{Verify}(vk' = (vk, ek, c), m, \sigma)$: Return $\Sigma.\mathsf{Verify}(vk, m, \sigma)$.

The theorem then follows as a consequence of the following two claims, the proofs of which are immediate. □

Claim 1. *If Π is IND-CPA-secure against a classical adversary and Σ is* $\mathsf{C^cC}$-eufcma*-secure, then Σ' is* $\mathsf{C^cC}$-eufcma*-secure.*

Proof. The proof follows from a simple substitution. Since Π is IND-CPA-secure, no passive adversary can distinguish Π-encryptions of sk from encryptions of $0^{|sk|}$ with significant advantage. So we can replace c in vk' with an encryption of zeros, while still successfully simulating answers to the signing oracle in the $\mathsf{C^cC}$-eufcma experiment. A Σ' forgery is immediately a Σ forgery. □

Claim 2. *If there exists an efficient quantum adversary $\mathcal{A}$ against the message recovery of Π, then Σ' is not* $\mathsf{C^cQ}$-eufcma*-secure.*

Proof. This proof is obvious. Given Σ' verification key $vk' = (vk, ek, c)$, run $\mathcal{A}$ on ek and c to recover sk. We can now forge signatures in Σ' by using the signing algorithm with sk. □

In Theorem 10, we prove that if there exist a signature scheme that is hard to break for an adversary who was once classical but now has quantum access, then it is $\mathsf{C^cQ}$-eufcma-secure. However, a scheme that is $\mathsf{C^cQ}$-eufcma-secure is not necessarily $\mathsf{Q^cQ}$-eufcma-secure when the adversary is always quantum even with only classical access to the signing oracle.

Theorem 10 ($\mathsf{C^cQ} \not\Rightarrow \mathsf{Q^cQ}$)**.** *If the RSA problem is hard for classical computers and there exists a signature scheme Σ that is* $\mathsf{C^cQ}$-eufcma*-secure, then there exists a signature scheme Σ' that is* $\mathsf{C^cQ}$-eufcma*-secure but not* $\mathsf{Q^cQ}$-eufcma*-secure.*

Proof. Let Π be a public-key encryption scheme that is IND-CPA-secure against classical adversaries and whose security relies on the hardness of the RSA problem. However, a quantum adversary could use Shor's algorithm to factor the modulus and decrypt ciphertexts encrypted using Π. Our signature scheme Σ' is designed so that the signing oracle can be used by an online quantum adversary to learn the signing key, but not by an off-line quantum adversary. Recall that in the proof of Theorem 9, an encrypted copy of the secret signing key was provided to the adversary in the public verification key.

Here, we put an encrypted *random challenge* in the public verification key, and if the adversary asks for that challenge to be signed, we have the signing oracle return the signing key. Intuitively, only an adversary that can break the challenge while it has access to the signing oracle (i.e., a quantum stage-1 adversary) can solve the challenge. The scheme Σ' is shown below.

- $\Sigma'.\mathsf{KeyGen}()$: $(sk, vk) \leftarrow_\$ \Sigma.\mathsf{KeyGen}()$. $(dk, ek) \leftarrow_\$ \Pi.\mathsf{KeyGen}()$. $s^* \leftarrow_\$ \{0,1\}^{256}$. $ch \leftarrow \Pi.\mathsf{Enc}(ek, s^*)$. $vk' \leftarrow (vk, ek, ch)$. $sk' \leftarrow (sk, s^*)$. Return (sk', vk').
- $\Sigma'.\mathsf{Sign}(sk' = (sk, s^*), m)$: If $m = s^*$, return sk. Else, return $\Sigma.\mathsf{Sign}(sk, m)$.
- $\Sigma'.\mathsf{Verify}(vk' = (vk, ek, ch), m, \sigma)$: Return $\Sigma.\mathsf{Verify}(vk, m, \sigma)$.

Note first that if Σ is ϵ-correct, then Σ' is $(\epsilon + \frac{1}{2^{256}})$-correct. The theorem is a consequence of the following two claims. □

Claim 3. *If Π is IND-CPA-secure against a classical adversary and Σ is* $\mathsf{C^cQ}$-eufcma*-secure, then Σ' is* $\mathsf{C^cQ}$-eufcma*-secure.*

Proof. The proof of this claim is as follows. Since Π is IND-CPA-secure, no passive adversary can distinguish Π-encryptions of s^* from encryptions of 0^{256} with significant advantage. An adversary could guess s^* with a probability of $\frac{q_S}{2^{256}}$, where q_S is the number of signing queries. Therefore, we can replace ch in vk' with an encryption of zeros, while still successfully simulating answers to the signing oracle in the $\mathsf{C^cQ}$-eufcma experiment. At the quantum stage, without access to the signing oracle, an adversary is unable to simulate answers. An Σ' forgery is immediately an Σ forgery, so the claim follows. □

Claim 4. *If there exists an efficient quantum adversary $\mathcal{A}$ against the message recovery of Π, then Σ' is not* $\mathsf{Q^cQ}$-eufcma-*secure.*

Proof. Suppose $\mathcal{A}$ breaks the message recovery of Π, i.e., decrypts ciphertexts encrypted using Π. Consider an algorithm $\mathcal{B}_1$ which uses $\mathcal{A}$ to decrypt ch and recover the correct solution s^*, which it can then query to its signing oracle to obtain the signing key sk. $\mathcal{B}_1$ will return a valid forgery. Taking $\mathcal{B}_2$ as the identity function, $\mathcal{B}_1, \mathcal{B}_2$ is a forger for Σ' if $\mathcal{A}$ is a decrypter for Π.

□

In Theorem 11, it is proved that a signature scheme is $\mathsf{Q^cQ}$-eufcma-secure when the adversary is quantum but only given classical access to the signing oracle. However a scheme that is $\mathsf{Q^cQ}$-eufcma-secure is not necessarily $\mathsf{Q^qQ}$-eufcma-secure, when the adversary is fully quantum with gained quantum access. A comprehensive construction and proof of this theorem by Matthew McKague is included in our paper [BHMS17]. A summary of the theorem and proof is included below for completeness. The full proof is not presented in this document, as the proof relies on notations and concepts specific to quantum computing and is out of the scope of this work.

Theorem 11 ($\mathsf{Q^cQ} \not\Rightarrow \mathsf{Q^qQ}$)**.** *Assuming there exists a quantum-secure pseudo-random family of permutations, and a signature scheme Σ that is* $\mathsf{Q^cQ}$-eufcma-*secure, then there exists a signature scheme Σ' that is* $\mathsf{Q^cQ}$-eufcma-*secure but not* $\mathsf{Q^qQ}$-eufcma-*secure.*

Similar to Theorem 10, a signature scheme is constructed where the secret key is hidden behind a problem which is hard for some adversaries and easy for others. Here, instead of a computational problem, an oracle query problem is used. The challenger gets access to the oracle that implements a specific function, and is asked to determine some property with as few calls to the oracle as possible. The minimum number of calls required to the oracle is given by the *query complexity*. For some oracle problems, querying the oracle in superposition reduces the query complexity exponentially, compared with classical calls to the oracle. The specific problem used in this proof is a variant of the *hidden linear structure* problem [DBCW+02a], which has a constant query complexity with quantum oracle access and exponential query complexity with classical oracle access. The problem is that given oracle access to $\mathcal{B}_{s,\pi}(x, y) = (x, \pi(y \oplus sx))$, where $x, y, s \in GF(2^n)$ and $\pi \in Perm(\{0,1\}^n)$ with s and π chosen uniformly at

random, determine s. Here, s is the secret which will unlock access to signing key. Since π is too large, instead of using the full set of permutations, the construction uses π_t, drawn from a family of quantum-safe pseudo-random permutations indexed by $t \in \{0,1\}^{256}$.

