Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Certificate Transparency in Theory and Practice

by

Josef Gustafsson

LIU-IDA/LITH-EX--A16/001--SE

2016-02-24

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköping University
Department of Computer and Information Science

Final Thesis

Certificate Transpareny in Theory and Practice

by

Josef Gustafsson

LIU-IDA/LITH-EX--A16/001--SE

2016-02-24

Supervisor: Vengatanathan Krishnamoorthi, *M.Sc.*
Examiner: Niklas Carlsson, *Associate Professor*

Abstract

Certificate Transparency provides auditability to the widely used X.509 Public Key Infrastructure (PKIX) authentication in Transport Layer Security (TLS) protocol. Transparency logs issue signed promises of inclusions to be used together with certificates for authentication of TLS servers. Google Chrome enforces the use of Certificate Transparency for validation of Extended Validation (EV) certificates. This thesis proposes a methodology for asserting correct operation and presents a survey of active Logs. An experimental Monitor has been implemented as part of the thesis. Varying Log usage patterns and metadata about Log operation are presented, and Logs are categorized based on characteristics and usage. A case of mis-issuance by Symantec is presented to show the effectiveness of Certificate Transparency.

Contents

Chapter 1

Introduction

1.1 Motivation

The original design of the Internet included no mechanism for associating a physical identity (person, organization) to a digital identity (IP address, MAC address, domain name). IP and MAC addresses can at best identify a device, not the person using it. IP addresses change over time and MAC addresses can easily be changed manually, making both types of addresses poorly suited for authenticating a person or organization. Domain names can be registered by anyone, often without the person providing proof of identity. Today it is unthinkable to trust that everyone on the Internet are who they claim to be, including banks, health services, authorities and service providers.

The first large-scale solution for the web was Secure Socket Layer (SSL), which dates back to experimental versions in 1993, and many improvements have been made since. The idea behind authentication remains the same in modern versions of the protocol (even though the initial approach was described by its creator as "a bit of a handwave", and "oh that, we just threw that in at the end")(Marlinspike 2011). Trust is placed in a few central Certificate Authorities (CAs) who take responsibility for verifying the identity of entities and issuing electronic proofs in the form of X.509 certificates. The holder presents the certificate as proof that the included key belongs to the correct entity, and a secure connection can be established.

The nature of the Internet has changed a lot since the SSL protocol was first devised and is now used to protect millions of connections every day (Soghoian and Stamm 2012). The number of CAs has grown into the hundreds and there is no consensus on what the set of trusted authorities should be. The transitive nature of trust in the CA infrastructure leads to not only the hundreds of CAs included in the root stores being trusted, but also an unknown number of intermediate CAs, to issue certificated that are

valid on the Internet. The security of the entire system depends on *everyone* behaving correctly, a single compromised or malicious CA could issue any certificate and violate the security of the entire system. Additionally, CA incentives do not promote practices in the best interests of end users (Roosa and Schultze 2013). The CA approach does not scale well, and we are observing more and more of its failures and shortcomings (Eckersley and Burns 2010).

The root stores of modern browsers and operating systems are used as trust anchors for validating X.509 certificate chains presented by TLS servers (Dang et al. 2010). CAs have the power to issue certificates for any domain, and the process is entirely opaque. As a result there is no feasible way to acquire an overview of issued certificates, not even by domain owners for certificates for their domains. Users have to trust CAs to behave honestly and correctly, and to notice, acknowledge and act upon any incidents. Significant cases of compromised CAs (Prins 2011), (Comodo 2011) have shown the need for further security measures (Clark and Oorschot 2013). TLS connections are routinely intercepted by Man-in-the-Middle (MITM) attacks. As many as 0.2% of the connections to Facebook show evidence of interception using forged certificates (Huang et al. 2014).

To improve the situation, Certificate Transparency (CT), a transparency mechanism standardized by the Internet Engineering Task Force (IETF) in RFC6962, requires certificates to be published in public append-only Logs and verified by browsers. Logging is not yet regularly enforced by browsers for all certificates, however Google Chrome 41 and later require CT logging for EV-certificates issued after 1 January 2015. It is the only browser to require any form of CT logging so far. In order for the browser to display the additional visual cues to the user that normally come with EV certificates, the certificate needs to be accompanied by Signed Certificate Timestamps (SCTs), which can be regarded as promises of inclusion in Logs. The number of SCTs depend on the validity period of the certificate, at least one Google operated Log and at least one non-Google operated Log are required (Laurie 2015).

1.2 Problem Description

Both the theoretical and practical aspects of CT are evolving rapidly. Although CT is standardized by the IETF as RFC6962, it is not commonly known how CT is deployed or used in practice. Google is pushing adoption by enforcing CT for EV certificates in Chrome, which influences how CT is used and what content is Logged, however there are no studies showing these effects or indeed any characteristics of Log content and usage.

1.3 Goals and Contributions

This thesis examines Certificate Transparency (CT) Logs, their behavior, characteristics, content and suggested protocol improvements.

The thesis specifically tries to answer:

- How are public CT Logs being used?
- How can the content of public CT Logs be characterized?
- How can correct Log behavior be asserted?
- Which operational issues can be observed in public CT Logs?
- How can partitioning attacks be detected/prevented?

1.4 Limitations

CT Log implementation is considered outside the scope of this thesis. Protocol design decisions are only discussed when they affect Clients/Monitors or when they have an impact on security or privacy for other entities than the Log itself. The thesis does not analyze or compare available open-source implementations on the Internet.

The implementations made for this thesis are not necessarily in any way optimal, or even a good way of implementing a CT Monitor. It serves as a proof-of-concept for how a Monitor can be implemented, but mainly it serves as a base for data gathering and has been designed with this purpose in mind.

The primary reference point for asserting standards compliance is RFC6962, newer drafts and updates are only considered where explicitly stated.
The suitability of Certificate Transparency in general, or its advantages over similar technologies, is not argued.

Chapter 2

Related Work

Certificate Transparency is a fairly new topic. There are specifications for how it is supposed to work, but unfortunately there is very little research about how CT is used in practice. This chapter focuses on other proposed measures, besides CT, for improving or replacing the current CA authentication infrastructure.

CT is far from the only attempt to reinforce the CA-based authentication system of SSL/TLS. All proposals from Section 2.2 propose improvements for X.509 certificate validation. Proposals may present variations on solutions to the same part of the problem.

Revocation is mainly performed using the Online Certificate Status Protocol (OCSP) (Santesson et al. 2013) and Certificate Revocation Lists (CRL) (Dang et al. 2010), neither of which works particularly well (Duncan 2013). An interesting hybrid technique is to use a transparency Log for revocations (Laurie and Käsper 2012), however is it not fully developed. These technologies are considered outside the scope of this thesis and will not be covered in further detail.

Section 2.1 describes client-centric approaches, relying on reducing the clients dependency on CAs by bypassing them partially or completely during the certificate validation process. Perspectives (Wendlandt et al. 2008) and Convergence (Marlinspike 2011) propose ideas for increasing trust agility by shifting trust away from CAs and putting control into the hands of clients.

Section 2.2 describes approaches relying on domain operators to either leverage the existing Domain Name System (DNS) to limit the trust in CAs, or by using third-party Logs to verify correct operations. DNS can be utilized using Domain Name System Security Extensions (DNSSEC) (Arends et al. 2005a; Arends et al. 2005b; Arends et al. 2005c), DNS-Based Authentication of Named Entities (DANE) (Hoffman and Schlyter 2012) and Certificate Authority Authorization (CAA) (Hallam-Baker and Stradling 2013). Log-based approaches include Certificate Transparency (CT) (Laurie et al. 2013), Accountable Key Infrastructure (AKI) (Kim et al. 2013) and

Attack Resilient Public Key Infrastructure (ARPKI) (Basin et al. 2014).

Section 2.3 describes suggested ways to use Logs for providing key distribution in other contexts: Coniks (Melara et al. 2015), or to provide transparency for other data than X.509 certificates: Binary Transparency (Zhang et al. 2015).

2.1 Perspectives & Convergence

One approach to improving authentication is to bypass the trust issues inherent in the current X.509 Public Key Infratructure (PKI) by adopting a distributed approach, bypassing the centralized CA infrastructure. One of the major issues regarding CAs is that traditionally there is no alternative authority available, either you trust the issuer of a certificate, or you do not. If you do not trust the issuer there is no means to securely connect to the intended website using TLS. A few large CAs certify a majority of connections (Ouvrier et al. 2015), thus revoking trust in one of these would break a clients ability to securely access a large share of the TLS secured sites. The same concern applies to browser and Operating System (OS) vendors where it may upset some users if they were to be unable to access their favorite sites. In essence, in the traditional system the site operator selects an authority to issue a certificate for the website, and the user has to trust that authority to securely access the website. A distributed approach to trust aims to reverse the power relationship, shifting the decision of whom to trust from the server to the client.

2.1.1 Perspectives

In the case of a Man-in-the-Middle attack, a malicious actor would modify traffic between the sender and the intended recipient. Perspectives suggest the use of multi-path probing to detect MITM attacks by requesting TLS certificates from several vantage points and comparing the replies (Wendlandt et al. 2008). An attacker would have to intercept and modify all responses to avoid detection. A TLS client contact various authorities (called Notaries) and request them to probe a specific site and return the received certificate. If all received certificates match, the client can proceed with increased confidence that the certificate in indeed the one served by the server. A similar suggestion, doublecheck (Alicherry and Keromytis 2009), uses Tor nodes to relay certificate queries as opposed to the Notaries proposed in Perspectives. As a result Doublecheck does not require new infrastructure to deploy, but is dependent on the availability of the Tor network[1].

Clients do not make any commitments to specific notaries. Querying a notary means that the client trusts this notary (optionally only if consistent with additional notaries) to verify *this certificate, this time*. The client can

[1] `https://www.torproject.org/`

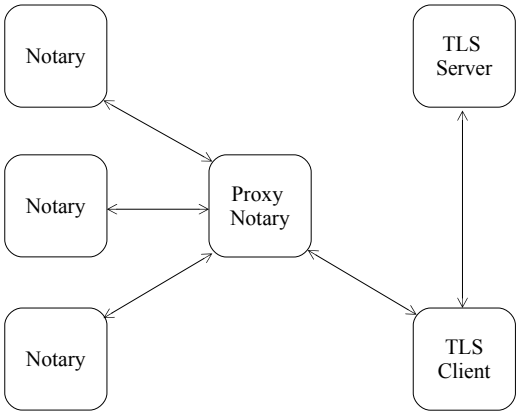

Figure 2.1: Webserver authentication using Convergence.

at any time revoke trust in a notary without introducing any usability- or security issues.

There are significant privacy implications of such a system, as these notaries have to know what sites to probe, and thereby also the browsing habits of the client.

2.1.2 Convergence

Convergence is an attempt to improve on Perspectives, addressing major issues concerning privacy and latency (Marlinspike 2011). The client software is available as a Firefox addon, and verifies server certificates without relying blindly on traditional CAs.

When a client wants to validate a certificate it contacts a set of notaries. To address the privacy issue in perspectives, all queries are proxied through a notary, see Figure 2.1. In this way one notary knows the identity of the client but can't see the traffic (or the requested website). The rest of the notaries can see what site is requested but can not see the identity of the client. Breaking the privacy would require that the proxy notary colludes with a second notary. Convergence also proposes several optimizations, including caching at Notaries, to reduce request latency.

The notaries themselves can use various techniques for certificate validation, such as regular CA signature verification, Perspectives or DANE. The notaries respond to the client with their opinion, and the ultimate trust decision lies with the client.

2.2 Improving CA Trust

The issues associated with the CA infrastructure can be mitigated by limiting the trust in CAs or by bypassing the CAs altogether. DNS is used to provide name resolution to clients, translating domain names into IP addresses. DNS, secured by DNSSEC (Arends et al. 2005a, Arends et al. 2005b, Arends et al. 2005c), can also provide associations between domain names and certificates or trust anchors. Tying certificates or CAs to domains through DNS can reduce the risk of unauthorized CAs issuing unwanted certificates. CAs can issue certificates for any domain without restriction on TLD. Therefore a compromised or rogue CA could issue valid certificates for any domain on the Internet. DANE (Hoffman and Schlyter 2012) or CAA (Hallam-Baker and Stradling 2013) provides means for limiting trust in CAs by requiring that a certificate association is present in the DNS record of the particular domain. This mechanism prevents passing of a mis-issued certificate as valid without the cooperation of anyone with administrative authority of the domain.

2.2.1 HTTP Public Key Pinning (HPKP)

Clients can safeguard against MITM attacks by creating a local mapping between identities and public keys, similar to what is present in an X.509 certificate. HPKP, as specified in RFC7469 (Evans et al. 2015), is an HTTP header set by the server to indicate that the client should "pin" a specific public key to the identity of the server. If, at a later point, the client receives a certificate chain that does not contain a key matching the local pinned key, the client should not trust the server. HPKP is intended to be used together with HTTP Strict Transport Security (Hodges et al. 2012). HPKP however has two major weaknesses. First, there are valid reasons for certificate mismatches, e.g. if the certificate provider has been changed. Second, HPKP is trust-on-first-use (TOFU) and assumes that the initial connection is not subject to a MITM attack.

2.2.2 DNS-Based Authentication of Named Entities (DANE)

DANE can be used to enforce various types of constraints on permitted certificates. If the specified constraints are not met, the client should not consider the connection secure regardless of whether the certificate provided by the TLS server passes X.509 validation or not. The constraints are specified as TLSA DNS Resource Records and secured using DNSSEC. Thomas Ptacek[2] points out several issues regarding DNSSEC and Adam Langley[3] draws conclusions about the shortcomings for domain validation using DANE.

[2] `http://sockpuppet.org/blog/2015/01/15/against-dnssec/`
[3] `https://www.imperialviolet.org/2015/01/17/notdane.html`

DANE provides means for setting the following constraints:

- *CA Constraint:* The specified CA must be present in the certificate chain presented by the TLS server. The presented certificate must pass X.509 validation.
- *Service Certificate Constraint:* The specified certificate must be used to certify the end entity. The presented certificate must pass X.509 validation.
- *Trust Anchor Assertion:* The specified CA must be present in the certificate chain presented by the TLS server. The CA does not have to be present in the root store of the client.
- *Domain-Issued Certificate:* The specified certificate must be used to certify the end entity.

2.2.3 Certificate Authority Authorization (CAA)

The CAA DNS Resource Record can be used to specify which CAs are authorized to issue certificates for a domain. CAA are not a part of the certificate validation process and will not prevent trust in mis-issued certificates. CAA records show which CAs are permitted to issue certificates for the domain, and should be checked by a CA before issuing a certificate. It is important to note that CAA records may change during the validity of a certificate, and a certificate issued by a CA not in the CAA does not necessarily imply that the certificate was mis-issued (Hallam-Baker and Stradling 2013).

2.2.4 Accountable Key Infrastructure (AKI) and Attack Resilient Public-Key Infrastructure (ARPKI)

AKI (Kim et al. 2013) and ARPKI (Basin et al. 2014) propose a PKI design with formally guaranteed security properties. The AKI proposes a Log server similar to that of CT in that it is based on a Merkle tree, however the Log is not append-only. The tree represents valid entries only and the entries are sorted. These properties allows the Log to provide absence proofs in addition to inclusion proofs. Since data can be removed from the Log, monitors are given a more central role in validating Log behavior and are a vital part of normal operations. ARPKI adds formal verification and guarantee of security properties to AKI.

2.2.5 PoliCert

PoliCert (Szalachowski et al. 2014) proposes the use of a combination of three mechanisms to improve certificate validation:

- An alternative to X.509 that permits multiple signatures from multiple CAs for the same subject.

- The use of Subject Certificate Policies for domain owners to specify restrictions on certificates for that domain.
- Public logs to prove the presence and absence of certificates and policies.

Together these changes would, if widely deployed, provide substantial improvement for certificate-based web authentication.

2.3 Other Transparency Applications

Creating public irrefutable proof of the presence of a piece of data at a certain point in time has proven useful for several applications. Certificate Transparency is used for X.509 certificates, but very similar technologies exist for sharing IM keys (Melara et al. 2015) and for ensuring transparency for binaries (Zhang et al. 2015).

2.3.1 Coniks

Key distribution for messaging is a pervasive problem and it is difficult even for large actors to come up with solutions that are reasonably secure. Apple's iMessage claims to be end-to-end encrypted, but still remains somewhat lacking when it comes to secure key distribution (Green 2015).

A proposal has been made for a key-verification protocol based on Merkle trees, called Coniks (Melara et al. 2015). The cryptographic primitives are used in the same fashion as in CT, with the major difference that Logs are queried by username and there is no mechanism for fetching all entries. A query by username returns all associated key bindings for that user. The Log does not guarantee correctness, but provides means for the owner to detect mis-issuance. This is analogous to the case of CT. Naturally it is not desirable to publish all users together with their public keys, such a list could easily be abused by spammers. Care is taken to assure that the privacy of users is maintained while still providing a valuable service. Coniks has been implemented for Off The Record (OTR) key exchange (Borisov et al. 2004) as a plugin for the Pidgin instant messaging client[4].

2.3.2 Binary Transparency

A promising application for Merkle Tree based transparency logs is Binary Transparency[5]. Binary Transparency ensure that downloaded binary packages are not modified before reaching the client, or that some clients are served with a modified (backdoored) software version (Zhang et al. 2015). After downloading a binary file, the client computes a hash of the received

[4] `https://pidgin.im/`

[5] `https://tools.ietf.org/html/draft-zhang-trans-ct-binary-codes-00`

binary and verifies that the same hash is seen by other clients. Although it is common for software distributors to publish checksums together with their packages, an adversary is here assumed to be in a position to modify an HTML page containing a checksum. Even if the software provider signs the binaries with a trusted key, it does not prevent the provider from serving selected clients with backdoored software. As Certificate Transparency is used to ensure that all clients see the same certificates even if the issuing CA is malicious, Binary Transparency is used to ensure that all clients see the same binaries even if the software distributor is malicious.

The need for mechanisms for ensuring correct binaries, and the fact that collusion to secretly implant backdoors exist, is highlighted in a public letter signed by many prominent security researchers and privacy advocates (Checkoway et al. 2014).[6]

[6]The authors use the term 'Binary Transparency' differently. They use it in the context of Reproducible Builds and *not* Merkle Tree transparency logs.

