TPValCert: Privacy-Preserving Trusted Proxy for Public Key Certificate Validation

TPValCert: Privacy-Preserving Trusted Proxy for Public Key Certificate Validation

Diana Gratiela Berbecaru
Department of Control and Computer Engineering
Politecnico di Torino, Italy
Corso Duca degli Abruzzi 24, 10129, Torino (ITALY)
diana.berbecaru@polito.it

***Abstract*—Public key X.509 certificates play a powerful role in promoting effective electronic identification, but some significant practical issues still affect their scalability. Every time a public key certificate is used, it must be validated by the system or application relying on it for security services, generically called also relying party. The validation involves several processing steps and checks, and it has been measured that many applications (still) perform it incompletely. Furthermore, privacy issues may occur when validating certificates, for example a website visited by a user could be revealed to external parties. We propose TPValCert, an architecture tailoring a trusted proxy to provide privacy-preserving certificate validation service to the relying parties. By exploiting TPValCert, a desktop, IoT, or mobile system that needs to validate a public-key certificate may execute a transaction with a trusted proxy, which performs validation by considering certificate policy parameters, privacy, and validation options received from the client and returns the validation status. Besides reducing complexity on the client, exploiting such trusted validation parties may also bring privacy benefits. To communicate with the clients, we consider the SCVP (Server-based Certificate Validation Protocol) or DVCS (Data Validation and Certification Server) protocols, even though, depending on the context, lighter formats could be considered. Our implementation efforts emphasize the possibility of pursuing a tradeoff between timeliness, privacy, and computational resource usage, via dynamic selection of several configurable options.**

***Index Terms*—PKI, trusted proxy, privacy, delegated certificate validation, certificate policy parameters.**

I. Introduction

The public key X.509v3 certificates, called also digital certificates, are widely employed nowadays in many security systems and protocols and are crucial for digital identities, secure data transfer, digital signatures, or trusted computing. One particularly open aspect regarding digital certificates is their validation. For instance, it has been measured that digital certificates are still incompletely validated in commonly used applications, like Internet browsers [1]. In particular, the revocation checking is often skipped, but new challenges have occured in the meantime. For instance, Google's Moving Forward, Together roadmap [2] indicates that the validity period for TLS certificates will likely soon drop from 398 days to a mere 90 days, according to the CA/Browser (CAB) Forum recommendations [3]. This is challenging for the Certificate Authorities (CAs) because they will have to issue certificates more frequently but also for the relying party applications because they will have to (be able to) validate appropriately an increasing number of certificates. Moreover, the Certificate Transparency (CT) system introduced by Google to counter mis-issued certificates for domain owners places an additional burden on the relying parties. As stated in [4], "*Web browsers enforcing CT should check that each certificate has been logged before accepting it as valid*". Furthermore, Google made CT mandatory in Chrome for all newly issued certificates [5] [6]. Thus, to avoid using mis-issued certificates that could open the door to dangerous attacks in web-based scenarios, the relying parties must check, possibly in a privacy-preserving manner, whether the certificate they use is present in one or more CT logs.

None of the mechanisms proposed so far for certificate validation could alone meet the computational (intended in terms of network and cryptographic support) and privacy requirements of all relying party applications and PKI (Public Key Infrastructure) topologies. For example, to check the revocation status of a certificate two main approaches exist: Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP) [7]. CRLs, issued by the CAs, contain all revoked certificates, while the OCSP can be used to ask for the revocation status of a single or few certificates from an OCSP server. To allow seamless revocation checking, the CAs integrate support for either or both methods in forms of extension placed inside the issued certificates. Nevertheless, neither the CRLs nor the OCSP protocol are ideal, they both present performance, scalability and privacy issues [8], [9]. The CT system also introduces some potential privacy issues about users' browsing history [1], when checking the SCTs (Signed Certificate Timestamps). An SCT is similar to an identifier placed inside each certificate used as index in a CT log. Google made its first foray at the beginning of 2021 into auditing the SCTs by "*letting Chrome browser asynchronously share a sample of SCTs with Google if the user has opted in to sharing browser history with Google for security purposes*" [4]. Later in 2021, it has been announced that Chrome would allow for this checking in a more privacy-preserving way. More precisely, Chrome makes, on a small sample of SCTs,

This work was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

a k-anonymous query [10] to Google to see if Google knows about the existance of an SCT. It is better, because (with this approach) Google learns the user was visiting one of the k sites, but not which one.

