Lukas Alber, BSc

A Short-Lived Answer to Full Delegation

Applying Time-Bound Identity-Based Signatures to TLS

Master's Thesis

to achieve the university degree of

Dipl.-Ing.

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dr.techn. Peter Lipp

Advisors

Dipl.-Ing. Stefan More & Dr.techn. Sebastian Ramacher

Institute of Applied Information Processing and Communications
Head: Univ.-Prof. Dr.techn. Stefan Mangard

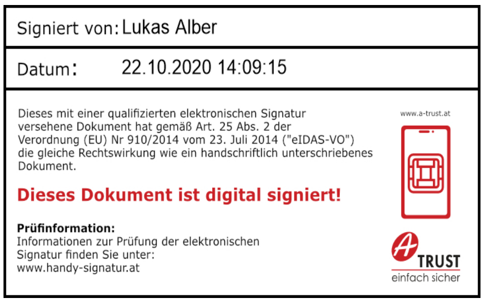
Graz, October 2020

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.



Signiert von: Lukas Alber

Datum: 22.10.2020 14:09:15

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Dieses Dokument ist digital signiert!

Prüfinformation:
Informationen zur Prüfung der elektronischen Signatur finden Sie unter:
www.handy-signatur.at

www.a-trust.at

TRUST
einfach sicher

Signature

Abstract

Nowadays, high performance and scalability demands on the internet lead to a massive deployment of Content Delivery Networks (CDNs). Positioned between client and server, they bear a man-in-the-middle nature. That opposes the end-to-end philosophy of Transport Layer Security (TLS), indispensable for modern internet security. "Full Delegation" practices are therefore utilized to bring these opposing sides together, i.e., the CDN can authenticate itself as server by receiving direct access to a server's private key. Research showed that Full Delegation had been a common practice of CDN providers for years (Liang et al., S&P'14). Some CDNs offer workarounds. However, these suffer from several disadvantages. Thus, we propose to use a modified identity-based signature scheme that enables the server to delegate short-lived privileges to the CDN without introducing significant changes to TLS. We implemented a signature provider library and integrated it into a TLS stack to perform benchmarks and evaluate the approach. Further, we show that it features reasonable performance and overcomes several disadvantages of its competing approaches.

Keywords: Full Delegation, TLS, CDN, identity-based signatures

Kurzzusammenfassung

Heutzutage führen hohe Geschwindigkeits- und Kapazitätsanforderungen im Internet zu einem massiven Einsatz von Content Delivery Networks (CDNs). Zwischen Client und Server sitzend entsteht so eine Man-in-the-middle-Situation. Dies steht im absoluten Gegensatz zur End-to-End-Philosophie von Transport Layer Security (TLS), die für moderne Internet-Sicherheit unverzichtbar ist. Um diese gegensätzlichen Aspekte zu vereinen wird die Praktik der sogenannten "Full Delegation" angewandt, d.h. der CDN kann sich als Server authentifizieren da ihm Zugriff auf den privaten Schlüssel des Servers gewährt wird. Studien haben gezeigt, dass Full Delegation bereits seit Jahren gängige Praxis bei CDN-Anbieter ist (Liang et al., S&P'14). Einige CDNs bieten Alternativen, die aber mehrere Nachteile haben. Darum schlagen wir vor, ein Identitätsbasiertes Signaturschema zu nutzen. Dieses ermöglicht dem Server eine kurzlebige Erlaubnis an den CDN zu übergeben, ohne wesentliche Änderungen an TLS vorzunehmen. Dazu implementierten wir eine Signatur Provider Bibliothek und integrieren sie in eine TLS Plattform, um Geschwindingkeitsmessungen und weitere Evaluierungen durchzuführen. Wir zeigen eine Lösung mit vernünftiger Geschwindigkeit, die mehrere Nachteile konkurrierender Ansätze zu überwinden weiß.

Schlüsselwörter: Full Delegation, TLS, CDN, identity-based signatures

Acknowledgements

I would like to thank my advisors for their lasting support and guidance throughout my work on this thesis. In particular, I want to thank Stefan More for the exciting working years together at the IAIK, and Sebastian Ramacher for providing precious advice even though he already left the institute. Furthermore, my special thanks go to my supervisor Peter Lipp for the support throughout the past working years and the possibility to write the paper during my time at the institute. Thanks also go to the SIC Foundation (IAIK libraries) for the technical support, especially Dieter Bratko. I also want to thank Olamide Omolola and all other people from the IAIK who supported me in my thesis's work. Last but not least, special thanks go to my family for their loving support throughout the years of study.

The thesis was supported by the European Union's Horizon 2020 research and innovation project KRAKEN (grant agreement number 871473).

Contents

Abstract **iv**

1 Introduction **1**

1.1 Bad Practices 2
1.2 Requirements 3
1.3 Contribution 5

2 Related Work **7**

2.1 Measurements 7
2.2 Workaround Attempts 8
2.2.1 Name Constraints 8
2.2.2 DANE Extension 9
2.2.3 SSL Splitting 10
2.2.4 KeylessSSL 10
2.2.5 Delegated STAR Certificates 13
2.3 Proxy Scheme Paradigm 14
2.3.1 Proxy Certificates 14
2.3.2 Delegated Credentials 16
2.3.3 Proxy Signature Schemes 17
2.3.4 Hierarchical Identity-Based Signatures 18
2.4 Nineteen Criteria Framework 18
2.5 Delegation in Similar Contexts 18
2.5.1 D2TLS 19
2.5.2 STYX and Other SGX Approaches 20
2.5.3 Multi-Context TLS 20
2.5.4 LIGHT^{est} 21

3 Preliminaries **22**

3.1 Full Delegation in General 22

3.2 Network . . . 23
3.2.1 Public Key Infrastructure . . . 23
3.2.2 Certificate Transparency . . . 26
3.2.3 Transport Layer Security . . . 27
3.2.4 Revocation . . . 31
3.2.5 The Short-Lived Paradigm . . . 32
3.2.6 Content Delivery Network . . . 34
3.3 Cryptography . . . 37
3.3.1 Forward and Backward Security . . . 37
3.3.2 Basic Complexity Assumptions . . . 37
3.3.3 Bilinear Pairing . . . 38
3.3.4 Identity-Based Encryption and Signature . . . 40
3.3.5 Hierarchical Identity-Based Encryption . . . 41
3.3.6 Signature Schemes . . . 44
3.3.7 Hierarchical Identity-Based Signature . . . 45
3.3.8 Time-Bound Identity-Based Signature . . . 47
3.3.9 TBIBΣ Construction . . . 50
3.3.10 Practical Considerations . . . 51

4 Integration **53**
4.1 Concept . . . 54
4.1.1 Epochs Limit a Delegation . . . 58
4.1.2 Revocation . . . 61
4.1.3 Security Consideration on Epoch Lengths . . . 63
4.2 Implementation . . . 64
4.2.1 HIBS Signature Provider Library . . . 65
4.2.2 TLS Stack . . . 67
4.3 Integration into Major TLS-Libraries and Browsers . . . 70
4.4 Other Applications . . . 70
4.4.1 TBIBΣ in TLS Restricting Connection and Content Types 70
4.4.2 HIBS and TBIBΣ in Other CDN Constellations . . . 71
4.4.3 HIBS and TBIBΣ in Other Applications . . . 72

5 Evaluation **74**
5.1 Measurements . . . 74
5.1.1 TBIBΣ Setup . . . 74
5.1.2 TBIBΣ Evaluation . . . 75

5.2 Scheme Performance 77
5.2.1 TLS Setup 77
5.2.2 TLS Evaluation 78
5.3 Qualitative Evaluation 81
5.3.1 Discussion 81
5.3.2 Systematic Characterization 84

6 Conclusion **88**
6.1 Summary 88
6.2 Future Work 89

7 Bibliography **90**
Literature 90
Online Resources 101

List of Figures

1.1 Overview of network distances with and without CDN. . . . 2
1.2 Sequence diagram of Full Delegation (key sharing). 4

2.1 KeylessSSL's round-trips. 9
2.2 Concept of KeylessSSL. 11
2.3 Sequence diagram of STAR and ACME delegating to a CDN. 13
2.4 Concept of Proxy Certificates. 15
2.5 Concept of Delegated Credentials. 17
2.6 Sequence diagram of D2TLS. 19

3.1 Concept of Public Key Infrastructure (PKI). 24
3.2 Structure of an X.509 certificate. 25
3.3 Sequence diagram of Transport Layer Security (TLS) 1.3. . . . 27
3.4 Mapping of pairing scheme. 38
3.5 Concept of time-grid explaining epochs. 48

4.1 Scenario of Full Delegation in the context of CDN. 53
4.2 Sequence diagram of TBIBΣ TLS. 56
4.3 Plot of web certificates' validity. 60
4.4 Overview on the used libraries. 65

5.1 Plot of different signature schemes' benchmark. 76
5.2 Plot of TLS 1.3 benchmarks. 79

List of Tables

5.1 Theoretical numbers on signatures. 75
5.2 Java benchmark table. 76
5.3 Relic benchmark table. 77
5.4 TLS benchmark table. 78
5.5 Sizes of handshake with different signature algorithms. 80
5.6 Comparison of different delegation approaches. 85

Acronyms

ACME Automated Certificate Management Environment. 13, 14, 33, 62, 63, 83, 86, 89
ALPN Application-Layer Protocol Negotiation. 71

CA Certificate Authority. 3, 23, 31, 68
CDH Computational Diffie-Hellman. 37, 38
CDN Content Delivery Network. iv, v, 1–3, 5–10, 12–14, 16, 17, 34–36, 53–55, 57, 58, 61, 62, 68, 69, 71, 72, 82, 83, 86–88
CDNI CDN Interconnections. 71, 73, 83, 89
CRL Certificate Revocation List. 31, 62, 63
CSR Certificate Signing Requests. 55, 68
CT Certificate Transparency. 16, 26, 83

DANE DNS-Based Authentication of Named Entities. 8, 9, 86
DDH Decisional Diffie-Hellman. 37, 38
DDoS Distributed Denial of Service attack. 1, 34, 35
DeC Delegated Credentials. 16, 21, 32, 63, 80, 83, 84, 86–89