Chapter 3

Theory

This chapter describes the theory and standards behind Certificate Transparency, and the context in which it operates. Fundamental cryptographic theory is also covered, including an elaboration on the topic of trust.

3.1 Cryptographic Components

A few cryptographic concepts are essential to Certificate Transparency. The reader will need to understand these concepts and their implications in order to fully understand the technical parts of this report. This chapter does not give a full account of modern cryptography. The reader is encouraged to seek out more thorough material from other sources.

3.1.1 Public Key Cryptography

Public key cryptography (also called *Asymmetric Cryptography*) utilizes a pair of keys for each identity, one private key and one public key. The pair works in such a way that one key is used to encrypt, and the other key is used to decrypt. A client (let's call her *Alice*) generates a pair of keys, publishes the public key and stores the private key securely. When another client (let's call him *Bob*) wants to send a message to Alice, he uses Alice's public key to encrypt the message and sends it to Alice. She can then decrypt the message using her private key.

Anyone can use the published public key to securely send a message to Alice, but since she is the only one in possession of the private key, she is the only one who can decrypt and read the message. Public key cryptography removes the need to securely transmit encryption keys via a secure side channel prior to sending an encrypted message.

3.1.2 Hash Functions

A hash function is a function that takes an input of arbitrary size and produces a fixed-size output (called a hash). Hash functions are often called one-way-functions, since they are not reversible. It should be easy to calculate the output for a given input, but unfeasible to calculate what input generated a given hash.

Good hash functions are collision resistant. It should be unfeasible to either find an input that generates a specific output, or two arbitrary inputs that produce the same hash. Note that the second is substantially easier due to the birthday problem[1].

3.1.3 Digital Signatures

Digital signatures reverse the public/private key usage described in Section 3.1.1. The *private* key is used by Alice to encrypt a message. She can then send the message to Bob, who uses Alice's *public* key to decrypt the message and read it. The message is not secure, anyone can use Alice's public key to decrypt and read the message. However, since Alice's public key decrypts the message, the reader can be assured that whomever encrypted the message was in possession of the corresponding private key, which should only be Alice herself. Thus the message must come from Alice.

3.1.4 Digital Certificates

Certificates are digitally signed documents that tie an identity to a public key. The signer of the certificate promises that the association is correct. Certificates can be issued in a hierarchical fashion where the key included in a certificate is used to sign other certificates. The resulting tree can be distributed to create a Public Key Infrastructure (PKI) where clients only need to trust a few Trust Anchors (the root of the tree).

When Alice sends her public key to Bob, an interceptor (let's call her *Mallory*) can't use Alice's intercepted key to read any subsequent encrypted messages from Bob to Alice, but she can modify the message to include her own key instead of Alice's key. This is called a Man-in-the-Middle (MITM) attack. Without a certificate, Bob has no way to tell that the received key does not belong to Alice.

3.1.5 Example

Consider the example in Figure 3.1 of Alice sending a secure message to Bob, with both parties authenticating using certificates. Initially each party possesses a keypair consisting of a public and a corresponding private key, as well as a certificate to tie the public key to their identity. The certificates are either signed by a trusted entity, or recursively signed by another certificate

[1] `https://en.wikipedia.org/wiki/Birthday_problem`

signed by a trusted entity.

1. Alice requests Bob's certificate. She can either do this from Bob of from a third party. The certificate contains Bob's public key.
2. Bob sends his certificate to Alice. At this point Alice does not trust the content of the certificate.
3. Alice verifies the signature (chain) on Bob's certificate, if it is signed by someone Alice trust, she can now be sure that the key belongs to Bob.
4. Alice signs the message she wants to send using her private key.
5. Alice encrypts a message using Bob's public key.
6. Alice sends the signed and encrypted message to Bob, together with her certificate. At this point Bob does not trust the content of the certificate.
7. Bob verifies the signature (chain) on Alice's certificate, if it is signed by someone Bob trust, he can now be sure that the key belongs to Alice.
8. Bob decrypts the message using his private key.
9. Bob verifies the message signature using Alice's public key.

3.1.6 Transport Layer Security (TLS)

The terms SSL and TLS are often used interchangeably although they are separate protocols, SSL is obsolete and replaced by TLS. Both protocols are used to establish an encrypted connection between a server and a client. Normally only server authentication is performed using X.509 certificates. The client does not normally authenticate itself as part of session establishment, as the server is expected to serve content to anyone. If the service requires client authentication, this can be handled on a higher level using other credentials such as username and password or Universal two-factor Authentication (U2F).

TLS is a highly complex protocol, however the reader is not required to understand all its intricacies to understand the rest of this report.

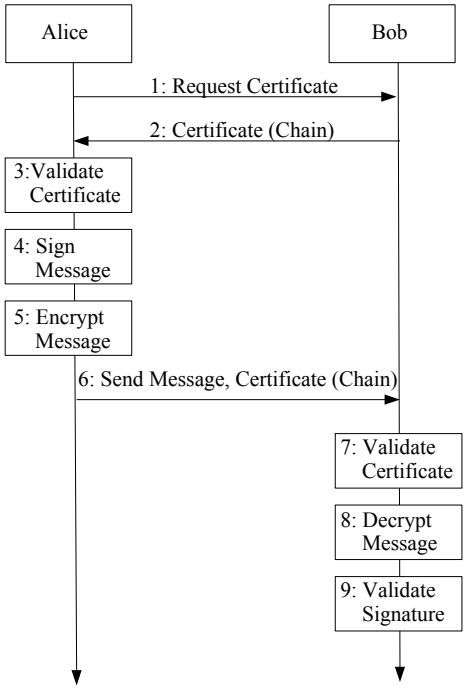

Figure 3.1: Simplified example of sending an encrypted and authenticated message

3.2 Trust

3.2.1 Before Certificate Transparency

The CAs stored in common browsers and operating systems are the foundation for trust in a pre-CT environment. These are by trusted by the browser to vouch for any domain and any intermediate CA. The relation is transitive, a trusted root CA can sign an intermediate CA. The intermediate is then authorized in turn to issue certificates or in sign another intermediate CA. If a client wishes to revoke trust in a CA, any certificates signed by that issuer will no longer be honored and the client looses the ability to securely connect to the associated domains. In the worst case this can lead to a too-big-to-fail scenario where trust in a misbehaving or compromised CA is not revoked on the grounds that it would lead to the loss of the ability to encrypt connections to a large portion of the internet. Blind trust in CAs is the norm (as well as in Browser/OS vendors and intermediate CAs) as there is no effective transparency.

3.2.2 After Certificate Transparency

CAs are still the root of the PKI and issue certificates, sign intermediate CAs etc. However, externally verifiable Logs and monitors work to make all available information public. In contrast to CAs, CT Logs are publicly auditable and provide insight into their activities, in order to enable anyone to verify their claims of correctness. No single entity has to be trusted blindly, anyone who detects suspicious activities can prove it, possibly leading to a modification of trust decisions. As anyone can operate the Certificate Transparency components: Logs, Monitors and Auditors, making it unfeasible for an adversary to control all available instances. An entity that is hesitant to place trust in existing services can operate their own Monitor.

3.2.3 Trusting the Client

Throughout this thesis the trustworthiness of the client system is assumed, up to and including the browser. Compromise at a lower level undermines security at higher levels. A compromised operating system or browser could for example choose to display security cues to the user even when validation fails or when transparency measures are absent. Some malware choose to replace the entire browser to bypass security mechanisms[2]. X.509 provides means for clients to authenticate entities on the Internet. The choice of what to do with that information lies with the client and can't be enforced from remote.

[2] `http://thehackernews.com/2015/10/malware-chome.html`

3.2.4 Trusting Standards

Certificate Transparency provides security and transparency based on cryptography. The security of the underlying cryptographical standards is essential to any and all claims made by CT. It relies on cryptographic primitives that are expected to remain secure for the foreseeable future, such as `SHA256`, `ECDSA` using the `NIST P-256` curve and `RSASSA-PKCS1-V1_5`.

In August 2015 the National Security Agency (NSA), the American intelligence agency, released a policy statement casting doubt on Elliptic Curve Cryptography (ECC) (NSA 2015). The update is a major turn on ECC, which the agency traditionally has been a strong advocate for. The rationale for transitioning away from ECC is still largely unknown with speculations ranging from *"The NSA can crack ECC"* to *"The NSA can not crack ECC"* (Koblitz and Menezes 2015). The NSA statement urges transitioning to quantum resistant algorithms, however currently there are no well tested and evaluated proposals for such algorithms. Rapid advances in quantum computing would threaten both ECC and RSA, for which there exist efficient attacks using quantum computers (Bernstein and Lange 2015). The NSA has previously been involved in weakening standards in order to be able to break online communication, as in the case of Dual Elliptic Curve Deterministic Random Bit Generator (DUAL-EC-DRBG) (Bernstein et al. 2015). The dual mission of the NSA, protecting American communications and breaking foreign (including allied) communications, means that the agency's motives and incentives are not straight forward.

The P-256 Curve standardized by the National Institute of Standards and Technology (NIST) is used for the Elliptic Curve Digital Signature Algorithm (ECDSA) signatures in Certificate Transparency. The updated NSA policy from August 2015 deprecated the NIST P-256 curve for ECDSA signatures. It has been suggested that there may exist intentional weaknesses in the NIST P-256 curve earlier as well (Gryb 2014), and the curve has some discouraging properties (Bernstein and Lange 2014). All in all the decision to mandate the use of the curve for signatures may be in need of some review. As of version 10 of RFC6962-bis, released October 2015, the proposed updated standard does not permit the use of stronger curves for ECDSA (Laurie et al. 2015).

3.3 Certificate Transparency: Protocol

Certificate Transparency is standardized by the IETF in RFC6962 (Laurie et al. 2013). Updates are being implemented and standardized in RFC6962-bis (Laurie et al. 2015).

Classic TLS operation (Figure 3.2) is extended with CT Logs, Auditors and Monitors, as well as several new interactions (Figure 3.3). As a certificate is issued, it is also appended to one or more CT logs. The log returns a

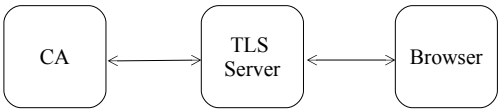

Figure 3.2: Components of normal TLS operation.

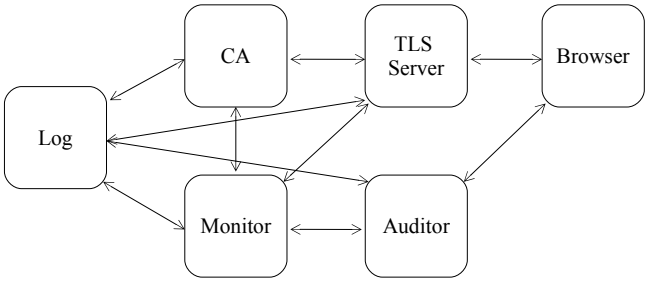

Figure 3.3: Certificate Transparency components and interactions.

signed promise of inclusion, called an SCT, which is used by the TLS server to prove to clients that the certificate is logged. Auditors and Monitors cooperate to ensure that Logs are behaving correctly and that the Log content corresponds to what the domain owners intended.

3.3.1 Certificates and Precertificates

Submissions to CT Logs can either be in the form of X.509 certificates or precertificates. Precertificates are X.509 certificates that contain a poison extension (`OID 1.3.6.1.4.1.11129.2.4.3: critical`) to prevent validation. A Log can use the body of a precertificate to create an SCT for a certificate that is still not created. The CA can then include the SCT in the certificate itself as an X.509 extension.

3.3.2 Interactions

CT introduces a number of new interactions between CT components and existing infrastructure, and extends classical TLS interactions. Figure 3.3 shows the components and interactions:

- *Log - CA:* The CA is assumed to contact a set of Logs and submit newly issued certificates to obtain SCTs. The SCTs are passed on to

the TLS Server operator together with the issued certificate.

- *Log - TLS Server:* The server operator may directly submit a certificate to one or more CT Logs to obtain SCTs.
- *Log - Monitor:* Monitors continuously keep track of Log operation and content. Monitors are used to detect mis-issued certificates by comparing seen and expected entries.
- *Log - Auditor:* Auditors continuously assert the integrity and consistency of Logs. It includes verifying STH signatures, freshness, and proofs of consistency and inclusion.
- *CA - TLS server:* The Server operator contacts a CA to acquire a certificate for TLS authentication. CT introduces SCTs to accompany the certificate.
- *CA - Monitor:* CAs possess knowledge of authorized certificates and should use monitors to assert that no unexpected certificates are issued in their name. Since complete lists of issued certificates may be considered a trade secret of a CA it may be difficult for third parties to make such an assertion.
- *TLS server - Browser:* The client - server communication of ordinary HTTPS sessions. CT introduces the transmission of SCTs from server to client using either a TLS extension, an X.509 extension or OCSP.
- *Monitor - TLS server:* TLS Server operators can use Monitors to assert that no unauthorized certificates are valid for the domain of the server.
- *Monitor - Auditor:* Partitioning attacks, where Logs present varying information, are discovered by comparing information from several vantage points. If a Log is caught misbehaving it is important to disseminate information about the incident to other Monitors and Auditors in order for them to reevaluate their trust decisions.
- *Auditor - Browser:* Auditors are likely to be a component of the browser itself[3]. The included Auditor can verify the validity of SCT seen by the browser, and that certificates are in fact included in Logs.

3.4 Certificate Transparency: Log

The central component of CT is the Log. The Log creates an append-only hash tree from the leaf certificates and has the ability to prove inclusion of any entry in the tree. A Log does not in itself prevent mis-issuance, but

[3] `https://www.certificate-transparency.org/how-ct-works`

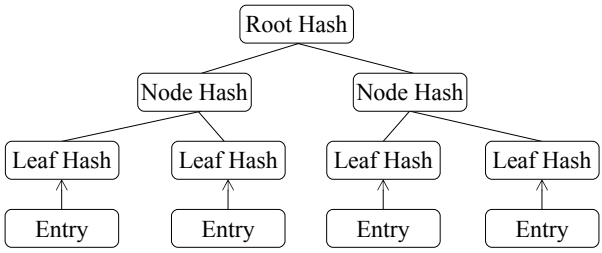

Figure 3.4: Structure of a Merkle Tree with 4 entries.

provides a means for legitimate operators to swiftly detect any mis-issued certificates and act to rectify the situation.

Logs are expected to be stable over time and to always be reachable. Protocols for adding and removing Logs are under development and expected to be included in the next version of CT. Such tasks are not trivial and will require some effort.

3.4.1 Log Structure

The structure of a CT Log is based on a binary Merkle Hash Tree (Merkle 1979). The tree consists of two types of nodes: leaf hashes and node hashes. All entries are hashed and used as leaves in the tree. Every node that is not a leaf, is a hash of its child nodes. Following this definition the root of the tree is a single hash that is influenced by every single leaf. Figure 3.4 shows an example of a Merkle Tree with four entries. The root hash is included in the Signed Tree Head (STH), a signed datastructure used to denote a version of the hash tree. RFC6962 mandates the use of the `SHA256` hash algorithm.

Example Consider a Merkle tree with 4 entries. Each entry is hashed into a corresponding leaf hash, and each layer of the tree consists of hashes of two nodes from the layer below. The root hash, also known as the tree hash, is the hash that is included in the STH.

3.4.2 Submissions

A Log entry is created upon submission of a (pre)certificate chain. The Log should only accept chains which end in a trusted root. Logs may accept entries that otherwise would not pass regular X.509 validation, e.g. due to having already expired. If the Log accepts the submission, it creates, signs and returns a fresh SCT.

If a Log receives a submission for a Certificate Chain that is already in its tree it may reply with the same SCT as for the initial submission. Producing a new SCT would not violate the standard, but may open the Log to spamming attacks. Every issued SCT would correspond to some form of database entry (details depending on implementation), resulting in an attacker being able to consuming Log disk space by resubmitting certificates over and over.

All surveyed Logs return old SCTs for entries that are already present in the Log. Plausible can under some circumstances create *one* additional SCT for a pre-existing entry. This is due to a side effect from a backend redesign that was implemented on the existing Log.

3.4.3 Consistency and Proofs

A Log may never rewrite history. Any STH can be proven to be consistent with any other STH from the same Log. If two STHs with sizes m and n, where $m < n$ are consistent, then it can be proven that the entries $0..m$ are equal in both trees. Consistency proofs consist of the tree nodes necessary to reconstruct both tree heads from known information.

If a Log can present a valid consistency proof, the Log has not altered old information in any way. Thus it is impossible for a Log to deny seeing a certificate that has at some point been logged, without failing to show consistency. Consistency proofs between two STHs are fetched over HTTPS:

```
GET https://<log server>/ct/v1/get-sth-consistency
```

3.4.4 Inclusion and Proofs

If a Log issues an SCT, the certificate chain must be merged into the tree within a specified time, called the Maximum Merge Delay (MMD). A Log can prove that a certain certificate has been included using an inclusion proof. RFC6962 specifies two APIs for requesting inclusion proofs[4]. Inclusion proofs consist of the nodes required for the client to use in reconstructing the root hash together with the leaf hash.

The size of an inclusion proof is logarithmic in relation to the size of the tree. If every address in the IPv4 address space had one host with one logged certificate chain, the tree would contain approximately $4.3 \cdot 10^9$ entries. Inclusion proofs would then contain only 32 entries. Currently Logs are far from this size, the largest Logs contain approximately than 10^7 entries, thus an inclusion proof should not contain more than 24 entries. Audit proofs can be fetched either by leaf hash or by index:

```
GET https://<log server>/ct/v1/get-proof-by-hash
```

[4]The terms *inclusion proof* and *audit proof* can be used interchangeably

`GET https://<log server>/ct/v1/get-entry-and-proof`[5]

3.4.5 Certificate Transparency 2 (RFC6962-bis)

The CT standard is still under development and subject to rapid and substantial change. The IETF are actively working on developing and standardizing a new and improved version of the specification. The draft specification uses the working name RFC6962-bis (Laurie et al. 2015), new versions are issued regularly, and the current version is 11. The new version contains several updates and improvements, solving important issues in the original standard. Unlike RFC6962, the new updated standard will be mature enough for more widespread adaptation. The initial push to use RFC6962 can be considered a pilot to prove the feasibility of using CT as a part of everyday TLS authentication.