Certificate validation requires both the construction and the validation of certification paths, called also certificate path processing. A certification path is said to be valid under a set of certificate policies that are common to all certificates in the path [11]. Certificate path processing must be implemented in every PKI-based application or system exploiting X.509v3 certificates. The IETF PKIX working group defined the requirements and has drafted a number of validation protocols including SCVP [12] and DVCS [13] allowing clients to delegate the certificate path processing to a server so that to minimize burden on the relying party. In these schemes, certificate path processing is partially or totally delegated from the RP to a dedicated server. The protocols undertake certification paths processing against a number of validation constraints, named *validation policies*. To perform its task, the server retrieves data from a number of resources, e.g., directories, local repositories, and communicates with other external servers.

This paper describes a trusted proxy and an architecture, called TPValCert, for privacy-preserving validation of X.509v3 certificates. The architecture is based on the client-server model and is designed at a suitable level of abstraction. General functionality is defined for the component modules so that to allow the developers to choose different protocols and validation mechanisms depending on the context. Moreover, in TPValCert, the clients can dynamically select a series of certificate validation, policy, and privacy options to foster optimization in various environments. In other words, the proposed trusted proxy allows clients to specify different validation profiles according to their timeliness, security, privacy, and power/network resources requirements. The client may indicate to the trusted proxy either the exact input certificate policy data to be used during path validation or it can specify a high-level application context.

The paper is organized as follows: Section II presents the TPValCert architecture, Section III describes our TPValCert implementation exploiting the SCVP or DVCS protocols, Section IV concludes the paper an indicates future work.

II. TPValCert Design

This section describes the architecture of the trusted proxy for the validation of public key certificates at a suitable level of abstraction, by extending a previous work [14]. TPValCert may be exploited by the following RP categories: thin or light users, applications running on devices in enterprise environments, naive users, or users with tight privacy requirements.

Certificate validation steps. We recall shortly the main steps to be performed in order to validate a *leaf* certificate:

1. construction of a certificate path from the *leaf* up to a trusted root certificate. By definition, a certificate path is an ordered list of certificates starting with the certificate to be validated, called also end-user or leaf certificate [15] followed by one or more intermediate certificates, and ending in a certificate considered trusted by the relying party (called also trusted anchor or root CA certificate). Depending on the direction in which the certificate path is built we find the forward and reverse order. In general, certificate path construction may require a path discovery action resulting in several certification paths found by the trusted proxy for a certain leaf.

2. execution of a path validation algorithm: specific verification steps following the RFC5280 specification [16] and CAB Forum requirements (applying to server certificates for Internet applications) must be performed for each certificate in the path and also on the chain itself for the path constraints, e.g., naming or policy restrictions. To name some of them, the algorithm verifies the digital signature on each certificate in the chain, it checks that the required certificate policies are present in the certificates, checks whether the names and identifier(s) in the certificates are consistent with a valid certification path, that is, the subject of every certificate in the path is the issuer of the next certificate (except for the root CA).

In TPValCert, the client delegates subtasks (e.g., only path discovery) or the entire certificate path processing task (e.g., path discovery and path validation) to a server typically by using a validation protocol running over the link L1 depicted in Fig. 1. The certification path discovery and validation tasks should be considered as two separate tasks. Their implementations may be distinct because the certification path discovery is not a security-critical function whereas the path validation it is considered so. Besides the leaf certificate, the request sent over the link L1 in Fig. 1 can contain data such as certificate revocation data or policy information. The RP relies on the trusted proxy to perform a service constrained by the so-called *validation policy*, under which the validation is requested and considered acceptable for a PKI-enabled application. The trusted proxy constructs a response containing either the verification results for the leaf certificate or an error message.

TPValCert components. The trusted proxy contains basic modules for crytographic support, certificate management, and secure storage of certificate or revocation data, and a `Validation Module`, which has been further split in several components. The `Validation Protocol` module manages the messages exchanged on link L1, and contains functions to create, print, sign, and parse requests and responses. The `Path Validation` module executes the path validation algorithm to obtain the validation status for a leaf certificate. The path validation algorithms are included in this abstract module. The `Path Building` module constructs the certification paths. Furthermore, this abstract level may contain also (efficient) algorithms for path development, which use certificate extensions and loop detection/elimination techniques to increase efficiency [17] [18]. The `Certificate Revocation Status` module determines the revocation status of a certificate by retrieving and processing CRLs, as well as by communicating with the OCSP responders.