The construction starts with a $\mathsf{Q^cQ}$-eufcma-secure signature scheme Σ and then a new scheme Σ' is defined to include $\mathcal{B}_{s,t}(m.x, m.y)$. The adversary is given access to the signing oracle. The signing oracle has three roles: Provide regular signing oracle for Σ to sign messages, oracle for $\mathcal{B}_{s,t}$ and get responses, and oracle to reveal the Σ secret key when queries with s. The proof shows that Σ' is not $\mathsf{Q^qQ}$-eufcma-secure since a quantum adversary allowed quantum oracle access to $\Sigma'.\mathsf{Sign}$ can obtain the secret s using a constant number of oracle queries, unlock the signing key, and then efficiently generate a polynomial number of signatures, whereas for a quantum adversary with classical access to the oracle it takes exponential number of oracle queries to obtain s, so this option is not available to $\mathsf{Q^cQ}$ adversaries and hence Σ' is $\mathsf{Q^cQ}$-eufcma-secure.

5.2.1 Separability of Hybrid Signatures

In the section above, we considered the security notions of hybrid signatures considering how quantum the adversary is in several instances. We identify another security property specifically related to hybrid signatures, called *non-separability*. When implementing hybrid signature schemes, an important concern is that for a signature involving two schemes, what if an adversary is able to separate the hybrid signature into a valid signature in any of the component schemes. This security property is interesting in the context of a transition because it will be a type of downgrading attack if successful. In a possible scenario, there is a signer who issues a hybrid signature during a transition. Further, there is a verifier who can interpret both hybrid signatures and single-scheme signatures but possibly act upon them differently. The goal of non-separability is to prevent an adversary from taking a hybrid signature and turning it into something that the verifier accepts as coming from a single-scheme signature—thereby misrepresenting the signer's original intention.

In Section 5.3 we discuss how to construct hybrid schemes using *combiners*. Suppose $\Sigma' = C(\Sigma_1, \Sigma_2)$ is the hybrid signature scheme using combiner C to combine signature schemes Σ_1, Σ_2. The initial idea for the security notion for non-separability is based on the standard EUF-CMA experiment: given a signing

$\underline{\text{Expt}^{\mathsf{X^yZ}\text{-}\tau\text{-}\mathsf{nonsep}}_{C,\Sigma_1,\Sigma_2,C(\Sigma_1,\Sigma_2).R}(\mathcal{A}_1,\mathcal{A}_2)}$:

1 $q_S \leftarrow 0$
2 $(sk', vk') \leftarrow\!\!\$\ C(\Sigma_1,\Sigma_2).\mathsf{KeyGen}()$
3 $st \leftarrow\!\!\$\ \mathcal{A}_1^{\mathcal{O}_S(\cdot)}(vk')$
4 $(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}_2(st)$
5 If $(\Sigma_\tau.\mathsf{Verify}((vk')_\tau, m^*, \sigma^*) = 1)$ $\wedge\ (C(\Sigma_1,\Sigma_2).R(m^*) = 0)$
6 Return 1
7 Else return 0

$\underline{\mathcal{O}_S}$:

8 If $\mathsf{y} = \mathsf{c}$, use classical signing oracle $\mathcal{O}_S$ from Figure 5.1 for $C(\Sigma_1,\Sigma_2)$.
9 If $\mathsf{y} = \mathsf{q}$, use quantum signing oracle $\mathcal{O}_S$ from Figure 5.1 for $C(\Sigma_1,\Sigma_2)$.

Figure 5.4: Unified security experiment for $\mathsf{X^yZ}$-τ-nonsep: τ-non-separability of a combiner C with signature schemes Σ_1, Σ_2 with respect to a recognizer $C(\Sigma_1,\Sigma_2).R$ for a two-stage adversary $\mathcal{A}_1$ (of type X), $\mathcal{A}_2$ (of type Z) with signing oracle of type y. $(vk')_\tau$ denotes the projection (extraction) of the public-key associated with scheme Σ_τ from the combined scheme's public-key vk', which we assume is possible.

oracle that produces signatures for the combined scheme Σ', it should be hard for an adversary to produce a valid signature for Σ_1 ("1-non-separability") or Σ_2 ("2-non-separability"). However, this approach needs to be improved. In all of the combiners we consider in Section 5.3 below, the combined signature contains subcomponents which are valid in the underlying schemes. This makes it impossible to satisfy the naive version of non-separability.

Figure 5.4 shows the τ-non-separability security experiment, $\mathsf{X^yZ}$-τ-nonsep, with $\tau \in \{1, 2\}$ for signature scheme Σ_1 or Σ_2. It checks the ability of a two-stage $\mathsf{X^yZ}$-adversary $(\mathcal{A}_1, \mathcal{A}_2)$ to create a valid Σ_τ signature. We use a *recogniser* algorithm, which a verifier can apply to a signature to attempt to help distinguish separated hybrid signatures. For the adversary to be successful, he/she has to deceive the recogniser algorithm $\Sigma.R$ which is supposed to identify values used in a combined signature scheme Σ.

Formally, a recogniser related to a combined signature scheme $\Sigma = C(\Sigma_1,\Sigma_2)$ is a function $\Sigma.R$ that takes one input and outputs a single bit. For a signature scheme Σ_τ with message space $\mathcal{M}_{\Sigma_\tau}$, it may be that $\Sigma.R$ yields 1 on some elements of $\mathcal{M}_{\Sigma_\tau}$ and 0 on others: the purpose of R is to recognize whether certain inputs are associated with a second signature scheme. Use of a recogniser algorithm can be viewed as a generalisation of "domain separation".

5.3 Signature Combiners

This section examines several methods of using two signature schemes Σ_1 and Σ_2 to produce hybrid signatures that are proposed and proved by Bindel et al.[BHMS17]. We employ their concept of combiners when implementing hybrid signatures in standards in Section 5.5. Some related work could be find in work done by Joux [Jou04]. Combining hash functions is out of scope of this work since we are more focused on the public key cryptography that is more of a concern for quantum.

For all the combiners, the key generation of the combined scheme will simply be the concatenation of the two schemes' keys:

- $C(\Sigma_1, \Sigma_2).\mathsf{KeyGen}()$: $(sk_1, vk_1) \leftarrow_\$ \Sigma_1.\mathsf{KeyGen}()$, $(sk_2, vk_2) \leftarrow_\$ \Sigma_2.\mathsf{KeyGen}()$. Return $(sk' \leftarrow (sk_1, sk_2), vk' \leftarrow (vk_1, vk_2))$.

5.3.1 $C_\|$: Concatenation

Concatenation is the first trivial "combiner" defined. $C_\|$ combiner places independent signatures from two schemes side-by-side. Initially, a message m is signed by Σ_1 outputting a signature σ_1. Then the same message m is signed with a second scheme Σ_2 obtaining the signature σ_2. Finally, a concatenation of two acquired signatures are returned.

- $C_\|(\Sigma_1, \Sigma_2).\mathsf{Sign}(sk', m)$: $\sigma_1 \leftarrow_\$ \Sigma_1.\mathsf{Sign}(sk_1, m)$, $\sigma_2 \leftarrow_\$ \Sigma_2.\mathsf{Sign}(sk_2, m)$. Return $\sigma' \leftarrow \sigma_1 \| \sigma_2$.

In terms of separability, $C_\|$ is neither 1-non-separable nor 2-non-separable: σ_1 is immediately a Σ_1-signature for m, with no way of recognizing this as being different from typical Σ_1 signatures. It is similar for σ_2.

5.3.2 $C_{\text{str-nest}}$: Strong nesting

In this combiner, the second signature scheme Σ_2 signs both the message m and the signature σ_1 from the first signature scheme Σ_1.

- $C_{\text{str-nest}}(\Sigma_1, \Sigma_2).\mathsf{Sign}(sk', m)$: $\sigma_1 \leftarrow_\$ \Sigma_1.\mathsf{Sign}(sk_1, m)$, $\sigma_2 \leftarrow_\$ \Sigma_2.\mathsf{Sign}(sk_2, (m, \sigma_1))$. Return $\sigma' \leftarrow (\sigma_1, \sigma_2)$.

$C_{\text{str-nest}}$ is not 1-non-separable: σ_1 is immediately a Σ_1-signature for m, with no way of recognising this as being different from typical Σ_1 signatures. However, since the inputs to Σ_2 in $C_{\text{str-nest}}$ have a particular form, we can recognise those and achieve 2-non-separability.

5.3.3 D_{nest}: Dual message combiner using nesting

For this combiner, the concept of two messages signed with two signature schemes in a nested manner is used. Initially, a message m_1 is signed with Σ_1 to obtain the signature σ_1. Thereafter, the second message m_2 combined with the message m_1 and signature σ_1 is signed with Σ_2. This type of combiner is required by some of our applications in Section 5.5.

