H. Roßnagel et al. (Eds.): Open Identity Summit,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 143

Let's Revoke! Mitigating Revocation Equivocation by re-purposing the Certificate Transparency Log

Tobias Mueller[1] Marius Stübs[2] Hannes Federrath[3]

Abstract: Distributing cryptographic keys and asserting their validity is a challenge for any system relying on such keys, for example the World Wide Web with HTTPS or OpenPGP encrypted email. When keys get stolen or compromised, it is desirable to shorten the time during which an attacker can decrypt or sign messages. This is usually achieved by revoking the affected certificates.

We investigate the security requirements for distributing key revocations in the context of asynchronous decentralised messaging and analyse the status quo with respect to these requirements. We show that equivocation, integrity protection, and non-repudiation pose a challenge in today's revocation distribution infrastructure. We find that a publicly verifiable append-only data structure serves our purpose and notice that operating such an infrastructure is expensive.

We propose a revocation distribution scheme that fulfils our requirements. Our scheme uses the already existing Certificate Transparency (CT) logs of the WebPKI as a publicly verifiable append-only data structure for storing revocations through specially crafted TLS certificates. The security of our system largely stems from the properties of these CT logs. Additionally, we analyse the computational and bandwidth requirements of our scheme and show limitations of the protocol we propose.

Keywords: key revocation; asynchronous decentralised messaging; email; PKI; trust; OpenPGP

1 Introduction

Email is one form of asynchronous messaging and several approaches to protecting the messages exist. Among them are S/MIME [TR10], OpenPGP [DCS07], and MLS [Ba18]. In all of these cases, clients wishing to communicate with one another, need to obtain the public key of the recipient. Eventually, that key can be compromised and marked as revoked. The clients encrypting a message or verifying a signature then need to check whether a given key has been marked as revoked. In case of a centralised PKI, the party revoking their key can ask the issuer to publish the certificate as being revoked. In a decentralised messaging architecture, however, no such central party exists.

Even if such a central party existed, it remains a challenge to hold it accountable for the answers it provides. That includes the scenario of the server responding with either old or

[1] Universität Hamburg, SVS, Vogt-Kölln-Str 30, 22527 Hamburg, mueller@informatik.uni-hamburg.de
[2] Universität Hamburg, SVS, Vogt-Kölln-Str 30, 22527 Hamburg, stuebs@informatik.uni-hamburg.de
[3] Universität Hamburg, SVS, Vogt-Kölln-Str 30, 22527 Hamburg, federrath@informatik.uni-hamburg.de

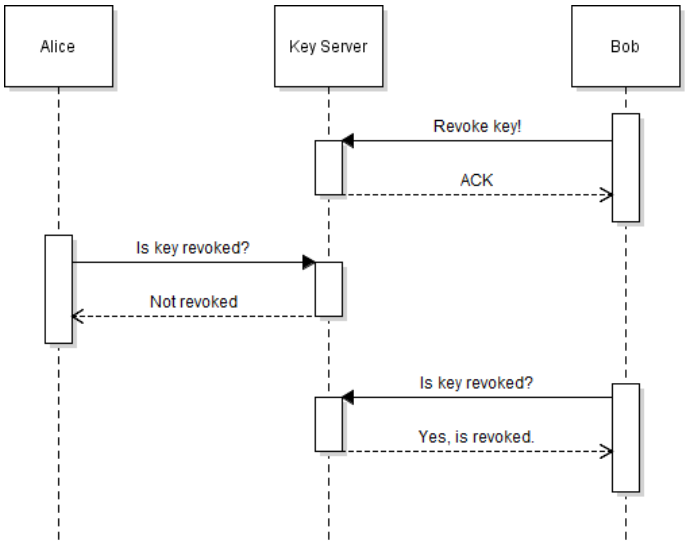

Fig. 1: Attack by a rogue key server delivering one version of the revocation information to one particular client while serving other information to other clients.

wrong information. The server could simply lie about the revocation status and make the requesting party believe that a given key has not yet been revoked.