3.5 Certificate Transparency: Auditor

An Auditor's main purpose is to verify the *cryptographic integrity* of one or more Logs, and may also be capable of verifying SCTs by requesting and verifying inclusion proofs from Logs. An Auditor would normally be located between a TLS client and CT Log as seen in Figure 3.3, and may even be incorporated into the browser itself. Note that there is no official document specifying the exact responsibilities of an Auditor, however the pseudocode from Figure 3.5 has been used as a basis for the Auditor implemented in this thesis.

3.5.1 Establish a trusted initial STH

An initial STH can be established in several ways. If a previously trusted STH exists it can be used as a starting point. If no previous STH exists, it can be build by constructing the complete Merkle Tree by fetching all entries in the Log and calculating the hash tree from the entries. An algorithm was developed for incrementally building and verifying an STH and is presented in Section 4.3.2. For a sizable Log this method will generate a significant amount of network traffic and calculation. This method ensures consistency of the current state, but can't verify that the current state is consistent with any previous state.

A significantly weaker method is Trust-on-First-Use. An STH is fetched and assumed to be correct. This method will catch inconsistencies introduced after the STH was fetched. In addition to being oblivious to anything up to this point, there is no guarantee that the current STH correctly represents the entries claimed to be included.

[5]This interface is declared as experimental in the RFC and is not supported by all Logs.

```
// Example Auditor pseudocode:

old_sth = import_trusted_sth()

while True:
    new_sth = get_sth()

    verify_signature(new_sth, log_pubkey)
    verify_fresh(new_sth)
    verify_larger(new_sth, old_sth)

    if new_sth["tree_size"] > old_sth["tree_size"]:
        new_entries = get_entries(old_sth["tree_size"],
                    new_sth["tree_size"])
        verify_consistency(new_sth, old_sth, new_entries)

    old_sth = new_sth
```

Figure 3.5: Example pseudocode for a (simplified) CT Auditor. The example code does not include functionality for verifying SCTs.

3.6 Certificate Transparency: Monitor

Monitors are responsible for verifying the content of Logs. Verifying content includes both the feasibility, i.e. that the alleged content indeed builds the advertised STH, and finding mis-issued entries. A Monitor may use various criteria for assessing Log entries, there is no official document specifying the exact responsibilities of a Monitor.

RFC6962 suggests the algorithm seen in Figure 3.6 for a CT Monitor (Laurie et al. 2013), which can be translated to pseudocode seen in Figure 3.6.

3.6.1 Certificate Evaluation

The Monitor developed for this thesis is described in detail in Section 4.3. It monitors certain domains as specified in the configuration file as a part of the Monitor software itself. It records observed certificates for the monitored domains, but does not try to determine the legitimacy of the certificates.

The task of determining the legitimacy of certificates is far from trivial and requires some form of additional information in order to verify that the content in Logs correspond to what the domain-owners intended. Operators have differing approaches, where some (e.g. Google) keep a close eye on their own domains and other domains of interest. Other (e.g. Comodo)

```
// Example Monitor pseudocode:

old_sth = get_sth()
verify_signature(old_sth, log_pubkey)
new_entries = get_entries(0, old_sth["tree_size"])
tree = build_tree(None, new_entries)
verify_root(tree, old_sth)

while True:
    new_sth = get_sth()
    verify_signature(new_sth, log_pubkey)
    verify_fresh(new_sth)
    verify_larger(new_sth, old_sth)

    if new_sth["tree_size"] > old_sth["tree_size"]:
        new_entries = get_entries(old_sth["tree_size"],
                                  new_sth["tree_size"])
        verify_consistency(new_sth, old_sth)
        tree = build_tree(tree, new_entries)
        verify_root(tree, new_sth)

    old_sth = new_sth
```

Figure 3.6: Example pseudocode for (simplified) CT Monitor.

offer services where anyone can inspect logged entries.[6] The latter approach effectively outsources the evaluation decisions to someone other then the Monitor operator. A more active approach is to notify domain operators about certificates relating to the domain in question, and let the domain operator validate the entries. However some improvements could be made if the Monitor is aware of normal behavior patterns, although when filtering it is always possible that legitimate issues are missed so care should be taken before implementing any such filtering.

First, many certificates are issued to replace previous certificates which are about to expire. In this scenario the subject and issuer of the certificate would likely be the same, but the validity period would be changed. Second, most organizations use a single provider of certificates. If an organization suddenly changes issuer or suddenly gets an additional issuer for the same domain, this might be cause for alarm. Many more optimizations are possible, and a likely area of future development.

Symantec were involved in an incident where they mis-issued several certificates, the incident is described in detail in Section 5.5. The first discovered mis-issued precertificate is shown in Appendix A. It serves as an interesting case study. The precertificate is correct and the corresponding certificate would pass the validation process as an EV-certificate. If the domain owner, in this case Google, monitors its own domain or is notifiedby someone who does, Google could quickly determine that this entry was not legitimately issued. In this case it should also be possible to determine that it is a case of mis-issuance already by the Monitor, as the entry is not consistent with other Google certificates. The issuing CA, Thawte, is not a regular CA for Google, and the certificate appears in parallel to other existing certificates for the same domain.

3.7 Partitioning Protection

This section details two complimentary methods for protecting clients against partitioning attacks where Logs present modified content to one or more clients. Partitioning attacks present certain clients with a Log version that includes rogue certificates while hiding the rogue entries from Monitors. To prevent such attacks, clients need to be able to verify that they are seeing the same version of the Log as everyone else (with high probability).

To sustain a MITM attack in a (RFC6962) CT environment, a colluding CA and Log would issue a certificate and accompanying SCT to present to a client. To achive a sustained attack the entry has to be hidden from Monitors, who would otherwise alert about the certificate to have it revoked, possibly also with reduced/revoked trust in the involved CA and Log. Creating two tree version in a Log, one including the rogue certificate and one where it is excluded, would result in two separate STHs. There would also

[6] `https://crt.sh/`

exist an SCT for which it is impossible to issue an inclusion proof (against the legitimate tree head).

3.7.1 Gossip

The IETF is actively working on developing a Gossip mechanisms to be included in future standards, specifying three ways to share information between clients and Auditors in order to detect partitioning: SCT Feedback, STH Pollination and Trusted Auditor Relationship (Nordberg et al. 2015). Gossiping can be done without serious impact on client privacy or incurring significant additional latency to TLS sessions (Chuat et al. 2015).

Note that all mechanisms described in this section are drafts, and are subject to rapid and substantial change.

SCT Feedback

A TLS client receives a certificate chain accompanied by SCTs (in some manner) from a TLS server. After validating the certificate and SCTs, the client proceeds with the connection and stores the received SCTs. When later connecting to the same server, the TLS client shares information about previously seen SCTs. SCTs must only be shared with the corresponding TLS server as not to disclose any privacy-sensitive information. The TLS server can then forward the SCTs to auditors and monitors without disclosing the association between the TLS server and client.

Detects attacks when the client has been subject to a MITM attack. It is assumed that an attacker can't sustain an attack indefinitely. During an attack the client is served with a malicious certificate accompanied with SCTs. These SCTs are stored, and when the attack is over they are shared with the legitimate TLS server. The SCTs are further shared from the TLS server to CT auditors, who can detect partitioning attacks by trying to verify inclusion of the SCT.

STH Pollination

An STH uniquely represents an entire tree version, and different versions of the same tree can be proven to consistent. Thus, it is enough to gossip STHs to verify that clients are seeing the same versions of Logs (or, at least consistent versions). STHs are normally not considered privacy sensitive, as long as they are shared by a large set of clients. TLS servers can act as STH Pools where anyone in possession of STHs can share them.

To assure that STHs are not privacy sensitive, they should not be unique or rare to a client. Two measures are taken to assure this. First, STHs gossiping is only concerned with *fresh* STHs, i.e. those less than 14 days old. Older STHs are silently discarded. Second, Logs are not allowed to issue new STHs more often than once per hour. Logs with a more rapid issuing

frequency are silently ignored. In combination, these measures ensures that there are no more than 336 fresh STHs per Log.

STH Pollination detects attacks when a non-consistent version of a Log is seen and the STH shared. STHs pollinated to STH pools are eventually seen by auditors and monitors, who try to verify consistency against the version of the tree they are seeing.

Trusted Auditor Relationship

Clients may trusts an Auditor enough to share privacy sensitive information, e.g. a service provider where the client is already "logged in", or a trusted organization. If so, they may enter a Trusted Auditor Relationship to augment the indirect communications in SCT Feedback and STH Pollination. The Auditor can verify all information the client is seeing to be consistent with what others are seeing, but at the cost of client privacy.

3.7.2 Multi-Signatures

There is an IETF draft proposal for using collective signatures in CT to prevent partitioning attacks (Ford 2015). Witness servers attest that they have seen an STH or SCT by contributing to a collective signature (Syta et al. 2015) on the STH/SCT. A client can then verify that the data has already been seen by the witnesses before it was sent to the client.

Security Properties

Adding a collective signature by multiple witnesses can prevent partitioning attacks, where one or more clients are silently served a modified version of a Log. If clients require N witnesses to sign of on seeing an STH or SCT, a Log would need to reveal the entry to that number of witnesses. Thus, a Log would need N colluding witnesses to convince a client to accept a modified Log. Choosing a sufficiently high threshold as well as reputable and trustworthy witnesses would dramatically increase the effort an attacker would have to go through to attack a client. Without multi-signatures or gossip it is enough for an attacker to compromise a CA and a Log server. If multi-signatures are used the attacker would also need to compromise N witnesses, where N could well be in the order of hundreds or even thousands. Collective signing scales well into thousands of witnesses with only small implications with regard to signature size and incurred latency (Syta et al. 2015).

If an SCT or STH has been witnessed and signed by trustworthy entities is can reasonably be assumed to be consistent with what the rest of the world is seeing, thus proving the absence of a partitioning attack against the client. One major advantage of collective signing over Gossip is its ability to prevent attacks, whereas Gossip works by detecting ongoing or earlier attacks.

Added Requirements

Producing a collective signature from 4000 well connected geographically distributed witnesses could be done in 1.5-3 seconds (Syta et al. 2015). Such a latency would be a significant increase for SCT issuance and may pose an obstacle. For STH issuance the idea appears highly feasible. SCT issuance is usually done in the form of an HTTP request-response, whereas STH issuance is done by a Log and published when ready. As seen in Table 5.2 Digicert already has a 12 hour delay between signing and publishing a new STH. In that context, an additional latency of 3 seconds would not be significant.

Witness servers are expected to have high availability. The size of a collective signature includes metadata about unavailable witnesses and therefor grows in size with the number of unavailable witnesses. Unavailable witnesses also impose security implications. A client would need to set a threshold on the number of permitted unavailable signers before the client revokes trust in the signed STH/SCT. If witnesses don't maintain a high availability, this threshold would need to be set low. A lower threshold would leave the client more exposed and lower the bar for a potential attacker.

3.8 Limitations to Certificate Transparency

Certificate Transparency is developed to provide a means to inspect issued certificates and identify mis-issuance. The technology is not a universal solution to all problems regarding Internet authentication today, nor is it without its limitations.

3.8.1 Legitimate Interception

Interception and surveillance is practiced by many corporations and organizations (Jarmoc 2012; Huang et al. 2014). Indeed it is sometimes considered a *security feature* and has been suggested to be implemented on a country level (JSC 2015). CT can't distinguish between legitimate and malicious MITM situations. Such interceptions can typically be performed by adding a local trust root to devices to permit a proxy server to decrypt traffic using a special certificate that chains to that root.

Whether routine interception of encrypted traffic is desirable or not, is a policy decision far beyond the scope of CT itself. The standard does not and will not contain any means for interception regardless of who performs it. To prevent browser security alerts on networks where legitimate interception occurs some workaround is needed. The Google Chrome browser solves this issue for proxies and key-pinning in a similar fashion, by not enforcing checks when the trust anchor is a *private root*, i.e. a root added by the administrator and not by the browser developer. A similar solution could be appropriate for Certificate Transparency.

3.8.2 Correctness vs. Transparency

CT does *not* guarantee correctness. It is important to understand that the fact that a certificate is correctly logged is not equivalent to the certificate being legitimately issued. CT does only provide transparency, it is up to monitoring parties to determine the legitimacy of entries and act accordingly. Detected cases of mis-issuance will not cause entries to be removed from CT Logs, but should trigger traditional revocation mechanisms such as CRL and OCSP.

3.8.3 Compromised Logs

Of course CT Logs and other CT components are not immune to being hacked, just like CAs. All public Logs except those operated by Google are operated by CAs. Under RFC6962 and the current Google Chrome policy two SCTs from publicly trusted Logs are required to validate an EV-certificate. If an attacker can gain control over two Logs, the attacker could present selected clients with malicious SCTs causing the client to accept a malicious certificate. Of course the certificate still needs to pass normal validation, but by additionally gaining control of two Logs the attacker can subdue the extra layer of security added by CT. Gossiping (Section 3.7.1) is intended to improve the situation and is a part of RFC6962-bis.

3.9 Chrome CT Validation

Of the major browsers, only the Google Chrome uses CT for certificate validation. As of May 2015, Chrome requires inclusion in at least one Google operated Log, and one non-Google operated Log. The total number of required SCTs are determined by the validity period of the certificate (Laurie 2015).

3.10 Attacks and Mitigations

3.10.1 Man-in-the-Middle

Man-in-the-Middle attacks can be classified by their approach to CT logging. CAs have been compromised before (Comodo 2011; Prins 2011), and it can happen again. If a certificate is issued without the knowledge and consent of the issuing CA, Certificate Transparency presents an excellent tool for detecting the mis-issuance and prompting an immediate revocation.

Man-in-the-Middle attack without CT: Consider the case of an attacker acquiring a rogue certificate by compromising a CA. This is the scenario of the Comodo incident in 2011. If the attacker can intercept HTTPS connections for the domain in the certificate, the attacker can authenticate

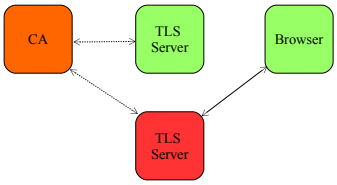

Figure 3.7: Man-in-the-Middle attack against a TLS client. A server with a valid certificate can intercept traffic without detection. Green actors are good, red actors are malicious and orange actors are performing malicious acts, either willingly or because they are compromise.

as the genuine site. Figure 3.7 illustrates this type of attack. The client will not be able to tell that it is not communicating with the intended webserver.

1. A CA is compromised by an attacker.
2. The attacker issues a rogue certificate.
3. The attacker tricks a user to connect to her instead of the website the user intended.
4. The attacker authenticates using the rogue certificate.

Man-in-the-Middle attack using logged rogue certificate: Currently a few CAs routinely log all issued certificates and hopefully the practice will become more common in the near future. If a mis-issued certificate is logged in one or more CT Logs. Either the domain owner or an independent monitor can detect the mis-issued entry as seen in Figure 3.8. Efficient attack detection requires more widespread enforcement of CT than what is present today. Even if the attack is detected by a monitor, the attacker can gain a brief window of opportunity where clients can be attacked before the certificate is successfully revoked.

1. A CA is compromised by an attacker.
2. The attacker issues a rogue certificate.
3. The certificate is logged using public CT Logs.
4. The certificate is detected my monitors and flagged as mis-issued.
5. The certificate is revoked by the CA.

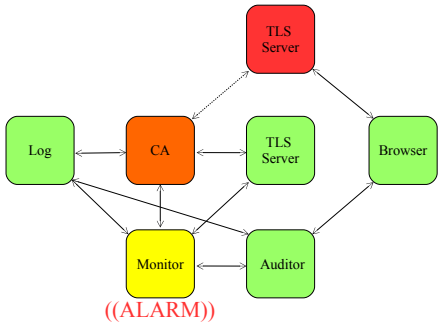

Figure 3.8: Man-in-the-Middle attack against a TLS client with logging of the rogue certificate. When a rogue certificate is issued and logged, CT monitors will detect it and notify affected parties to have the certificate revoked. Green actors are good, red actors are malicious, orange actors are performing malicious acts, either willingly or because they are compromised, and yellow actors detect the attack.

Man-in-the-Middle attack using non-logged rogue certificate: At the time of writing, the effect of having a non-logged rogue certificate would prompt a downgrade in EV certificates in Chrome, showing them without the added security ques. There is no mechanism enforcing CT logging in general, and the majority of certificates in the wild are not logged. If the CA responsible for the mis-issuance does not log the certificate the Browser will see that there are not accompanying SCTs, but the browser may choose to proceed anyway, see Figure 3.9.

Man-in-the-Middle with colluding CT Log: A possible attack scenario is where one or more CT Logs are controlled by, or colluding with, the attacker, see Figure 3.10. In this scenario the attacker has the ability to issue valid SCTs and certificates. The Log is capable of lying to selected parties such as Monitors in order to hide maliciously issued entries. All information seen by a single correct entity appears consistent and legitimate. In order to detect this type of attacks, entities need to communicate among each other to ensure that they are all seeing the same information. The current standard, RFC6962, does not contain measures for detecting malicious Logs. Gossip, as presented in Section 3.7.1, can be used to counter malicious Logs and will likely be included in the next version of the standard. For an Auditor to be able to detect the attack, as in Figure 3.10, the Auditor need to have access to an SCT for an entry the Log does not acknowledge the existence of.

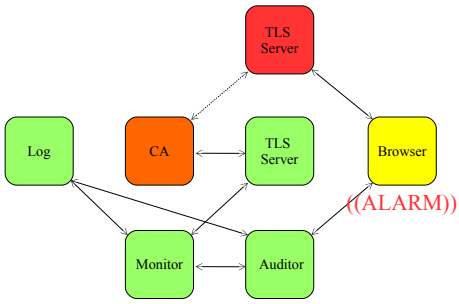

Figure 3.9: Man-in-the-Middle attack against a TLS client without logging of the rogue certificate. The rogue certificate is not logged and thus not acompanied by one or more valid SCT, therefor it will not be accepted for server authentication. Green actors are good, red actors are malicious, orange actors are performing malicious acts, either willingly or because they are compromised, and yellow actors detect the attack.