HIBE Hierarchical Identity-Based Encryption. 41, 46
HIBS Hierarchical Identity-Based Signature. 70, 72

IBE Identity-Based encryption. 40
IBS Identity-Based signature. 40
IETF Internet Engineering Task Force. 1, 16, 63
IoT Internet of Things. 19, 20, 36, 72

JCA Java Cryptography Architecture. 65–67

MAC Message Authentication Code. 10, 29

NPN Next Protocol Negotiation. 71

OCSP Online Certificate Status Protocol. 31, 32, 59, 62–64
OID Object Identifier. 25

PKI Public Key Infrastructure. x, 1, 8, 14, 15, 23, 24, 31, 33, 54, 57, 61, 64, 68, 69, 82, 83
PoP Point of Presence. 2, 34–36
PPT probabilistic, polynomial-time. 41, 44
PSK pre-shared key. 29, 30, 62

SAN Subject Alternative Name. 3, 26
SNI Server Name Indication. 28
SSL Socket Security Layer. 10
STAR Short-Term Automatically Renewed. 13, 32, 33, 62, 63, 83, 86

TBIBΣ Time-Bound Identity-Based Signature. x, 5, 6, 22, 47–51, 53–58, 62–87, 89
TLS Transport Layer Security. iv, v, x, 1, 27, 53–55, 57, 59, 69, 72
TOTP Time-based One-time Password algorithm. 59

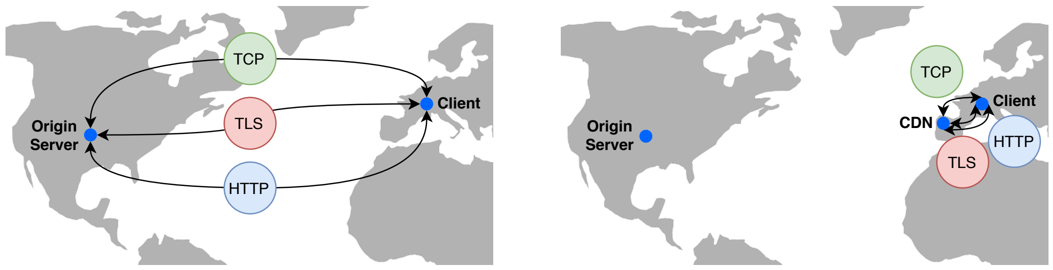
1 Introduction

The demand for security on the internet has been growing steadily in the past decades [Cis18]. An essential part of it is end-to-end secure communication between interacting parties. On untrusted networks, like the internet, the prevailing standard introduced by the Internet Engineering Task Force (IETF) is Transport Layer Security (TLS). It ensures confidential and authenticated communication between two endpoints [Res18]. TLS uses a Public Key Infrastructure (PKI) that comprises certificate chains for authenticating both endpoints (especially the server). In the handshake, one of the endpoints signs the handshake data, and the counterpart can subsequently verify the handshakes authenticity by using the former's certificate.

On the other hand, modern internet applications demand improved throughput, latency, scalability, and availability [Joh+01; VP03]. As a result, we can see an increasing popularity of Content Delivery Networks (CDNs) [KWZ01]. Cisco Systems [Sys19] recently released a survey on CDN traffic estimating a five-year increase of over 500% until 2022, reaching global traffic of 252 exabytes per month. Originally intended to reduce latencies by delivering cached content from a surrogate server network-wise near the client (see Figure 1.1) [Bae+97; Dil+02], CDN providers soon offered a variety of deployment benefits such as mitigation against various attacks like the Distributed Denial of Service attack (DDoS) [Can+16; Clob].

We can see that both factors play a vital role in today's internet. However, they reveal an opposing nature: While a CDN positions itself between client and server to broker content, TLS was meant to mitigate man-in-the-middle practices since it supports an end-to-end philosophy. This contradiction makes it hard to deploy both at the same time. Nonetheless, providers found questionable ways to do so.

If the server-side (origin server) decides to deploy a Content Delivery Network (CDN) service, the CDN has to serve content on behalf of the



(a) Without a CDN. (b) With a CDN.

Figure 1.1: Two Figures showing round trips with and without CDN. Without a CDN three trans-Atlantic instead of three continental round trips are needed in TLS 1.3.

server over a TLS connection to the client. To authenticate the connection, the CDN needs to identify as the origin server to fulfill the certificate's claim bound to the origin server's domain. The straightforward way is to use the origin server's secret key to sign the TLS handshake. Thus, the CDN needs access to the secret key. Such bad practices, i.e., giving a third party access to its secret key, is called "Full Delegation" in literature (see Section 3.1 for further explanations) [BPW12; MUO96].

1.1 Bad Practices

Full Delegation can take various forms in the TLS context. In the literature we observed two separate forms, Custom and Shared Certificates [Lia+14]. The first and simpler one describes a practice letting the origin server upload its private key to CDN provider (see Figure 1.2), who distributes it to its Point of Presences (PoPs)[1], violating the fundamentals of public key cryptography. Quoting from the X.509 RFC [Coo+08]:

> "The protection afforded private keys is a critical security factor. On a small scale, failure of users to protect their private keys

[1]A PoP in the context of CDN consists of multiple strategically positioned caching servers responsible for serving content for all network-wise near clients [imp].

> will permit an attacker to masquerade as them or decrypt their personal information. On a larger scale, compromise of a CA's private signing key may have a catastrophic effect."

Thus, this practice is also known as Private Key Sharing [Can+16]. Besides, scattered over different nodes across the entire web, the private key is more exposed than ever. Further, the origin server has no means to revoke the delegation to the CDN except revoking the certificate as a whole.

In the second form, the CDN provider manages directly the certificate bound to the origin server's domain. CDNs often list several different customers on the same certificate using the Subject Alternative Name (SAN) extension. Cangialosi et al. [Can+16] referred to these certificates as "cruise-liner certificates" and raised a number of questions:

> "Who on a cruise-liner certificate deserves access to the certificate's corresponding private key, given that who-ever has it can impersonate all others on the certificate? Who among them has the right to revoke the certificate, if so doing potentially renders invalid a certificate the others rely on? Cruise-liner certificates are not covered explicitly by X.509, but we can infer that, in all likelihood, only the hosting provider has the private keys and right to revoke."

Further, the usage of such a cruise-liner Certificate leads to an inaccurate communication of HTTPS's security indicators to the end-users. Also, it impedes the use of Extended Validation certificates[2] [Can+16] .

1.2 Requirements

In situations of Full Delegation, a considerable amount of trust is needed, since the peril of impersonation and misconduct is imminent. However, a

[2]Extended Validation certificates are issued by Certificate Authoritys (CAs), who also check the entity's legal identity [For]. However, since recent, Extended Validation certificates have been seen as obsolete [Fis19]

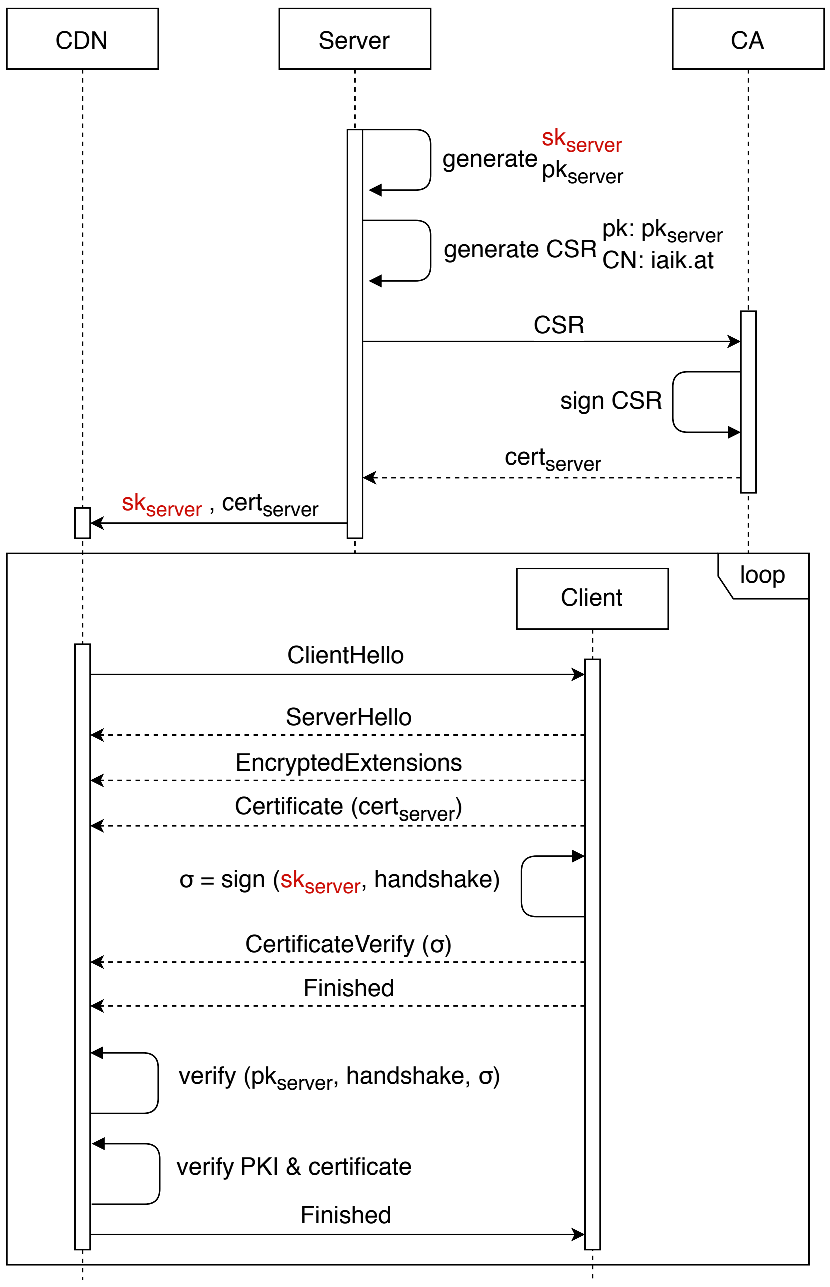


Figure 1.2: A sequence diagram of the Full Delegation scenario in TLS (private key sharing). The private (secret) key (sk) is highlighted in red.