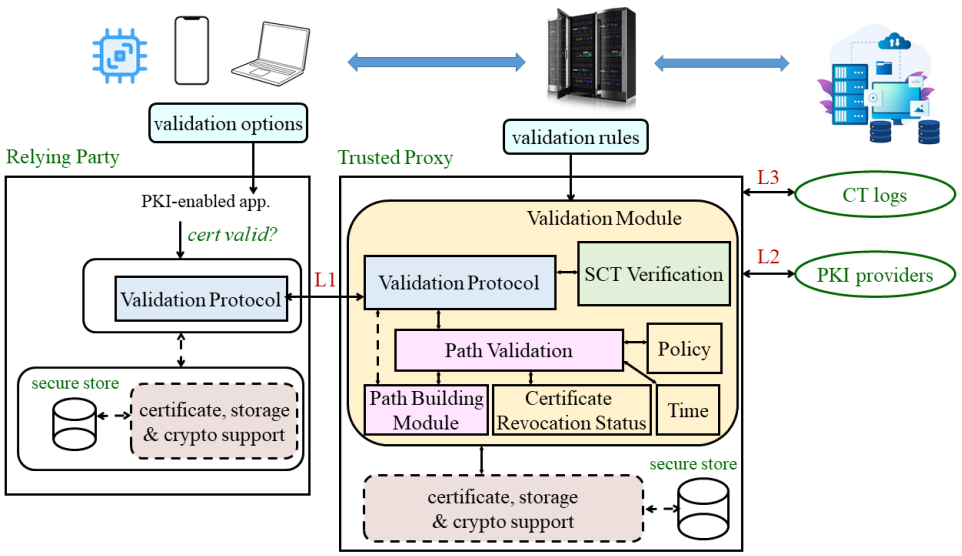

Fig. 1. TPValCert architecture.

Additionally, support for other revocation mechanisms, like for example CCADB CRLs [8], might be included also in this module. The `Policy Processing` module handles the certificate and the validation policies. The `SCT Verification` module checks the SCTs by interrogating the CT logs over the link L3. The `Time` module must provide an accurate and reliable time reference. The simplest form is time represented as GeneralizedTime but alternate forms include a TST [20] or a DVC [13]. A typical server application receives and parses the request by making calls to the path validation module.

This is a general architecture, void of details on the use of a specific protocol. Thus, L1 and L2 can be mapped onto particular protocols of choice. The link L3 is used instead to interact with the CT system. When CAs issue a certificate, they submit it to multiple (CT) logs, and each log returns a precertificate SCT (Signed Certificate Timestamp) promising to log the certificate [4]. The SCTs can be either embedded in the final certificate to be validated or it may be provided to a RP application through a Transport Layer Security (TLS) protocol extension, such as when a browsers connects to a web server over a TLS connection supporting this extension. Since each SCT uniquely identifies a certificate, and certificates are roughly unique per website, an application (browser) that directly interacts with Google asynchronously sharing a sample of SCTs with Google basically shares browsing history with Google [4] if the user has opted in for this feature. This might not be acceptable for some applications with strict privacy requirements. The RPs can thus delegate SCT verification to the trusted proxy to hinder browsing history of a relying party. Thus, the L3 may be used by the trusted proxy to communicate with the CT system, asking for a proof that the (verified) certificate has been included in the CT logs.

Configuration options. Through an end-user interface, the

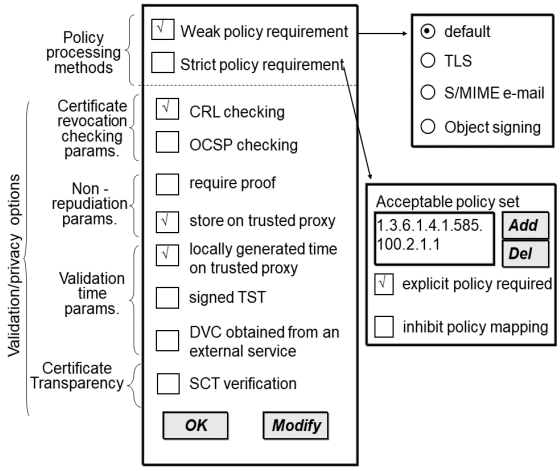

Fig. 2. Sample RP configuration allowing the selection of certificate policy parameters (strict or weak) and validation/privacy options.