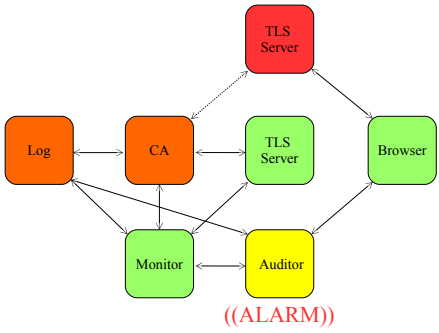

Figure 3.10: Man-in-the-Middle attack against a TLS client, with a colluding Log. A colluding Log can issue a valid SCT without presenting the rogue certificate to Monitors, however Auditors will detect the failure to provide inclusion proofs for the SCT presented by the rogue TLS server. Green actors are good, red actors are malicious, orange actors are performing malicious acts, either willingly or because they are compromised, and yellow actors detect the attack.

3.10.2 Legal Interception

Many countries have the legal means to coerce CAs and other companies under its jurisdiction to cooperate, in addition to the number of trusted state-run CAs where authorities have the ability to directly issue certificates (Soghoian and Stamm 2012). There are at least 46 governments with judicial power over one or more CA (Eckersley and Burns 2010), implying that there are at least 46 states with the power to intercept TLS traffic using rogue certificates. Since certificate issuing power is not restricted to specific domains, rogue certificates could be issued for any domain by any CA.

Such measures are by no means beyond what some authorities are prepared to do in order to subvert private communication. Beyond forcing companies to subvert encrypt, companies can also be barred from informing its users about the fact that their traffic is being intercepted. A prominent example that received much media attention is the email provider Lavabit, whos creator Ladar Levison chose to shut the company down and go public instead of handing over the encryption keys to his users' email[7].

Certificate transparency presents means for detecting Man-in-the-Middle attacks using certificates that were never authorized by the organization operating the website in question. This capability does not take intent or legal considerations into account, and can significantly reduce the ability of various governments and authorities to covertly intercept and decrypt TLS traffic. If providing such a service to clients is positive or negative depends on the perspective of the observer.

[7] `http://www.theguardian.com/commentisfree/2014/may/20/why-did-lavabit-shut-down-snowden-email`

Chapter 4

Methodology

This chapter presents methods, algorithms, settings, setups, and material used for the thesis. Large parts of the software and algorithms used have been developed specifically for the purpose of this thesis project, and these are shown in more detail below. Important design decisions are motivated and explained as applicable.

4.1 CT Logs

There are currently a number of accepted public Logs listed for inclusion and use for certificate validation in Chrome [1]. These Logs, as well as one non-listed Log operated by NORDUnet, have been used throughout this thesis. For the sake of readability these Logs will be referenced by abbreviated names instead of full URLs or IDs. All used Logs, their names are listed in Table 4.1. One log explicitly used for testing purposes has been omitted from all measurements.

4.1.1 Documentation

Some important information about Certificate Transparency and public Logs is readily available in documentation online. The protocol is standardized in RFC6962 and several additional drafts exists, specifying suggested improvements and additions. Log operators publish metadata about their Logs, some even publish the source code of their implementations. All these written sources have been used to gather information about the inner workings of CT. The people behind the Plausible Log and the Catlfish implementation have been consulted to give further insights into design decisions and operation.

[1] `http://www.certificate-transparency.org/known-logs`

Name	Operator	Base URL	Submitted to Chrome
Aviator	Google	ct.googleapis.com/aviator	2013-09-30
Pilot	Google	ct.googleapis.com/pilot	2013-03-25
Rocketeer	Google	ct.googleapis.com/rocketeer	2014-09-01
Digicert	DigiCert	ct1.digicert-ct.com/log	2014-09-30
Izenpe	Izenpe	ct.izenpe.com	2014-11-10
Certly	Certly	log.certly.io	2014-12-14
Symantec	Symantec	ct.ws.symantec.com	2015-05-01
Venafi	Venafi	ctlog.api.venafi.com	2015-06-11
WoSign	WoSign	ct.wosign.com	2015-09-22
Vega	Symantec	vega.ws.symantec.com	2015-11-13
Plausible	NORDUnet	plausible.ct.nordu.net	Not Submitted

Table 4.1: Known Certificate Transparency Logs

Discussion between stakeholders mainly take place on three mailing lists: the `trans` IETF mailing list, and the `certificate-transparency` and `ct-policy` google groups. There is also an IRC channel, `#ct`, on the `OFTC` server. These forums have been watched and participated in to include up-to-date information about the state of CT and ongoing discussions.

4.2 Auditor

For this thesis, a CT Auditor has been designed and implemented in the Python programming language. Constructing an Auditor serves dual purposes. First, it promotes understanding of the workings and behavior of vital CT components, as well as of their interactions with each other. Second, an Auditor is useful for providing data about CT Log actions. The Auditor has been used to gather parts of the data presented in Section 5. To provide continuous and reliable data the Auditor has been implemented and run as a Nagios plugin. This allows running on a professionally maintained server and can provide notifications of unusual behavior.

The Catlfish project[2] includes a toolbox of useful python tools for auditing and monitoring of Logs. Such tools are excellent for experiments and constructing various specialized setups for measurement and testing purposes. They constitute the foundation for the implemented Auditor and have proven highly useful for education.

The specification of an Auditor in RFC6962 (Laurie et al. 2013) is vague and leaves some room for interpretation. The following tasks are suggested and have been used as a basis for the implementation, while also adding checks for verifying compliance with Log specifications:

1. Verify consistency between two STHs using consistency proofs.

[2] `https://www.ct.nordu.net/`

2. Verifying inclusion of certificates (by validating the corresponding SCTs) using inclusion proofs.

3. Continuously fetch and verify STHs.

4.2.1 Nagios

Operations and infrastructure at NORDUnet are monitored using Nagios[3]. In order to provide uniformity and easy monitoring, the Auditor has been constructed as a Nagios test. This allows it to be run continuously and trigger alarms when appropriate. If any public Log encounters serious issues, a notification would be triggered.

4.3 Monitor

A CT Monitor has been designed and implemented in the Python programming language and used for data collection. The code-base is partially shared with the Auditor that is described in Section 4.2. The Monitor is used to gather data about Log behavior and content. In cases where questions are raised that can't be addressed through direct measurements, other sources such as online documentation and posting to mailing lists are used.

4.3.1 Architecture

The Monitor is designed to be flexible and compatible with existing systems. Figure 4.1 shows an overview of the design. The Monitor runs as a daemon and produces output in the form of multiple output files. Independent processes can inspect the output files to draw conclusions about the status of Logs or about the entries within them. The file-based output can be integrated with Nagios for convenient supervision. The use of command line arguments or `STDIN` input has been avoided for operation as a daemon, all configuration is handled via the configuration file. The Monitor uses a LevelDB key-value store for internal storage of certificate information for the purpose of domain monitoring.

4.3.2 Building the STH

The STH advertised by a Log contains a hash computed over data from all entries in the Log. To ensure that the Log has indeed incorporated all entries as promised, the Monitor can download all entries and use them to reconstruct the STH using the algorithm from Figure 4.2.

The hash tree can be viewed as a number of layers, see Figure 3.4, where the bottom layer consists of the entries and the top layer is the root hash. The append-only property of the CT Log ensures that if two nodes can be

[3] `https://www.nagios.org/`

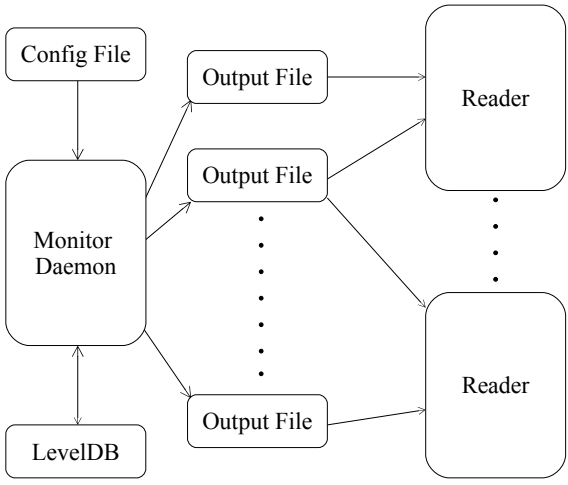

Figure 4.1: Overview of CT Monitor system architecture. The Monitor daemon reads its configuration from file. It uses a LevelDB storage for storing information about observed certificates and the domains to which they apply. The daemon produces a number of output files, that are in turn used by separate reader processes to draw conclusions about the state of CT Logs, as well as the monitor itself.

```
// Pseudocode for building the STH

sth_size = get_sth()["tree_size"]
local_size = 0

while local_size < sth_size:
    entries = get_entries(local_size, sth_size)
    leaf_hashes = hash_each(entries)
    layers[0] += leaf_hashes

    for layer in layers:
        while len(layer) >= 2:
            new_hash = hash(layer.pop(), layer.pop())
            layers[layer + 1] += new_hash

for layer in layers:
    if len(layer) == 1:
        layers[layer + 1] += layer.pop()

root = layers[max]
```

Figure 4.2: Algorithm for incrementally fetching entries and constructing a Signed Tree Head (STH). When two nodes are available at the same level of the tree, they can be combined into a single node on the next layer.

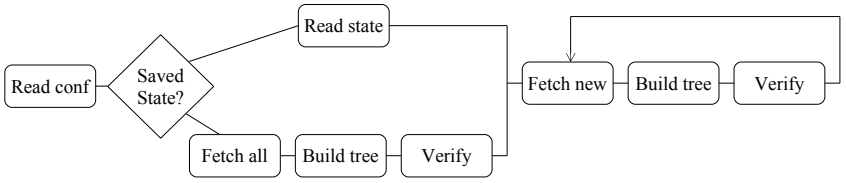

Figure 4.3: Monitor daemon process flow chart. On startup the Monitor creates a state either by reading a stored state from file or by fetching all entries from Logs to build the Merkle trees. When the Monitor is up to date, it goes into its operational loop of fetching and verifying new Log information.

hashed into a single node on a higher layer, this binding will always hold since no nodes can be prepended and the order of nodes can't be changed. The two initial nodes can therefor safely be discarded. Using this property, only a subset of nodes needs to be kept for enabling subsequent addition of new nodes. The number of nodes needed scales logarithmically with the number of nodes, as at most one node needs to be kept at each layer in the binary tree.

This property does not hold for Log implementations that use an ordered set of entries to create a tree, such as ARPKI (Basin et al. 2014) or CONIKS (Melara et al. 2015).

4.3.3 Daemon

The Monitor daemon is designed to run continuously. See Figure 4.3. The internal state, consisting of a partial Merkle-tree and the current STH, is saved upon exit and can be restored upon restart. On first start the Monitor will fetch all entries to build the Merkle tree, which is a costly process with regard to network traffic and processing. Due to the convenient format of saving a partial Merkle tree for each Log, the internal state scales logarithmically with the number of processed entries. For current Log sizes it will remain in the order of hundreds of kilobytes.

Once the daemon has established a state for all monitored Logs it will continuously look for updates, as seen in the bottom the right-most loop in Figure 4.3. The Monitor also checks for new or expired entries that relate to certain domains.

4.3.4 Log Files

The monitor writes to one log file for each monitored CT Log. Log files are written as plain text and can be read either with general text-processing tools or a custom reader, see Section 4.3.7. One log file is used for logging information regarding monitored domains, such as newly published or expired certificates that are valid for a monitored domain.

4.3.5 Certificate Data

Information about all logged certificates, such as issuer, subject and period of validity is stored in a manner that permits later analysis. Storage needs to be conveniently accessible and capable of handling significant amounts of data. The stored data is enough to permit an initial analysis of the relevance of the certificate, however the full chain can be retrieved from the Log if a more thorough analysis is desired.

The database solution has been selected based on the following assumptions:

1. Needs to provide all matches for a given domain.
2. Needs to handle wildcard matching for domains.
3. Needs to handle data from $\approx$ 100 million certificates ($\approx$ 70 GB as text).
4. Needs to provide a response within 5 seconds.
5. Needs to be extensible with new queries.
6. Needs to be runnable on a commodity machine.
7. A low number of dependencies is desirable.

Linear search through a text file on disk, assuming a file of $\approx$ 70GB, will take too long and thereby violate requirement 4. LevelDB seems to satisfy these requirements and has been used for the implementation. Using a key-value store with python bindings was deemed simpler than an external SQL database, and therefore preferable according to criteria 7.

Database Design

Certificate data is stored in a LevelDB[4] key-value store. Storage is divided into 3 databases. Two are used for domain lookups and one for certificate information lookups. The key-value pairs form a structure inspired by DNS for looking up certificates associated with a particular domain. A domain lookup would therefore require one database lookup for each level of the domain. E.g. a lookup for `www.nordu.net` would give three database lookups,

[4] Available at code.google.com/p/py-leveldb/

```
// Example: Fetching complete certificate from hash

sth = get_sth()
index = get_proof_by_hash(sth["tree_size"])["leaf_index"]
entry = get_entries(index, index)
```

Figure 4.4: Pseudocode for fetching certificate from hash.

for `net`, `nordu.net`, and `www.nordu.net`. The first two lookups would return lists of available next-level domains and the third lookup would provide certificate information. Wildcard support is implemented by storing wildcard domain and accepting either exact or wildcard matches. The .com domain is handled separately as it on its own makes up a significant portion of all domains. The database is used in such a way that all matches for a value are stored as a linear list. If the .com domain would have been stored with the others this would have meant that lookups for example.com would have to search through the entire list of .com domains.

Certificate information is stored in a separate domain, using the CT leaf hashes as key. This simplifies pointing several entries in the domain lookup database to point to a single certificate entry, as they can just store the same entry hash. Certificates can store subject domains either in the subject field or in the Subject Alternative Name extension, permitting one certificate to be valid for several domains. As this is common practice, it is desirable to store a single certificate entry for these domains.

Using leaf hashes as keys provides some flexibility. The leaf hash can be used both to fetch information from the local database as well as fetching the original entry form the Log. Requesting the full certificate chain can be achieved using three requests to the Log where the certificate was submitted, as seen in Figure 4.4.

1. `<get-sth>` will return the current STH of the Log.
2. `<get-proof-by-hash>` takes the hash and the tree size (found in the STH from the previous step), and returns leaf index + an inclusion proof.
3. `<get-entries>` takes the index as input and returns the full entry.

4.3.6 Log state

Mainly used for preserving internal Monitor state between runs, these files can also be utilized by external processes to inspect the current state of the Monitor itself (as opposed to the state of the monitored Logs).

4.3.7 Readers

Readers are independent from the monitor, new readers can be added as the need for them arise. The following readers have been developed during this thesis:

- *Domain Monitoring:* One of the major use cases for a CT Monitor is domain monitoring. A domain provider, domain owner or other entity can inspect all issued certificates for a particular domain. A reader has been implemented for that purpose as a Proof-of-Concept. For a certificate to be valid for a domain, the domain has to be included in either the subject field, or in the Subject Alternative Name extension.
- *Histogram Generator:* A reader has been implemented for creating data files for generating diagrams of historical Log data.
- *Statistics:* A reader has been implemented to compile statistics from log files, including update interval, average delay between signing and publishing STHs, current STH age, size and IP addresses used.

4.4 Content Analysis

4.4.1 Entry Overlap

The only major browser to enforce Certificate Transparency at the time of writing is Google Chrome. In order to pass CT checks for EV certificates in Chrome, a certificate must be accompanied by multiple valid SCTs , at least one operated by Google, and at least one by another operator, and additional SCTs depending on the validity period of the certificate (Laurie 2015). The potentially valid entries, e.g. those included in at least one non-Google Log included in Chrome, are checked against Google Logs to assert overlap. Inclusion can be checked by fetching all entries, or by resubmission. Resubmission was used as it does not require fetching and searching the content of all logs, but only the log from which the overlap is assessed. (Note that `get-proof-by-hash` can't be used since the hash is calculated over an entry that includes a Log-specific timestamp.)

All entries are fetched from one Log and subsequently submitted to all other Logs[5] . Upon submission of an entry, a Log returns an SCT or an error messages. If the entry was accepted, the Log returns an SCT where the included timestamp can be used to assess if the entry is new or was already present in the Log. Although a Log is allowed to issue a new SCT for a submission of an entry that is already present in the Log this is not done in practice. Systematically issuing new SCTs would present significant operational problems, such as opening the Log to Denial-of-Service attacks that consume resources by constantly resubmitting existing entries.

[5]The Vega log was published after the time of the measurement and was therefor not included.

4.4.2 Entry Characteristics

In order to draw conclusions about the content of Logs, all entries in all Logs up to 31 November 2015 are fetched and analyzed. The entries are compared with a "baseline" consisting of data gathered by monitoring the TLS handshakes seen at the University of Calgary, Canada (Ouvrier et al. 2015). The reference data was gathered 11 - 17 October 2015, putting the measurements less than two months apart. Any discrepancies between the certificates observed in TLS handshakes and the ones seen in CT Logs would indicate that logged certificates do not constitute a representative sample of used certificates.

Chapter 5

Results

This chapter describes data and findings from the active measurements described in Section 4 and from public documents.

5.1 Log Characteristics

A number of measurements have been performed to better understand and characterize the behavior of CT Logs. Log operators publish a small amount of important metadata for each Log when applying for inclusion in Chrome[1]. Each measurement listed below specifies if the data has been collected through published resources or through active measurements.

5.1.1 Access Protocols

Logs are normally accessed using HTTPS, however some operators have also made their Logs accessible over HTTP. Table 5.1 shows which Logs accept queries over HTTP connections. Transferring sensitive data over an unencrypted and unauthenticated channel such as HTTP is not recommended as an attacker could modify and snoop on the content. The signatures included in the SCT and STH data structures can however be used to verify the integrity of this data. Log queries may reveal personal information such as browsing habits, for instance if a client fetches inclusion proofs for SCTs presented by visited webpages. Normally, an encrypted HTTPS connection should always be preferable over unencrypted HTTP for interacting with Log servers. The data was gathered by active probing all Logs using both protocols.

[1] `http://www.certificate-transparency.org/known-logs`

5.1.2 Quirks of Distributed Systems

To meet demands of scalability and availability, CT Logs are implemented as distributed systems consisting of several components and replicas for critical services. As an example, the NORDUnet Log "Plausible" (running the Catlfish software) is using two independent front-end nodes for handling client requests. In the event of a merge, a dedicated merge node fetches all new entries from the client-facing front end nodes, creates a new tree, and hands the tree head over to a signing node in order to create a new STH. The new STH is published to both front-end nodes, together with any missing entries on the particular nodes.