Consider the following example depicted in Fig. 1: Alice wants to send a message to Bob. She has already obtained the relevant cryptographic key. Bob realises that his key has been compromised and uploads a revocation. Alice then asks the key server for the revocation status and gets a wrong answer. Either the key server itself or a network-based attacker strips the revocation information which in turn makes Alice believe that the key has not yet been revoked.

To make matters worse, Alice has no proof that the key server has given her inaccurate information and even if she finds out later that she had been sent wrong information, she could not prove it to the world.

2 Background

In this section we present mechanisms used for revocations in decentralised asynchronous messaging and compare them to what is used in the WebPKI. Finally, we briefly introduce Certificate Transparency as we will use this mechanism for storing revocation information.

```
0000  88 78                                                 frame
0002        04                                              version
0003           20                                           sigtype
0004              16                                        pk_algo
0005                 08                                     hash_algo
0006                    00 20                               hashed_area_len
0008                           16 21 04 cd 6f 2d 93 e8      hashed_area
0010  87 0d 61 17 6a d5 c7 89  b4 a0 07 47 a2 88 70 05
0020  02 5b dd 82 a9 02 1d 00
0028                           00 0a                        unhashed_area_len
002a                                 09 10 89 b4 a0 07      unhashed_area
0030  47 a2 88 70
0034              06                                        hash_prefix1
0035                 f9                                     hash_prefix2
0036                    01 00                               mpi_len
0038                           97 61 89 af 42 3d f5 e2      eddsa_signature_r
0040  eb 95 9a 50 d2 c4 20 ce  fc 2a f7 f9 1b 72 27 33
0050  0e 5a 4b 7a 2f 27 73 2a
0058                           01 00                        mpi_len
005a                                 9d 3b 4a 71 c2 2f      eddsa_signature_s
0060  1e 2b 65 f4 24 20 11 3d  45 29 ee 16 fc 61 ef 3f
0070  fd 98 16 f7 98 e1 33 48  1e 07
```

Fig. 2: A 120 bytes long OpenPGP revocation signature packet of a ed25519 key

OpenPGP Key Revocation Packet OpenPGP is a message syntax for asynchronous messaging and tries to avoid centralised infrastructure. It describes how to serialise cryptographic keys and messages. OpenPGP defines revocations as a special type of self-signature [DCS07, §5.2.1]. Once a key has such a revocation signature, it cannot be healed and is considered to be invalid. The format is packet based and the packets can be appended in any order. In particular, an OpenPGP revocation signature is composed of the following fields (cf. Fig. 2): version, signature type, public-key algorithm, and hash algorithm, each of which take up one byte. In addition, the OpenPGP specification allows for additional data of variable length to be signed. Finally, the actual signature is calculated over all these fields and is serialised in a variable length field. In the case of RSA, the signature is about the size of the modulus, i.e. the bit length of the key. Other signature schemes, such as EdDSA, produce considerably shorter signatures.

In case a key is known to be compromised, such a revocation signature packet is appended to the other packets making up an OpenPGP key. Note that the list of packets is not authenticated which in turn allows attackers to alter it, e.g. remove packets from the key.

OpenPGP Keyserver An established way of checking for revocations is contacting a so-called key server. Such a key server commonly serves requests via a HTTP-based

protocol [Sh03]. In order to reduce the amount of trust placed in any single key server, many operators run an instance of the dominating solution for distributing OpenPGP keys over the Internet: SKS Synchronising Keyserver [Mi02]. Those servers generally gossip the keys among each other so that the servers can provide data which was not uploaded directly to them, but rather to a peer they gossip with [MTZ03]. No mechanism for ensuring integrity of the keys during transit exists. That is, a malicious key server can gossip modified keys, e.g. with a truncated revocation signature.

We identify three cases for which a key server is used today:

1. initial key discovery,
2. retrieving key updates,
3. and checking for revocations.

