Shedding Light on Certificates: The Web PKI and Certificate Transparency

Term Paper

Lukas Gebhard
lukas.gebhard@students.uni-freiburg.de

Submitted on 31/07/2018

Albert-Ludwigs-Universität Freiburg
Chair of Communication Systems

Supervisor: Rafael Gieschke

Abstract

Today, certificates are an integral part of securing web traffic. They are managed by a large ecosystem of service providers called certification authorities (CAs). Together with CAs and security protocols such as TLS/SSL and OCSP, they form a vast security infrastructure known as web PKI. However, this infrastructure has some serious vulnerabilities. Most importantly, each CA is a single point of failure and can thus break the whole security. In this article, I first give a high-level overview of the existing web PKI. Then I move on to outline a novel extension called certificate transparency (CT), which is designed to tackle some of the current web PKI's weaknesses. Finally, I conclude that although CT does not solve all of the existing web PKI's inherent issues, it does contribute to a safer web as it opens the certificate landscape to public scrutiny.

Contents

1 Introduction

Many web applications rely on secure end-to-end communication. In particular, applications such as e-banking, web shops and public web services would not even be possible without proper end-to-end encryption. The general infrastructure that enables secure web traffic is called *web public key infrastructure (web PKI)*. It is a large ecosystem comprising security protocols and *certificates* (cryptographic documents needed by the protocols to work). Certificates are issued by *certification authorities (CAs)*.

Unfortunately, in the current form of the web PKI, each CA is a single point of failure [4]. That is, if an attacker manages to compromise a CA, they can issue fake certificates for any domain on the web. This opens the door to *man-in-the-middle (MITM)* attacks: Imagine if an attacker obtained a fake certificate for the domain `example.com`. They can then potentially intercept and read all the communications between `example.com` and any of its clients without the victim even noticing.

This turned out not to be a hypothetical threat but a severe problem in practice. For example, in 2011 the Dutch CA DigiNotar was hacked [10][2] and more than 500 bogus certificates were issued, including certificates for websites owned by Google, Microsoft, Twitter, the CIA and Mossad. As a consequence, the Dutch government deemed it necessary to revoke all trust in certificates issued by DigiNotar. Since some of the CA's customers were Dutch authorities, many public services were no longer available and web communications were disrupted. In another incident, the US-based CA Symantec announced that they erroneously issued 187 certificates for domains they did not own, plus an additional number of 2458 certificates for unregistered domains [28].

Obviously, the web PKI's weaknesses need to be fixed urgently. A promising approach to tackle some of its shortcomings is *certificate transparency (CT)*, a proposed extension which is standardised through the IETF [24]. As the name suggests, CT aims to open the certificate landscape to public scrutiny. Using CT, it is much faster to revoke mistakenly issued certificates and to shut down rogue CAs. This has already proven valuable in identifying improper certificates issued by major CAs [28].

The objective of this article is to give a brief introduction to CT. In chapter 2 I outline today's web PKI, focussing on the role of certificates. Then, in chapter 3 I explain CT and how it integrates into the web PKI. Finally, in chapter 4, I draw some conclusions.

2 The Web PKI

In order to grasp the idea of CT, it is important to understand today's web PKI. Therefore, in this chapter I give a short overview of the web PKI's structure and mechanisms, focussing on the role of certificates as they are central to understanding CT.

2.1 Definition

Roughly speaking, the web PKI is the general infrastructure that enables secure web traffic. More specifically, the IETF working group on web PKI operations defines the web PKI as "the set of systems, policies, and procedures used to protect the confidentiality, integrity, and authenticity of communications between web browsers and web content servers" [20].

2.2 Structure

The web PKI's main entities are web browsers, web content servers and CAs. These are interlinked as follows. When a web browser wants to establish a secure end-to-end connection with a web content server, it usually initiates a TLS/SSL handshake. This handshake involves the server sending a TLS/SSL certificate to the browser. A certificate is a digitally signed document which binds an entity to its public key, and thereby guarantees that this key actually belongs to the entity. Thus, if Alice receives a certificate from Bob which she considers trustworthy, she can safely use Bob's public key from his certificate to establish a TLS/SSL connection. However, Alice will only trust the certificate if she also trusts the respective issuer.