It is not feasible to guarantee simultaneous publishing of a new STH on all nodes, the process may even be implemented in serial. Thus, at a given time, all front-end nodes may not have a consistent view of the current STH. (Note that all STHs are representing consistent Merkle tree versions , but at different points in time). An unlucky client making multiple requests to the public interface may en up seeing a regression of the advertised STH if a request to an updated front-end node is followed by a request to a lagging front-end node. The issue has been observed at several points and needs to be taken into account by Auditors.

If this behavior was seen in a single node, it would be a clear sign of misbehavior, however in a distributed setting it is a scenario that is hard to avoid without compromising other aspects. This effect is addressed in RFC6962-bis, which specifies how a front-end node should gracefully handle requests relating to STHs the node is not yet aware of, as well as an interface for clients to request several pieces of data from a single node.

5.1.3 Limiting Entry Request Chunks

Logs are free to limit the number of entries that can be requested in one chunk, delivering only the maximum number of entries for larger requests. Chunk sizes have been measured and can be seen in Table 5.1. If a client request more entries from a Log than the Log is willing to serve in a single chunk, it will return a chunk of entries starting at the first requested entry.

5.1.4 Maximum Merge Delay

The Maximum Merge Delay (MMD) is the longest permitted time the Log can wait between issuing STHs, and the longest permitted time between issuing an SCT and incorporating it into the STH. The MMD can be viewed as a promise from the Log, however a broken or malicious Log may fail to live up to that promise. MMDs for the surveyed Logs can be seen in Table 5.2. Publishing an MMD is only required for public Logs, the unofficial MMD for Plausible is 24 hours.

Name	Chunk	Signature	Access Protocol	Roots
Aviator	1024	ECDSA	HTTP, HTTPS	474
Pilot	1024	ECDSA	HTTP, HTTPS	474
Rocketeer	1024	ECDSA	HTTP, HTTPS	474
Digicert	65	ECDSA	HTTPS	57
Izenpe	1001	ECDSA	HTTPS	40
Certly	1001	ECDSA	HTTPS	183
Symantec	1024	ECDSA	HTTPS	19
Venafi	101	RSA	HTTP, HTTPS	357
WoSign	1001	ECDSA	HTTP, HTTPS	12
Vega	N/A	ECDSA	HTTPS	19
Plausible	1000	ECDSA	HTTPS	442

Table 5.1: Log characteristics. Chunk size is the maximum number of entries transferred at once. Signature is the algorithm used by the Log to sign data structures (STHs and SCTs). Access protocol shows if the Log responds to queries over HTTP and HTTPS. Roots is the number of accepted certificate-chain roots, entries that do not chain to one of these roots will not be accepted. No chunk size is listed for Vega as all entries are served in one chunk.

Before one MMD has passed from issuing an SCT, a Log may be unable to issue inclusion proofs for the entry corresponding to that SCT, or returning the corresponding entry. In practice, the update interval in Table 5.2 sets an upper limit on the delay between issuing an SCT and incorporating the corresponding entry into the STH.

5.1.5 Update Interval

Logs have selected different strategies for choosing STH update intervals. RFC6962 specifies that the STH must never be older than the Maximum Merge Delay, if a Log has not received any new entries during that time it must sign the same tree size with a new timestamp and publish. This is a fairly weak restriction and leaves the operator some freedom of choices.

A shorter interval can be convenient for both operators and clients. The Catlfish implementation is quite resource-heavy for large merges (creating a new tree with many newly submitted entries) which would suggest to merge often, in order to reduce load. For clients, a shorter interval means that received SCTs would be incorporated in the STH faster, which gives the ability to request inclusion proofs.

A very short updating interval can also open up to fingerprinting attacks against clients. In Gossiping, as described in Section 3.7.1, clients share STH information among each other. In theory, a Log could issue a unique STH for each client it wishes to trace by rapidly creating new STHs, ensuring

Name	MMD	Update Interval	Time to Publish
Aviator	24 hours	1 hour	22 minutes
Pilot	24 hours	1 hour	22 minutes
Rocketeer	24 hours	30 minutes	34 minutes
Digicert	24 hours	1 hour	12 hours
Izenpe	24 hours	1 minute	$<$ 1 minute
Certly	24 hours	10 minutes	$<$ 1 minute
Symantec	24 hours	6 hours	$<$ 1 minute
Venafi	24 hours	2 hours	3 minutes
WoSign	24 hours	1 minute	$<$ 1 minute
Vega	24 hours	6 hours	$<$ 1 minute
Plausible	Unspecified	12 minutes	2 minutes

Table 5.2: Maximum Merge Delay (MMD) is publicly advertised by the Logs, Update Interval is measured by time difference between STH timestamps. Time to Publish is the delay between signing and publishing an STH, measured as the time difference between the timestamp in an STH and the time it is seen by a monitor. Both measured timings have shown to be very consistent over repeated measurements for all Logs.

that the clients end up with unique STHs that can be linked back to them. In RFC6962 there is no measure to prevent this behavior, however Gossip introduces a minimum merge delay, meaning the a Log may not issue two separate STHs within a too short interval. This is to ensure that clients do not receive unique (or very rare) STHs. Izenpe and WoSign both have an update interval of 1 minute, see Table 5.2. Additionally Rocketeer, Certly and Plausible have update intervals of less than 1 hour.

5.1.6 Publish Delay

There is always a slight delay between signing a new STH and having it published, partially because of design aspects described in Section 5.1.2. The notable outlier is Digicert with a 12 hour delay between signing and publishing STHs. When questioned, Digicert claims the delay is for synchronizing between systems but leave no explanation for why this process would require 12 hours to complete[2]. All other logs publish STHs within 1 hour, see Table 5.2. The timings are very consistent over time, showing only small variations. The values in Table 5.2 are median values, which represent the typical case. The consistency of the measurements indicate automated procedures, where the delay is in some way decided by the operator. One explanation for the variations that do exist is that the amount of work required to build the new STH is influenced by the number of new

[2] `https://groups.google.com/forum/#!topic/certificate-transparency/eHdZM0wwlcQ`

entries. Thus, variations in the number of newly submitted entries can lead to variations in publishing delay.

5.1.7 Signature Algorithms

RFC6962 specifies that Logs may either provide signatures using ECDSA over the NIST P-256 curve, or RSA signatures (RSASSA-PKCS1-V1_5) with SHA-256. The signature algorithms for specific Logs can be seen in Table 5.1. The public keys and signing algorithms of the Logs are published as metadata. All Logs except Venafi use ECDSA for signatures. Neither algorithm is considered significantly better than the other with respect to security.

5.1.8 Roots

CT Logs only accept submissions of certificate chains that end in an accepted root. As with root stores in browsers and operating systems, there's no generally accepted consensus of the content of the root store of CT Logs. CT Logs provides an API for requesting all roots accepted by the Log. Table 5.1 shows that root are commonly shared between 3-5 Logs.

Generally a higher number of accepted roots is desirable. As a Log only accepts certificate chains that end in a trusted root, the usability of the Log increases as it accepts more roots. The rationale for enforcing restriction on submissions is DoS, and spam mitigation, no entries can be removed once accepted even if deemed unwanted or inappropriate.

A total of 503 unique roots were found. Many roots were found to be accepted by several Logs, see Figure 5.1. The three Google operated Logs overlap with each other, giving a misleading impression of overlapping roots. Figure 5.1a shows overlap for all Logs, 5.1b shows overlapping roots, excluding the Google Logs. There is no known current policy that requires SCTs from several CA operated Logs, however there is no drawback to log entries in all Logs that accept the entry. There may be business advantages for CAs to set up Logs for their own entries, while not accepting entries with roots from a competing CA. Internal CA rationale on how roots are chosen is not public information.

5.1.9 Size (Number of Entries)

Logs have no obligation to accept certificates (however they would serve little purpose if no entries were accepted). Once an SCT is issued, the corresponding certificate has to be incorporated in the tree within the time frame specified by the MMD. Some Logs, notably those operated by CAs, show a much smaller number of included certificates than their Google operated counterparts, see Figure 5.2. CT Logs are strictly append-only, with the obvious consequence that they will be ever growing. There is no possibility of shrinking a Log, if a Log gets too big it will have to be shut down and

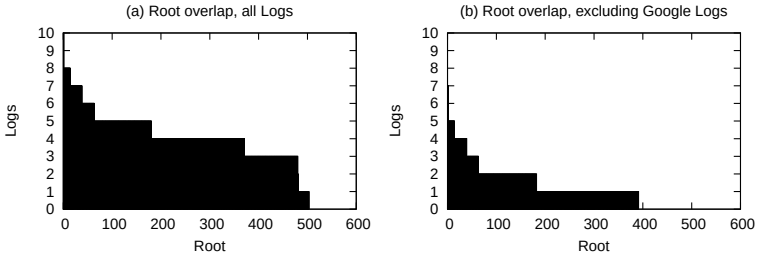

Figure 5.1: Number of Logs accepting each root for submitted entries. (a) shows overlap for all Logs, (b) shows overlap excluding Google Logs.

replaced by a new Log. The number of included certificates is not limited to the number of TLS hosts, as a single host can be the source of multiple included entries, for instance due to updating expired certificates. The increasing popularity of short-lived certificates can potentially increase the speed with which Logs are growing (Topalovic et al. 2012). The advent of enforcing CT for DV certificates could also be a source of rapid Log growth.

As seen in Figure 5.2, the Logs can be divided into three groups by size, with large Logs consisting of > 5000000 entries, medium Logs consisting of $50000 - 1000000$ entries, and small Logs of < 50000 entries. There are no Logs outside these intervals. Four Logs classify as large by this definition: Aviator, Pilot, Rocketeer, and Plausible. All four Logs have been populated by crawling the internet and submitting encountered certificates. Medium Logs: Digicert, Izenpe, Certly, and Symantec, appear to be populated mainly by submissions with the intention of use in CT. Small Logs: Venafi, WoSign, and Vega, are younger than the others and appear to still contain a large portion of test entries rather than entries intended for CT.

Let's Encrypt is an initiative to let anyone acquire free certificates and provides automated configuration for major web servers such as Apache and Nginx[3]. Let's Encrypt's certificates are valid for three months and allow for automatic renewal when they expire. CT is not only used to log Certificates, but is also an important tool for assessing how issued certificates are used (cite C3 talk). Potentially driving the development in the direction of ubiquitous encryption, it could bring with it an increase in the rate of certificate logging.

[3] `https://letsencrypt.org/about/`

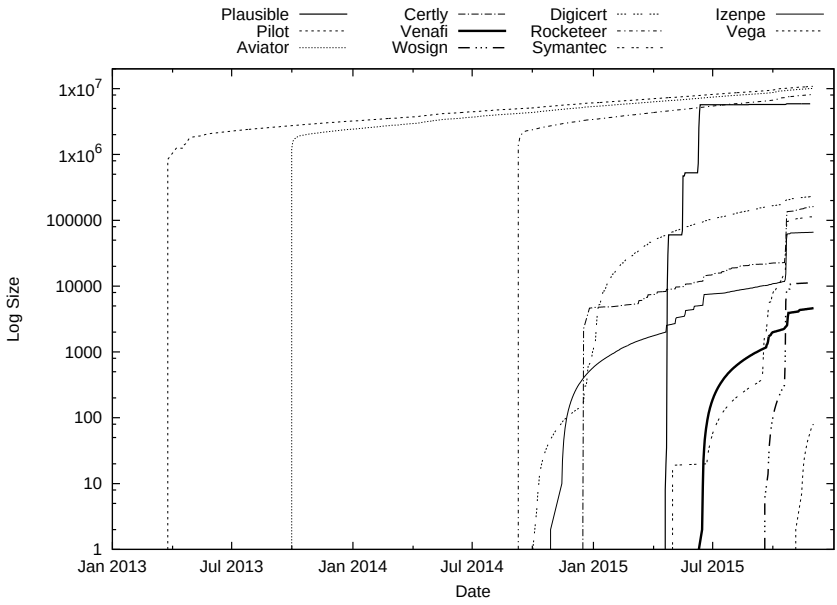

Figure 5.2: Entry count for known Logs. Historical data collected by timestamps on logged entries.

5.1.10 Entry Overlap

Several SCTs can be used when validating certificates in order to improve the assurance level. Chrome mandates a set of multiple SCTs issued by both Google and non Google Logs. Entries that are not present in both categories of logs will thus not fulfil Extended Validation requirements in Chrome.

For this reason, the usage of a Log can be assessed by examining the portion of entries that are also present in other Logs. Figure 5.3 shows the percentages of entries in CA operated Logs that are also found in at least one Google operated Log. Such entries could potentially have been submitted with the intention to validate in Chrome. There are four CA Logs with a high ($> 80\%$) portion of entries that overlap with Google Logs: Digicert, Izenpe, Certly, and Symantec. There is a clear correlation between age and overlap with Google Logs, these four CA Logs are older than the remaining two Logs Venafi and WoSign. As seen in Figure 5.2, the Venafi and WoSign Logs were started after the beginning of the measurement. The remaining portion contains certificates submitted for other purposes than TLS authentication. The exact nature of such submissions can only be speculated about, but whatever the reason for adding the entries, the append only property of Logs ensure that they will remain in the Log.

Table 5.3 shows a more complete listing of the overlap of certificates present in CA operated Logs. The CA Logs can here be separated and characterized. The WoSign Log stands out as it contains no substantial overlap with any other Log. This can be an indicator that the Log is mainly used for testing purposes so far. The young age of the Log would support this claim.

The usage of the three Google Logs vary. Certly, Venafi, and WoSign show a similar overlap with all three Google Logs. On the other hand Digicert, Izenpe, and Symantec show differing overlaps between the three Google Logs. For the purpose of Chrome validation it is enough with inclusion in one of the three, however there is no substantial drawback to utilizing all three.

The fact that each Log shows 100% overlap with itself is due to the reflexive property of overlapping. All entries found in Log X can be found in Log X.

5.2 Entry Characteristics

This section contains an overview of the content in CT Logs. Gathered statistics are used to compare and differentiate the Logs, as well as to compare with statistics for general Internet traffic.

Name	Digicert	Izenpe	Certly	Symantec	Venafi	WoSign
Aviator*	78%	82%	84%	55%	33%	2%
Pilot*	94%	75%	85%	96%	33%	2%
Rocketeer*	16%	20%	78%	22%	33%	0%
Digicert	100%	83%	50%	22%	24%	0%
Izenpe	1%	100%	1%	0%	10%	1%
Certly	4%	9%	100%	1%	29%	0%
Symantec	2%	0%	0%	100%	8%	0%
Venafi	0%	0%	0%	0%	100%	0%
WoSign	0%	0%	0%	0%	0%	100%
Vega	N/A	N/A	N/A	N/A	N/A	N/A

Table 5.3: Overlap from the CA operated Log in each column, to the Log in each row. Percentages calculated for the number of entries in the CA Log. Data collected 19 October 2015. Logs marked with * are operated by Google. The Vega Log was launched after the measurement.

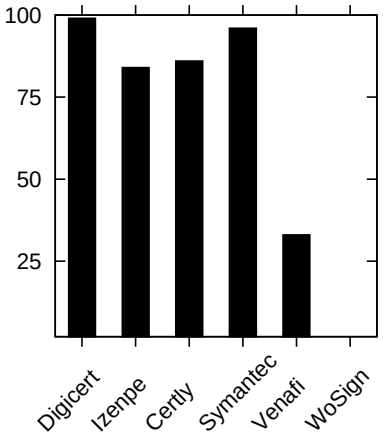

Figure 5.3: Percentages of entries in CA operated Logs seen in at least one Google operated Log. The data was collected on 19 October 2015.

5.2.1 X.509 Certificate Types

Certificates can be classified by type of validation. Certificate Transparency is only required for validation of Extended Validation (EV) certificates in Chrome, however a large number of Domain Validated (DV) and Organization Validated (OV) certificates are present in CT Logs as well. Possible reasons for logging certificates other than EV include testing and preserving public records of certain certificates. An incident occurred where Comodo issued certificated containing subjects with internal named in violation of regulations. The certificates in question were logged to create a public record where anyone can inspect the certificates.[4]

Table 5.4 shows the distribution of validation types for each Log. The share of EV certificates is roughly similar to what is generally observed on the internet ($\approx 5\%$) for all logs except Digicert, Izenpe, Certly and Symantec (61-78%). The significant skew in log size however means that there is a higher number of EV certificates in the three Google logs and Plausible than in the logs with a larger share of EV certificates.

For most Logs, with Digicert being the notable exception, EV entries have been added slightly after DV and OV entries. The CDF plots in Figure 5.4 show such tendencies for both Google operated Logs and CA operated Logs. Initializing and testing a Log by submitting a number of DV entries, and continuously adding EV entries for Chrome validation would produce an EV curve to the right of the DV & OV curve in the plots in Figure 5.4. Continuously adding all types of entries by crawling the web would produce overlapping curves in the CDF plots. EV entries for Aviator, Pilot, and Rocketeer have been submitted slightly DV and OV entries, indicating that Log content show signs of not only webcrawling, but also submission with the purpose to validate EV certificates with transparency. The dominance of DV and OV entries early in the life of Logs, especially Certly, Izenpe and Venafi, indicate that they were initially populated by DV/OV entries for other purposes. All Logs have seen one or more cases of bulk submission of one or both kinds of entries. These occurrences remain largely unexplained and influence the plots to a large extent.

This clear divide point to varying methods of population. The Google Logs and Plausible have been populated by crawling the web for certificates, which explains the close resemblance of the statistics. The pattern shown by Digicert, Izenpe, Certly, and Symantec indicates that the Logs are used for logging entries with the intent of using the SCTs in EV validation. This would mean that the Logs are populated by submitting newly issued EV certificates. How Venafi, WoSign and Vega are populated is not known, but their small size indicate that they are not routinely used for logging newly issued certificates yet.