The first case refers to the problem of finding a certificate for a given email address, the second refers to extending already known certificates with new packets, and the last refers to the validity of a key in case of a compromise. While the revocation case can be considered a specialisation of the update case, we argue that the semantics differ enough to view them as separate operations. We base our observation on the fact that certain updates to keys can be transient, such as temporarily adding new user IDs or sub-keys, while a revocation cannot be healed. In other words, certain updates can overrule others, while a revocation, once set, cannot be undone.

In this paper we concentrate on the revocation use case. We exploit the fact that revocations cannot be healed by using a publicly verifiable append-only data structure.

Certificate Revocation List In the Web context, revocations can be distributed as part of Certificate Revocation Lists (CRLs). A CRL is a complete list of revoked certificates signed by the issuing CA. Clients wishing to learn about revoked certificates download the CRL and check whether the certificate is contained in the list. This list can grow too big for clients to handle efficiently.

The Online Certificate Status Protocol (OCSP) addresses this shortcoming by providing a live response to a request for a certificates validity. With that schema, clients contact the CA of the certificate and ask about the status of the certificate at hand. That approach requires an additional connection to the CA, or rather the designated OCSP server, which is considered to be too expensive. It also leads to the situation in which the client cannot establish a separate connection, e.g. because the attacker blocks connection attempts.

An extension is to make the server the client is contacting prove that its certificate is still valid. To that end, the server itself contacts the OCSP server and obtains a proof which it

hands back to the client. This again, is considered to be too fragile, because those additional connections add to the latency and can fail.

Certificate Transparency Log Certificate Transparency (CT) attempts to solve a slightly different problem than publishing and distributing revocations, namely reducing the time it takes to detect falsely issued certificates [LKL13]. It provides infrastructure and a protocol for a publicly verifiable append-only data structure in which issued certificates ought to be stored. The nature of the data structure makes the issuance of a certificate transparent to the public. With CT a CA submits the certificate to be generated to a CT log server which in turn includes the certificate in the log and produces a signature as a proof and promise of inclusion. While the main idea of CT is that site operators can check who issued certificates for their DNS names, it can also be used by clients to convince themselves of seeing the same certificates as everybody else. To that end, the server hands the certificate with its proof of inclusion to the client, which in turn can ask the log server for the presence of said certificate. This scheme makes it expensive for the server and the log operator to equivocate and to deny presence of a certificate in the log, because it would need to maintain the separate Merkle trees and prevent clients from exchanging the Merkle tree heads they are seeing.

We will exploit these properties for storing revocation signatures of OpenPGP certificates.

3 Requirements for Revocation Distribution Schemes

We identify the following four main requirements for a revocation distribution and querying scheme.

1. **Integrity-Preserving:** The distributor of revocations must not be able to modify the packets.
2. **Equivocation-Resistance:** The distributor must not be able to give two requesting parties other versions of the same information, i.e. the revocation.
3. **Non-Repudiation:** The information a client retrieves needs to be authenticated such that misbehaviour can be proven to a third party.
4. **Privacy-Preserving:** The distributor must not learn who the client wishes to communicate with, i.e. for which entity the revocation information is being requested.

The current scheme of key servers fails to fulfil these requirements, because an attacker can manipulate the information in transit and thus, e.g. invalidate the revocation signature (**Integrity**). The attacker can also serve two parties separate versions of the key, e.g. discriminate the receiver of the information and hand out a stale key rather than the

most recent one (**Equivocation**). The client has no way of detecting whether it has been discriminated by the server, i.e. that the server has provided information dedicated to the requesting client and that is not made available for other parties. Once a client has received information from a server, the server can deny having sent it (**Non-Repudiation**). In the current scheme, the client asks the key server about a specific key. By means of that request, the client needs to inform the server about the party they want to communicate with (**Privacy**).

4 Equivocation-Resistant Key Revocation Protocol

In this section we first describe an intuitive approach for distributing certificate revocations which has led to what is being used for the WebPKI today. We then describe our proposed protocol of using the existing CT log for storing the revocation information of OpenPGP keys.