CDN service is usually a third party that does not necessarily have to be trustworthy. Besides, the breach of trust can not only happen deliberately maliciously but also in negligence, e.g., when the service unpreparedly falls victim to an external hacking attack. As that is all quite hard to prove, we encourage the policy of relying on trust as little as possible.

Therefore, we postulate the following requirements that we regard necessary for the ideal delegation solution in the context of CDN internet solutions:

R1 The private key must be kept secure and solely in the hand of the owner.
R2 A delegation must be unforgeable and non-repudiable.
R3 The origin server is aware of the CDNs actions and approves them.
R4 Issuance and revocation must be efficient and independent.
R5 The delegation should identify the domain owner and the proxy (CDN).
R6 Delegating privileges in a fine-grained way, and enabling domain-based policies should be possible.
R7 The domain owner needs control over the validity period of the delegation.

1.3 Contribution

For the thesis, we explored relevant work conducted by other researchers regarding Full Delegation with the focus on solutions in the field of internet protocols, especially TLS. We found methods eliminating symptoms but not the fundamental problem, approaches solving the issue but being impractical, and yet to be standardized solutions. We evaluated all of them to understand their advantages and downsides. We give a summary in Chapter 2 (Related Work).

To better introduce our approach, we collected additional background on the internet standards and entities involved in the CDN context. Further, we explained the basics of signature cryptography, elliptic curves, and pairing-based schemes such as the Time-Bound Identity-Based Signature (TBIBΣ). We present those in detail in Chapter 3 (Preliminaries).

With the latter mentioned TBIBΣ, our solution tries to solve the Full

Delegation issue in TLS by switching to that signature algorithm. Instead of letting the CDN sign with the origin server's private key, we push a derived secret key containing the delegation to the CDN. While the signature created with the derived key still validates with the origin server's public key, additional delegation restrictions must hold at verification time to fulfill the validation. Like subdomain and time, these restrictions are easily accessible to the client and require no further changes to the TLS protocol. Further, since an origin server can not afford a traditional revocation system for the delegations it distributes, we propose short-lived delegated keys to curb the issue. We present our solution in more detail and discuss multiple design decisions in Chapter 4 (Integration). Further, we provide an implementation of the scheme and a modified TLS stack for evaluation. They are available on GitHub [AMR].

Subsequently, we evaluated the proposed solution. We measured the cryptographic scheme and TLS performance by evaluating various indicators. Further, we carried out an extensive analysis of its qualitative aspects comparing to related solutions introduced in related work. We also classified our approach into the 19-criteria framework of Chuat et al. [Chu+20], offering a systematization of knowledge regarding delegation and revocation. We report these outcomes in Chapter 5 (Evaluation).

We end with Chapter 6 (Conclusion) recapitulating what our solution managed to achieve and what we could not fulfill pursuing this approach. Further, we give an outlook on future research topics regarding Full Delegation and delegation in the TLS protocol.

The paper by Alber et al. [AMR20] was written during the elaboration of this thesis and appears at the CCSW 2020. The paper was a collaboration between the author and his advisors. Consequently, the thesis influenced the paper and vice versa. The paper adopted the benchmark results of the author's signature implementation and TLS integration, as well as most of the related work research, integration description, and evaluation. On the other hand, the thesis cites the cryptographic preliminaries on identity-based signatures, as well as the definition and construction of Time-Bound Identity-Based Signature (TBIBΣ). Further, it also cites the benchmark results from the Relic implementation of the TBIBΣ algorithm.

2 Related Work

In this chapter, we are going to explore different existing literature that has been released in recent years on the matter of Full Delegation and its security implications. For a recent Systematization of Knowledge paper on delegation (and revocation) for the internet, see Chuat et al. [Chu+20].

2.1 Measurements

In 2014 Liang et al. [Lia+14] conducted a systematic investigation on the diffusion of bad practices in the TLS context. They surveyed 20 of the most popular CDN providers and 10.721 websites maintained by them. Besides Full Delegation, which they found most concerning, they also observed invalid certificates, neglected revocation of stale certificates, insecure back-end communication. These concerns were not only from operational nature, but also included the design of the CDNs mechanisms back then.

Two years later, in 2016, Cangialosi et al. [Can+16] performed a large-scale measurement study unveiling how remarkably common Full Delegation practices are. Subsequently, they analyzed the impact of Full Delegation on responsible key management practices like revocation. They found that third parties managing the client's private key act more thoroughly but slower. With great concern, they also noticed that a small number of CDN, respectively, hosting providers have access to the majority of the popular websites' keys. More specifically, 76.5% of the organizations they identified, share one or more keys with a third party. Further, the ten biggest CDNs have access to 45.3% of the observed domains' private keys. Therefore, they called for a way that would reconcile management centralization and offer less trust aggregation at the same time.

2.2 Workaround Attempts

Since the Full Delegation was recognized by the internet community as a pressing issue, several workarounds have been proposed, using common Public Key Infrastructure (PKI) standards (Section 2.2.1), extending upcoming internet standards such as DNS-Based Authentication of Named Entities (DANE)[1] (Section 2.2.2), or even interfering with the TLS handshake itself (Section 2.2.3 and Section 2.2.4). In the following section, we take a closer look at these approaches.

2.2.1 Name Constraints

A straightforward technique to curb Full Delegation can be employed using a feature from X.509 certificates called the Name Constraints Extension [Coo+08]. It offers CAs the possibility to constraint their subordinate CAs to specific namespaces. In the CDN scenario, the origin server would apply for a subordinate CA-certificate, i.e., the certificate it receives has the "CA flag" set to true, and a Name Constraint set to its domain. Consequently, the origin server can issue certificates for the namespace of its domain, and process incoming Certificate Signing Requests from its CDN(s). This approach would allow to create delegations and revoke them again with existing standards of traditional PKI.

However, Liang et al. [Lia+14] showed its infeasibility. Besides, showing improper enforcement by end-user software, which could be fixed, they highlight the unaffordability and the lack of technical capabilities for an origin server to perform CA tasks. Further, CAs would not tolerate the massive issuance of intermediate certificates. High operational costs for vetting and other audits would not pay off. Indeed, Liang et al. [Lia+14] found in their survey that from their 1.5 million collected certificates, none contained a Name Constraint extension [Lia+14].

[1]It uses the DNSSEC (signed DNS-zone data) infrastructure to store certificates of DNS names used for TLS authentication [HS12].

2.2.2 DANE Extension

Liang et al. [Lia+14]'s solution, after unveiling Full Delegation issues in the CDN context in 2014, concentrates on making a delegation through the name resolution process using DANE [HS12]. They achieve that by extending the origin server's DNSSEC records with a special TLSA record, including its and the CDN's certificate. When establishing a connection to the domain, the CDN participates in the handshake and sends its certificate. The client can then recognize this delegation by consulting the origin server's TLSA record [Lia+14].

Although this promising solution was described several years ago, it did not establish itself. It has several disadvantages: Changes on DNS, respectively DANE side are unavertable. Also, the client needs changes to its certificate validation process. Further, the usage of DANE introduces an additional round trip between the client and the DNS server, which results in an increased page-load time [Lia+14] (However, implementations can support the lookup of the TLSA-record before initiating the TLS handshake [HS12, Chapter 4.1.], i.e., parallel with the A-record lookup). However, on the advantage side, it offers full transparency about any delegation and introduces no significant changes to the server or the CDN [Lia+14].

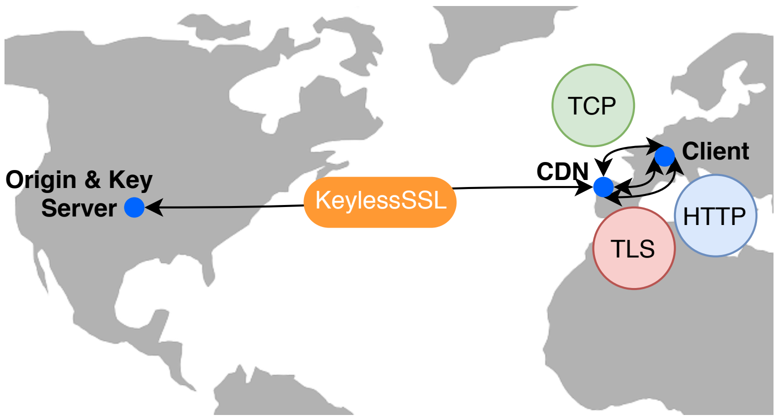


Figure 2.1: Graphic of the round-trips necessary for KeylessSSL. A key server is distant to the destination and introduces additional latency.

2.2.3 SSL Splitting

In 2005 Lesniewski-Laas et al. [LK05] presented a novel technique for guaranteeing the authenticity and data integrity of content cached and served by a proxy such as a CDN. As the name Socket Security Layer (SSL) splitting implies, this technique was designed for the predecessor of modern TLS and can be seen as a forerunner to KeylessSSL used on modern TLS. Originally the idea behind SSL splitting is to simulate a regular SSL connection between client and proxy by using authentication records form the server and data records from the proxies cache [LK05].

In more detail, SSL records are organized in data and an authentication component. The loose coupling of these offers the possibility to cache the data component at the proxy instead of resending it in full. However, it implies changes at the server and proxy side. Consequently, a proxy-server protocol extension is described. It features two new message types: one with a short unique payload identifier and a Message Authentication Code (MAC) authenticator, the other containing a session encryption key for the server. In practice, a cache hit results in a payload lookup. Then, the data is spliced together with the MAC received from the server-side [LK05].

Further, they conducted measurements with an experimental setup using this technique. They found a reduced bandwidth consumption of 25% to 90% depending on redundancy amount and cache warmth. On the other hand, uncached requests having to pass through the proxy encounter a 5% latency gain [LK05].

2.2.4 KeylessSSL

Based on SSL splitting (cf. Section 2.2.3), Cloudflare deployed their adaption called KeylessSSL [SN14] in 2014 for the first time to curb the issues of Full Delegation. Originally developed for TLS 1.2, Cloudflare's KeylessSSL (cf. Figure 2.2) tries to solve the issue by splitting the TLS handshake so that the CDN handles connection establishment and only authenticating operations are outsourced to a key server in control of the domain owner. Initially, both RSA- and Diffie-Hellman-handshakes were supported.