- $D_{\text{nest}}(\Sigma_1, \Sigma_2).\mathsf{Sign}(sk', (m_1, m_2))$: $\sigma_1 \leftarrow\!\!\$\ \Sigma_1.\mathsf{Sign}(sk_1, m_1)$, $\sigma_2 \leftarrow\!\!\$\ \Sigma_2.\mathsf{Sign}(sk_2, (m_1, \sigma_1, m_2))$. Return $\sigma' \leftarrow (\sigma_1, \sigma_2)$.

This dual-message combiner is not designed to give unforgeability of *both* messages under *either* signature scheme, though it does preserve unforgeability of each message under its corresponding signature scheme, as well as give unforgeability of both messages under the outer signature scheme Σ_2.

Furthermore, D_{nest} is also not 1-non-separable: σ_1 is immediately a Σ_1-signature for m, with no way of recognising this as being different from typical Σ_1 signatures. Nevertheless, since the inputs to Σ_2 in D_{nest} have a particular form, we can recognise those and achieve 2-non-separability.

Theorems and proofs for all the combiners appear in Bindel et al.[BHMS17].

5.4 Post-quantum Signature Schemes

Most of the public-key cryptography that is used on the Internet today is based on algorithms that are vulnerable to quantum attacks. Public-key algorithms such as, RSA, DSA or elliptic-curve-based schemes are perceived today as being small and fast, but since Shor presented a polynomial-time quantum algorithm, they are considered broken [Sho99]. All these conventional public-key cryptosystems have one thing in common; they are based on specific computational problems—namely, integer factorisation and discrete logarithm. Despite the fact that they are hard for a classical computer to solve, they are easily solved by a quantum computer.

In order to sustain the security of communication over the Internet and other technologies depending on cryptography, it is necessary to identify new techniques which are resilient against quantum attacks. Thus, recent studies have focused on developing signature schemes for a post-quantum world. Researchers are mainly focusing on two categories of quantum safe cryptosystems: lattice-based and hash based.

Lattice-based signature schemes are very fast and considered quantum safe as all possible key selections are strong and hard to solve. BLISS [DDLL13], TESLA [ABBD15], GLP [GLP12] and short signatures produced by Boyen [Boy10] are a few examples for efficient lattice-based signature schemes. Hash based systems offer one-time signature schemes based on hash functions such as Lamport-Diffie or Winternitz signatures. Security of these signatures relies on the collision-resistance property of the chosen hash function. XMSS [BDH11] and SPHINCS [BHH+15] are two hash based signature schemes introduced recently.

Lattice-based signature schemes like TESLA and BLISS call for fairly large key sizes, while hash based schemes generate comparatively smaller keys. We have analysed these two categories of signature schemes based on their key sizes and level of post-quantum security offered. Our assumption was that when post-quantum signature schemes require large key sizes, the corresponding certificates would be relatively larger in size. To test this assumption, we generated customised key pairs for each signature scheme and constructed certificates using those key pairs. Our goal was to experiment with these new certificates in a few certificate processing applications. Key and signature sizes, claimed security level and estimated certificate size for each signature scheme appears in Table 5.1. The experiments we carried out using these figures are described in Section 5.5.

It should be noted that, we review these proposed post-quantum signature schemes for the purpose of acquiring information on keys and signature sizes to estimate the size of potential certificate sizes. We do not evaluate them on their efficiency, practicality or even the level of security offered. There are other types of proposed signature schemes such as multivariate-quadratic and code-based, that are expected to be quantum-safe. However, we have decided on using only hash-based and lattice-based signature schemes, as they both have solid and active research bases and are representative of the variety of key sizes in use.

Table 5.1: Post-quantum signature schemes; keys and signature sizes, estimated certificate sizes, and claimed security level

Scheme	**Size** (bytes)				**Security**
	public-key	secret-key	signature	certificate	(bits)
Lattice-based					
GLP	1 536	256	1 186	3.0 KiB	100
Ring-TESLA-II [ABB+16]	3 328	1 920	1 488	5.1 KiB	128
TESLA#-I [BLN+16]	3 328	2 112	1 616	5.2 KiB	128
BLISS	7 168	2 048	1 559	9.0 KiB	128
TESLA-416	1 331 200	1 011 744	1 280	1 332.8 KiB	128
Hash-based					
XMSS	912	19	2 451	3.6 KiB	82
SPHINCS	1,056	1,088	41,000	42.3 KiB	>128
Rainbow [CCC+09]	44 160	86 240	37	44.5 KiB	80

5.5 Hybrid Signatures in Standards

When security protocols are designed, it is made sure that they are incorporated with the most effective cryptographic tools at the time of design. Over time, when the security level of these protocols needs to be upgraded, most of them are built to allow corrections for supporting changes to key sizes and other cryptographic parameters. As quantum computers present a serious challenge to widely used current techniques it is imperative that we pursue measures for quantum safe communication. Adopting post-quantum cryptography may require more substantial transformations than first anticipated. Some of the existing standards might not be compatible, especially if they are incorporated with quantum vulnerable products. In a well ordered transition, there is a period where new techniques are eventually taking hold of the system while unsupported ones are phased out. Since people are always reluctant to let go of familiar environments, most established secure communication techniques would be vulnerable in the face of quantum attacks. Thus we suggest implementing hybrid techniques that abide by both conventional and post-quantum standards. As PKI is the backbone of modern secure communication over an insecure network, a smooth transition to a quantum-resistant environment is vital.

We discuss two techniques of hybrid signatures for post-quantum cryptography. Our approach will allow early adopters to have quantum-safe communication while still having the security offered by existing mechanisms. We rely on the fact that a post-quantum X.509 certificate should be able to support at least two

signature schemes; a traditional signature scheme and a post-quantum signature scheme. If there are still traditional certificate-using systems in use (which only knows about one algorithm) then the certificate should support them and at the same time new post-quantum software should process both signature schemes.

In the following sections we examine three standards which make significant use of digital signatures to identify how hybrid signatures might be used in PKI. Namely we investigate X.509 for certificates, TLS for secure channels and S/MIME for secure email. Furthermore we evaluate backwards-compatibility of various approaches with existing software.

5.5.1 X.509v3 Certificate

The X.509 standard version 3 specifies the standard format for public-key certificates, mechanisms for managing and revoking certificates, certificate attributes and a certificate validation algorithm.

Table 5.2 depicts the basic certificate field structure of a X.509v3 certificate. X.509 uses ASN.1 distinguished encoding rules (DER) to encode data that is to be signed for signature calculation. A certificate is a sequence of three required fields. The body of the certificate, `tbsCertificate` contains regular information such as the name of the certificate authority, details about the subject including distinguished name and public-key. The `signatureAlgorithm` contains the identifier for the cryptographic algorithm used by CA to sign this certificate. The `signatureValue` field includes a digital signature computed using the CA's private key upon the encoded `tbsCertificate`.

Additionally, the certificate body also allows for optional extensions. Extension fields in X.509 certificates allow additional attributes with users and public-keys to be defined and provide a method to manage associations between CAs. The certificate format also allows private extensions to be defined and can be designated as either critical or non-critical. If a critical extension cannot be recognised or processed, it is rejected by the system. A non-critical extension is processed if it is recognised and may ignored if it is not.

Our goal is to construct a hybrid certificate which somehow includes two CA signatures and two public-keys for the subject. Ideally, one public-key for the traditional algorithm and the other for the post-quantum algorithm. Notably, the X.509v3 standard says that a certificate can contain exactly one `tbsCertificate`, which can contain exactly one subject public-key, and all of this can be signed

Table 5.2: X.509 Certificate fields

Certificate	**tbsCertificate**	**extension**
tbsCertificate	Version	extnID
signatureAlgorithm	serialNumber	critical
signatureValue	signature	extnValue
	Public key information	
	issuer ID	
	validity period	
	subject name	
	subject unique ID	
	extension	

using exactly one CA signature. This makes creating backwards-compatible hybrid certificates challenging.