Intuitively, a relatively simple list of revoked certificates fulfils the requirements. That list needs to be signed, fetched, and distributed by a trusted party. In fact, Google and Mozilla use this scheme to fetch revocations from CAs and distribute to their customers as part of the browser's update mechanism (OneCRL, CRLSet). Note that if the user was made to contact the CRL server of a CA directly, that server could easily equivocate in a non-repudiable manner. For the Web, the users are arguably placing trust in the vendor to produce and distribute secure software. It seems reasonable to further assume that trusted party does not violate any of the requirements mentioned before. For a decentralised use case like messaging, such a centralised vendor does not exist let alone a central instance being able to invalidate a certificate and distribute such a list.

A publicly verifiable append-only data structure can be used to store revocation information. However, such a data structure tends to be difficult and expensive to maintain, largely because of the cost of the required infrastructure. However, if such an infrastructure already existed it seems worthwhile to investigate how to use it for our purpose. Fortunately, the CT log possesses the desired properties and is already being operated and maintained for the WebPKI. As of the time of writing Google's Chrome browser requires certain certificates to be present in the CT log before establishing TLS connections.

If we place certificates with specially formed names in the CT log then the mere presence of such a TLS certificate signals the revocation of an OpenPGP key. Without loss of generality, we introduce a new centralised, but untrusted entity: The Revocation Service. Its only job is to generate certificates with a well known name which will then be stored in the CT log in order to enable clients to find both the certificate and proof.

Storing the OpenPGP revocation signatures The Revocation Service's purpose is to accept the revocation signatures for a key, e.g. via e-mail or a Web interface, and then to

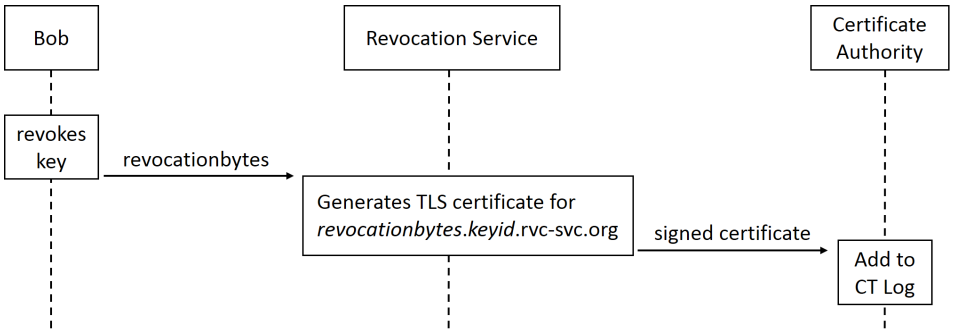

Fig. 3: Revocation Publishing Protocol

generate a specially formed TLS certificate with a well-known suffix, i.e. domain name. The name for which the certificate is valid includes the actual bytes of the revocation signature. *revocationbytes*.revocation-service.org. This certificate is then placed in the CT log, such that the public can detect its presence. These three steps, sending the revocation bytes, generating the specially crafted TLS certificate, and placing it in the existing CT logs, form the publishing protocol shown in Fig. 3.

Note that we do not define how exactly the Revocation Service gets hold of the revocation bytes. One of many ways is to send an e-mail or upload via a Web interface. Notice that the name for which the certificate is valid includes the key id. This is an optimisation for speeding up clients searching for revocations and is not necessary to fulfil our security requirements. While we do not specify how the certificate should be generated, we envision the use of the ACME [Y116] protocol to automatically generate the certificates. Once the certificate has been generated, it is placed in the CT log. Note that this is done by the CA signing the certificate. We also note that the number of such revocation services is not limited to one. In fact, the sole reason for a centralised service is to provide a well-known suffix which makes querying for the information much more feasible. It is conceivable that clients wishing to ask for revocations have multiple well-known suffixes to search for and that clients revoking their certificate contact multiple services. Also, because of the querying protocol shown below, clients do not need to trust the contents of the *revocationbytes*, so the server does not need to defend against wrong or fraudulent submissions.