[4] `https://cabforum.org/pipermail/public/2015-November/006226.html`

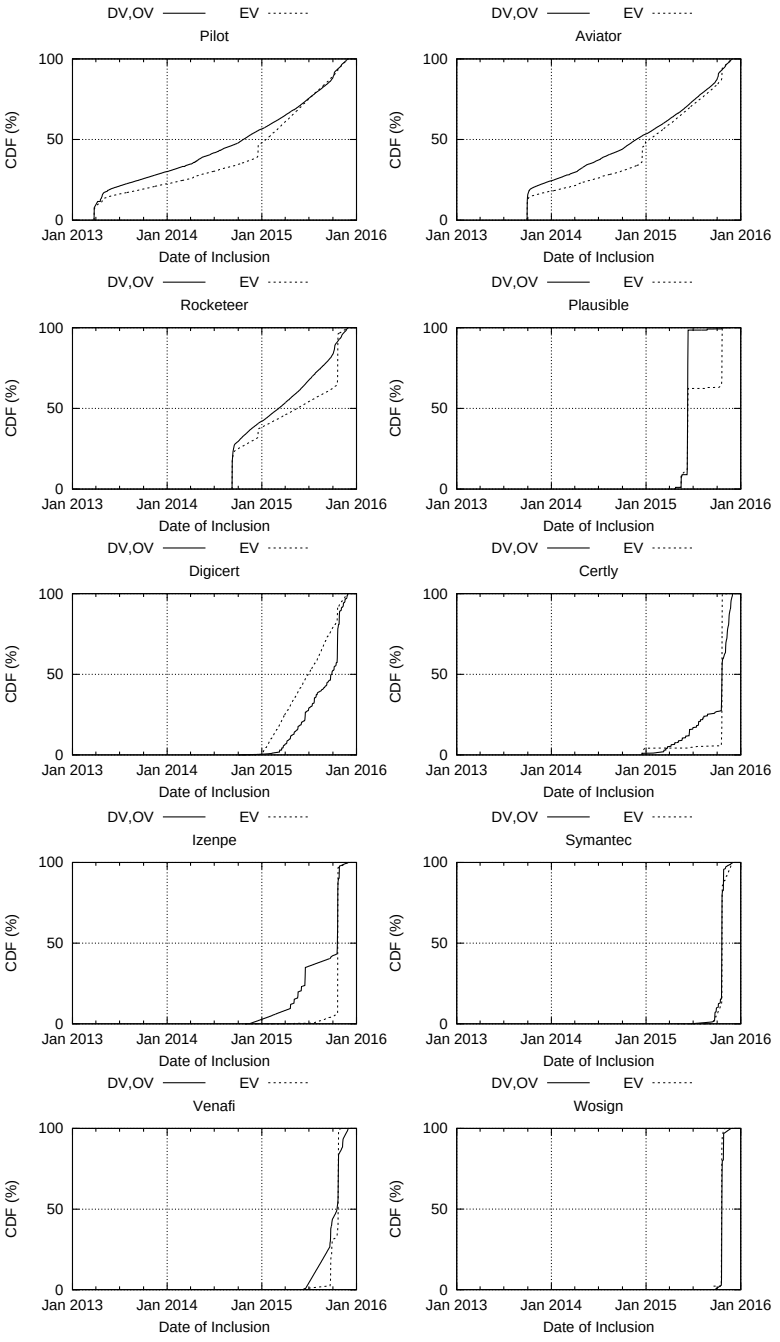

Figure 5.4: CDF plots of submission dates for Log entries for different types of Validation. DV and OV certificates are treated equally by the Chrome CT implementation and have thus been combined in the plots. Vega is omitted due to containing too few data points to provide any insights.

Name	DV	OV	EV
Aviator	87%	8%	5%
Pilot	87%	8%	5%
Rocketeer	87%	8%	5%
Digicert	18%	5%	78%
Izenpe	31%	1%	68%
Certly	36%	3%	61%
Symantec	21%	5%	74%
Venafi	85%	10%	5%
WoSign	97%	1%	2%
Vega	95%	0%	5%
Plausible	88%	7%	5%

Table 5.4: Distribution of certificate validation types per Log. Data was gathered 1 December 2015.

5.2.2 Algorithms and Keys

SHA1 and SHA256 are used as hash algorithms in signatures issued by CT logs. Table 5.5 shows the prevalence of the respective algorithms. SHA1 (used in $17 - 97\%$ of signatures) is outdated and is in the process of being deprecated[5], new certificates should be signed using SHA256. The prevalence of SHA1 is lower than in 2013 (98.7%)(Durumeric et al. 2013) for all logs, but for some logs lower than seen on the web today (24.9%) (Ouvrier et al. 2015).

Figure 5.5 shows when entries were added to Logs, divided by hash function used in the leaf certificate. The three Google Logs show clear pattern of `SHA1` certificates being added earlier than `SHA256` certificates. The same pattern is not as evident in other Logs, but Certly, Venafi and Izenpe show the same tendencies. Digicert even shows the opposite patter where certificates with stronger hash algorithms are added earlier than weaker ones. Pilot, Aviator, and Rocketeer are older than all the other Logs, the curve for added `SHA256` entries is not behind the curves for either typ in other Logs.

The vast majority of certificates in all Logs use RSA with 2048 bit keys ($\geq$75%), Table 5.6 shows the distribution of algorithms used for certificates in each Log. Notably, the three Google Logs have a significant number (16-21%) of certificates using id-ecPublicKey signatures.

RSA with 1024 bit keys is considered broken as an attacker who has put significant effort into pre-computing may be able to intercept and decrypt connections in real time using the Logjam attack (Halderman and Heninger 2015). The medium sized Logs (Digicert, Izenpe, Certly and Symantec) show only an insignificant share of deprecated encryption algorithms (RSA

[5] `https://blog.cloudflare.com/sha-1-deprecation-no-browser-left-behind/`

Name	SHA1	SHA256
Aviator	45%	55%
Pilot	48%	52%
Rocketeer	34%	66%
Digicert	17%	82%
Izenpe	25%	75%
Certly	19%	81%
Symantec	23%	77%
Venafi	55%	45%
WoSign	97%	3%
Vega	93%	6%
Plausible	22%	78%

Table 5.5: Distribution of certificate hash algorithms per Log. Data was gathered 1 December 2015.

1024) used in certificates, whereas the Logs populated by Internet crawling show a share $(1-3\%)$ that is representative of the Internet. Likewise the use of SHA256 in the medium logs show better standards than for the Internet in general.

5.2.3 Validation Against the Mozilla Root Store

All certificate chains in a Log ends in a root that is accepted by that particular log. There is no consensus about how roots should be selected, and selections varies extensively. Table 5.1 shows the number of accepted roots for each log, and Table 5.7 shows the number of Log entries for which the root certificate can be validated using the Mozilla root store. As logs are append-only it is expected that root certificates will eventually expire. For general web traffic, 94.8% of the seen certificate chains can be validated against the Mozilla root store (Ouvrier et al. 2015).

The number of entries $(4-9\%)$ in the Google Logs and Plausible that do not chain to a root in the Mozilla root store is comparable to what is seen in the wild. For CA operated Logs the number percentile is higher. One factor is that these Logs are smaller, increasing the impact of a number of testing certificates that do not chain to a root in the Mozilla store. There is a strong correlation with size, indicating that size is an important factor.

The number of entries chaining to an expired root is expected to increase over time as more and more root certificates expire, as expired entries are not removed from Logs.

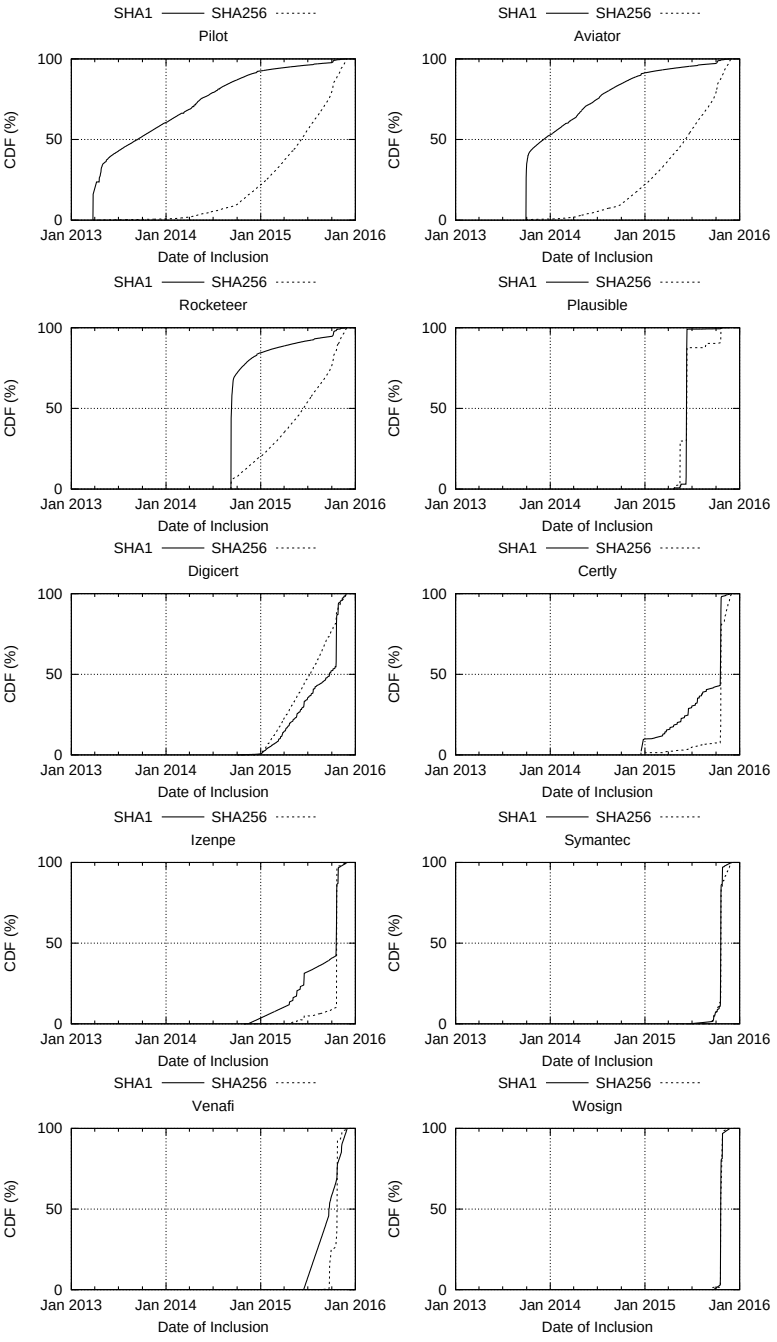

Figure 5.5: CDF plots of submission dates for Log entries for different signature hash function. Vega is omitted due to containing too few data points to provide any insights.

Name	RSA (1024)	RSA (2048)	RSA (4096)	EC (256)
Aviator	1%	78%	3%	17%
Pilot	2%	79%	3%	16%
Rocketeer	1%	75%	4%	21%
Digicert	0%	96%	3%	0%
Izenpe	0%	95%	5%	0%
Certly	0%	94%	5%	0%
Symantec	0%	97%	2%	0%
Venafi	0%	93%	5%	1%
WoSign	0%	99%	1%	0%
Vega	0%	95%	0%	2%
Plausible	3%	90%	3%	4%

Table 5.6: Distribution of certificate encryption algorithms. EC denotes *id-ecPublicKey* signatures as specified in RFC5480. Data was gathered 1 December 2015.

Name	Valid (Mozilla)	Root Expired	Invalid (Mozilla)
Aviator	86%	5%	9%
Pilot	89%	5%	6%
Rocketeer	89%	4%	7%
Digicert	83%	0%	17%
Izenpe	82%	0%	18%
Certly	85%	0%	15%
Symantec	71%	0%	29%
Venafi	48%	0%	52%
WoSign	4%	0%	96%
Vega	5%	0%	94%
Plausible	90%	6%	4%

Table 5.7: Portion of Log entries with a root certificate included in the Mozilla root store. As root certificates are expected to eventually expire, they are separated form certificates that are otherwise considered invalid. Data was gathered 1 December 2015.

5.3 RFC6962 non-compliance

Several Log implementations have been found to not be compliant with RFC6962. Digicert, Izenpe and Certly have failed to implement part of the API specified by the RFC. It should be possible to fetch an entry accompanied by an audit proof for that entry by index using the API call:

`GET https://<log server>/ct/v1/get-entry-and-proof`

Digicert, Izenpe and Certly all return an error saying *not found*. The other surveyed Logs reply with correct responses. The API is specifically stated to be experimental. Under normal operation there should not be a need to fetch entries accompanied with proofs by index. The same information can be acquired by first requesting an entry by index and using the leaf hash to request an audit proof for the entry. All Logs correctly implement these interfaces. The `get-entry-and-proof` interface is removed in RFC6962-bis.

Failure to implement this interface is not a major concern due to the facts that it is not used in normal operation and the same result can be achieved through other calls. It does however highlight the fact that clients interacting with servers should not blindly rely on specifications but rather investigate behavior of specific implementations.

5.4 Operational Issues

This section specifies operational issues and problems that have occurred during the time that Logs were monitored for this thesis. All issued described below are serious and have direct impact on Certificate Transparency operations on a global scale. All issues were publicly acknowledged by the Log operators in question, the problems detailed in Section 5.4.1 - 5.4.3 were also detected by the implemented Monitor.

5.4.1 Aviator

The Aviator Log was unavailable for over 5 hours on 6 October 2015. Google published a message[6] detailing the problem to the Google group *ct-policy*, see Figure 5.6. During the incident all three Google Logs were unreachable[7]. When the Logs are unavailable it is impossible to issue new SCTs and proofs, however certificate validation using SCTs works offline and can proceed without contact with Logs. Inability to submit new entries may be an inconvenience for CAs and other submitters. If reachability problems are recurring it may decrease the trustworthiness of the Log in question.

[6] `https://groups.google.com/a/chromium.org/d/msgid/ct-policy/CAP9QY5a26QcMmv4bSsX9%3DoeLsaOg5TbgO6RK5saPXQdwhOv3dQ%40mail.gmail.com`

[7] A fourth Log, Testtube, was also affected. Testtube is not covered in this thesis.

Particularly alarming in this incident is the fact that all three Google Logs were affected. Google operates three geographically dispersed Logs, which should increase the resilience of overall CT services. The incident highlights the single-point-of-failure in the DNS system, as well as the insufficient monitoring in place. Use of at least one of the Google Logs is mandated in Chrome, which may be a problem if all those Logs are unreliable. For other applications there is no need to require particular Logs, or Logs from any particular subset of trusted Logs, which would decrease the impact of any particular Log or Logs being unavailable.

If a Log would be unavailable for longer than MMD, currently 24h for all Logs, it would be a direct violation of the protocol. Such violations should not be taken lightly, as they may look like the Log behaving maliciously. A malicious Log trying to cover up inconsistencies could do so by simply ignoring requests it does not wish to answer, or reply with a suitable error code.

5.4.2 Pilot

The Pilot Log was unavailable for over 5 hours on 6 October 2015. The issue was caused by the same incident as described in Section 5.4.1.

5.4.3 Rocketeer

The Rocketeer Log was unavailable for over 5 hours on 6 October 2015. The issue was caused by the same incident as described in Section 5.4.1.

5.4.4 Digicert

Retrieving entries failed to return all requested entries for some requests on 24 October 2015. Digicert published a message[8] detailing the problem to the Google group *ct-policy*, see Figure 5.7.

The issue can best be describes as a failure to adhere to the specified API. Request for a set of entries did not necessarily return all those entries, excluding entries in the middle of the range. All entries would have been reachable using modified queries, however automated fetches and checks may have failed to verify Log content. As submission and issuance of proofs was not affected, the impact for clients should have been minimal. The only severely affected CT components would have been Monitors, as they fetch and verify entries. The Log was at all points capable of proving its consistency and inclusions of all entries.

Failure to adhere to standards is an inconvenience that if it recurs may discourage usage. At worst, it may desensitize operators to warnings and error leading to overlooking real problems.

[8]Original message available at: `https://groups.google.com/a/chromium.org/d/msgid/ct-policy/7d269627-6e85-4a63-8de6-b51fd16ba17a%40chromium.org`

SUMMARY:
On Tuesday 6 October 2015, the 'pilot', 'aviator', 'rocketeer', and 'testtube' Certificate Transparency Logs operated by Google experienced an outage for a duration of 5 hours and 20 minutes. If your service or application was affected, we apologize; this is not the level of quality and reliability we strive to offer you, and we are taking immediate steps to improve the platform's performance and availability.

DETAILED DESCRIPTION OF IMPACT:
On Tuesday 6th of October 2015 between 18:35 and 23:55 PDT, DNS lookups for ct.googleapis.com failed to correctly resolve resulting in 100% outage for clients attempting to connect to the service. The underlying Logs themselves were not affected and no data was at risk.

ROOT CAUSE:
During preparation for the planned addition of a DNS interface for inclusion proof checking a DNS configuration change was made to enable resolution of records below ct.googleapis.com. This change unintentionally affected DNS resolution for the existing ct.googleapis.com domain. Additionally, automated monitoring that was in place for all 4 Logs unexpectedly bypassed DNS resolution such that these probes did not detect service outage. The DNS change became effective at 18:28 and caused resolution to fail soon after.

REMEDIATION AND PREVENTION:
Google was notified by a user at 19:13 and the engineering team began their investigation shortly afterwards. The root cause was identified, a fix was prepared, tested, and the roll-out completed by 23:55 when normal operation was restored.

To prevent similar incidents in future, we have changed our automated monitoring to ensure that a more complete path (including DNS resolution) is tested during probes.

Figure 5.6: Public statement from Google about downtime for Aviator, Pilot and Rocketeer.

SUMMARY:
On Saturday 24 October 2015, not all, but, some requests to the DigiCert Certificate Transparency Log get-entries endpoint had missing Log entries in the middle of the requested Log entry sequence.

DETAILED DESCRIPTION OF IMPACT:
During the day of the 24th of October 2015 some requests to the get-entries endpoint would not correctly respond with all requested entries. The response would leave out entries in the middle of the requested sequence. For example if /ct/v1/get-entries?start=0&end=2 was called, it was possible that Log entry 1 would be missing from the response. There were no issues adding entries to the Log and there was no impact to the integrity of the Log structure.

ROOT CAUSE:
Our cloud provider had a maintenance window over the weekend of the 24th of October 2015, that caused some of our DB servers to be down for several hours. We replicate all Log data to all our data centers and most of our queries require a quorum to return the results. However, the Log entries query only requires one server in the cluster to respond. We designed the query this way for performance reasons. When the DB servers came back online they were out of sync and we had to manually run a process to re-sync the data.