This is where CAs become involved. A CA is a service provider which issues certificates. Its customers are usually operators of web content servers. However, a CA can also issue certificates for other CAs. This leads to a hierarchical structure of CAs. The CAs at the top of this hierarchy are called *trust anchors* and have self-signed certificates. The path from a trust anchor to a web content server is called *certification path*. The overall structure is visualised in Figure 2.1. In the example, A and B are trust anchors with subordinate CAs C, D, E, F and G. Instances of certification paths are A-D-H and B-E-G-L.

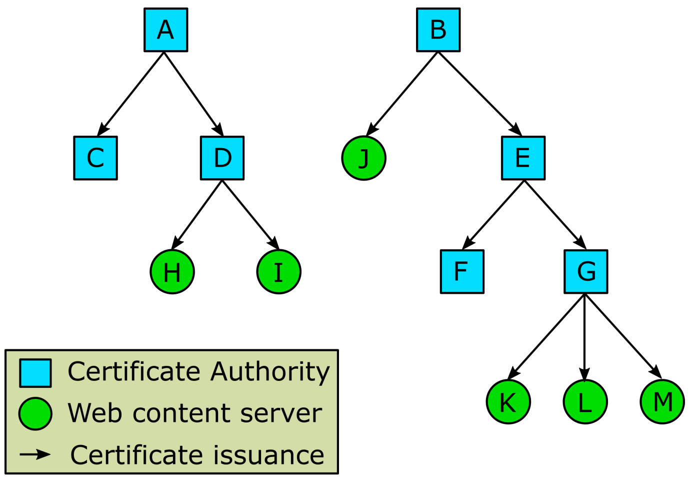

Figure 2.1: Hierarchical organisation of CAs

2.3 How It Works

Now that we have examined the web PKI's structure, we want to have a closer look at the interactions between its entities.

2.3.1 Certificate Issuance

Consider the server operator of a company Example Inc. that wants its server `example.org` to support HTTPS. At first, they have to get a certificate from a CA. Roughly speaking, they can choose between two types of certificates [7]: *extended validation (EV)* certificates and *domain validated (DV)* certificates. If the operator chooses an EV certificate, the CA will verify that Example Inc. is in fact a valid legal entity which owns or has control of `example.org`.[1] If the operator decides for a DV certificate, the CA will only verify that the operator has control over `example.org`. This allows DV certificates to be issued automatically (i.e., without a human element), and thus they are usually much cheaper than EV certificates or even free of charge [21].

There is no difference between the general structure of an EV or DV certificate. TLS/SSL certificates follow the X.509 standard [6]. Most importantly, they contain information about the subject (e.g., the owner's name and/or domain name), the subject's public key, the issuer's name, the issuer's digital signature and a validity period.

[1]For details, see the *Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates* [5] defined by the CA/Browser Forum consortium.

2.3.2 Certificate Validation

Let us now consider the web content server at `example.org` is represented by node H in Figure 2.1, that is, Example Inc. bought a certificate at the CA represented by node D. Suppose further we use a browser that only trusts the CAs A and B. This is what happens when we open `https://example.org` in a browser:[2]

1. H delivers its certificate along with those of D and A.
2. The browser validates the trust anchor (A):
    - 2.1. It verifies that A is contained in its list of trusted CAs.
    - 2.2. It checks the validity period of A's certificate.
    - 2.3. It verifies that the specified subject matches the issuer specified in D's certificate.
    - 2.4. Since A is a trust anchor, its certificate is self-signed. Therefore, the browser can validate the signature using A's public key.
3. The browser validates the certificate of D:
    - 3.1. It verifies that the certificate has not been revoked.
    - 3.2. It checks the validity period.
    - 3.3. It verifies that the specified subject matches the issuer specified in H's certificate.
    - 3.4. It validates the signature using A's public key.
4. The browser verifies the certificate of H:
    - 4.1. It verifies that the certificate has not been revoked.
    - 4.2. It checks the validity period.
    - 4.3. It checks if the specified domain name actually equals `example.org`.
    - 4.4. Finally, it validates the signature using D's public key.

If any of the above checks fail, the browser rejects the certification path and refuses to establish a TLS connection.