RP can configure several parameters to control the certificate validation process or to express the privacy requirements, as shown in Fig. 2. We have divided the options into the following distinct categories:

1. Validation/Privacy options. The RP may indicate to the trusted proxy information relevant or required for path processing, e.g., certificates, CRLs, OCSP responses, validation time. Moreover, it can ask for SCT verification, indicating to the proxy to check the presence of the leaf in the CT logs, hiding at the same time RP identity. Inspired by the approach in [4], the trusted proxy performs SCT verification by fetching from the CT logs all SCTs that share the same prefix when hashed, as shown in Fig. 3. If the RP's SCT is not present in the returned list, then the trusted proxy indicates certificate mississuance to

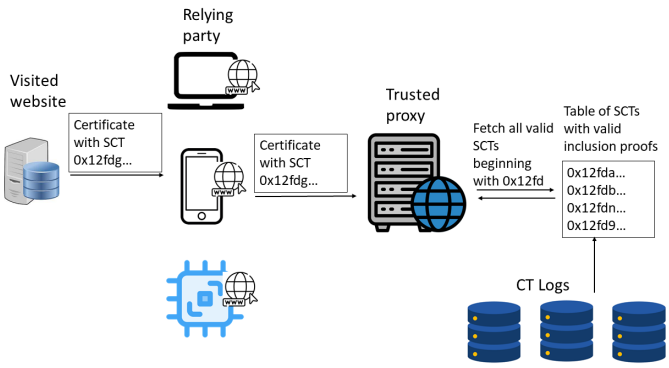

Fig. 3. SCT verification in TPValCert architecture.

the RP. Potentially, the absence of the SCT from the list can indicate a CT log misbehaviour and the incident might be reported to Google.

2. Certificate Policy Processing Parameters. When validating one leaf certificate, the policy processing module must examine the certificate policies of the CAs from a certification path. The RP may select these parameters to specify a strict (numerical) or weak (derived from the application context) requirement. The strict certificate policy requirement is used when the RP knows the set of certificate policies that are acceptable for its intended use. Additionally, the RP can indicate how the proxy should process the policy constraints. In this case, the trusted proxy can return an *on/off* response, i.e., it will indicate the path as valid only there is a match with one of the certificate policies specified in the set. Alternatively, an RP may select a weak certificate policy requirement to indicate the application context, e.g., secure e-mail.

On the server side, the configurable validation rules include parameters to manage the functionality of each internal module and parameters used in the communication with external parties. For example, in an enterprise environment, the trusted CAs should be configured on the secure store.

III. TPValCert Implementation

The IETF PKIX working group defined two protocols that could be exploited in TPValCert: SCVP (Server-Based Certificate Validation Protocol) and the Data Validation and Certification Server (DVCS). We explain their use in TPValCert.

a) SCVP: SCVP allows delegating path discovery and/or validation processing to a server by using a simple request-response model. At the same time, the protocol allows central administration of validation policies within an organization.

In the first place, the client generates a SCVP request, which contains a *requestNonce* holding a request identifier generated by the client. The purpose of this field is to protect against replay attacks. If a client includes a *requestNonce* value in the request, the server should return a response including the value of *requestNonce* from the request as a *respNonce* field. The *requestorText* field in the SCVP request can provide text for inclusion in the corresponding response, such as it may describe the nature or reason for the request. The *query* item in the request contains *queriedCerts* and a *validationTime* relative to which the SCVP client wants the server to perform the checks. The client may allow the server to used a cached SCVP response: in this case the client may exploit the *producedAt* field to indicate requirements on the freshness of the cached response. In practice, the *producedAt* field contains the earliest date and time at which an acceptable SVCP response could have been produced. The *validationPolicy* defines the one to be used by the proxy during certificate validation. It includes trusted anchors, key usages, user policy set. The SCTs to be verified could be placed instead in the *requestExtensions* field.