Querying the revocation service When a messaging client wishes to learn whether a given certificate has been revoked it investigates the CT logs and checks for the presence of certificates including bytes of a revocation signature for the certificate. We assume that the client verifies the CT log for authenticity and integrity as per the regular CT protocol.

Fig. 4 shows the querying protocol. Note that we do not specify how exactly the client obtains the CT log in order to check for the presence of a certificate with a certain host

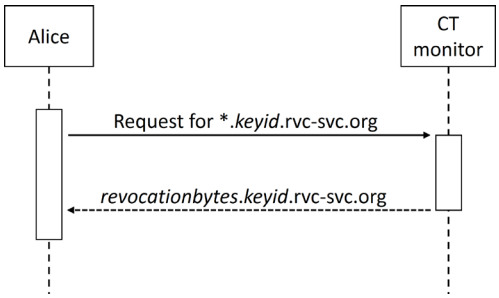

Fig. 4: Revocation Querying Protocol

name. We refer to the mechanisms CT provides for what is called a "monitor" [LKL13, §5.3]. One trivial option is to use the HTTP interfaces of the CT log servers to obtain all Merkle trees and all X509 certificates contained therein. The client can then iterate over all X509 certificates for host names with the well known suffix `keyid.rvc-svc.org`.

Verifying the Revocation Signature If the presence of such a certificate is detected in the log, the client trivially extracts the *revocationbytes* from the host name found in the X.509 certificate and appends them to the OpenPGP certificate the client already knows. If the *revocationbytes* make a valid OpenPGP packet with a valid signature under the public key of the certificate it has been appended to, the client will mark the certificate as revoked.

5 Discussion

In this section we discuss the properties of our presented protocol for publishing and querying revocations.

Privacy The proposed protocol is private under the assumption that the client either crawls the CT logs itself or queries a trusted monitor service (as per the CT standard [LKL13]) for the presence of certificates in the log. If the monitor is not trusted, this query may present a privacy risk, because it lets the monitor know who the client wishes to communicate with. We note that Web clients, e.g. browsers, are exposed to a similar but slightly different problem. In the Web context, the client obtains the certificate to check whereas in our context we do not have such a certificate up front.

Integrity The *revocationbytes* are transported as part of a X509 certificate in a CT log which in turn is integrity protected by the Merkle tree. A modification of the X509 certificate would invalidate the Merkle tree and thus be detectable by a client. Additionally, the

presented scheme is secure against the centralised Revocation Service becoming malicious either intentionally or by being coerced into misbehaving. Because the client does not rely on the mere existence of a TLS certificate in the CT log but rather verifies that the revocation bytes do indeed verify under the public key it ought to revoke. The assumption is that the client checking for the revocation already possesses the public key it wants to check and that the service cannot produce a valid signature, e.g. it does not have access to an oracle or the private key.

Equivocation-Resistance Because the Revocation Service itself does not respond to queries about revoked keys but merely creates the certificates with a well known name, it cannot equivocate in the first place. The scheme is thus as equivocation-resistant as the CT log. That is, clients obtain signed responses in form of Signed Tree Heads (STH) from the CT logs in order to check for integrity [LKL13]. The clients can then exchange those STHs with their peers and compare whether they have received differing information. Additionally, the server would need to maintain separate branches of the STHs of the Merkle tree and remember which branch it provides to which client. So while the presented scheme does not prevent equivocation it makes it expensive and detectable. Even if we assume an attacker being capable of equivocating, it needs to prevent clients from exchanging the STHs they received from the CT logs because these allow for uncovering equivocation hence leading to a loss of reputation of the CT log.

Non-Repudiation Our proposed scheme achieves non-repudiation, because clients receive signed responses as mentioned above. Clients can propagate the information they retrieve and convince others of the authorship of the information. That is, the STHs will have a valid signature which allows for attributing a potentially malicious Merkle tree.