[2]For the sake of comprehensibility, the process is simplified and the order of steps may be different. See RFC 5280 [6] for the complete X.509 certification path validation algorithm.

2.3.3 Certificate Revocation

In some cases, a certificate needs to be revoked. For instance, revocation may be necessary if the certificate was issued by mistake or if the issuing CA was compromised.[3] Another typical scenario is the loss or compromise of a private key.

There are two basic mechanisms to handle certificate revocation. One approach is that each CA keeps a list of all certificates it revoked. This list is called *certificate revocation list (CRL)* and specified in RFC 5280 [6]. Web browsers periodically download CRLs from their trusted CAs and treat them as blacklists (i.e., reject all certificates listed there). The other mechanism is called *online certificate status protocol (OSCP)* and standardised in RFC 6960 [27]. Again, each CA keeps track of the revocation statuses of each of the certificates it issues. In contrast to CRLs, it additionally runs a server called *OCSP responder* which browsers can query on demand. That is, if a browser wants to know a certificate's revocation status, it sends a request to the corresponding CA's OCSP responder which in response sends the requested revocation status, along with the response's validity interval and the CA's signature.

2.4 Weaknesses

The ecosystem of CAs can be considered to be the web PKI's central component as it enables authentication of web servers. However, in its current form, it exposes serious vulnerabilities.

First, each CA is a single point of failure [4]. This is because in general, every CA can issue certificates for any domain. In the past, there have been severe incidents where CAs have been compromised or mistakenly issued test certificates for domains they did not own (see chapter 1).

Second, the risk that a trusted CA is compromised or goes rogue is very high. This is simply due to the fact that major browsers trust a large number of CAs per default. For instance, in 2010, Mozilla Firefox was distributed with a list of 124 trusted CAs, which corresponded to approximately 60 organisations [9]. In the same year, an analysis [9] revealed that a total of at least 1,482 CA certificates were trusted by Mozilla and Windows.

Third, the traditional certificate revocation mechanisms (see subsection 2.3.3) are essentially broken [22]. Regarding OCSP, the problem is that revocation status requests frequently time out. For the sake of optimal user experience, major browsers accept the certificates in such cases anyway, or simply do not support OCSP at all. There is a TLS extension called *OCSP stapling* [8][26] which addresses this issue. OCSP stapling shifts

[3]For real-life incidents, see chapter 1.

the burden of sending revocation status requests from the browser to the web server.[4] A web server that supports OCSP stapling makes OCSP requests at regular intervals. When it obtains the requested data, it attaches them to the corresponding certificates in its certification path. As a result, clients receive the OCSP data along with the certificates and thus do not need to send any revocation status requests themselves. To date, implementations of OCSP stapling are not mature yet [3], and therefore it is not surprising that major browsers do not enforce OCSP stapling so far [1]. As a consequence, the issue of OCSP request timeouts remains. The alternative mechanism, CRL, lacks scalability. In practice, sooner or later a CRL contains so many entries that it can no longer be downloaded or processed in a reasonable amount of time. As a compromise solution, Mozilla Firefox and Google Chrome blacklist only certificates they consider to be of particularly high importance [11][17]. Obviously, this solution is far from ideal.

Fourth, the process of certificate issuance is not transparent to the public. There are no public records about who issued which certificates, when and to whom. Hence, it often takes weeks or even months until rogue certificates are discovered and revoked [12]. Furthermore, the lack of transparency makes it difficult to empirically analyse the state of web PKI. It would be beneficial to have statistics about the certificates that are currently in use (e.g., information about the encryption algorithms used, the length of public keys, distribution of EV vs. DV certificates, geographic distribution) as well as the CAs' practices (e.g., the number of revocations per month).

[4]With that, OCSP stapling also addresses privacy concerns: A browser that uses OCSP discloses some of the user's browsing history to CAs.

3 Certificate Transparency

As we have seen in the previous section, the issuance and existence of certificates is non-transparent to the public, which leads to serious problems. The objective of CT is to change exactly that. More specifically, CT intends to "hold CAs publicly accountable for all certificates they issue [...] without introducing another trusted third party" [23].

This chapter is mainly based on `certificate-transparency.org` [13], a website hosted by Google, Inc. I outline the concept of CT (section 3.1) and how it integrates into the existing web PKI (section 3.2). In addition, I give a brief overview over the current state of the art (section 3.3).