5.5.1.1 Dual Certificates

The simplest approach is of course to create separate certificates: one for the traditional algorithm, the other for post-quantum algorithms. This can be constructed as a dual message analogue of the concatenation combiner $C_{\|}$ defined in Section 5.3.1 which places independent signatures from two schemes side-by-side. The dual certificates approach leaves the task of conveying the "hybrid" certificate (actually, two certificates) to the application, which will suffice in some settings (e.g., in S/MIME and some TLS settings), but is unsatisfactory in others. Additionally, this approach and the next both require assigning additional object identifiers (OIDs) for each post-quantum algorithm.

5.5.1.2 Second Certificate in Extension

Since X.509v3 does not provide any direct way of putting two public-keys or two signatures in the same certificate, one option is to use the standard's extension mechanism. The idea is to embed a post-quantum certificate as an extension in a regular X.509 certificate. In the X.509 certificate format each extension includes an OID and an ASN.1 structure. When the embedded certificate appears as an extension, its OID appears as the field `extnID` and the DER encoded certificate is recorded as the value of the octet string `extnValue`. An extension also includes a boolean critical, which we define as FALSE since it is necessary that current

mechanisms which do not recognise post-quantum certificate do not reject it.

We instantiate this using the dual message nested combiner D_{nest} defined in Section 5.3.3. Let c_1 be the certificate obtained by the CA signing `tbsCertificate` m_1 (containing subject public-key vk_1) using signature scheme Σ_1. Construct certificate c_2 by the CA signing `tbsCertificate` m_2 (containing subject public-key vk_2 as well as (an encoding of) c_1 as an extension in m_2) using signature scheme Σ_2. The extension containing c_1 would use a distinct extension identifier saying "this is an additional certificate" and would be marked as non-critical. Alternatively, the extension could contain a subset of fields, such as just the public-key and CA's signature, rather than a whole certificate. Let Σ_1 be a post-quantum signature scheme PQ and Σ_2 be a traditional signature scheme, like RSA. Let Sub and CA stand for subject of the certificate and certificate authority respectively. Then we construct the signature of certificate c_1 shown by 5.1 and signature of certificate c_2 shown by 5.2 below.

$$c_1 = Sign_{PQ}(sk_{PQ}^{CA}, (m_1, vk_{PQ}^{Sub})) \tag{5.1}$$

$$c_2 = Sign_{RSA}(sk_{RSA}^{CA}, (m_2, vk_{RSA}^{Sub}, c_1, m_1)) \tag{5.2}$$

By marking the "additional certificate" extension as non-critical, existing software which are not aware of the hybrid structure *should* ignore the unrecognised extension and continue validating the certificate and using it in applications without change.

5.5.1.3 Experiments and Results

Out of the two approaches, we followed the second technique which creates hybrid certificates with a second certificate as an extension. We proceed to investigate the extent of backwards-compatibility of this hybrid certificate and whether larger public-keys or signatures would cause issues to real time operations. In this experiment, our focus is on testing the ability of these chosen software libraries to process certificates with varying extension sizes. Here the extension sizes represent different post-quantum signature schemes, which are chosen based on the potential certificate sizes that will be produced. For our purposes of evaluating backwards compatibility, it does not matter whether the extension actually *contains* a valid post-quantum certificate. Our only concern is that it is the *size* of such a certificate. For that reason, efficiency and security levels of

Table 5.3: Compatibility of hybrid X.509v3 certificates containing large extensions.

	Extension size (and corresponding example signature scheme)				
	1.5 KiB (RSA)	3.5 KiB (GLP [GLP12])	9.0 KiB (BLISS [DDLL13])	43.0 KiB (SPHINCS [BHH+15])	1333.0 KiB (TESLA-416 [ABBD15])
GnuTLS 3.5.11	✓	✓	✓	✓	✓
Java SE 1.8.0_131	✓	✓	✓	✓	✓
mbedTLS 2.4.2	✓	✓	✓	✓	✓
NSS 3.29.1	✓	✓	✓	✓	✓
OpenSSL 1.0.2k	✓	✓	✓	✓	✓

these signature schemes are not taken in to consideration.

The "outside" certificate c_2 contains a 2048-bit RSA public-key, and is signed by a CA using 2048-bit RSA key. The extension for embedding c_1 in c_2 is identified by a distinct and previously unused algorithm identifier (OID), and is marked as non-critical following the above mentioned arguments. Because post-quantum public-keys and signatures vary considerably in size, we use a range of extension sizes to simulate the expected size of an embedded certificate for various post-quantum signature algorithms. The extension sizes we use are across the columns of Table 5.3, derived from public-key and signature sizes summarised in Table 5.1 in Section 5.4. The hybrid certificates were created using a custom-written C program using the mbedTLS library and as well as a custom-written Java program using the BouncyCastle library for more assurance.

We chose a few software libraries that offer a platform to access X.509 certificates; GnuTLS, Java SE, mbedTLS, NSS and OpenSSL. Table 5.3 shows the results of using command-line certificate verification programs in various libraries along with their version numbers at the time we tested. All libraries we tested were able to parse and verify X.509v3 certificates containing unrecognised extensions of all sizes. Large size variation from an RSA certificate (1.5 KiB) to a TESLA-416 certificate (1.3 MiB) did not affect the operability of these libraries we tested.

5.5.2 TLS

In majority, TLS handshake protocol relies on X.509 certificates with RSA public-keys to authenticate servers. Thus we chose TLS as the next standard to be tested with hybrid certificates, specifically, TLSv1.2 [DR08] which was the standardised version at the time of this research (2017) and is widely deployed.

Ciphersuites with digital signatures allow servers and (optionally) clients to authenticate each other by presenting their public-key, usually in an X.509 certificate, and signing certain messages. While the parties can negotiate which signature algorithms to use, which public-key formats to use ([MG11, WTG+14]), and which CAs to trust, once having done so they can each use only a single public-key and signature algorithm to authenticate.

The current draft of TLSv1.3 [Res17] does not change how server authentication works. However, it has a "post-handshake authentication" mode for clients [Res17, §4.5.2], where clients can be requested to further authenticate using a certificate for a given algorithm. This would allow client authentication using two or more signature schemes. This is an example of the concatenation combiner $C_{\|}$ from Section 5.3.1, since each client signature is over the same handshake context data structure. A proposal for "exported authenticators" [Sul17] is currently before the TLS working group and would allow a similar approach for server authentication, although it envisions that this takes place out-of-band (e.g., at the application layer). Neither would require hybrid certificates as in Section 5.5.1.

TLS data structures allow certificates of size up to 2^{24} bytes = 16 MiB, which would accommodate even very large post-quantum algorithms. However, TLS record layer fragments can be at most 16 KiB. TLS messages can be split across multiple fragments, but this increases the risk of incompatibility with poorly implemented software and can be problematic with datagram transport (UDP).

5.5.2.1 Experiments and Results

The goal of this experiment is to test if TLS connections can successfully be established when using hybrid X.509 certificates. Ideally, the hybrid certificates support a traditional signature scheme, say RSA and a post-quantum signature scheme. Current standards been able to support these certificates will advance the transition to quantum-safe public-key infrastructure. Thus, this experiment tests the backwards-compatibility of some popular TLS supporting applications with our hybrid certificates. In this experiment, run-time performance is not a criterion when assessing the ability of web browsers to handle certificates with large extensions, nor the type of post-quantum signature scheme, their individual efficiencies and security levels.

For experiments we choose several applications that support TLS. These

applications include, most notably web browsers—Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Opera and Apple Safari—and TLS programming libraries—OpenSSL, GnuTLS, mbedTLS and Java SE. Table 5.4 shows the results of testing the compatibility of these chosen TLS libraries and web browsers when using the hybrid certificates from second approach in Section 5.5.1.2.

In the top half of the table, we test whether popular TLS libraries can be used to establish a TLS 1.2 connection using an RSA certificate with an extension of the given size. In each case, the experiment is carried out between that library's own TLS server and TLS client command-line programs. In the case of Java, we wrote a simple HTTPS server and client using built-in libraries. Only Java completes connections with extensions the size of a TESLA-416 certificate (1.3 MiB). The mbedTLS library cannot handle certificates with extensions the size of a SPHINCS certificate (43 KiB) and upwards. GnuTLS and OpenSSL, failed at processing TESLA-416 certificates. Since there is a substantial size gap between SPHINCS and TESLA-416 certificates, we further experimented on the point of failure and found they could handle an 80 KiB extension but not a 90 KiB extension.