Runtime In addition to the security requirement we discuss the computational and bandwidth effort, a client has to make. Firstly, we note that a simpler protocol would have the same security guarantees but worse computational characteristics. A simpler protocol like this would not make the key id part of the host name and merely encode the revocation bytes. The client would then have to verify all revocation bytes it sees in the CT log against all the public keys it knows about. We include a hint, the key id, in the host name so that the client can discard non relevant host names. A host name is deemed relevant if it matches the key id of the certificate of which the client is querying the revocation status.

Secondly, it is possible to optimise the scheme further by placing a certificate for multiple host names in the CT log. In particular, a certificate which has *keyid*.rvc-svc.org as well as *revocationbytes*.*keyid*.rvc-svc.org as Subject Alternative Names (SANs) might be more easily located by a client.

Thirdly, we note that a simple implementation of our protocol incurs a download of the whole CT log along with the corresponding certificates. Because we have not specified how exactly to query for host names in the CT log, clients use the monitor infrastructure envisioned by Certificate Transparency to check for host names with the desired known suffix. At the time of writing, several such services exist. Our proposed protocol for querying allows for anonymous queries so those services can be used through anonymisation networks.

Lastly, the Revocation Service could help the client locate the actual certificate by offering resolvable names with a TLS server. The client would attempt to connect to the host name on a well know port, e.g. 443, and receive the actual X509 certificate, e.g as part of a TLS handshake. While this speeds the clients up and makes using a monitor service more private, it makes operating the Revocation Service more expensive due to the requirement of an online presence. So far, the Revocation Service merely provides the well known suffix for placing certificates in the CT log. While this typically requires an online presence, it is only needed for a short amount of time.

6 Challenges

This section describes problems with the approach of using the Certificate Transparency system for our purpose of storing OpenPGP revocation information.

DNS label length TLS certificates are currently using X.509v3 syntax for the certificates. While we are theoretically free to use any field in that notation, we need to have our certificate signed by a CA. Those CAs tend to be overly cautious about signing X.509 structures. In fact, they usually generate those themselves. The only fields we can certainly influence are the public key and the host name. We investigate which of those fields are fit for our purpose.

In DNS, every host name can only be 253 octets long and every part, that is the name between the dots, can be up to 63 octets long [Mo87],[Br89],[BE97]. RSA keys are still very common in the OpenPGP ecosystem and these keys tend to produce relatively long signatures. Assuming no other overhead in the host name, the actual revocation, and in the padding of the resulting signature, we can encode a signature for a key of up to 253 octets or 2024 bits. The recommendation for the length of RSA keys generated today is 3000 bits or longer [Bu18].

CT logs may cease to exist In fact, Cloudflare and Google have set up CT logs which only accept certificates expiring in a particular year. The Baseline Requirements [CA18] demand PKIX CAs to only issue certificates with an expiry date not longer than two years in the future. After the expiry date, the certificate is invalid, regardless of whether it had been included in the CT logs. The idea, thus, is to not maintain one CT log indefinitely, but only for as long as all certificates included in the log have expired. This presents a challenge for

our use-case, because due to the packet-based structure of OpenPGP certificates it remains unknown whether a key has expired as a packet which extends the lifetime of the key might exist but has not yet been disseminated.

Operation of the Revocation Service While the presented scheme reuses existing CT log infrastructure, it still requires an actual service to be run and maintained. In particular, submission of revoked OpenPGP public keys as well as the generation of TLS certificates need to be provided. We argue that the cost of running such a service is comparatively low, but we appreciate that the cost is not zero. We also note that the amount of trust placed in the newly introduced Revocation Service is lower than the existing key server infrastructure. Instead of having to trust the service for not modifying the data in transit, we need to trust it to actually generate the TLS certificates rather than denying to do it. We envision that a promise of service can be given, similar to what CT does for the SCTs. However, the proposed protocol is kept simple to ease its adoption.