3.1 The Framework

CT is designed as an extension to the existing web PKI [12]. That is, its components do not replace the current system but provide supplemental services. Furthermore, CT is an open standard which means anybody can operate its components and configure them to their specific needs [24]. To date there is only an experimental RFC [24] but an IETF working group [19] has been established to produce a standards-track version.

CT comprises three components: *logs*, *monitors* and *auditors*. Logs are append-only lists of certificates with a public interface for querying. When a CA issues a certificate, it is required to have it appended to at least one log. Monitors regularly query the logs to watch for suspicious certificates and to check if all logged certificates are visible. Last but not least, the task of auditors is to verify that logs are behaving correctly and if they include certain certificates.

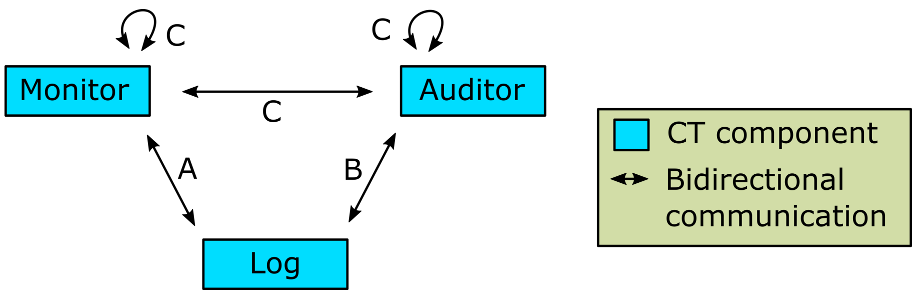

Figure 3.1: Interactions between CT components

In the following we have a closer look at each of the components and their interactions as shown in Figure 3.1.

3.1.1 Logs

Logs are the core of the CT framework. A log is a public network service that maintains an append-only record of certificates. It has an HTTPS interface which allows anybody to interact with it in the following ways:

- One can make queries to fetch certain certificates.
- One can have certificates appended to the log. After telling a log to append a given certificate, it must respond with a timestamp known as *signed certificate timestamp (SCT)*. The log thereby promises to append the given certificate by that time.
- One can request a *Merkle audit proof*. This is a cryptographic proof which assures that a given certificate has been correctly appended to the log. This feature is of key importance as the idea behind CT is that a certificate needs to be listed in at least one log to be considered trustworthy.
- One can request a *Merkle consistency proof*. This is a cryptographic proof which verifies the log's consistency. A log is consistent if for any two versions "the later version includes everything in the earlier version, in the same order, and all new entries come after the entries in the older version" [14]. If a log is consistent, this in particular guarantees that no certificate was removed or retrospectively inserted.

3.1.2 Monitors

The task of a monitor is to keep logs under surveillance. More specifically, it maintains copies of the logs it watches by periodically fetching all newly appended certificates. It uses these copies to...

- Watch for suspicious certificates such as certificates that were issued illegitimately or that have unusual characteristics (e.g., strange permissions or extensions), and to
- Check if all logged certificates actually appear in the logs (i.e., have been properly appended).

In Figure 3.1, these interactions are represented by (A).

3.1.3 Auditors

Finally, an auditor is a small program that verifies the consistency of logs (by requesting Merkle consistency proofs). Optionally it may also have the ability to check if a particular certificate is contained in a log (using Merkle audit proofs). Together, this corresponds to (B) in Figure 3.1.

3.1.4 Gossiping

Merkle consistency proofs make it possible to verify a log's consistency in the sense that it is consistent with its preceding versions. However, they cannot be used to check if different logs are consistent with each other. This is violated if one log is a fork or branch of another. To identify forks and branches among logs, it is necessary that monitors exchange information both with auditors as well as among themselves. An IETF working group elaborated how this communication could work and drafted a *gossiping* protocol [25]. In Figure 3.1, the gossiping is represented by the arrows labelled (C).

3.2 Integration Into the web PKI

As mentioned earlier, CT is an open framework and thus, various setups are possible. A common one is depicted in Figure 3.2.