Stebila et al. [SS15] conducted an informal security analysis, including a

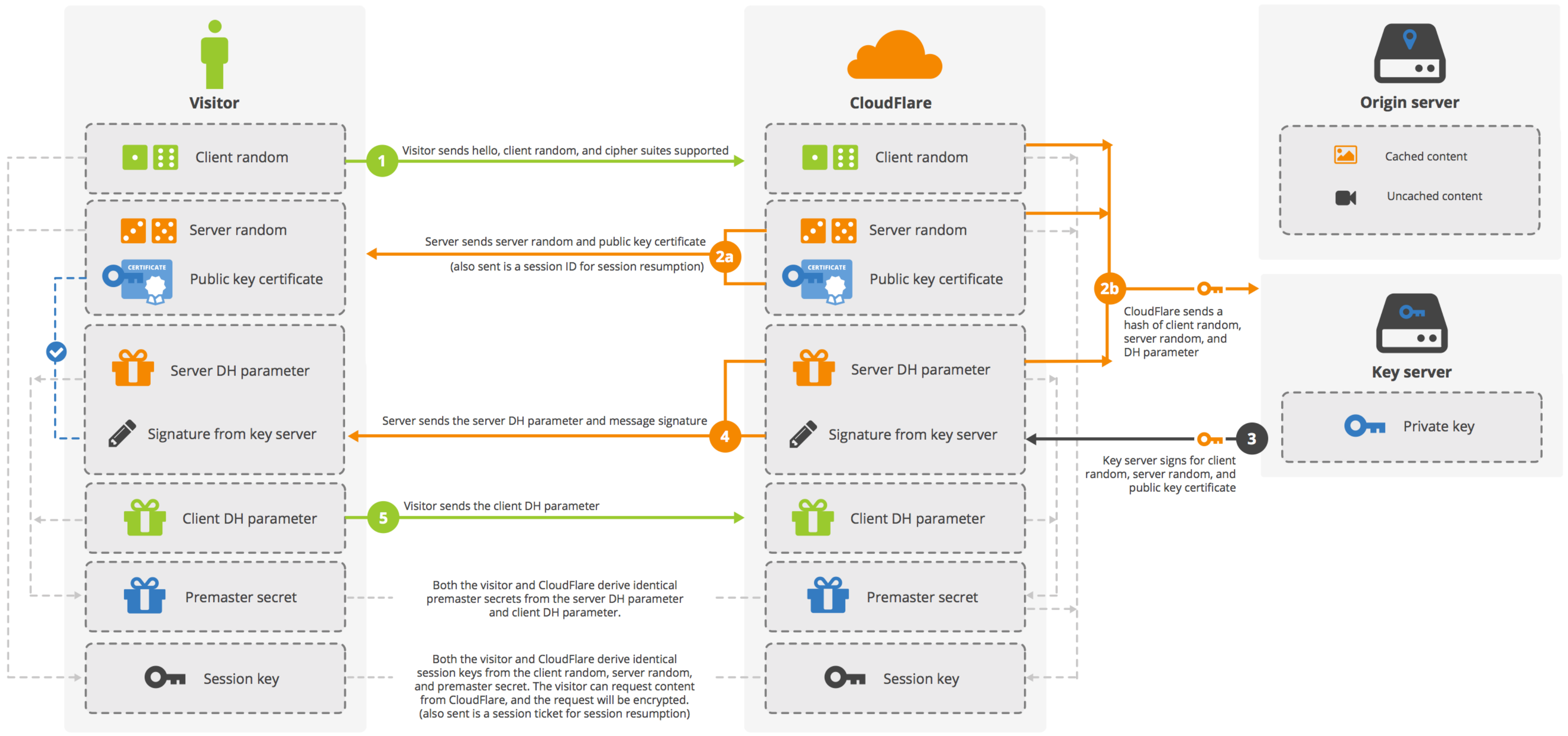

Visitor
CloudFlare
Origin server
Client random
Server random
Public key certificate
Server DH parameter
Signature from key server
Client DH parameter
Premaster secret
Session key
Cached content
Uncached content
Key server
Private key
1 Visitor sends hello, client random, and cipher suites supported
2a Server sends server random and public key certificate (also sent is a session ID for session resumption)
2b CloudFlare sends a hash of client random, server random, and DH parameter
3 Key server signs for client random, server random, and public key certificate
4 Server sends the server DH parameter and message signature
5 Visitor sends the client DH parameter
Both the visitor and CloudFlare derive identical premaster secrets from the server DH parameter and client DH parameter.
Both the visitor and CloudFlare derive identical session keys from the client random, server random, and premaster secret. The visitor can request content from CloudFlare, and the request will be encrypted. (also sent is a session ticket for session resumption)

Figure 2.2: The concept of KeylessSSL in a Diffie-Hellman based TLS handshake [SN14].

discussion on timing side-channel resistance and its effects on TLS session resumption. Later, a formal analysis by Bhargavan et al. [Bha+17] revealed insecurities of the approach. However, using a novel 3-party security definition dubbed `3(S)ACCE`[2]-security (since conventional TLS analyses are 2-party), they proposed fixes but argued for an upgrade of KeylessSSL implementations to TLS 1.3. Further, they urged to forbid session resumption except for selected cases.

In more detail, the RSA mode, which TLS 1.3 no longer supports, works the following: the key server decrypts a pre-master secret encrypted by the client using the domain owner's public key and sends it back to the CDN over a secure channel allowing the handshake to continue [Bha+17; SN14]

The Diffie-Hellman handshake (cf. Figure 2.2), on the other hand, works a little different: the key server receives a hash which he uses for signing and sends the signature back to the CDN. Until TLS 1.2, the hash consisted of the nonces from CDN and client plus the DH-parameters. Since TLS 1.3, the whole transcript is hashed (handshake's hash). Consequently, the CDN can incorporate the signature in the $CertificateVerify$ message so that the handshake can continue as usual [Bha+17; SN14].

While KeylessSSL has operationally proven itself for years in Cloudflare's CDN infrastructure, it suffers from some downsides. Firstly, the additional connection to the key server during the handshake introduces additional latency. Secondly, the domain owner must maintain a key server, which has to be reachable at all times. That is why Cloudfare pushes Delegated Credentials to replace KeylessSSL in this matter [SL19].

[2] `3` stands for three parties. `(S)` indicates that only the server is authenticated. `ACCE` (authenticated and confidential channel establishment) is an extension of `AKE` (authenticated key-exchange), adding full length-hiding authenticated encryption. It makes part of the terminology used in cryptography to classify a channel or state its characteristics [Jag+12].

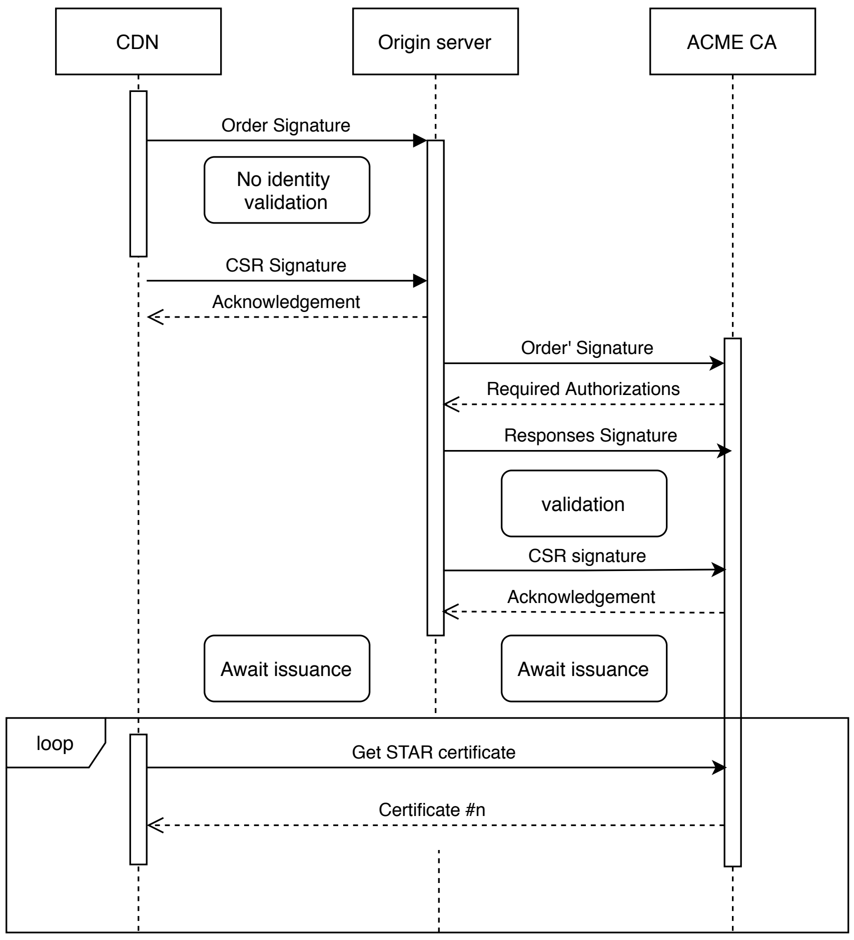


Figure 2.3: The diagram shows how Automated Certificate Management Environment (ACME) can be setup together with Short-Term Automatically Renewed (STAR) certificates to delegate to a CDN without sharing the private key [She+20a].

2.2.5 Delegated STAR Certificates

For Automated Certificate Management Environment (ACME) supporting CAs Sheffer et al. [She+20a] propose a modification of the standard procedure [She+20b] (see Figure 2.3). Short-Term, Automatically Renewed (STAR) certificates should be provided periodically from the CA for a subdomain used by the CDN. In more detail, the latter asks the origin server for a delegation, which on approval authenticates at the CA and ask for STAR certificates in the name of the CDN. The ACME CA then begins the periodical issuance on approval [She+20a].

2.3 Proxy Scheme Paradigm

While we have seen approaches using lookup techniques, traditional PKI, or a modified TLS handshake, we now want to concentrate on approaches that directly pass the delegation proof to the CDN (proxy). The proxy then uses this proof to demonstrate that it rightfully acts in the name of the origin server. We summarize these approaches under the term Proxy Scheme Paradigms and describe them in the following section.