In the bottom half of the table, we test whether popular web browsers can be used to establish a TLS 1.2 connection to a TLS server run using the OpenSSL command-line `s_server` program using an RSA certificate with an extension of the given size. Microsoft browsers on Windows 10 cannot handle SPHINCS-sized extensions. Furthermore, when it came to quite large certificates, no browser except Safari could handle TESLA-416-sized extensions (1.3 MiB). Curiously, Safari was able to handle a 1.3 MiB extension with an OpenSSL command-line server despite OpenSSL's own command-line client not being able to handle it.

5.5.3 CMS and S/MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME) [RT10] is widely adopted in government and enterprises for sending digitally signed and encrypted email messages. Cryptographic Message Syntax (CMS) [Hou09] is used as the key cryptographic component of S/MIME, which enables encryption, decryption, signing and verifying emails. In this context we focus on the signing property in which a valid certificate and its private key is required. In an S/MIME signed email, a header is used to specify the algorithms used, and then the body of the email is divided into chunks: one chunk is the body to be signed, and the other

Table 5.4: Compatibility of TLS connections using hybrid X.509v3 certificates containing large extensions.

	Extension size in KiB				
	1.5	3.5	9.0	43.0	1333.0
Libraries (library's command-line client talking to library's command-line server)					
GnuTLS 3.5.11	✓	✓	✓	✓	×
Java SE 1.8.0_131	✓	✓	✓	✓	✓
mbedTLS 2.4.2	✓	✓	✓	×	×
NSS 3.29.1	✓	✓	✓	✓	×
OpenSSL 1.0.2k	✓	✓	✓	✓	×
Web browsers (talking to OpenSSL's command-line server)					
Apple Safari 10.1 (12603.1.30.0.34)	✓	✓	✓	✓	✓
Google Chrome 58.0.3029.81	✓	✓	✓	✓	×
Microsoft Edge 38.14393.1066.0	✓	✓	✓	×	×
Microsoft IE 11.1066.14393.0	✓	✓	✓	×	×
Mozilla Firefox 53.0	✓	✓	✓	✓	×
Opera 44.0.2510.1218	✓	✓	✓	✓	×

is a Base-64 encoding of a CMS `SignedData` object. The `SignedData` object contains several fields, including a set of certificates and a set of `SignerInfo` objects. Each `SignerInfo` object contains a signer identifier, algorithm identifier, signature, and optional signed and unsigned attributes.

Many of the security properties of S/MIME rely on the parameters of CMS. Fortunately, most CMS parameters are customisable including algorithm selection. The S/MIME Capabilities attribute is designed to be flexible and extensible so that it would be backwards-compatible when new capabilities are added later. Contemplating these features, we identify two approaches to how S/MIME could be implemented in a quantum safe environment.

5.5.3.1 Approach 1: Parallel SignerInfo

The CMS `SignedData` object content allows zero or more signature values. It is constructed so that any number of signers in parallel can sign any type of content. Each signer can choose a specific message-digest algorithm and digitally sign using signer's private key. The `SignerInfo` value for each signer is created by collecting the signature value and other signer-specific information. This method can be

used to incorporate two signature algorithms in place of two individual signers. Thus, here we find that there is no need for the hybrid certificates proposed in Section 5.5.1. Instead, to construct a standards-compliant hybrid signature in S/MIME, we could put the certificate for each algorithm in the `SignedData` object's set of certificates, and then include `SignerInfo` objects for the signature from each algorithm. This is an example of the concatenation combiner $C_{\|}$ from Section 5.3.1.

5.5.3.2 Approach 2: Nested signature in SignerInfo attributes

For an alternative and still standards-compliant approach, we could use the optional attributes in the `SignerInfo` object to embed a second signature. We need to convey the certificate for the second algorithm, as well as the signature using the second algorithm for the message. There are several options based on the standards:

2.a) Put a second certificate in the set of certificates, and put a second `SignerInfo` in an attribute of the first `SignerInfo`.

2.b) Put a hybrid certificate in the set of certificates, and put a second `SignerInfo` in an attribute of the first `SignerInfo`.

2.c) Put a second `SignedData` in an attribute of the first `SignerInfo`.

These approaches require defining a new attribute type, but this is easily done. The CMS standard indicates that verifiers can accept signatures with unrecognised attributes, so this approach results in backwards-compatible signatures that should be accepted by existing software.

If the extra data is put in the *signed* attribute of the first `SignerInfo`, then we are using the strong nesting combiner $C_{\text{str-nest}}$ from Section 5.3.2. If the extra data is put in the *unsigned* attribute of the first `SignerInfo`, then we are using the concatenation combiner $C_{\|}$ from Section 5.3.1.

5.5.3.3 Experiments and Results

Our goal in this experiment is to test the backward-compatibility of S/MIME applications when tested with post-quantum signature schemes and hybrid digital certificates. It does not take in to account the efficiency and security levels of

Table 5.5: Compatibility of hybrid S/MIME approaches.

	Approach								
	0	**1**	**2.a**	**2.b** (attribute size in KiB)					**2.c**
				1.5	3.5	9.0	43.0	1333.0	
Apple Mail 10.2 (3259)	✓	✓	✓	✓	✓	✓	✓	✓	✓
BouncyCastle 1.56 with Java SE 1.8.0_131	✓	–	✓	✓	✓	✓	✓	✓	✓
Microsoft Outlook 2016 16.0.7870.2031	✓	×	✓	✓	✓	✓	✓	✓	✓
Mozilla Thunderbird 45.7.1	✓	×	✓	✓	✓	✓	✓	×	✓
OpenSSL 1.0.2k	✓	×	✓	✓	✓	✓	✓	✓	✓

Approach 0: both RSA-2048.
Approach 1: one RSA-2048, one with unknown algorithm of similar key and signature size; unable to test Approach 1 on BouncyCastle.
Approach 2.a, 2.c: both RSA-2048.
Approach 2.b: outer RSA-2048, inner random binary extension of given size.

signature schemes nor the run-time performance of the applications when using these signatures and certificates.

We tested five S/MIME libraries/applications for acceptance of S/MIME messages from each approach above. For the experiment we chose Apple Mail, BouncyCastle, Microsft Outlook, Mozilla Thunderbird and OpenSSL. Following the Table 5.1, initially we created hybrid certificates with large extensions which would adhere to potential post-quantum signature schemes. Using post-quantum signature schemes will result in a large content of `SignedData`. Moreover, some of the new schemes may not be identified by the existing software. We kept those facts in mind while testing S/MIME protocol. The results of the experiment on the proposed approaches and the configurations appear in Table 5.5.

We tested approach 1 beginning with two certificates with known key algorithms (depicted by Approach 0 in Table 5.5). Thereafter we tested with one known and unknown key algorithm. Regarding approach 1, the S/MIME and CMS standards does not specify how to validate multiple `SignerInfo` objects: should a signed message be considered valid if *any* of the `SignerInfo` objects is valid, or only if *all* of them are? Apple Mail accepted in this case, whereas the three other programs rejected the new technique of using one known algorithm and one unknown algorithm. Therefore it was evident that approach using parallel `SignerInfo`s is not fully backwards-compatible. In principle all the tested libraries support approaches 2.a–2.c. We only tested multiple attribute sizes in one of these three approaches (2.b), but the results should generalise to 2.a and

2.c. For most signature schemes, the chosen S/MIME libraries/applications are able to process through. Only Thunderbird struggled with very large attributes such as TESLA-416 scheme.

5.6 Discussion

We have investigated and presented the use of hybrid digital signature schemes in this chapter. First we defined appropriate security notions for hybrid signatures and identified the conditions on when the resulting scheme is unforgeable. Additionally, we addressed a new notion about the inability of an adversary to separate a hybrid signature into its underlying components. When developing combined signature schemes from scratch, we recommend to build combiners that include "domain separation" to properly achieve the security notion *non-separability*. By having domain separation in built, it would eliminate the need for recogniser algorithms which helps to identify values used in a combined signature scheme.