REMEDIATION AND PREVENTION:
We modified the Log entries query to require a quorum before the results are returned. This change has been deployed to our production CT Log servers.

Figure 5.7: Public statement from Digicert about Log issues.

5.4.5 Symantec

SCT Signatures failed to comply with DER encoding principles. Symantec published a message[9] detailing the problem to the Google group *ct-policy*, see Figure 5.8.

Incorrect signature encoding may lead to verification failures, depending on the implementation of the verifying client. Symantec were issuing SCTs that were considered invalid by some clients, including Chrome and OpenSSL. These are arguably the two most important clients as they are very widely used, Chrome is the most prominent usage of CT.

The effect of validating a legitimate certificate accompanied by such signatures in Chrome would trigger a downgrade to DV status even if the certificate is classified as EV. Currently failure to prove transparency logging does not cause complete rejection of the certificate by the client. As CT is deployed more widely to include DV certificates and additional browsers, the issue would have caused the TLS server to be unable to initiate HTTPS sessions with most clients.

Depending on Log implementation, resubmitting the certificates in question could either yield the same invalid SCTs or fresh valid SCTs. If the Log insists on returning invalid SCTs another Log would have to be used to generate valid SCTs to be used in TLS validation.

5.5 Symantec Mis-issuance: A Success Story

Thawte, a CA owned and operated by Symantec, mis-issued a number of certificates. The problem was detected when an EV precertificate valid for `google.com` and `www.google.com` on 14 September 2015 was found in CT Logs (Somogyi and Eijdenberg 2015; Budington 2015). The issuance was intended as part of an internal testing process, but somehow ended up in two public CT Logs: Pilot and Digicert. The precertificate was discovered by monitors, as it was neither requested nor authorized by Google. Ensuing investigations, both internal at Symantec and by Google, revealed that the in total at least 164 certificated for 76 domains had been erroneously issued, and an additional 2458 certificates for unregistered domains (Sleevi 2015).

The initially discovered precertificate can be seen in its entirety in Appendix A. The subject of the certificate is particularly noteworthy, see Figure 5.9. The subject field contains the Common Name *www.google.com* and the Organization Name *Symantec Corp* indicating that the domain www.google.com belongs to Symantec, which is clearly not the case. Furthermore we can assume that Symantec possess the corresponding private key that was generated together with the public key included in the certificate. This key would enable Symantec to authenticate as `google.com` and

[9] Original message available at: `https://groups.google.com/a/chromium.org/d/msgid/ct-policy/67016d78-96cf-4ca0-a61e-e89ff8354e4b%40chromium.org`

Summary:
On October 19, 2015 Google reported an issue with verification of the signature part of the Signed Certificate Timestamp (SCT) returned by the Certificate Transparency Log Server operated by Symantec. This issue did not affect all SCT signatures, but a small subset.

Symantec reviewed the issue and confirmed the issue with Google that same day. Symantec then took steps to identify the root cause and subsequently released a fix on October 27, 2015 for any new SCTs returned.

Detailed Description:
The signature part of an SCT is ASN.1 DER-encoded ECC signature structure. Some of the encoded signatures were not in strict compliance with ITU-T Recommendation X.690 §8.3.2, which states:

"If the contents octets of an integer value encoding consist of more than one octet, then the bits of the first octet and bit 8 of the second octet:
a) shall not all be ones; and
b) shall not all be zero.
NOTE - These rules ensure that an integer value is always encoded in the smallest possible number of octets."

These signatures were issued in SCTs to certificates as well as precertificates logged to Symantec's Log Server. Many clients will take the encoded signature and proceed to verification operations. However, some clients such as Chrome and OpenSSL will take the encoded signature, parse it for individual components, and then re-encode it with strict DER compliance to verify if the re-encoded signature matches the original encoded signature. Such comparisons will fail for signatures that are not encoded with strict DER compliance. If the comparison fails, the client can opt out of proceeding with further signature validation.

Root Cause:
Symantec uses a Hardware Security Module (HSM) to sign SCTs. The HSM module generates fixed length integer components for an ECC signature. In certain instances generated integer components were padded with leading zeros. The DER encoding routine did not alter the integers before encoding them into the signature part of the SCT.

Remediation:
Symantec has updated its CT Log Server code to prevent issuing signatures that fail to comply strictly with DER encoding rules.

Figure 5.8: Public statement from Symantec about Log issues.

```
Subject:
    commonName                     = www.google.com
    localityName                   = Mountain view
    stateOrProvinceName            = California
    countryName                    = US
    serialNumber                   = 2158113
    businessCategory               = Private Organization
    organizationName               = Symantec Corp
    jurisdictionStateOrProvinceName = Delaware
    jurisdictionCountryName        = US
```

Figure 5.9: Subject field of mis-issued precertificate.

`www.google.com` to any client.

The Symantec incident is the first example of Certificate Transparency identifying mis-issued certificates in the wild. The incident proves that CT is a potentially effective mitigation for certificate mis-issuance. Of course the certificates in question were immediately revoked when discovered. Additionally, the people responsible had to walk (Kerner 2015). Most importantly however, Symantec will be subject to additional transparency requirements to ensure the incident is not repeated. Starting 2016-06-01 *all* Symantec certificated have to be logged using Certificate Transparency. The resulting measures shows that the CA climate is changing into one where it is actually possible to hold people and organizations responsible and mistakes have consequences.

Chapter 6

Discussion

6.1 Results Discussion

6.1.1 Certificate Transparency Usage

The main real-world usage of CT is currently validation of EV certificates in the Chrome browser. Minor applications include publishing certificates that for some reason are considered interesting. The goal of providing transparency for CA operations, including *all* issued certificates require more widespread deployment and enforcement than seen at this point.

Logs are populated using one of three strategies. Early in their life Logs are populated for testing purposes. Such entries would not necessarily pass X.509 validation. As Logs become established they are submitted for inclusion in Chrome. If accepted, SCTs from the Logs are accepted for certificate validation in TLS. At this point entries are submitted by CAs or domain operators to the Logs in order to acquire SCTs for their certificates. Additionally, some Logs crawl the web looking for entries to add.

6.1.2 Certificate Transparency Log behavior and Issues

The Auditor and Monitor components of CT are tasked with asserting Log behavior. Implementations for these components have been made as a part of this thesis and used for data collection. A number of issues have been identified, ranging from failures to comply with standards to operational issues such as temporary unavailability or incorrect responses to certain queries.

6.1.3 Mitigating Partitioning Attacks

Two potential mitigation strategies for partitioning attacks have been identified. Gossip is used to disseminate information between clients and auditors to detect Logs presenting inconsistent information, and is in the process of

being included in an updated version of the standard. Multi-signatures can be used to have witnesses attest to seeing certain information before publishing. Signatures are a guarantee that the information is available to auditors, and clients can refuse to trust information without a sufficient signatures.

6.1.4 Asserting Correct Log Behavior

Auditors and Monitors are essential components of CT as they together assert that Logs operate correctly and honestly. Monitors additionally watch Log content in search of mis-issued or entries of interest. Without these components, clients would need to place trust in CT Logs without verifying their trustworthiness. Such a scenario would look a lot like the flawed infrastructure of today. This thesis presents an implementation of a combined Auditor/Monitor that can be used to inspect Log behavior. This implementation serves as proof that both Monitors and Auditors can be operated by anyone, with low resource demands.

6.2 Representativeness

All known CT logs, except one explicitly for testing purposes, have been included in this study. All measurements have been made against live public systems and give an accurate picture of real-world CT operations. For content analysis all available content has been used leaving no uncertainty from data selection. New content is continuously added to Logs and there is no guarantee that the presented results correctly represent entries added after measurements were performed. New Logs introduced after measurements may differ from any patterns shown in this report, it is possible to use Logs for more specialized purposes than global logging of X.509 certificates.

The only browser to enforce CT at the time of this thesis is Chrome, thus its implementation and policies have been used for analysis. This particularly applies to the set of included Logs and SCT requirements for validating certificates.

6.3 Ethical Considerations

This chapter discusses the ethical considerations of the work contained behind this thesis, of Certificate Transparency (as the thesis contributes to the field) and to enhancing web authentication and security for everyone.

6.3.1 Active Measurements

Some of the measurements performed to gather the data presented in this thesis use the CT infrastructure, including public Log servers, in a non-standard manner. Measurements such as high volume submissions or retrieval of complete Logs can induce a significant load on servers.

At least two Log operators (Certly, NORDUnet) have noticed these measurements and the increased Log loads that follow. Certly initiated contact over IRC following high volume resubmission of certificates to their Log. They did not consider it abusive, only unusual and interesting.

Critical security infrastructure such at the CT infrastructure needs to be robust and be able to handle clients with non-standard behavior. The performed measurements have been designed to be as benign as possible. No active measurements have been performed that do not pose a valid, however unusual, legitimate use case.

6.3.2 Publishing Log Data and Metadata

Per design all data in Logs are public and can be accessed by anyone from anywhere. The data was created with the intention to be public and does not contain any sensitive information about clients. Publishing Log content and analysis thereof does not pose any ethical problems.

Some metadata for all Logs used for this thesis, except Plausible, is readily available online[1]. All unpublished metadata, e.g. update intervals etc, can easily be determined using open-source software. Such metadata can be considered public as it is readily available although it is not published by the operators. Publishing Log metadata does not constitute any ethical problem. An operator of Plausible has conceded that all covered information about the Log is public.

6.3.3 Client Privacy Considerations

Certificate Transparency should not introduce any new privacy- or security concerns for clients. The protocol contains some possibly sensitive parts which have been addressed is RFC6962-bis. Any remaining issues concerning privacy should be highlighted and made optional for clients. Advancing the adoption of RFC6962-bis is considered in the interest of clients.

Direct Log queries can potentially leak the browsing patterns of clients through requesting inclusion proofs, as the entries are mapped directly to certificates that include the subject domain. Utilizing DNS for Log queries limits the problem by not disclosing information to any new parties. The local DNS resolver likely received a DNS query for the same domain just prior to the CT query, thus it does not disclose any new information. Direct interactions between clients and Logs should be kept to a minimum.

Gossiping concerns sharing information between clients, and care must be taken in designing the protocol to ensure that no sensitive information is leaked to other clients. This concern in addressed at length in Section 3.7.1.

Political implications arise from purely technical research (Rogaway 2015). Politics is about power and technology serves to shift power one

[1] `https://www.certificate-transparency.org/known-logs`

way or another, and to drive change in society. Certificate Transparency is no exception.

6.3.4 TLS Interception

Certificate transparency as a whole makes subversion of TLS authentication more difficult. This should generally be considered positive, although it should be acknowledged that there might be attackers with the good of the client in mind, and these attackers are also hampered in their attempts to intercept and inspect encrypted traffic. Examples of such actors are employers gathering data from encrypted sessions of their employees with the intent to detect threats to employee and the company. Another example of interception is governments "protecting" citizens from harmful content on the internet through censoring.

If there is a need to intercept individual connections this can be done through configuring browsers to bypass CT checks for private trust anchors (as done for HTTP Public Key Pinning). Intercepting encrypted traffic without the knowledge and consent of the user is not considered to be in the best interest of the user.

6.3.5 Code

All code for this thesis is open-source. Hopefully the code can be useful to others in understanding Certificate Transparency.

6.4 Method Criticism

6.4.1 Log Selection

The Logs submitted for inclusion in Chrome have been used throughout this thesis. Google is by no means an independent authority on the matter, but one of the driving stakeholders. The Log selection has been adopted as it is the only set of Logs available. New Logs have been included in measurements as they are published, with the implication that some measurements do not include data points for all Logs. Excluding Logs that were introduced after the start of the study would not have provided any additional benefits. Google has a Log, nicknamed "Testtube", explicitly for testing purposes. Testtube was not considered representative of real-world CT operation and was excluded.

The NORDUnet Plausible Log was included even though it has not been submitted for inclusion in Chrome. Care has been taken to explicitly state where the status of the Log should be taken into account in the analysis of the results.

6.4.2 Implementations

This thesis relies to a large extent on an Auditor/Monitor implementation that has been used for measurements and gathering data. No guarantee can be given that the software is free from bugs affecting the quality of gathered material. The decision to use a single custom Monitor was made as it provides maximal control and understanding of the used software. Using several monitors would shift the focus away from CT and CT Logs toward Monitor comparison, which was not deemed consistent with the scope of the thesis.

6.4.3 Overlap Measurement

The method described in Section 4.4.1, used to produce the results in Section 5.1.10, has a few drawbacks that may impact the results. According to the standard a Log is allowed to issue a new SCT for an entry that is already present in the Log. Doing so is allowed but if done routinely the practice would introduce serious issued including opening up for spamming attacks. No evidence has been found of any Log except Plausible issuing new SCTs for old entries, however is has not been conclusively proven to always be the case.

Furthermore testing inclusion by submitting entries will effect the content of the Log, and thereby the outcome of future measurements as well as changing the properties that are being measured. Fixating a point in time before the start of the measurement to use as a reference while comparing entry timestamps means that the measurement will be consistent, however it is still true that the property being assessed is changed by the measurement.

6.4.4 Content Analysis

Logs vary significantly with regard to several important parameters. The number of entries differ by many orders of magnitude and log age vary by years. The effects of these, and other, parameters have not always been taken into account and may impact the results. A more thorough analysis taking further parameters into account may produce other results.

6.5 The State of Certificate Transparency

6.5.1 Current State

The field of Certificate Transparency as a whole is in somewhat of a beta stage. With its implementation in Chrome it is actively used on a global scale on the internet, but its impact on trust decisions is limited. At most failure to verify sufficient CT logging will trigger a downgrade in the status of a certificate from EV to DV.

6.5.2 Trajectory

The standard itself is in the process of being heavily revised. There are substantial differences between RFC6962 and RFC6962-bis, with the latter being expected to be adopted by the IETF as the updated standard shortly. The standards are not compatible, thus all existing implementations will need to be adapted as the standard is adopted. It is possible that some parties are holding of on deploying CT while waiting for the next version.

In order to achieve its potential in protecting clients CT would need to be implemented for the four major browsers, and the browsers would need to refuse validating a certificate that without SCTs or transparency proofs. The requirements would also need to extend to DV and OV certificates.

More actors are expected to independently run CT components over time. It is beneficial to have as many Monitors and Auditors as possible, each one makes it a little bit easier to detect malicious activities. The number of CT Logs is expected to grow significantly, but Logs are subject to stricter requirements on vetting and availability than the other components. For this reason it is not unrealistic that the majority of Logs will be operated by organizations with an interest in the field, such as CAs, browser vendors and security companies.

6.5.3 The Role of Google

The driving party behind the whole initiative of CT is Google. Having a major stakeholder developing new standards and technologies for the Internet should be done carefully to ensure that common interests are observed. On one hand this position gives Google the opportunity to tailor CT to its own interests, ensuring that the parties that are favoured the most are parties it controls, such as browser vendors. On the other hand, in order to be deployed and enforced, CT has to be pushed by an actor who is capable of demanding adoption. CT is not convenient for CAs and would likely not be spontaneously adopted. CT forces CAs to publish their business relationships, adds additional workload, and aim to call out CA malpractice. To counteract the negative aspects, some equally large positive aspect of adopting CT has to be presented. One such positive aspect is the possibility to have issued certificates accepted by browsers. This gives Google the opportunity to force CAs to action by refusing to validate certificates without sufficient SCTs in Chrome. As more browsers follow suit, the incentives for CAs will increase.

The policy decision for Chrome to require Google and non-Google Logs may not apply to other browsers, and as more browsers deploy CT verification the influence of Google will likely decrease.

Chapter 7

Conclusion

The goal of this thesis was to examine Certificate Transparency (CT) Logs, their behavior, characteristics, content and suggested protocol improvements. Code and methodology for Auditing, Monitoring and related measurements have been developed for this purpose. Important aspects of Log behavior has been investigated and varying usage patterns and maturity have been shown. The results of this thesis provides useful insights for researchers and practitioners alike, and should be particularly interesting for CA operators who are interested in operating their own CT Log.

7.1 Log Usage Patterns

Logs have been shown to fit into three categories with regard to size and age, correlating with usage patterns. Young and small Logs contain entries that are not valid using normal X.509 validation. Medium sized CA operated Logs contain a large number of entries submitted for the purpose of using the corresponding SCTs for authentication on Chrome. Large Logs operated by Google, as well as one operated by NORDUnet, are populated bu crawling the web. The contents of these are representative of the certificate landscape of the Internet in general.

7.2 Observed Issues and Incidents

Several operational issued in Logs have occurred during the time of the thesis, and are presented and analyzed. A case of mis-issuance by Symantec have been observed and analyzed. Several incidents were severe, where the impact could have been severe in a fully rolled out CT deployment. The possibility of partitioning attacks with colluding logs have been assessed and proposed solutions have been evaluated, with the most prominent solution being Gossip.

7.3 Future Research

Certificate Transparency is in the early stages of global deployment. The field is expected to see rapid development in both standards, deployment and enforcement over the coming few years. Following this development using the methods and tools from this thesis could provide a better understanding of the PKIX infrastructure, both as is looks today and as it evolves to adopt Certificate Transparency. If and when CT is enforced for DV certificates, it will provide a comprehensible data set of web certificates to be used by researchers for any number of purposes.

The longitudinal study of log growth could benefit from being extended to span a longer time period, preferably in the order of one or more years.

Let's Encrypt[1] presents an opportunity for a very interesting case study. Let's Encrypt is a CA that offers free X.509 certificates for use on web servers. They are committed to Certificate Transparency and use fairly short-lived (90 days) certificates.

The Log content analysis in this thesis only covers some cryptographic aspects of leaf certificates, but there are certainly more conclusions to be drawn from this data set.

[1] `https://letsencrypt.org/`

Appendix A

Terminology

CA	*Certificate Authority.* Responsible for issuing X.509 certificates and signing intermediate CAs.
CAA	*Certificate Authority Authorization.* DNS Resource Record, indicating which CA(s) is authorized to issue certificates for the given domain. CAA is *not* a part of the certificate validation process.
CC	*Certificate Chain.* A chain of certificates where the end certificate is associated with the subject in question, and the root is en entity in whom trust is placed.
CT	*Certificate Transparency.* Specified in RFC6962 and presented in detail in Section 3.
DANE	*DNS-Based Authentication of Named Entities.* DNS extension for providing certification information for a domain, in the form of TLSA records.
DNS	*Domain Name System.* Hierarchical resolution of information associated with domains, such as IP addresses. Specified in RFC4033, RFC4034 and RFC4035.
DNSSEC	*DNS Security Extension.* PKI based method for authenticating DNS entries.
DoC	*Duration of Compromise.* A metric for assessing the severity of incidents. Many protection mechanisms aim at reducing the duration of an attack, rather than preventing the attack.
DoS	*Denial of Service.* An attack class that aims to disrupt or crash a service, or otherwise make it unavailable.