2.3.1 Proxy Certificates

Proxy certificates have been around since the early 2000s. At the beginning they were mainly used in grid computing [Tue+04; Wel+04]. Mostly present in middleware, they were not envisioned for the web. Generally (cf. Figure 2.4), a proxy certificate can be issued by entities (e.g. origin server) holding a regular non-CA certificate (cf. Section 2.2.1 (Name Constraints), which needs the origin server to be a subordinate CA). It contains the public key bound to the target entity (CDN) and is signed by the origin server using its private key. The origin server may grant a fine-grained subset of privileges to the targeted entity specified within the certificate allowing the CDN to serve certain content [Chu+19].

The question of revocation is troubling since the issuer of a certificate is the origin server, and therefore the latter would have to maintain an own certificate revocation mechanism. Thus, for revocation causes such as key compromise, the proxy certificate's lifetime can be short-lived to curb the lack of a traditional revocation mechanism [Chu+19].

Incorporating proxy certificates into the current PKI would require small changes on the browser side, but introduces bigger ones on the server- and CDN-side. Further, CAs will not easily grant subordinate CA rights

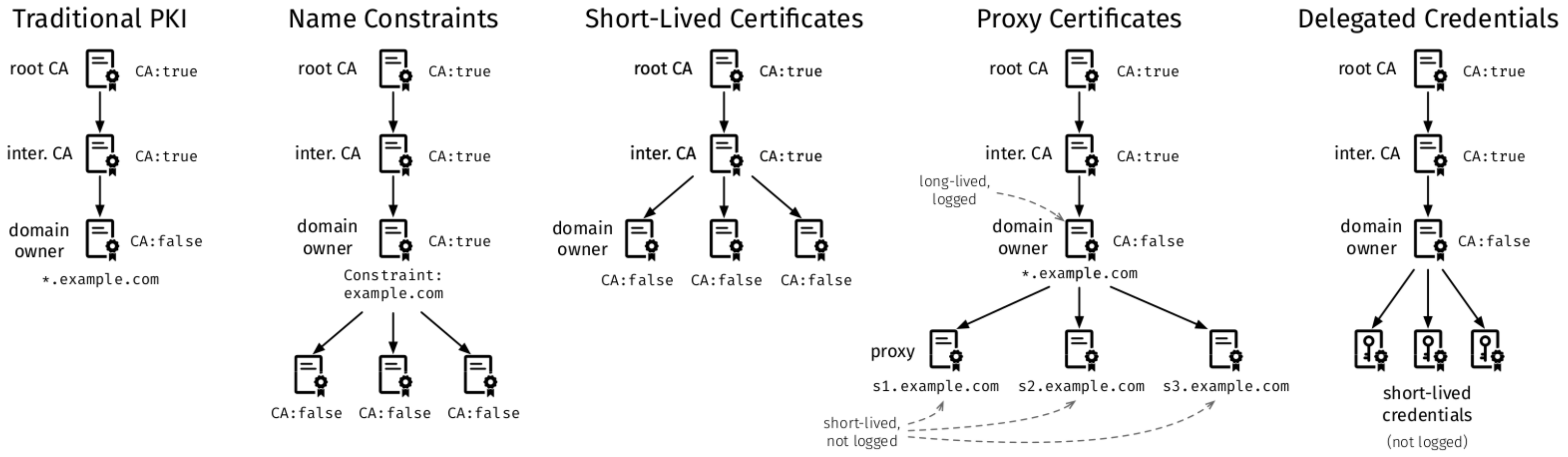


Figure 2.4: A comparison between traditional PKI, Name Constrains, Short-Lived Certificates, and Proxy Certificates [Chu+19].

or introduce a proxy certificate flag for end-entity certificates. Besides, the introduction of proxy certificates would put heavy pressure on Certificate Transparency (Chuat et al. [Chu+19] propose to not log proxy certificates at all).

2.3.2 Delegated Credentials

Another approach similar to short-lived proxy certificates is Delegated Credentials (DeC) [Bar+20] that has been proposed recently as an IETF standard. It is meant to replace established workaround such as KeylessSSL. Technically (cf. Figure 2.5), a delegated credential is a data blob persisting of a validity period and the proxy's public key. It is then signed by the origin server on the request of the CDN and therefore constitutes a delegation. On communication between the client and the CDN, the *ClientHello* message contains an extension indicating support. The CDN then signs the TLS handshake with its private key and sends the Credential blob with an extension of the *Certificate* message. The client can now verify the handshake with the delegated blob and the delegation with the origin server's certificate.

DeC is designed for the modern TLS 1.3 standard and has no traditional mechanism for revocation. It uses the validity period for short-lived longevity and lets the CDN renew its delegated credentials frequently. Further, DeC enjoys excellent support from big internet companies and is currently evaluated by Cisco, Facebook, Cloudflare, and Mozilla, while the latter three already announced support in their products [GNI19; JJM19; SL19]. From all potential solutions we discuss, this one is the most promising and constitutes the biggest competitor.

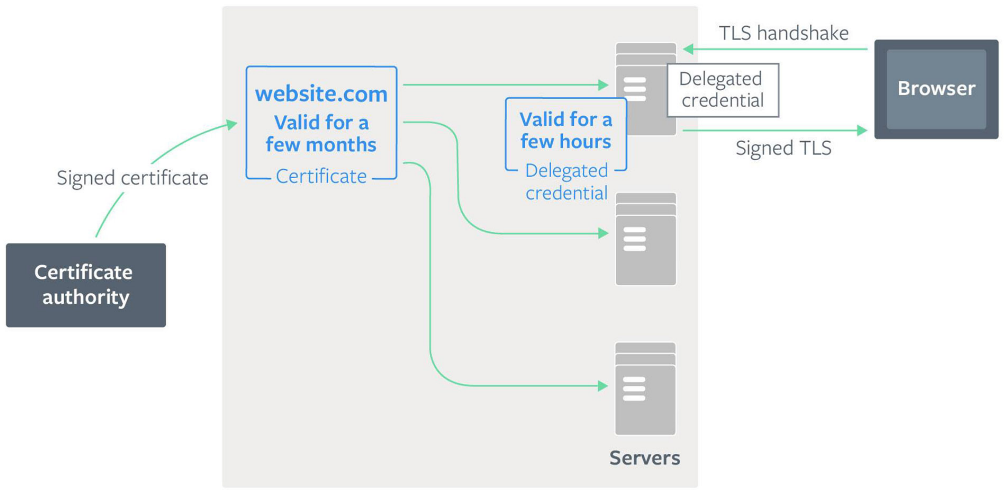


Figure 2.5: Concept of Delegated Credentials [GNI19].

2.3.3 Proxy Signature Schemes

Proxy signatures, introduced in 1996 by Mambo et al. [MUO96], produce an aggregated signature [BPW12], i.e., a single signature formed by the contribution of both, CDN (proxy) and origin server (original signer). In other words, it allows the delegation of signing rights to a proxy. The verifier can check whether the proxy signed the signature, and if it had the right to do so. We note a paring based version by Boneh et al. [Bon+03] known as BGLS and a Schnorr signature [Sch91] based version by Kim et al. [KPW97] named KPW. Generally, they have not seen much adoption in internet technologies such as TLS, although they are featuring useful properties.

2.3.4 Hierarchical Identity-Based Signatures

Similarly, hierarchical identity-based signatures (cf. Section 3.3.7) are also used for delegation purposes. In such signatures, an arbitrary number of identities can be included in the signature. Subsequently, the signature is verified by recalling the identities present in the signature and the public key. Boneh et al. [BF01] in 2001 sketch a transformation from identity-based encryption schemes to signatures attributing it to Moni Naor. Therefore this transformation is called Naor Transform in the literature (cf. Section 3.3.7). In this thesis, we chose such an identity-based signature approach to present a fully operational alternative to Full Delegation practices.

2.4 Nineteen Criteria Framework

In a second version of their paper, Chuat et al. [Chu+20] concentrated on the systematization of knowledge present in the literature, explaining different approaches solving pressing problems in web PKI, with particular focus on Full Delegation. They discuss the impact of resumption and revocation on alternatives and further analyzed combinations of different techniques from the delegation and revocation field. To do so, they introduce a nineteen-criteria framework characterizing the examined schemes. The combination they recommend is short-lived delegated credentials or proxy certificates with functional revocation (in particular PKISN [SCP16]). We will use the criteria framework to characterize our approach and use it for further discussion and comparison (see Section 5.3.2). Please find a description of every single criterion in [Chu+20, Chapter 8, A-F].

2.5 Delegation in Similar Contexts

Full Delegation is not a problem limited to the authentication of CDNs. Other similar contexts share the same issue. In the following section, we present some of these cases we encountered during literature research and are worth mentioning in the setting of the thesis.