We have considered several methods of combining signature schemes. Furthermore we identified several approaches to preserve backwards-compatibility in the chosen standards while supporting hybrid signatures. Placing post-quantum objects as non-critical extensions of pre-quantum objects makes it easy to maintain backwards-compatibility. Such example is using attributes in S/MIME. We observed that constructing hybrid signatures in our chosen standards leads to one of the nested signature combiners, either D_{nest} for X.509 certificate extensions or $C_{\text{str-nest}}$ for S/MIME and CMS. Both these combiners offer unforgeability under the assumption that either scheme is unforgeable.

We created certificates with estimated sizes containing post-quantum signature schemes to use as extensions. When analysing our experimental observations on hybrid certificates, it is evident that the software we tested had no problems with certificates or extensions up to ∼10 kB, accommodating ideal lattice schemes and some software up to ∼80 kB, accommodating hash-based schemes. Generally, extensions up to 40 KiB are mostly handled successfully. This size covers many of the chosen post-quantum signature schemes except TESLA-416; none could handle the megabyte-plus size public-keys of this largest lattice-based scheme. Any similarly large signature schemes would be troublesome to use in hybrid modes with existing software. It is advisable to upgrade these security protocols and standards to enable accommodating large key sizes.

The post-quantum signature schemes mentioned in this chapter are in theoretic research state. In our experiments we use the estimated sizes of certificates these signature schemes would yield and it is limited to information gained from those individual research projects. We have made our assumptions and carried out the experiments based on those facts. We have also limited our experiments to only three standards that significantly use digital signatures.

Chapter 6

Conclusion and Future Work

In this final chapter we summarise the contributions of our thesis. We evaluate our success in capturing new and interesting aspects of public key infrastructure technologies, and suggest other potential research directions that may benefit from the techniques we used. We also remark upon the limitations of our research, and contemplate how to overcome the obstacles we faced going forward.

Public-key infrastructure is a comprehensive system that supports public-key encryption and digital signature services. The purpose of a public-key infrastructure is to manage keys and certificates. The web public-key infrastructure is about enabling web applications to interact with digital certificates and focuses on protecting the confidentiality, integrity, and authenticity of communications between web browsers and web content servers. The demand for successfully implemented PKI systems is increasing as many organisations are looking to secure their communications. Although it is a good sign in terms of information security, unfortunately it also puts the Certificate Authority ecosystem in the spotlight for sophisticated cyber attacks. However, the PKI trust model has a few attractive qualities; it is Internet scalable and cost effective, it provides integrity, confidentiality, and authentication as well as revocation. These qualities outshine the alternatives to the PKI trust model. Hence, it is imperative we look for methods to enhance the current PKI trust model for present and future developments, rather than replace it, which is our main objective in this research. Our work focuses on web public-key infrastructure. First we work on modelling an existing technology, then on designing a new technology and finally speculating

and progressing towards a quantum-safe web PKI.

Further, this research opens up new research directions, particularly pointing the research community towards improving the quality of current public-key infrastructure.

The main contributions and future directions of this thesis are summarised as follows.

Chapter 3: Secure Logging Schemes and Certificate Transparency. Over the years, many new PKI technologies have been proposed with the aim of improving the PKI trust model either by detecting fraudulent certificates, reducing the impact of compromises or improving checks on revoked credentials. However, for most of these schemes, no security analysis has been done. Such an experimental Internet security standard is Certificate Transparency (CT), which is a promising approach for providing assurance in the web PKI by using untrusted auditable public logs to detect fraudulently issued certificates. Given that CT is now being deployed, it is important to verify that it achieves its security goals. We contribute a model of logging scheme that allows to attain a formal understanding of the security goals of Certificate Transparency and analyse whether these goals are achieved or not. We performed an extensive analysis of Certificate Transparency in terms of its security properties. We contribute a model of logging schemes for Certificate Transparency and we define four security properties. For these properties of logging schemes that can be proved cryptographically, we follow a provable security approach. Four security properties of the logging scheme are categorised in to two types: security notions which concern a malicious logger attempting to present different views of the log to different parties or at different points in time, and security notions concerning a malicious monitor attempting to frame an honest log for failing to include a certificate in the log. Our logging scheme is backed with a Merkle tree hash system, which we have reformulated to a top-down recursive approach to help proving our theorems. We prove that Certificate Transparency satisfies the security properties we have defined under various assumptions on Merkle hash trees.

Moreover, the logging scheme we have proposed has the potential to be applied to any other similar construction. As for the limitations in this work, there are some components involved in Certificate Transparency that are difficult to capture in a cryptographic model. We solely focus on the sections that use and have the potential to use cryptographic techniques to establish the trust necessary in a

public logging framework like Certificate Transparency. For example, determining the time it takes to propagate signed certificate timestamps (SCT) and signed tree heads (STH) among entities to ensure detection of misbehaviour depend on the facts like various conditions on adversary control of the network and diverse patterns of honest entity behaviour. These facts are difficult or even impossible to capture in cryptographic techniques, thus are out of our scope.

The gossiping protocol in CT allows the involved entities to share information they receive from log servers. In our model, we assume that gossiping protocol works as intended and assume that all STH values are shared among monitors and auditors. It is possible that the performance of gossiping protocol has a huge impact on how and when misbehaviour of entities are detected. We have not addressed some aspects of gossiping protocol in our model, for example *SCT feedback*. Moreover, there are number of organisational measures that can be taken once misbehaviour is detected to prevent further damage. Analysing these components in general as well as their specific relevance in the Certificate Transparency framework is an interesting task for future work.

Chapter 4: Associative Blockchain for Decentralised PKI Transparency. The conventional public-key infrastructure model, which powers most of the Internet, suffers from an excess of trust into Certificate Authorities. This overly placed trust together with the lack of transparency makes PKI vulnerable to impersonation attacks that are hard to detect. Although there are existing approaches aiming to make certificate management more transparent, they are strictly speaking still centralised. We contribute a model of a new design for a decentralised approach to PKI transparency based on generic blockchains extended with efficient data structures, called Decentralised PKI Transparency. Decentralised PKI Transparency (DPKIT) is a system that allows light weight auditing and management of issuance and revocation certificates. It can be considered as a strongly immutable record of X.509 certificates. It is compatible with existing public-key infrastructure and effective at detecting and dissuading trust abuses by malicious certificate authorities. We define three security properties against a malign entity using the formalism of provable security. These properties show that it is hard for an adversary, to remove an entry once logged through honest functionalities, to provide a proof for an entry that is invalid or not yet logged, and to hide an entry's revocation information if it exists in the log. For the security purposes of a blockchain, we define ideal functionality of a distributed

append-only ledger. Then we prove the security results on DPKIT and show that our construction guarantees these three security properties.

It is clear that gaps still exist between current log-based approaches to make certificate issuance more transparent and blockchain-based approaches to make it decentralised. We believe DPKIT is a clear step towards bridging this gap. But it is not without limitations. In this work we view blockchains as a black-box that let users append small and infrequent amount of data commonly shared and replicated on a timeline. As a result we do not perform a thorough analysis of security properties of a blockchain. Furthermore, we do not provide a specific proof that lets a user obtain all the certificates pertaining to a subject. This task could be achieved by having the data structure return node values of the secondary tree of a subject in the order that they were inserted and having root hash reconstructed.

Throughout this thesis, our work is mainly based on web PKI. PKIs of one type or another, have many uses, including providing public keys and bindings to user identities which are used for: encryption and/or sender authentication of e-mail messages (email signing), encryption and/or authentication of documents (document signing), authentication of users to applications, code signing and many more. An interesting aspect for future work would be to use DPKIT for these other applications of public-key infrastructure.

Additionally we do not provide a practical evaluation of DPKIT. DPKIT could be implemented as a browser extension, as a further step in fraud detection and prosecution. However, for future work, it remains to be developed as such and then executed. Another prospective suggestion for future work is overcoming a particular obstacle which applies for any blockchain-based application, namely scalability. A potential solution for this matter could be sharding the blockchain [LNZ+16]. As sharding is a topic of on going research, investigating and associating it to our scheme will be interesting. Although it is not without drawbacks, these types of blockchain-based systems have an inclination to be chosen mainly due to strong security guarantees provided. We believe the success of blockchain-based solution will largely depend on the participation of CAs, browser vendors and their clients.