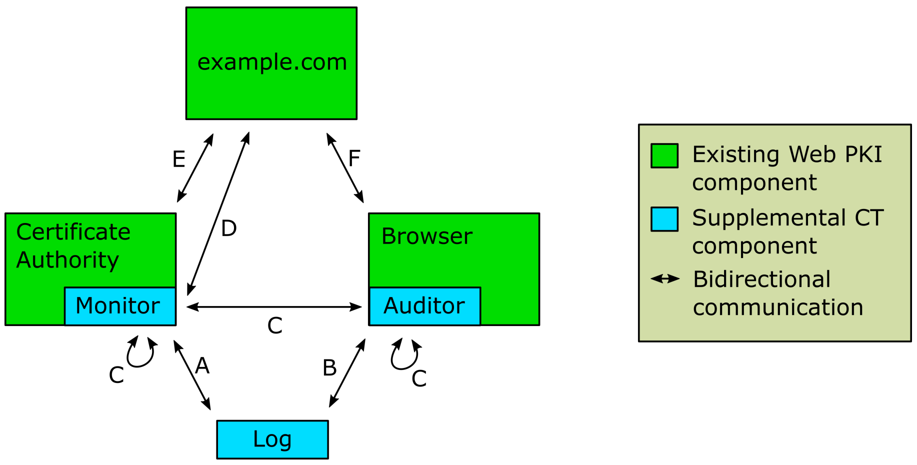
example.com
E
D
F
Certificate Authority
Monitor
Browser
Auditor
C
C
A
B
C
Log
Existing Web PKI component
Supplemental CT component
Bidirectional communication

Figure 3.2: A typical CT setup. Source: Adaption of a figure from `https://www.certificate-transparency.org/how-ct-works` [12].

In this setup, monitors are operated by CAs. This has several advantages. For one, a CA can configure its monitor such that it observes all certificates that belong to its customers, and advertise this additional service. For another, a CA can offer certificate monitoring as a service independent of certificate issuance to extend its business model.

Furthermore, it stands to reason that auditors are incorporated into browsers. This way auditors can check certificates on demand. That is, if a browser wants to establish a TLS connection with `example.org`, it can tell its integrated auditor to verify that the domain's certificate is contained in a properly functioning log.

It is important to note that any interested party can run a log server. Since logs are publicly verifiable, the operator of a log does not have to be trusted.

Regarding Figure 3.2, I have already covered the interactions (A), (B) and (C) in section 3.1. In the remainder of this section I will cover (D), (E) and (F). I do this by revisiting the certificate life cycle outlined in section 2.3, however this time assuming a CT configuration as in Figure 3.2.

3.2.1 Certificate Issuance

With CT, issuing a certificate to `example.com` breaks down as follows:

1. The operator of `example.com` requests a certificate from the CA (E).
2. The CA creates a preliminary certificate and appends it to at least one log (A).
3. The log responds with an SCT (A).
4. The CA creates a new certificate based on the preliminary one, attaches the SCT as an X.509v3 extension and delivers the finalised certificate to `example.com` (E).

Note that from the viewpoint of the server operator, the process of certificate issuance with CT is unchanged as compared to without CT, except that it might take longer until their certificate is ready to use.[1] Obviously, the CA is required to adapt their processes, though.

3.2.2 Certificate Validation

Next, consider a user opens `https://example.com` in their browser. Here is how the TLS handshake looks like with CT:

1. `example.org` sends its certificate (with an SCT attached) to the browser's TLS client (F).
2. If the certificate does not have a valid SCT attached (e.g., the timestamp is in the future, or the signature is invalid, or there is no SCT at all), the browser rejects the connection.

[1]RFC 6962 [24] also specifies two alternative mechanisms for certificate issuance. In contrast to the outlined one, these require server modifications.

3. Otherwise, it validates the certification path as usual.
4. The browser's auditor asynchronously requests a Merkle consistency proof and a Merkle audit proof from a log (B).[2]
5. If validation succeeds, the browser accepts the certificate and continues with the handshake, otherwise it refuses to establish a TLS connection.

Similar to the process of certificate issuance, certificate validation with CT does not require any modifications of the server at `example.com`. However, the browser obviously needs to be updated to incorporate an auditor module.