In the SCVP response, we also find fields used to express the result of the validation process and other extensions. Besides the *serverConfigurationID*, which carries the version of the server configuration when it processed the request, the *responseStatus* field contains status information that will be returned to the SCVP client about its request. This field contains a status code and an error message. The *respValPolicy* field contains either a reference to the full validation policy or the full policy by value used by the server to validate the request. The *requestRef* is used to identify the requestor, that is it associates the response to a particular request using either a copy of the request or a hash of the request. The *cvResponseExtensions* contains the result of the SCT verification process and the proof.

b) DVCS: A DVCS server may assert the validity of signed documents, public key certificates and the possession or existence of data. Among these functionalities we were interested in the service allowing a client to offload certificate validation to a server. In this case, the trusted proxy is a DVCS server that exploits multiple data sources and communicates with various servers to provide signed response structures called Data Validation Certificates (DVCs) to the clients. The DVCs contain validation results and trustworthy time information asserting certificate validation status. Moreover, we exploit the extensions in this message format for requesting SCT verification and returning SCT verification proofs.

In a DVCS transaction, the client prepares a request containing data whose validity must be certified, data to be used in the verification processes and extensions to carry additional data. The clients can ask the server to validate more than one certificate at once. Thus, all leaf certificates, the chains (if any) and the policy information are embedded into one single request. The client prepares the request by calling the Validation Protocol module, which processes the validation/privacy options selected by the RP. For instance, the client chooses strict or weak certificate policy requirements.

Selecting strict certificate policy requirement. In this case the client specifies an *acceptable policy set* consisting of one or more acceptable certificate policies identifiers that get inserted by the Validation Protocol module in the *acceptable policy set* inside the request. For example, the client may choose to accept only paths consistent with some high-assurance certificate policy identifier. The Policy Processing

module indicates the path as valid only if it there is a match under one or more of the policies specified in this set and it will return the subset of the *acceptable policy set* for which the path is valid. Additionally, the client can indicate to the server how to process the certificate policy OID extensions. Thus, two boolean flags in the request, an *explicit policy required* and *inhibit policy mapping* (whose meaning is identical to the fields in the *Policy Constraints* [16]) can be set to indicate whether the policy mapping is allowed and to explicitly specify the policies required. In this context their effect is to allow the client to set the policy constraints state immediately, without waiting for the certificate extension to appear in the path. The *explicit policy required* determines if an acceptable policy identifier [16] needs to explicitly appear in the certificate policies *extension* field of all certificates in the path. By setting this flag to *true*, the client indicates that the certificate policies in the chain must be consistent with the *acceptable policy set*. The *inhibit policy mapping* determines whether policy mapping will be inhibited or not during path processing. By setting this flag to *true* the client indicates to the server that the identifiers in the *acceptable policy set* must not be mapped by a CA in the chain. If the client chooses the strict certificate policy requirement but it doesn't specify an *acceptable policy set* (and the flags) then the fields of the request used for storing these values are left blank. This will be interpreted by the trusted proxy that *any policy* is acceptable for the client.

Selecting weak certificate policy requirement. In this case, the client's application context is inserted as an *extension* in the request. However, the absence of certificate policy data in the request (i.e., *acceptable policy set* and flags) must not be interpreted by the server that *any policy* is acceptable. When the server constructs the response it will set the *acceptable policy set* to indicate the set of certificates for which the certification path is valid and the policy mappings that were processed. If the *Default* option is chosen in Fig. 2 then this means that *any policy* is acceptable for the client.

The *Validation/privacy options* (presented in Sect. II) allow specifying the validation policy under which the validation is requested. The client can express this policy with an identifier inserted in the request in the *requestPolicy* field of the *DVCSRequestInfo* structure. This case can be applied if the client knows in advance both the validation policy that is appropriate for one particular application as well as its unique identifier. Another possibility could be either to map the validation options to an identifier or to define *extensions* in the request (possibly also in the *DVCSRequestInfo* structure) to hold the validation options but in this case the server must be able to process these *extensions*. The client can verify that the server has made the verification in accordance to the validation parameters because a copy of the *DVCSRequestInfo* of the corresponding request is inserted in the response.

The client can set also a number of Validation Protocol module parameters, such as: check the validity of the server's signing certificate store part of the information for later use; accept unsigned responses (this may be the case if the client uses a transport mechanism that provides server authentication); require CMS encapsulation [19] for the request or to establish an TLS channel to ensure not only server authentication but also the confidentiality of data exchanged.

Upon receiving a request, the trusted proxy checks first whether the request is valid, e.g., whether it contains a time expressed in correct format, the name of the server, the request information and the appropriate service. The Validation Protocol module parses the request, retrieves the leaf certificate, and passes it to the path validation module. The certificates and algorithms for validation of certification paths follow the conventions of X.509 [11] and the PKIX profile [16].