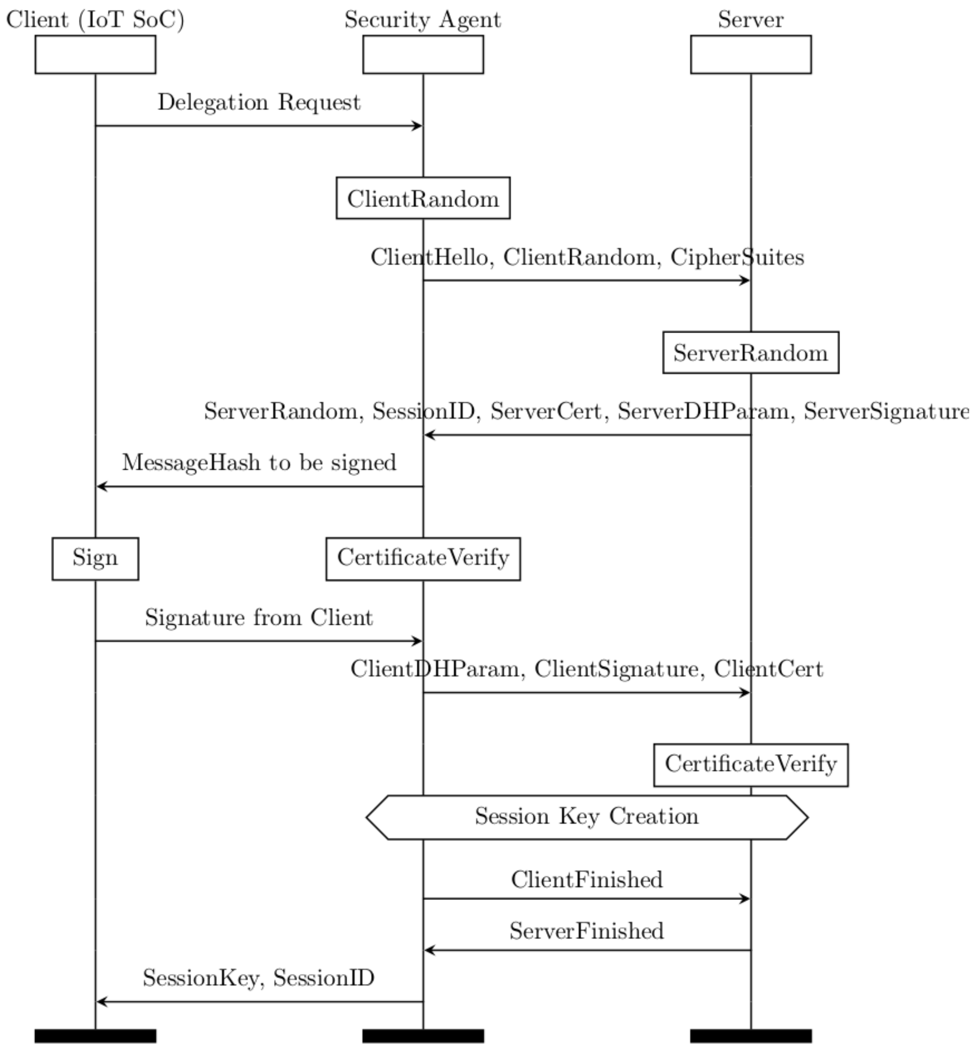


Figure 2.6: Sequence diagram of the D2TLS scheme when setting up a session [Cho+19].

2.5.1 D2TLS

Cho et al. [Cho+19] denounce in their work Full Delegation practices in the field of Internet of Things. As a remedy, they propose a technique similar to KeylessSSL (cf. Section 2.2.4 for datagram TLS (DTLS), they call D2TLS. A secure agent functioning as a proxy is used to set up a secure connection to the corresponding cloud service for a low-power IoT device, which fits the key server's role (see Figure 2.6). Like in KeylessSSL, the proxy (agent) does not own the private key of the device (since that would be Full Delegation). Instead, the agent asks the IoT device to sign the handshake's hash to establish the connection successfully. However, compared to KeylessSSL, in the aftermath, the agent sends the session key and id to the IoT device. The latter can then use session resumption for all further communication with its IoT-cloud service.

D2TLS requires changes to the Internet of Things (IoT) device software and a compatible agent. On the other hand, Cho et al. [Cho+19] showed in their paper that resource-limited devices could profit from the introduced encryption security-wise while incurring little computational overhead and evade Full Delegation practices.

2.5.2 STYX and Other SGX Approaches

We are also aware of approaches that require the presence of a trusted execution environment such as Intel SGX. Examples of such are Conclaves by Herwig et al. [HGL20] and STYX by Wei et al. [Wei+17]. However, since such trusted execution environments exceed the scope of the thesis, trying only to tackle Full Delegation in the context of TLS, and have a history full of serious attacks [Bul+18; Bul+20; Sch+19], we do not consider them further in this thesis.

2.5.3 Multi-Context TLS

Naylor et al. [Nay+15] try to solve confidentiality issues in TLS in the context of in-network middleboxes. Such middleboxes offer benefits like intrusion detection, caching, parental filters, content compression or transcoding, or compliance with corporate practices in enterprise environments. Naylor et al. [Nay+15] argue that the handshake should be modified so that the client and server can grant in-network middleboxes explicit read/write permissions. Contrary to CDNs, such proxying actors do not open two separate TLS connections but try to work on an existing TLS secured end-to-end-connection.

Prior approaches by Ericsson together with AT&T [Lor+14] and by Google [PR12] could not fulfill the requirements Naylor et al. [Nay+15] postulated. Therefore, they suggest Multi-Context TLS (mcTLS). It includes socalled Encryption Contexts which enable endpoints to limit a middlebox's read-/write access to a fine-grained portion.

An Encryption Context gets associated with a particular purpose. It consists of a set of symmetric keys. One for encryption (read access) and one for updating the message authentication code (write access). Naylor et al. [Nay+15] present a method, where both endpoints generate half of a key and send their half to the middlebox. Putting both halfs together, the middlebox obtains the full symmetric secret. Consequently, both parties have to agree and send their part of the key to allow access [Nay+15].

Some years later, Bhargavan et al. [Bha+18] conducted a formal analysis and suggested that the approach should be revised. According to them the original approach is susceptible to a group of attacks called "Middlebox Confusion". Further, they presented a formally provable alternative to mcTLS.

2.5.4 LIGHT^{est}

Wagner et al. [WOM17] discuss delegations in the context of e-government and electronic transaction. In more detail, when a proxy signer wants to finalize an electronic transaction in the name of an other entity, we need the means to verify such a claim of delegated signing rights in the corresponding trust management system. Unfortunately, they did not find any generic solutions in their field for solving the issue. Therefore, they suggested a delegation scheme similar to DeC to provide a generic approach easily applicable to different domains. Based on XML, it further offers the possibility to define generic constraints open to any application area.

3 Preliminaries

This chapter provides an overview of secure internet communication and cryptographic concepts useful to understand our approach in the thesis. In the first part (see Section 3.1), we discuss Full Delegation in general and describe a simplified example case. In the second part (see Section 3.2), we discuss traditional security measures on the internet, continue by generalizing different short-lived approaches, and finally explain CDNs. In the third part (see Section 3.3), we start explaining the basics of pairing-based cryptography. Then continue with identity-based encryption and path a way to understand the Time-Bound Identity-Based Signature (TBIBΣ) construction.

3.1 Full Delegation in General

Full Delegation is a bad practice not only present in the TLS context but generally in the authentication context. We define Full Delegation as follows: An entity E forwards a personal secret to another party P so that it can authenticate as E.

A trivial example can be stated in the context of keycard locking systems: Full Delegation would mean lending out one's keycard or even allowing copying it. Then, Entity E is at the mercy of a third party P. If the latter abuses the access privileges granted, deliberately or in negligence, E has no proper means to stop an impersonation. In fact, Full Delegation lets the owner lose sole control of the associated secret key. Consequently, the danger of key leakage increases, and the risk of malicious proxy behavior, without an easy chance to repudiate, is all-time present. The owner's certificate or credential itself needs to be revoked to counter such risks.

Good delegation practices, on the other hand, would mean giving an identifiable party well-restricted privileges. In the exemplary keycard context, it would mean that instead of lending the keycard, we would distribute an electronic delegation that somebody may gain access to a specific door (with the possibility to restrict this privilege further, e.g., temporally).

3.2 Network

This section describes and discusses aspects of networks such as the internet, that are essential for the comprehension of our approach described in this master thesis.

3.2.1 Public Key Infrastructure

Public Key Infrastructure (PKI) describes a system on a computer network that facilitates secure transfer of information. It uses public key cryptography to authenticate users and devices on this network. It does so by binding a public key to a certain entity using a certificate. The public key (or digital) certificate is an electronic document containing the subject's and the issuer's name, as well as the subject's public key. It is signed by the issuer's private key. In traditional (centralized) PKI (see Figure 3.1), there exist trusted parties that form a trust root. They are called Certificate Authoritys (CAs). Root certificates (or CA certificates) are self-signed (signed with their own private key), and software vendors distribute them to the clients. CAs then sign certificates that associate a public key to an entity. Besides, the CA can delegate some of its tasks to a Registration Authority (RA) validating registration requests or to a CRL issuer (or OCSP responder) distributing revocation status information in the name of the CA [Coo+08; Ken93].

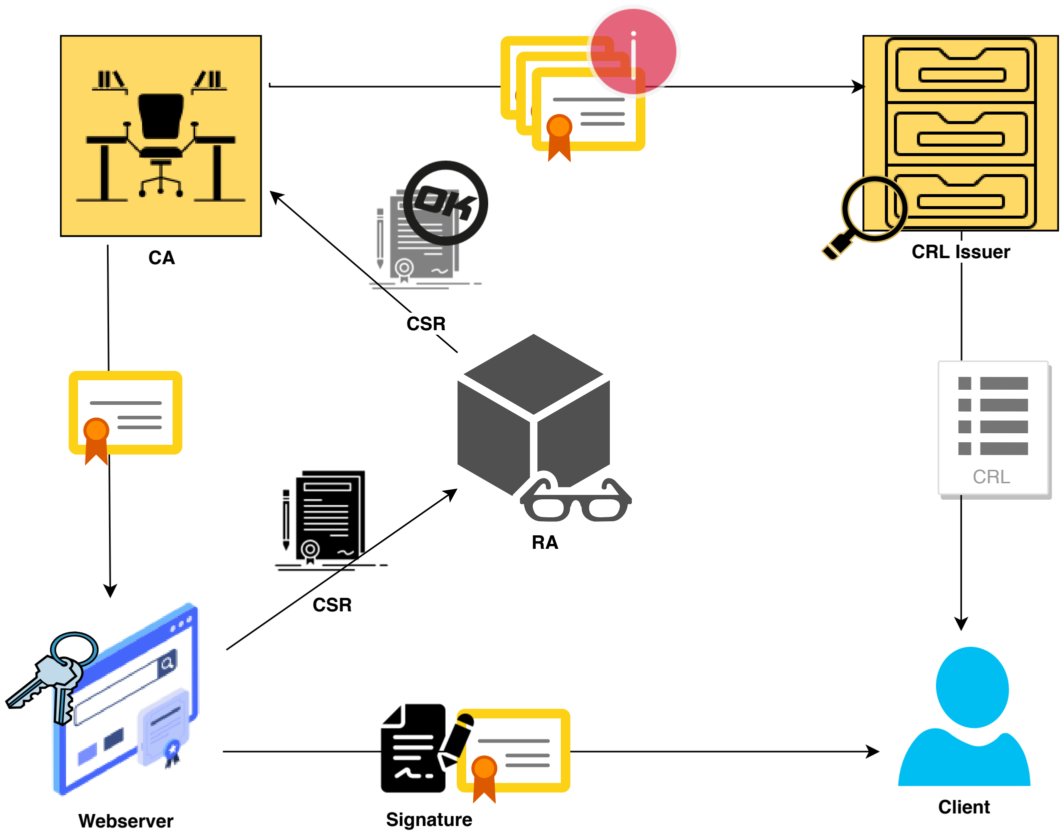


Figure 3.1: Setup of a traditional Public Key Infrastructure (PKI).