Chapter 5: Transitioning to a Quantum-Resistant Public-Key Infrastructure. To ensure uninterrupted cryptographic security, it is important to begin planning the transition to post-quantum cryptography. In addition to

creating post-quantum primitives, we must plan how to adapt the cryptographic infrastructure for the transition, such as public-key infrastructure with many participants. As taking a step towards a better transition, we investigate of the use of a hybrid digital signature scheme in which both a traditional algorithm and one or more post-quantum algorithms are employed. Then we examine the approaches of supporting hybrid signatures in three real-world standards that involve digital signatures and public-key infrastructure: X.509 certificates, secure channels, and S/MIME email. We identify possible approaches to supporting hybrid signatures in these standards while retaining backwards compatibility. We created certificates with estimated sizes containing post-quantum signature schemes. Then we included them in to X.509 certificates as extensions and tested them on several applications that support TLS connections.

Most of the software we chose in this work has no problems with these designed certificates or extensions up to certain sizes. A potential future work is to test the compatibility of other existing software as importance of using quantum-safe X.509 certificates depends on the application in which they are used. We only tested several post-quantum signature schemes that were popular at the time. For our purposes of evaluating backwards compatibility, it does not matter whether the certificate extension actually contains a valid post-quantum certificate. A new direction would be to create certificates with actual post-quantum signature schemes and utilise them in our proposed concept. Our contribution provides suggestions for enhancing current PKI model as well as for future developments when transitioning to a quantum resistant PKI.

Bibliography

[ABB+16] Sedat Akleylek, Nina Bindel, Johannes A. Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 44–60. Springer, Heidelberg, April 2016.

[ABBD15] Erdem Alkim, Nina Bindel, Johannes Buchmann, and Özgür Dagdelen. TESLA: Tightly-secure efficient signatures from standard lattices. Cryptology ePrint Archive, Report 2015/755, 2015. `http://eprint.iacr.org/2015/755`.

[AP03] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of LNCS, pages 452–473, 2003.

[Bac02] Adam Back. Hashcash-a denial of service counter-measure. 2002. `http://www.hashcash.org/papers/hashcash.pdf`.

[BBSU12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make Bitcoin a better currency. In *FC'12*, volume 7397 of LNCS, pages 399–414. Springer, 2012.

[BCK+14a] David A. Basin, Cas J. F. Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: attack resilient public-key infrastructure. In *ACM CCS 2014*, pages 382–393, November 2014.

[BCK+14b] David A. Basin, Cas J. F. Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack resilient public-key infrastructure. In Gail-Joon Ahn, Moti Yung,

and Ninghui Li, editors, *ACM CCS 14*, pages 382–393. ACM Press, November 2014.

[BCNS15] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society Press, May 2015.

[BDE+11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 363–378. Springer, Heidelberg, July 2011.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 117–129. Springer, 2011.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of LNCS, pages 213–229. Springer, 2001.

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, Heidelberg, April 2015.

[BHMS17] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a quantum-resistant public key infrastructure (full version). Cryptology ePrint Archive. Published in PQCrypto 2017, April 2017.

[BL18] Blockchain Luxembourg. Blockchain charts. `https://www.blockchain.com/charts`, 2018.

[BLN+16] Paulo S. L. M. Barreto, Patrick Longa, Michael Naehrig, Jefferson E. Ricardini, and Gustavo Zanon. Sharper ring-LWE signatures. Cryptology ePrint Archive, Report 2016/1026, 2016. http://eprint.iacr.org/2016/1026.

[Boy10] Xavier Boyen. Lattice mixing and vanishing trapdoors. In *Public Key Cryptography*, 2010.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 92–111. Springer, Heidelberg, May 1995.

[Bra16] Matt Braithwaite. Google Security Blog: Experimenting with post-quantum cryptography, July 2016. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.

[BvOP+09] Robert Biddle, Paul C. van Oorschot, Andrew S. Patrick, Jennifer Sobey, and Tara Whalen. Browser interfaces and extended validation ssl certificates: an empirical study. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, pages 19–30. ACM, 2009. https://docbox.etsi.org/workshop/2014/201410_crypto/quantum_safe_whitepaper_1_0_0.pdf.

[BZ13] Dan Boneh and Mark Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 361–379. Springer, Heidelberg, August 2013.

[Cam15] Matthew Campagna. Quantum safe cryptography and security: An introduction, benefits, enablers and challengers. Technical report, ETSI (European Telecommunications Standards Institute), June 2015.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[CCC+09] Anna Inn-Tung Chen, Ming-Shing Chen, Tien-Ren Chen, Chen-Mou Cheng, Jintai Ding, Eric Li-Hsiang Kuo, Frost Yu-Shuang Lee, and Bo-Yin Yang. SSE implementation of multivariate PKCs on modern x86 CPUs. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 33–48. Springer, Heidelberg, September 2009.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding, 8th IMA International Conference*, pages 360–363, December 2001.

[Com11] Comodo Group. Comodo fraud incident: Update 31-Mar-2011. https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, March 2011.

[Cor09] Thomas H Cormen. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.

[Cro09] Scott A. Crosby. *Efficient Tamper-Evident Data Structures for Untrusted Servers*. PhD thesis, Rice University, Houston, Texas, USA, 2009.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[CvO13] Jeremy Clark and Paul C. van Oorschot. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy*, pages 511–525. IEEE Computer Society Press, May 2013.

[CW09] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *18th USENIX Security Symposium 2009*, pages 317–334. USENIX Association, 2009.

[DBCW+02a] J Niel De Beaudrap, Richard Cleve, John Watrous, et al. Sharp quantum versus classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002.

[dBCW02b] Niel de Beaudrap, Richard Cleve, and John Watrous. Sharp quantum versus classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.

[Dec16] Casey Deccio. Organizational Domains and Use Policies for Domain Names . Internet-Draft draft-deccio-dbound-organizational-domain-policy-03, July 2016.

[DGHS16] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and Certificate Transparency. In *ESORICS*, pages 140–158. Springer, 2016.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DOTV08] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In *PQCrypto 2008*, volume 5299 of LNCS, pages 109–123. Springer, 2008.

[DR06] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) protocol version 1.1. *Internet Engineering Task Force (IETF)*, RFC 4346, April 2006.

[DR08] Tim Dierks and Eric Rescorla. The Transport Layer Security TLS protocol version 1.2. *Internet Engineering Task Force (IETF)*, RFC 5246, August 2008.

[DTMH+16] Aymard De Touzalin, Charles Marcus, Freeke Heijman, Ignacio Cirac, Richard Murray, and Tommaso Calarco. Quantum Manifesto. a new era of technology. *European Comission*, 2016.

[Duc13] Paul Ducklin. The TurkTrust SSL certificate fiasco – what really happened, and what happens next? *Sophos, Naked Security*, January 2013.

[Eas11] Donald Eastlake. Transport Layer Security (TLS) extensions: Extension definitions. *Internet Engineering Task Force (IETF)*, RFC 6066, January, 2011.

[EB10] Peter Eckersley and Jesse Burns. The EFF SSL observatory. *Internet: https://www.eff.org/observatory*, 2010.

[Eck11] Peter Eckersley. Sovereign key cryptography for internet domains. *Electronic Frontier Foundation (EFF)*, November 2011. `https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD`.

[Eck14] Peter Eckersley. Launching in 2015: A certificate authority to encrypt the entire web. Deeplinks blog, Electronic Frontier Foundation (EFF), November 2014. `https://www.eff.org/deeplinks/2014/11/certificate-authority-encrypt-entire-web`.

[Ele] Electronic Frontier Foundation. Sovereign keys. `https://www.eff.org/sovereign-keys`.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 1984*, volume 196 of LNCS, pages 10–18. Springer, 1984.

[EP15] Chris Evans and Chris Palmer. Public key pinning extension for HTTP. *Internet Engineering Task Force (IETF)*, RFC 7469, April 2015.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[For07] CA/Browser Forum. Guidelines for the issuance and management of extended validation certificates. 2007. `https://cabforum.org/documents/`.

[Fox12] Fox IT. Black Tulip: Report of the investigation into the DigiNotar certificate authority breach. `http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf`, August 2012.

[Fre17] Bastian Fredriksson. A Distributed Public Key Infrastructure for the web backed by a Blockchain. Dissertation, KTH, School of Computer Science and Communication (CSC), 2017. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-210912.

[FS00] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, 2000.