DV-Certificate	*Domain Validation Certificate.* A weaker form of ownership verification performed by a CA before issuing a certificate. The process is sometimes automated.
EV-Certificate	*Extended Validation Certificate.* A stronger form of ownership verification performed by a CA before issuing a certificate. Full acceptance in the Google Chrome browser requires CT logging of the certificate.
HTTP	*HyperText Transport Protocol.* Unencrypted and unauthenticated protocol for browsing the web.
HTTPS	*HyperText Transport Protocol Secure.* HTTP over TLS. Encrypted and authenticated (server-only or server-client) protocol for browsing the web.
IETF	*Internet Engineering Task Force.* An open community that produces Internet standards most notably RFC documents.
MITM	*Man In The Middle.* An attack scenario where a malicious actor positioned between the two communicating actors establish two secure sessions, one with each endpoint. The attacker can then in each session claim to be the intended remote actor, making herself transparent and the two legitimate actors think they are communicating directly. Man In The Middle attacks are made possible by failures in authentication.
MMD	*Maximum Merge Delay.* The largest permissible delay from a Log issuing an SCT to incorporating the new entry into the tree and publishing a new signed tree hash.
MTH	*Merkle Tree Hash.* The root of a binary hash tree, where the leaves correspond to entries and every node on a higher level is the hash of the two nodes below it. The root node is then affected by every entry in the tree.
OCSP	*Online Certificate Status Protocol.* Protocol for verifying the status of received certificates. A client queries for whether a certificate is revoked or not (among other information).
OS	*Operating System.* Common operating systems for computers include Linux, Mac OSX and Windows.
OTR	*Off The Record.* Encryption protocol for securing Internet communication, such as Instant Messaging. Provides Perfect Forward Secrecy.

PKI	*Public Key Infrastructure.* Provisions means for securely distributing public encryption keys. Normally hierarchical structure in the shape of a tree, where the leafs are associations of identity and key, signed in one or more steps. Trust can then be placed in the root of the tree instead of each leaf individually.
PKIX	The X.509 PKI standardized in RFC5280 used to secure SSL/TLS.
SCT	*Signed Certificate Timestamp.* A signed promise from a CT Log to include the submitted certificate chain within the Maximum Merge Delay. One or more SCTs are provided by a TLS server to its clients.
SSL	*Secure Socket Layer.* Legacy protocol for establishing encrypted and authenticated channels over insecure mediums. Replaced by TLS.
STH	*Signed Tree Head.* Signed data structure specified in RFC6962, containing (simplified) timestamp, tree size and root hash of a Merkle tree.
TLD	*Top Level Domain.* The last section of a domain name, e.g. .se, .com or .onion.
TLS	*Transport Layer Security.* Modern protocol for establishing secure connections over insecure channels. Replaces SSL. Commonly used with services such as web, email, VoIP, IM, etc.
TOFU	*Trust On First Use.* The practice of assuming that the first connection is not subject to attack and using it to set a baseline for future verification.
X.509	Standards for Certificates, keys etc. used in SSL/TLS.

Appendix B

Mis-issued Certificate

```
SHA1 Fingerprint=8C:89:DB:BE:5B:91:17:7B:A5:95:77:CD:26:C9:76:
                 BF:71:89:37:18
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0a:b4:c7:3c:41:3a:01:94:9f:23:78:f2:b2:29:f6:6c
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=thawte, Inc., CN=thawte EV SSL CA - G3
        Validity
            Not Before: Sep 14 00:00:00 2015 GMT
            Not After : Sep 15 23:59:59 2015 GMT
        Subject: 1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60
                 .2.1.2
                 =Delaware, O=Symantec Corp/businessCategory=
                 Private Organization/serialNumber=2158113, C=US,
                 ST=California, L=Mountain view, CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:8e:d9:6a:64:8f:6b:e8:d8:71:bf:fd:da:e3:58:
                    62:35:8c:e0:31:0c:b8:65:27:48:7f:a3:f6:54:04:
                    3f:4d:23:5c:dc:46:18:eb:5c:1a:f8:e7:99:9e:f9:
                    20:e4:5f:61:bb:bb:35:28:5b:eb:f9:57:a7:1c:56:
                    48:fe:74:b3:27:fe:5a:dd:79:36:cb:dc:fa:a0:8b:
                    14:52:8d:e0:3a:7a:24:77:8e:4c:f8:dd:e9:34:35:
                    f8:8b:b0:30:3c:f1:dc:78:6e:e7:39:e2:4a:4d:6e:
                    49:5e:32:a7:25:14:c6:5a:85:f8:28:a8:1f:f5:00:
                    bb:ac:c9:31:d0:fa:cc:70:80:9d:95:86:df:84:0e:
```

```
                11:16:36:1d:b4:47:d6:aa:9a:a3:49:cf:af:f5:80:
                b6:48:6e:8f:c1:df:a5:aa:38:0a:1f:ed:97:77:7f:
                c1:46:f8:3c:a4:67:71:2d:ba:3d:45:fa:64:67:fa:
                5e:fd:33:0e:5a:0c:b5:13:bd:fa:66:c9:ab:aa:2d:
                29:5a:5b:c1:d4:eb:b6:db:88:f6:fc:dd:89:5f:22:
                30:40:65:18:72:59:69:ec:a2:89:35:58:0b:40:ad:
                bd:d7:44:ea:8a:91:cc:a0:8a:e9:7f:14:09:1c:41:
                2d:b5:2b:f6:2e:c9:28:f6:8b:d4:69:31:0b:b2:4c:
                e1:bf
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Alternative Name:
            DNS:www.google.com, DNS:google.com
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage: critical
            Digital Signature, Key Encipherment
        X509v3 CRL Distribution Points:

            Full Name:
              URI:http://ti.symcb.com/ti.crl

        X509v3 Certificate Policies:
            Policy: 2.16.840.1.113733.1.7.48.1
              CPS: https://www.thawte.com/cps
              User Notice:
                Explicit Text: https://www.thawte.com/
                               repository

        X509v3 Extended Key Usage:
            TLS Web Server Authentication,
            TLS Web Client Authentication
        X509v3 Authority Key Identifier:
            keyid:F0:70:51:DA:D3:2A:91:4F:52:77:D7:86:77:74:
                  0F:CE:71:1A:6C:22

        Authority Information Access:
            OCSP - URI:http://ti.symcd.com
            CA Issuers - URI:http://ti.symcb.com/ti.crt

        1.3.6.1.4.1.11129.2.4.3: critical
            ..
Signature Algorithm: sha256WithRSAEncryption
     8b:08:d4:31:46:6a:cc:14:77:a2:f1:57:3d:de:18:75:ec:dd:
     8a:99:f3:fe:e0:da:95:01:f3:94:0f:65:57:73:0a:bf:4e:63:
```

```
3e:41:c2:2b:b9:89:7a:c1:b4:d6:05:77:85:52:db:fa:46:a2:
1d:d0:29:01:23:18:d4:c4:a8:98:cf:85:04:e5:31:be:ca:91:
67:24:55:63:77:31:52:4a:09:92:12:72:b2:32:55:ee:c1:57:
e1:c0:7d:bd:78:7c:7d:db:10:11:f6:56:3c:fd:5f:94:9b:df:
fe:ce:54:5c:b3:7b:65:ae:6e:c2:bb:ea:c5:2a:6e:a4:5f:cc:
69:f4:65:d5:b4:e5:4d:e2:a0:b0:25:e3:90:2b:51:21:89:8c:
a7:87:17:2e:07:d0:bb:5c:cd:88:d1:2b:34:3a:6c:25:2e:a0:
4c:b9:ae:1e:61:8d:f2:11:c4:25:59:38:11:9e:58:db:0c:5b:
d1:7c:47:f7:c3:39:fb:a2:24:52:f4:90:8b:24:bb:94:2e:44:
57:cd:55:58:4d:cf:45:c5:9a:b2:58:8a:63:36:57:c1:3c:e4:
08:ec:c9:61:c2:77:3f:d3:14:e5:f9:cc:00:1a:41:64:bb:cc:
c5:49:a6:cc:bc:27:07:1d:e4:81:c9:79:40:07:55:c0:4e:6f:
c0:1f:59:a4
```

Bibliography

Alicherry, Mansoor and Angelos D. Keromytis (2009). "DoubleCheck: Multi-path verification against man-in-the-middle attacks". In: *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, pp. 557–563.

Arends, Roy, Rob Austein, Matt Larson, Dan Massey, and Scott Rose (2005a). *RFC4033: DNS Security Introduction and Requirements*. IETF. URL: `https://tools.ietf.org/html/rfc4033`.

— (2005b). *RFC4034: Resource Records for the DNS Security Extensions*. IETF. URL: `https://tools.ietf.org/html/rfc4034`.

— (2005c). *RFC4035: Protocol Modifications for the DNS Security Extensions*. IETF. URL: `https://tools.ietf.org/html/rfc4035`.

Basin, David, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski (2014). "ARPKI: Attack resilient public-key infrastructure". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, pp. 382–393.

Bernstein, Daniel J and Tanja Lange (2014). *SafeCurves: choosing safe curves for elliptic-curve cryptography*. URL: `http://safecurves.cr.yp.to`.

— (2015). *PQCHacks: a gentle introduction to post-quantum cryptography*. URL: `https://cr.yp.to/talks/2015.12.27/slides-dan+tanja-20151227-16x9.pdf`.

Bernstein, Daniel J, Tanja Lange, and Ruben Niederhagen (2015). *Dual EC: A Standardized Back Door*. URL: `https://projectbullrun.org/dual-ec/documents/dual-ec-20150731.pdf`.

Borisov, Nikita, Ian Goldberg, and Eric Brewer (2004). "Off-the-record communication, or, why not to use PGP". In: *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, pp. 77–84.

Budington, Bill (2015). *Symantec Issues Rogue EV Certificate for Google.com*. URL: `https://www.eff.org/deeplinks/2015/09/symantec-issues-rogue-ev-certificate-googlecom`.

Checkoway, Stephen, Roger Dingledine, Brendan Eich, Matthew Green, Nadia Heninger, Tanja Lange, Nick Mathewson, Ruben Niederhagen, Eleanor Saitta, Bruce Schneier, Peter Schwabe, Christopher Soghoian, Ashkan Soltani, Jon A. Solworth, Brian Warner, and Zooko Wilcox-O'Hearn (2014). *How to Protect Your Users from NSA Backdoors: An Open Letter to Technology Companies*. URL: `https://www.eff.org/deeplinks/2014/02/open-letter-to-tech-companies`.

Chuat, Laurent, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri (2015). "Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs". In: *Communications and Network Security (CNS)*.

Clark, Jeremy and Paul C van Oorschot (2013). "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements". In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 511–525.

Comodo (2011). *Comodo Report of Incident*. URL: `https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`.

Dang, Quynh, Tim Polk, and Daniel RL Brown (2010). *RFC5280: Internet X. 509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA*. IETF. URL: `https://tools.ietf.org/html/rfc5280`.

Duncan, Robert (2013). *How certificate revocation (doesnt) work in practice*. URL: `http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html`.

Durumeric, Zakir, James Kasten, Michael Bailey, and J Alex Halderman (2013). "Analysis of the HTTPS certificate ecosystem". In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM, pp. 291–304.

Eckersley, Peter and Jesse Burns (2010). "An observatory for the SSLiverse". In: *Talk at Defcon* 18.

Evans, Chris, Chris Palmer, and Ryan Sleevi (2015). *RFC7469: Public Key Pinning Extension for HTTP*. IETF. URL: `https://tools.ietf.org/html/rfc7469`.

Ford, Bryan (2015). *Collectively Witnessing Log Servers in CT*. IETF. URL: `https://tools.ietf.org/html/draft-ford-trans-witness-00`.

Green, Matthew (2015). *Let's talk about iMessage (again)*. URL: `http://blog.cryptographyengineering.com/2015/09/lets-talk-about-imessage-again.html`.

Gryb, Oleg (2014). *Why I don't Trust NIST P-256*. URL: `http://ogryb.blogspot.se/2014/11/why-i-dont-trust-nist-p-256.html`.

Halderman, Ale and Nadia Heninger (2015). "Logjam: Diffie-Hellman, discrete logs, the NSA, and you". In: *32nd Chaos Communication Congress*. URL: https://media.ccc.de/v/32c3-7288-logjam_diffie-hellman_discrete_logs_the_nsa_and_you#video.

Hallam-Baker, Phillip and Rob Stradling (2013). *RFC6844: DNS Certification Authority Authorization (CAA) Resource Record*. IETF. URL: https://tools.ietf.org/html/rfc6844.

Hodges, Jeff, Collin Jackson, and Adam Barth (2012). *RFC6797: HTTP Strict Transport Security (HSTS)*. IETF. URL: https://tools.ietf.org/html/rfc6797.

Hoffman, Paul and Jakob Schlyter (2012). *RFC6698: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. IETF. URL: https://tools.ietf.org/html/rfc6698.

Huang, Lin Shung, Alex Rice, Erling Ellingsen, and Charlie Jackson (2014). "Analyzing forged ssl certificates in the wild". In: *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, pp. 83–97.

Jarmoc, Jeff (2012). "SSL Interception Proxies and Transitive Trust". In: *Black Hat Europe*.

JSC, Kazakhtelecom (2015). *Kazakhtelecom JSC notifies on introduction of National security certificate from 1 January 2016*. URL: https://web.archive.org/web/20151202203337/http://telecom.kz/en/news/view/18729.

Kerner, Sean Michael (2015). *Symantec Issues Fraudulent Google SSL Cert*. URL: http://www.esecurityplanet.com/network-security/symantec-issues-fraudulent-google-ssl-cert.html.

Kim, Tiffany Hyun-Jin, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor (2013). "Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure". In: *Proceedings of the International World Wide Web Conference (WWW)*, pp. 679–690.

Koblitz, NEAL and A Menezes (2015). "A Riddle Wrapped in an Enigma". In:

Laurie, Ben (2015). *Improving the Security of EV Certificates*. URL: http://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxjZXJ0aWZpY2F0ZXRyYW5zcGFyZW5jeXxneDoyZGU5Yjg1MmVjNzc5NjQz.

Laurie, Ben and Emilia Käsper (2012). "Revocation transparency". In: *Google Research, September*.

Laurie, Ben, Adam Langley, and Emilia Käsper (2013). *RFC6962: Certificate Transparency*. IETF. URL: https://tools.ietf.org/html/rfc6962.

Laurie, Ben, Adam Langley, Emilia Käsper, Eran Messeri, and Rob Stradling (2015). *RFC6962-bis: Certificate Transparency draft-ietf-trans-rfc6962-bis-10*. IETF. URL: `https://tools.ietf.org/html/draft-ietf-trans-rfc6962-bis-10`.

Marlinspike, Moxie (2011). "SSL and the future of authenticity". In: *Black Hat USA*. URL: `https://www.blackhat.com/html/bh-us-11/bh-us-11-archives.html#Marlinspike`.

Melara, Marcela S, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman (2015). "CONIKS: Bringing Key Transparency to End Users". In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association.

Merkle, Ralph (1979). *Merkle Tree Patent, US4309569A*.

Nordberg, Linus, Daniel Kahn Gillmor, and Tom Ritter (2015). *Gossiping in CT*. IETF. URL: `https://tools.ietf.org/html/draft-ietf-trans-gossip-01`.

NSA (2015). *Cryptography Today*. URL: `https://www.nsa.gov/ia/programs/suiteb_cryptography/`.

Ouvrier, Gustaf, Michel Laterman, Martin Arlitt, and Niklas Carlsson (2015). *Characterizing Trust in HTTPS Communication: A View from the Edge*. Report. Linköping University and University of Calgary.

Prins, Ronald (2011). *DigiNotar Certificate Authority breach, Operation Black Tulip*. Fox-IT.

Rogaway, Phillip (2015). *The Moral Character of Cryptographic Work*. URL: `http://web.cs.ucdavis.edu/~rogaway/papers/moral.pdf`.

Roosa, Steven B and Stephen Schultze (2013). "Trust darknet: Control and compromise in the internet's certificate authority model". In: *Internet Computing, IEEE* 17.3, pp. 18–25.

Santesson, Stefan, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams (2013). *RFC6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. IETF. URL: `https://tools.ietf.org/html/rfc6960`.

Sleevi, Ryan (2015). *Sustaining Digital Certificate Security*. URL: `https://googleonlinesecurity.blogspot.se/2015/10/sustaining-digital-certificate-security.html`.

Soghoian, Christopher and Sid Stamm (2012). "Certified lies: Detecting and defeating government interception attacks against ssl (short paper)". In: *Financial Cryptography and Data Security*. Springer, pp. 250–259.

Somogyi, Stephan and Adam Eijdenberg (2015). *Improved Digital Certificate Security*. URL: `http://googleonlinesecurity.blogspot.se/2015/09/improved-digital-certificate-security.html`.

Syta, Ewa, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, and Bryan Ford (2015). "Decentralizing Authorities into Scalable Strongest-Link Cothorities". In: *arXiv preprint arXiv:1503.08768*.

Szalachowski, Pawel, Stephanos Matsumoto, and Adrian Perrig (2014). "PoliCert: Secure and flexible TLS certificate management". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, pp. 406–417.

Topalovic, Emin, Brennan Saeta, Lin-Shung Huang, Collin Jackson, and Dan Boneh (2012). "Towards short-lived certificates". In: *Web 2.0 Security and Privacy*.

Wendlandt, Dan, David G Andersen, and Adrian Perrig (2008). "Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing." In: *USENIX Annual Technical Conference*, pp. 321–334.

Zhang, Dacheng, Daniel Kahn Gillmor, Danping He, and Behcet Sarikaya (2015). *CT for Binary Codes*. IETF. URL: `https://tools.ietf.org/html/draft-zhang-trans-ct-binary-codes-03`.

På svenska

In English