If multiple leaf certificates are present in the request, they are processed sequentially. Unless the client has inserted also certificate chains in the request, useful for constructing the certification paths, or if the server's local policy does not allow the use of this chain, the path validation module calls the path-building module to construct the certificate chains. Other request *extensions* can be defined to support the certificate path construction on the server but they are beyond the scope of this paper. The path discovery produces a number of possible 'valid' chains called *candidate chains*. To choose the correct chain, the path building module receives from the Validation Protocol module the values (retrieved from the request) containing either the policy OIDs and policy constraints flags or the *extension* containing a client's weak policy requirement. In other words, the proxy can determine the certificate policy processing method chosen by the client only after processing the fields in the request that carry the certificate policy data and the appropriate request *extension*. If none of the above data is present in the request then the server concludes that the client considers acceptable *any policy*. The server cannot determine exactly at this point which paths are good without doing a path validation also. Thus, one candidate chain is passed as input to the path validation module for processing. Throughout the path validation process the name constraints are checked also and function calls are made to the policy processing module to check the policy constraints. If required, functions from the certificate status checking or time modules are called to return OCSP or TSP [20] responses. Upon completion of path validation, more than one path could still satisfy client's criteria. In the simplest and most common case, the client is asking for a single chain. Finally, the path validation module returns to the Validation Protocol module the validation results and other related data, such as the set of policies used and the mappings that were processed during the execution of certificate path validation algorithm. The Validation Protocol module generates a DVC or an error message, which is verified by the client.

Implementation details. The first step in implementing TPValCert was the implementation of the validation protocol module to generate, parse, sign and verify SCVP and DVCS messages by extending the work in [14] and exploiting the OpenSSL library [21]. To support OCSP, we've embedded an OCSP client into the certificate status checking module and the

calls to this module are made via a simple OCSP client API. For the path building and path validation modules we've used the code currently developed in the OpenSSL library. Nevertheless, other freeware implementations for building/validating certificate chains are already available and may be used. We work on the integration in TPValCert of the SCT Verification module based on the SCT auditing implementation in Chrome [22]. We investigate also the technique for private SCT auditing exploiting Private Information Retrieval (PIR) [23], which allows users to query a database privately without the database operator learning the query or the response [24]. With the imminent adoption of static CT/Sunlight API [25] and the updates to CT logs, certificates can be queried in the CT logs much quicker than searching them based on hash [26].

IV. Conclusions and Future Work

Certificate validation is a foundational process in PKI exploitation. Nevertheless, it is still performed incorrectly or incompletely nowadays or could leak user privacy data. To counter this problem, we propose a privacy-preserving modular architecture, TPValCert, exploiting a trusted proxy for providing certificate validation services to clients. In our implementation, the relying parties may use either SCVP or DVCS protocols for communicating with the trusted proxy. The clients may delegate certificate validation and can control to various extent the validation process by specifying several validation and privacy parameters to the server. Future work will cover the implemention of SCT verification and the evaluation of trusted proxy's performance in terms of delays introduced by the interaction with the CT system and external servers. Although TPValCert is in its incipient stage, we aim to integrate and test it with TLS middleboxes, or widely used applications, including mobile or desktop browsers, or applications for devices with limited processing capabilities, such as IoT or embedded devices.

References

[1] D.G. Berbecaru and A. Lioy, "An Evaluation of X.509 Certificate Revocation and Related Privacy Issues in the Web PKI Ecosystem," in IEEE *Access*, vol. 11, pp. 79156-79175, 2023, doi: 10.1109/ACCESS.2023.3299357.

[2] The Chromium Projects. Moving Forward, Together. Updated: 2024-10-09. Available: https://www.chromium.org/Home/chromium-security/root-ca-policy/moving-forward-together/

[3] CA/Browser Forum, "Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates", Version 1.8.6, 14 Dec. 2022, https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.8.6.pdf

[4] E. Stark, J. DeBlasio and D. O'Brien, "Certificate Transparency in Google Chrome: Past, Present, and Future," in IEEE Security & Privacy, vol. 19, no. 6, pp. 112-118, Nov.-Dec. 2021, doi: 10.1109/MSEC.2021.3103461.

[5] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, "The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem," Internet Measurement Conference 2018 (IMC '18). ACM, New York, NY, USA, 343–349, doi: 10.1145/3278532.3278562.