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptology ePrint Archive*, 2014:803, 2014.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

[Goo19] Google. HTTPS encryption on the web. Google Transparency Report, March 2019. `https://transparencyreport.google.com/https/certificates?hl=en`.

[GSE11] Eva Galperin, Seth Schoen, and Peter Eckersley. A post mortem on the Iranian DigiNotar attack. *EFF Blog,*

September, 2011. https://www.eff.org/deeplinks/2011/09/post-mortem-iranian-diginotar-attack/.

[GSM+13] Slava Galperin, Stefan Santesson, Michael Myers, Ambarish Malpani, and Carlisle Adams. X. 509 Internet Public Key Infrastructure Online Certificate Status Protocol-OCSP. *Internet Engineering Task Force (IETF)*, RFC 6960, June 2013.

[Hou09] Russell Housley. Cryptographic Message Syntax (CMS). RFC 5652 (INTERNET STANDARD), September 2009.

[HPFS08] Russell Housley, William Polk, Warwick Ford, and David Solo. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. Rfc 5280, May 2008.

[HS12] Paul Hoffman and Jakob Schlyter. The (DNS)-based Authentication of Named Entities (DANE) Transport Layer Security (TLS) protocol: TLSA. RFC 6698, August 2012.

[Hua16] David Huang. Early impacts of certificate transparency, April 2016. https://www.facebook.com/notes/protect-the-graph/early-impacts-of-certificate-transparency/1709731569266987/.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 306–316, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Ken15] Stephen Kent. Attack model and threat for Certificate Transparency. *Internet Engineering task Force (IETF)*, June 2015. Internet Draft https://tools.ietf.org/html/draft-ietf-trans-threat-analysis-14.

[Ken18] Stephen Kent. Attack and threat model for certificate transparency. *draft-ietf-trans-threat-analysis-16 (work in progress)*, 2018.

[KHP+13] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil D. Gligor. Accountable key infrastructure

(AKI): a proposal for a public-key validation infrastructure. In *WWW 2013*, pages 679–690, 2013.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 19, August 2012. https://pdfs.semanticscholar.org/0db3/8d32069f3341d34c35085dc009a85ba13c13.pdf.

[Lau14] Ben Laurie. Certificate Transparency. *ACM Queue - Security*, 12(8):10, 2014. https://queue.acm.org/detail.cfm?id=2668154.

[LCMR16] Benjamin Leiding, Clemens H. Cap, Thomas Mundt, and Samaneh Rashidibajgan. Authcoin: Validation and authentication in decentralized networks. *CoRR*, abs/1609.04955, 2016. https://arxiv.org/abs/1609.04955.

[LK12] Ben Laurie and Emilia Kasper. Revocation transparency. http://www.links.org/files/RevocationTransparency.pdf, 2012.

[LKL13] Adam Langley, Emilia Kasper, and Ben Laurie. Certificate Transparency. *Internet Engineering Task Force (IETF)*, RFC 6962, June 2013.

[LKMS04] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. Secure untrusted data repository (sundr). In *OSDI*, volume 4, pages 9–9, 2004.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[LYH03] Sean Lancaster, David C. Yen, and Shi-Ming Huang. Public key infrastructure: a micro and macro analysis. *Computer Standards & Interfaces*, 25(5):437–446, 2003.

[Man12] Ravi Mandalia. Security Breach in CA Networks -Comodo, DigiNotar, GlobalSign. *(ISC)2 Blog*, April 2012. `http://blog.isc2.org/isc2_blog/2012/04/test.html/`.

[Mar13] Moxie Marlinspike. Trust assertions for certificate keys. 2013. `https://tools.ietf.org/html/draft-perrin-tls-tack-00`.

[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 2015*, pages 383–398. USENIX Association, 2015.

[Mer79] Ralph C. Merkle. Secrecy, authentication, and public key systems. Technical Report 1979-1, Information Systems Laboratory, Stanford University, Palo Alto, CA, USA, June 1979.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

[MG11] Nikos Mavrogiannopoulos and Daniel Gillmor. Using OpenPGP Keys for Transport Layer Security (TLS) Authentication. RFC 6091 (Informational), February 2011.

[MR16] Stephanos Matsumoto and Raphael M. Reischuk. IKP: Turning a PKI around with blockchains. *IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA*, 2017:1018, May 2016.

[Mye98] Michael Myers. Revocatoin: Options and challenges. In *Financial Cryptography and Data Security*, pages 165–171. Springer, 1998.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2008.

[NGR15] Linus Nordberg, Daniel Gillmore, and Tom Ritter. Gossiping in CT. *Internet Engineering Task Force (IETF)*, August 2015. `https://tools.ietf.org/html/draft-ietf-trans-gossip-00`.

[QHW+17] Bo Qin, Jikun Huang, Qin Wang, Xizhao Luo, Bin Liang, and Wenchang Shi. CeCoin: A decentralized PKI mitigating MitM attacks. *Elsevier, Future Generation Computer Systems*, 2017.

[Rea13] Scott Rea. Alternatives to certification authorities for a secure web. RSA Conference Asia Pacific 2013, 2013. https://www.rsaconference.com/writable/presentations/file_upload/sec-t02_final.pdf.

[Res00] Eric Rescorla. HTTP over TLS. *Internet Engineering Task Force (IETF)*, RFC 2818, May 2000.

[Res17] Eric Rescorla. The Transport Layer Security (TLS) protocol version 1.3, draft 19, March 2017. https://tools.ietf.org/html/draft-ietf-tls-tls13-19.

[RSA78] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RT10] Blake Ramsdell and Sean Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), January 2010.

[Rya14] Mark D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *NDSS 2014*. The Internet Society, February 2014.

[SBvP08] Jennifer Sobey, Robert Biddle, Paul C. van Oorschot, and Andrew S. Patrick. Exploring user reactions to new browser cues for extended validation certificates. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 411–427. Springer, Heidelberg, October 2008.

[SE15] Stephan Somogyi and Adam Eijdenberg. Improved digital certificate security. http://googleonlinesecurity.blogspot.de/2015/09/improved-digital-certificate-security.html, September 2015.

[SEA+09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *18th USENIX Security Symposium, Montreal, Canada*, pages 399–416, August 2009.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology - CRYPTO 1984*, volume 196 of LNCS, pages 47–53. Springer, 1985.

[SHF02] David Solo, Russell Housley, and Warwick Ford. Internet x. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) profile. *Internet Engineering Task Force (IETF)*, RFC 3280, April 2002.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[SM84] Goldwasser Shafi and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[SS12] Christopher Soghoian and Sid Stamm. Certified lies: Detecting and defeating government interception attacks against SSL (short paper). In *Financial Cryptography and Data Security*, volume 7035 of LNCS, pages 250–259. Springer, 2012.

[Ste10] Douglas Stebila. Reinforcing bad behaviour: the misuse of security indicators on popular websites. In *Proceedings of the 22nd Australasian Computer-Human Interaction Conference, OZCHI 2010*, pages 248–251, 2010.

[Ste14] Douglas Stebila. An introduction to provable security. July 2014. `https://s3.amazonaws.com/files.douglas.stebila.ca/files/teaching/amsi-winter-school/Lecture-2-3-Provable-security.pdf`.

[Sul17] Nick Sullivan. Exported authenticators in TLS, draft 01, March 2017. `https://tools.ietf.org/html/draft-sullivan-tls-exported-authenticator-01`.

[Vac04] John R. Vacca. *Public key infrastructure: building trusted applications and Web services*. CRC Press, 2004.

[VER18] VERISIGN. The Domain Name Industry Brief. Q1 2018. Press Release. Volume 15, Issue 2, June 2018. `https://www.verisign.com/en_US/domain-names/dnib/index.xhtml`.

[Woo14] Gvain Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014.

[WTG+14] Paul Wouters, Hannes Tschofenig, John Gilmore, Samuel Weiler, and Tero Kivinen. Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7250 (Proposed Standard), June 2014.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36. Springer, Heidelberg, August 2005.

[YCR16] Jiangshan Yu, Vincent Cheval, and Mark Ryan. DTKI: A new formalized PKI with verifiable trusted parties. *The Computer Journal*, 59(11):1695–1713, 2016.

[Zim95] Phil Zimmerman. Pretty good privacy. *The Official PGP Users Guide*, 1995.