3.2.3 Certificate Monitoring

With certificate monitoring, CT adds a completely new use case to the existing web PKI. While CAs watch for suspicious certificates (A), e.g., certificates with strange permissions or extensions, server operators query monitors to check if anybody issued fake certificates for their domains (D).

Apart from CAs and web content server operators, certificate monitoring also affects browser vendors because monitors need to gossip with auditors (C). Hence, certificate monitoring requires modifications of all of the three web PKI entities.

3.2.4 Certificate Revocation

It is important to note that CT does not affect the process of certificate revocation at all. Logs are append-only and thus, a certificate cannot be removed even if it was revoked. With CT, certificate revocation is still based on mechanisms such as OCSP or CRLs.

3.3 State Of The Art

CT is still in an early stage. Nevertheless, CT infrastructure is already set up and ready to use. In 2013, DigiCert became the first CA to implement CT [30]. Up to now, 18 organisations announced a total of 64 logs [16]. What is more, Google's browser Chrome already enforces CT for all EV certificates and all newly issued DV certificates [29].

CT already has some success stories to tell. Most importantly, it has proven valuable to identify improper certificates from major CAs [28]. Moreover, a first analysis of existing logs [18] revealed that a considerable share of certificates uses weak keys or signatures. For instance, a significant number of certificates were signed using the deprecated SHA-1 algorithm.

[2] For details, see `https://www.certificate-transparency.org/log-proofs-work` [14].

4 Conclusions

In this article, I have given a high-level overview of the existing web PKI and what CT contributes to it. In summary, CT opens the CA ecosystem to public scrutiny and thus has two main benefits. First, it creates better oversight of the whole web PKI. Second, suspect certificates and CAs can be detected faster.

Concerning technical aspects, CT has some desirable qualities, too [15]. Some of them are:

- It can be deployed quite easily, that is to say with few changes to the existing web PKI infrastructure .
- It is flexible and customisable.
- In contrast to alternative technologies, it does not need side-channels.
- It does not require users to trust any third party other than CAs.
- It can be rolled out gradually. That is, browser vendors can implement stricter policies as time moves on. For example, in a first step a browser could display a warning when a certificate does not appear in a log. In a second step, it could reject EV certificates that are not logged. Ultimately, it could require every certificate to be logged.

On the other hand, as CT does not replace but rather extend the existing web PKI, inherent problems of the CA ecosystem remain. In particular, CT tackles only one of the web PKI's four weaknesses listed in section 2.4. It does not make certificate revocation mechanisms more reliable, it does not change the fact that each CA is a single point of failure, and neither does it tackle the problem that the risk of an attack is proportional to the number of CAs a user trusts.

As a conclusion, CT does not solve all of the web PKI's problems. Nevertheless, it is certainly a valuable extension and thus can be seen as an important step in fostering web security.

Bibliography

[1] R. Andrews and B. Morton. *OCSP Must-Staple*. CA Security Council. 2014. URL: https://casecurity.org/2014/06/18/ocsp-must-staple/ (visited on 2018-07-25).

[2] C. Arthur. *DigiNotar SSL Certificate Hack Amounts to Cyberwar, Says Expert*. 2011. URL: https://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar (visited on 2018-05-21).

[3] H. Böck. *The Problem With OCSP Stapling and Must Staple and Why Certificate Revocation Is Still Broken*. 2017. URL: https://blog.hboeck.de/archives/886-The-Problem-with-OCSP-Stapling-and-Must-Staple-and-why-Certificate-Revocation-is-still-broken.html (visited on 2018-07-25).

[4] J. Braun et al. "Trust Views for the Web PKI". In: *Public Key Infrastructures, Services and Applications*. Ed. by S. Katsikas and I. Agudo. Springer Berlin Heidelberg, 2014, pp. 134–151.

[5] CA/Browser Forum. *Baseline Requirements Documents*. URL: https://cabforum.org/baseline-requirements-documents/ (visited on 2018-06-30).

[6] D. Cooper et al. *Internet X.509 Public Key Infrastructure. Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. 2018.

[7] DigiCert Inc. *Domain Validated SSL vs. High Assurance*. URL: https://www.digicert.com/dv-ssl-certificate.htm (visited on 2018-05-21).