[6] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission accomplished? HTTPS security after diginotar," 2017 Internet Measurement Conference (IMC '17). Association for Computing Machinery, New York, NY, USA, 325–340., doi: 10.1145/3131365.3131401

[7] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC-6960, June 2012.

[8] M. Sosnowski et al., "An Internet-Wide View on HTTPS Certificate Revocations: Observing the Revival of CRLs via Active TLS Scans," 2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Vienna, Austria, 2024, pp. 297-306, doi: 10.1109/EuroSPW61312.2024.00038.

[9] T. Chung et al., "Is the Web Ready for OCSP Must-Staple?" Internet Measurement Conference 2018 (IMC '18), Association for Computing Machinery, New York, NY, USA, 105–118. https://doi.org/10.1145/3278532.3278543

[10] P. Samarati, "Protecting respondents identities in microdata release," in IEEE Transactions on Knowledge and Data Engineering, vol. 13, no. 6, pp. 1010-1027, Nov.-Dec. 2001, doi: 10.1109/69.971193.

[11] ITU-T Recommendation X.509-ISO/IEC 9594-8: Information Technology - Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks.

[12] T. Freeman, R. Housley, A. Malpani, D. Cooper, and W. Polk, "Server-Based Certificate Validation Protocol (SCVP)," IETF, RFC 5055, December 2007.

[13] C. Adams, P. Sylvester, M. Zolotarev, R. Zuccherato, "Data Validation and Certification Server Protocols," Feb. 2001, IETF, RFC-3029.

[14] D. Berbecaru and A. Lioy, "Towards simplifying PKI implementation: Client-server based validation of public key certificates", presented at IEEE Symposium on Signal Processing and Information Technology (ISSPIT), Marrakech, Morocco, 18 - 21 Dec. 2002, Available at: https://arxiv.org/pdf/1910.06641.

[15] X. de Carné de Carnavalet and Paul C. van Oorschot, "A Survey and Analysis of TLS Interception Mechanisms and Motivations: Exploring how end-to-end TLS is made "end-to-me" for web traffic," ACM Comput. Surv. 55, 13s, Article 269 (December 2023), 40 pages, doi: 10.1145/3580522.

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," RFC-5280, May 2008, doi: 10.17487/RFC5280.

[17] Y. Elley, A. Anderson, S. Hanna, S. Mullan, R. Perlman, S. Proctor, "Building certification paths: Forward vs. reverse," Proceedings of Network and Distributed System Security Symposium, 2001. Available at: https://www.ndss-symposium.org/ndss2001/building-certifications-paths-forward-vs-reverse/

[18] Y. Liu and Z. Yang, "The Research and Design of the Proxy for Certificate Validation Based on Distributed Cross-Certification," 2017 5th Intl Conf on Applied Computing and Information Technology/4th Intl Conf on Computational Science/Intelligence and Applied Informatics/2nd Intl Conf on Big Data, Cloud Computing, Data Science (ACIT-CSII-BCD), Hamamatsu, Japan, 2017, pp. 135-140, doi: 10.1109/ACIT-CSII-BCD.2017.18.

[19] R. Housley, "Cryptographic Message Syntax (CMS)," IETF, RFC 3369, August 2002. Available at: https://www.rfc-editor.org/rfc/rfc3369.txt.

[20] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, "Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)," IETF, RFC 3161, 2001.

[21] The OpenSSL project web page, http://www.openssl.org.

[22] E. Stark and C. Thomson, "Opt-in SCT Auditing," Sept 30, 2020, Available at: https://chromium.googlesource.com/chromium/src/+/HEAD/services/network/sct_auditing/README.md

[23] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, "Private information retrieval," Proceedings of IEEE 36th Annual Foundations of Computer Science, Milwaukee, WI, USA, 1995, pp. 41-50, doi: 10.1109/SFCS.1995.492461.

[24] L. Heimberger, C. Patton, B. Westerbaan, "Private SCT Auditing, Revisited," Cryptology ePrint Archive, 2025, Avialable at: https://eprint.iacr.org/2025/556

[25] The Static Certificate Transparency API project, Available at: https://c2sp.org/static-ct-api.

[26] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, V. Vaikuntanathan, "One server for the price of two: simple and fast single-server private information retrieval," Proc. of the 32nd USENIX Conference on Security Symposium (SEC '23). USENIX Association, USA, Article 218, 3889–3905.