7 Related Work

A large body of work in the area of distributed consensus, PKI, and secure messaging exist. For brevity reasons, we only discuss the work that, according to our knowledge, is closest to what we presented in this paper, namely the concept of Revocation Transparency.

Revocation Transparency [LK12] is a proposed concept to address verifiable revocations. However, it assumes that revocations can be deleted. In the WebPKI this is true, because the Baseline Requirements demand certain lifetimes of keys. For decentralised asynchronous messaging, however, no such list of requirements has been established yet. It is conceivable that this approach can be adapted by removing the ability to delete revocations, though. In fact, such a system would indeed fulfil the requirement of a publicly verifiable append-only data structure in which revocations can only be added and never removed while at the same time making equivocation expensive. The biggest obstacle of using Revocation Transparency is the lack of operators. Certificate Transparency enjoys the backing of major Internet companies which have an interest in unveiling misbehaviour of CAs. Decentralised asynchronous messaging does not enjoy the support of deep pocketed stake holders and infrastructure is thus more scarce.

8 Conclusion

We identified distributing certificate revocation information as a challenge in systems depending on public keys. We also identified requirements for the secure distribution of such revocations in a decentralised asynchronous messaging context: Integrity, Equivocation, Non-Repudiation, Privacy. We further proposed a protocol for publishing and querying

revocation information for OpenPGP certificates based on a publicly verifiable append-only data structure. Such a data structure is usually difficult and expensive to operate. Our research has shown that it is possible to overcome this problem by reusing existing infrastructure in form of Certificate Transparency log.

In the proposed scheme, OpenPGP revocation signatures are translated into host names which in turn are encoded in X.509 certificates. This allows for storing them in the already existing and successfully operated Certificate Transparency log. We derive our security guarantees to a large degree from the append-only nature of the Certificate Transparency logs. This includes the resistance against equivocation which cannot be defended against in the current OpenPGP ecosystem.

Bibliography

[Ba18] Barnes, Richard; Millican, Jon; Omara, Emad; Cohn-Gordon, Katriel; Robert, Raphael: The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-02, Internet Engineering Task Force, October 2018. Work in Progress.

[BE97] Bush, Randy; Elz, Robert: Clarifications to the DNS Specification. July 1997.

[Br89] Braden, R.: , Requirements for Internet Hosts - Application and Support, October 1989.

[Bu18] Bundesamt für Sicherheit in der Informationstechnik: BSI TR-02102-1: Kryptographische Verfahren: Empfehlungen Und Schlüssellängen. May 2018.

[CA18] CA/Browser Forum: Baseline Requirements for the Issuance and Management of Publicly - Trusted Certificates. October 2018.

[DCS07] Donnerhacke, Lutz; Callas, Jon; Shaw, David: , OpenPGP Message Format. RFC 4880, November 2007.

[LK12] Laurie, Ben; Kasper, Emilia: Revocation transparency. Google Research, September, 2012.

[LKL13] Langley, Adam; Kasper, Emilia; Laurie, Ben: RFC 6962: Certificate Transparency. June 2013.

[Mi02] Minsky, Yaron: SKS Synchronising Key Server. Bitbucket repository `https://bitbucket.org/skskeyserver/sks-keyserver/`, 2002.

[Mo87] Mockapetris, P. V.: Domain Names - Implementation and Specification. November 1987.

[MTZ03] Minsky, Yaron; Trachtenberg, Ari; Zippel, Richard: Set Reconciliation with Nearly Optimal Communication Complexity. IEEE Transactions on Information Theory, 49(9):2213–2218, September 2003.

[Sh03] Shaw, David: The OpenPGP HTTP Keyserver Protocol (HKP). Internet-Draft draft-shaw-openpgp-hkp-00, Internet Engineering Task Force, March 2003. Work in Progress.

[TR10] Turner, Sean; Ramsdell, Blake C.: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. (5751), January 2010.

[Yl16] Ylonen, Tatu J: Automated access, key, certificate, and credential management. December 6 2016. US Patent 9,515,999.