[8] D. Eastlake 3rd. *Transport Layer Security (TLS) Extensions: Extension Definitions*. RFC 6066. 2011.

[9] P. Eckersley and J. Burns. *An Observatory for the SSLiverse*. Presentation at DEFCON 18. Electronic Frontier Foundation, 2010. URL: https://www.eff.org/files/DefconSSLiverse.pdf (visited on 2018-05-13).

[10] Fox-IT BV. *Black Tulip. Report of the Investigation into the DigiNotar Certificate Authority Breach*. 2012. URL: https://github.com/juliocesarfort/public-pentesting-reports/blob/master/Fox-IT/Fox-IT_-_DigiNotar.pdf (visited on 2018-06-30).

[11] M. Goodwin. *Revoking Intermediate Certificates: Introducing OneCRL*. Mozilla Foundation. 2015. URL: https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/ (visited on 2018-07-01).

[12] Google Inc. *Certificate Transparency. How Certificate Transparency Works*. URL: https://www.certificate-transparency.org/how-ct-works (visited on 2018-05-13).

[13] Google Inc. *Certificate Transparency*. URL: https://www.certificate-transparency.org/ (visited on 2018-05-13).

[14] Google Inc. *Certificate Transparency. How Log Proofs Work*. URL: https://www.certificate-transparency.org/log-proofs-work (visited on 2018-05-13).

[15] Google Inc. *Certificate Transparency. Comparison with Other Technologies*. URL: https://www.certificate-transparency.org/comparison (visited on 2018-05-13).

[16] Google Inc. *List of All Known and Announced CT Logs*. URL: https://www.gstatic.com/ct/log_list/all_logs_list.json (visited on 2018-05-24).

[17] Google, Inc. *CRLSets*. URL: https://dev.chromium.org/Home/chromium-security/crlsets (visited on 2018-07-01).

[18] J. Gustafsson et al. "A First Look at the CT Landscape: Certificate Transparency Logs in Practice". In: *Passive and Active Measurement*. Ed. by M. A. Kaafar, S. Uhlig, and J. Amann. Springer International Publishing, 2017, pp. 87–99.

[19] IETF. *Charter for Working Group 'Public Notary Transparency'*. URL: https://datatracker.ietf.org/doc/charter-ietf-trans/ (visited on 2018-05-13).

[20] IETF. *Charter for Working Group 'Web PKI Operations'*. URL: https://datatracker.ietf.org/doc/charter-ietf-wpkops/ (visited on 2018-05-13).

[21] Internet Security Research Group. *Let's Encrypt*. URL: https://letsencrypt.org/ (visited on 2018-05-21).

[22] A. Langley. *No, Don't Enable Revocation Checking*. 2014. URL: https://www.imperialviolet.org/2014/04/19/revchecking.html (visited on 2018-07-01).

[23] A. Langley et al. *Certificate Transparency*. Presentation at NIST. Google Inc., 2013. URL: http://csrc.nist.gov/groups/ST/ca-workshop-2013/presentations/Kasper_ca-workshop2013.pdf (visited on 2018-05-24).

[24] B. Laurie, A. Langley, and E. Kasper. *Certificate Transparency*. RFC 6962. 2013.

[25] L. Nordberg, D. K. Gillmor, and T. Ritter. *Gossiping in CT*. Internet-Draft. Work in Progress. IETF, 2018. URL: https://datatracker.ietf.org/doc/html/draft-ietf-trans-gossip-05 (visited on 2018-05-13).

[26] Y. Pettersen. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*. RFC 6961. 2013.

[27] S. Santesson et al. *X.509 Internet Public Key Infrastructure. Online Certificate Status Protocol - OCSP*. RFC 6960. 2013.

[28] R. Sleevi. *Sustaining Digital Certificate Security*. Google Inc. URL: https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html (visited on 2018-05-13).

[29] U.S. General Services Administration. *Chrome Certificate Transparency Requirements*. 2018. URL: https://fpki.idmanagement.gov/announcements/chromect/ (visited on 2018-07-01).

[30] UBM Tech. *DigiCert Announces Certificate Transparency Support*. URL: https://web.archive.org/web/20131010015324/http://www.darkreading.com/privacy/digicert-announces-certificate-transpare/240161779 (visited on 2018-05-24).