A CA can delegate to subordinate CAs and therefore constitute a hierarchy. On top of the hierarchy, there is the root CA, which is publicly accepted, while its children (subordinate CAs) are only trusted because they have a delegated mandate by the root CA to act as a CA themselves. This mandate is represented by a certificate with CA privileges enabled and gets called an intermediate certificate.

Thus, if a root CA has a subordinate CA and the subordinate CA certifies an entity's identity, the construction constitutes a certificate chain. If a client wants to verify a specific entity's authenticity, it can use chain traversal to validate the certificates along the chain until it reaches a root CA.

- Version Number
- Serial Number
- **Signature Algorithm ID**
- **Issuer Name**
- Validity period
  - Not Before
  - Not After
- **Subject name**
- **Subject Public Key Info**
  - **Public Key Algorithm**
  - **Subject Public Key**
- Issuer Unique Identifier (optional)
- Subject Unique Identifier (optional)
- Extensions
  - ...
- **Certificate Signature Algorithm**
- **Certificate Signature**

Figure 3.2: Full structure of an X.509 certificate. The most important parts for the thesis are marked in bold [Coo+08].

The approved standard for certificates serving internet protocols is the so-called X.509 standard. Besides the public key, it contains information about the representing identity, its issuer and its validity (see Listing 3.2 for the whole structure) [Coo+08].

An X.509 certificate can have several well-defined certificate extensions. Extensions extend the certificate's functionality. A specific extension is identified by an Object Identifier (OID) and can be marked "critical" (if extension is unknown, the verification must fail) or "non-critical" (if the extension is unknown, it gets ignored). In Section 1.1, we already encountered an extension used for "cruise-liner certificates":

Subject Alternative Name The X.509 extension Subject Alternative Name (SAN) allows a list of additional identities (e.g. a DNS name) besides the subject to be bound to the certificate. The certificate will validate for all of the identities present in the SAN list [Coo+08, p. 4.2.1.6.].

Besides, the X.509 standard defines not only a certificate structure, but also certificate revocation, i.e., revocation management and revocation information distribution [Coo+08].

3.2.2 Certificate Transparency

Certificate Transparency (CT) is a standard promoted by Google for monitoring and auditing web certificates. It is based on a system of public logs that record web certificates. CT was a response to CA compromises and illegitimate certificate issuance, even for high-profile domains [Hoo+12]. With CT, maliciously or mistakenly issued certificates can easily be discovered [LLK13]. In fact, everybody can submit a certificate or consult the log servers of CT. For efficient presence and consistency proofs, CT organizes certificates in Merkle hash trees. On submission of a certificate, the CT endpoint returns a Signed Certificate Timestamp. When a CT supporting client requests the certificate, the server also delivers the timestamp. The Signed Certificate Timestamp proofs to the client that CT approves the certificate.

There exist two actors: Monitors periodically check for suspicious certificates on the log servers. They are meant to be run by companies and organizations. Auditors (mostly browser clients) check if the log is consistent and query for certificates.

CT has become an integral part of the internet's ecosystem, and therefore it needs to be accounted for as we introduce our delegation approach for TLS [cer; Chu+20].

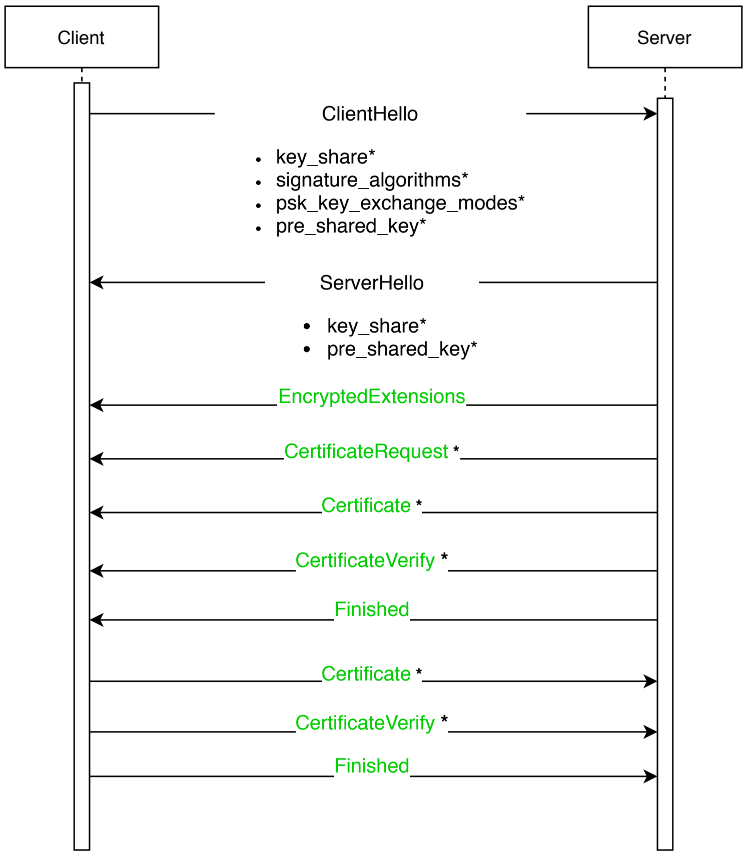


Figure 3.3: Handshake of a Transport Layer Security (TLS) 1.3 connection. Messages marked with * are optional or situation-dependent messages/extensions. Messages in green are encrypted by the secret resulting from key exchange.

3.2.3 Transport Layer Security

Transport Layer Security (TLS) is an internet protocol providing end-to-end security for connections on a public computer network. It ensures confidentiality, authentication, integrity, and non-repudiation of the data transmitted.

A great variety of applications use TLS, e.g., web browsing (HTTPS), email (SMTP), as well as voice, video, and messaging applications (SIP). It has been revised several times and is specified in its current version 1.3 since August 2018 (see Figure 3.3) [Res18].

TLS can be regarded as a two-layered protocol, consisting of the record and the handshake. While the former defines the structure of payloads transported by TLS, we concentrate more on the latter, which negotiates the setup, establishes the connection, and controls the transfer.

Further, TLS allows extensions to augment the protocol with additional functionality without modifying the protocol [Bla+03]. To mention some important: Server Name Indication (SNI), Supported Elliptic Curves Extension (for Elliptic Curve Cryptography for TLS 1.2 and earlier) [NJP18], and Session Tickets (discussed in Section 3.2.3). The first is vital to comprehend this thesis:

Server Name Indication Server Name Indication (SNI) is an extension for the TLS standard. It offers the possibility that TLS secured connections with different domains share the same port (i.e., 443) on the same server, even though it is only reachable by a single IP address. In other words, the client sends the virtual domain name as part of the TLS negotiation to the server, enabling the latter to select the right virtual domain, and thus presenting the client with the correct certificate [Bla+03].

A variation of the standard is ESNI, where the "E" stands for encrypted and which encrypts the hostname to be less vulnerable to domain eavesdropping (standard SNI reveals which site is requested). [GA18; Shb+15]

In the handshake, a key exchange using public key cryptography is used to establish symmetric keys. These keys are then used to encrypt the TLS connection. Since RSA does not offer forward secrecy (explained in Section 3.3.1) and for simplification of available scheme choices, in TLS 1.3, it was decided to exclusively use multiple ephemeral Diffie-Hellman options known to be secure (multiple to evade in future potentially discovered vulnerabilities) [Res18; Sul18].

Also in TLS 1.3, only AEAD[1] supporting cipher suites are allowed. Encryption does not prevent data tampering. Therefore, encryption and integrity are provided by a single operation in AEAD [Res18; Sul18].

[1]AEAD is an extension of AE (authenticated encryption) and stands for authenticated encryption with associated data. The idea is to provide confidentiality and message authentication with one single cryptoalgorithm [McG08].

In TLS 1.2 and prior, only part of the handshake was signed for authentication. That made it vulnerable since the negotiation of symmetric ciphers was authenticated by a symmetric Message Authentication Code (MAC) (CurveSwap [Val+18], LogJam [Adr+15], FREAK [Beu+15],etc.). It was fixed in version 1.3, as the whole transcript gets signed now [Res18; Sul18].

Session Resumption

A TLS connection uses shared key material, yielded by a key-exchange protocol between two parties, to secure the data transfer. Such a connection can stay alive as long as needed and times out after remaining unused for one to five minutes. On the other hand, a session is a collection of chained connections, where each can provide a session resumption mechanism. Therefore, if a connection has ended, the session can still be resumed without redoing the key-exchange and certificate retrieval. That helps to curb the extra latency and computational costs of a full TLS handshake [Sul17]. Different forms of session resumption exist: Session Identifiers, Session Tickets, and pre-shared keys (pre-shared key (PSK)) since TLS 1.3.

The first option works with a session ID assigned by the server on the first handshake. Both, client and server, store the ID with the negotiated session data, including the session's secret key. A client in possession of such an ID sends it on *ClientHello* to the server. Is the server willing to resume the session, it returns the same ID to the client, and they resume the session using the session data they stored. The requirement for this approach is a well-operated session cache on the server-side [Sy+18].

The second approach mentioned tries to address the limitations of Session Identifiers. On indication of support by the client, the server returns the negotiated session data encrypted with a security key as a Session Ticket record. Subsequently, the client sends the ticket as part of the handshake, when a new session needs to be established. Referred to as a stateless resumption mechanism, it removes the server's session cache simplifying deployment [Sal+08].

With TLS 1.3, both options got replaced by a concept of session resumption called pre-shared key (PSK). After the handshake, both derive a shared "resumption master secret". The client stores an opaque blob. It may either be a lookup key (ID) or an encrypted ticket (sent back on resumption).

Depending on ID or ticket option, both or only the client stores the PSK. On reconnection, both parties share a single-use resumption secret, and TLS 1.3's 0-RRT is possible [Sul18; Tau17].

It is recommended, but not binding for the relying party to cache the certificate chain along with the PSK material to lookup if it expired. A PSK material intended to stay valid for up to seven days. When a resumption is effected, a new PSK is provisioned, which again can be valid for seven days. Such chaining is allowed indefinite times [Res18].

Handshakes

In TLS 1.3, that we use for the implementation of our approach, we have two different handshakes:

Full Handshake Compared to its predecessor TLS 1.3 has a much simpler cryptographic negotiation. TLS 1.3 sends various key-share material already in the *ClientHello*. The server then can decide which key agreement material it prefers and calculates the shared secret accordingly. In return, it sends its shared key material together with shared secret encrypted data to the client, all happening already in the first round trip [Res18].

That is possible since the key-agreement options are few. All of them are Diffie-Hellman based (most likely ECDHE with X25519 or P-256). Therefore it is not too much overhead for the client to send several key material options at once [Res18].

That leaves us with only one round trip per TCP, TLS and HTTP handshake, plus one DNS request (which could be cached). Compared to its predecessor that is one round less, which directly affects the latency positively [Sul17].

Session Resumption When resuming a session TLS 1.3 eliminates the extra round trip for the handshake. Cloudflare argues that the round-trip elimination presents a big advantage to previous TLS versions since they observed a resumption rate of 40% for HTTPS connections, and measured an improvement of more than 30% on connection time [Sul17].

Branded 0-RRT (zero round trip time), it allows for sending session key encrypted data on the first trip. That is possible since the PSK blob is also sent on the first trip. The blob consists either of a database key for the server or the server encrypted PSK key (see Section 3.2.3) [Res18]. However, there is also a catch to this: it is vulnerable to replay attacks. Nonetheless, Cloudflare calls this slightly weaker security assumption "managable". They curb the problem with a nonce in an extra header and claim that one should "use application-layer mechanisms to prevent replay requests" [Sul17].

3.2.4 Revocation

Part of the Public Key Infrastructure (PKI) ecosystem is revocation. After a TLS certificate is signed and distributed by the CA, the verifier authenticates a subject using the certificate and a signature that has been created with the corresponding private key before the certificate's expiration. However, sometimes the certificate needs to be invalidated before expiration, often because of key material compromise. So, revocation names the process of invalidating a certificate before its natural expiration.

When a certificate is revoked, end-user internet software, such as browsers, should be notified about the invalidation. The approaches to do so are manifold. We listed the most important ones for this thesis:

CRL The Certificate Revocation List is defined in RFC 5280 [Coo+08] together with the X.509 certificate. It is a list of certificates revoked before they expire at the end of their validity period.

Such a list is signed by the CRL issuer (often the CA itself). Further, it has specific longevity, such as 24 hours, after which the client updates to the actual list. However, CRLs can get pretty big (delta CRLs help if supported), give no assurance that certificates are signed by a legitimate CA, and have a timeliness issue (client only updates periodically to actual list) [Coo+08; Lia+14].

OCSP Online Certificate Status Protocol is a lightweight alternative to CRL. A responder provides real-time status checks even on facing plentiful requests while reducing bandwidth burdens (fewer data transported). On the other hand, Online Certificate Status Protocol (OCSP) discloses the requested domain to responders and requires no encryption (other parties can intercept the data, if not OCSP over HTTPS is supported) [Lia+14; San+13].

OCSP stapling Both endpoints need to implement OCSP stapling, and the server-side sends the OCSP response as part of the protocol, instead of having the clients to retrieve the response at a responder. Parties relying on OCSP can avoid opening another connection to a responder that would slow them down. Further, it improves privacy, since the domain is no longer disclosed to the responder or any other third party [III11; Pet13].

3.2.5 The Short-Lived Paradigm

The term short-lived in the literature on authentication mechanisms often gets used to describe a mandate for privileges granted only for a short period. That has many advantages, such as eliminating revocation leading to improved privacy (visited websites stay undisclosed), and fewer maintenance efforts. Concepts on which this paradigm gets applied are certificates, credentials, and delegated keys:

Short-lived certificate These are certificates with a short validity period and prominently represented by Short-Term Automatically Renewed (STAR) certificates, which are further described in the subsection below [She+20b]. In comparison to other short-lived concepts, short-lived certificates are highly versatile thanks to possible extensions. Therefore, they can specify granted privileges in a fine-grained way.

Short-lived credential Delegated Credentials (DeC) (see Section 2.3.2) makes part of this category. It is a data blob specifying explicit competences and other properties singed with the issuer's private key [Bar+20].

Short-lived delegated key A delegated key is a private key derived by one or multiple identities. Those identities are part of the delegated key and are also inherited by a resulting signature. The identities can be used to specify properties that need to be fulfilled at verification time (cf. Section 3.3.4). We use those properties in our approach for restricting validity to a short period.

STAR and ACME

Short-Term Automatically Renewed (STAR) certificates make part of the Automated Certificate Management Environment (ACME). The latter is a protocol meant to automate verification, certificate issuance, and other certificate management duties of CAs. Prominently used by Let's Encrypt [Encb], a nonprofit CA, it is a widely used standard on the internet (alone Let's Encrypt provides TLS certificates for 225 million websites [Enca]). As a response to the unreliable revocation process, Sheffer et al. [She+20b] propose an alternative to revocation compatible to ACME. Their approach issues periodically new short-lived certificates. Therefore, the certificate forms a sequence that can be terminated at any point. To ensure the reoccurring task of renewing the certificates is done without human effort and flaws, it is meant to be integrated into the Automated Certificate Management Environment (ACME) [Bar+19]. Advantages of STAR are: No records of issued certificates are needed resulting in a simpler CA. Attack surface and complexity is minimized by not using always-on response points for revocation data distribution.

Timeliness Issue

The short-lived paradigm renounces to a security feature which prior was important to Public Key Infrastructure (PKI), certificate revocation. On compromise of the private key, revocation is meant to limit the damage a rogue node or similar adversary can inflict. In short-lived paradigm based concepts, the lack of revocation is curbed by expiration. A significant amount of time can pass from compromise over detection to the endpoints awareness of an issued revocation. This time-gap constitutes the timeliness issue. On the other hand, short-lived concepts have a timeliness issue too: They use expiration instead of revocation. Thus, the certificate can not be invalidated before expiration. However, we can argue that with a short enough validity period, the timeliness issue is smaller with short-lived concepts than with revocation [Nir+18].

3.2.6 Content Delivery Network

With the evergrowing popularity of web applications over the last decades, performance, scalability, and security demands increased exponentially. So-called CDNs try to answer those by placing surrogate servers on convenient internet spots to be closer to clients. Their main task is the mirroring of content from the origin server. However, over time different providers started to offer a variety of services overlapping with cloud services. Such additional services can be hosting expensive computing tasks, enhanced analytics, or improved web security (e.g., DDoS mitigation), besides the by default improved load times, reduced bandwidth costs, and enhanced availability and redundancy [Clob; KWZ01; VP03].

When performance-critical data needs to be transferred (e.g., streaming movies), CDNs are present in many subnetworks of the internet to deliver increased performance in terms of throughput and latency. E.g., is the content owner situated network-based distant from the client, it may negatively impact the data throughput because it involves significant rerouting effort. To speed up transfer rates, content owners may contract CDNs, which cache highly requested content at each operated Point of Presence (PoP)[2]all around the world. A Content Delivery Network (CDN) provider then mostly uses the DNS operating principle to direct the request to the PoP in network-based proximity to the requesting client [Bae+97; BC95; Cloa].

In more details, deploying a CDN for a web application works the following: Often, when data is deemed worthy of being cached by a CDN, domain owners outsource it to a specific subdomain (e.g., cdn.iaik.at). The latter is especially true for static data because it can be cached easily. More advanced CDN providers such as Cloudflare offer optional computing services for dynamic content and customizations on how specific content is cached (depending also on factors such as region, and more) [Cloc; Dil+02].

When a client requests content, the CDN tries to find the cached content in its storage. If a cache miss happens (i.e., the data is not present at the PoP), the CDN fetches the data from an origin server and fills its cache for

[2]Network-wise location where the provider is present with a datacenter.

future requests. When the origin server sends the content to the server for caching, it attaches information (e.g., to the HTTP Header) on the caching's longevity, known as Time-To-Live (TTL). If no client requests the content within this period, it gets dropped. On the other hand, the origin server can also request a manual purge of the content [Bae+97; Cloc; Dil+02].

Further, CDNs make sure that requests resolve to the network-based nearest (available) PoP. Unaware of establishing its connection to another than the origin server, the client sees a significant improvement in overall performance [Dil+02].

There exist several possibilities to redirect HTTP requests to the CDN's PoPs [Chu+20]:

Authoritative The CDN can take full control over the DNS resolution if its name server is declared authoritative. In such a case, the client will resolve to an IP of a certain PoP, which needs access to the private key to establish a connection authenticated by certificate of the corresponding domain name [Chu+20; Dil+02].

CNAME The DNS Canonical Name record allows a specific (sub-)domain redirection. So, e.g., for the subdomain *cdn.iaik.at*, a CNAME could redirect the resolution to *cdn.iaik.cdn.net*. Still, the client expects an authentication with a valid certificate for *cdn.iaik.at*. Nevertheless, it allows for fine-grained mapping [Chu+20].

URL rewriting Here, URLs for specific resources are modified to point to the CDN servers. The by-resource fine-grained approach can be advantageous since the CDN's certificate is accepted. However, other features such as a web application firewall and DDoS protection is not possible [Chu+20; KWZ01].

We concentrate on the authoritative redirection approach using DNS in this thesis. Further, we differentiate CDNs from terms used synonymously in literature:

Edge Server It is a similar concept and, consequently, profits strongly from the commitment of CDN service providers. It involves getting closer to the edge of the internet with computing and content resources. The internet's edge stands for proximity to devices and information sources. Like in CDNs, an increase of bandwidth and a reduction of latency are pursued. Since the Internet of Things (IoT) communication traffic demands grow [SM19], high pressure is posed on existing infrastructure. Edge computing should relieve some of the pressure by moving computation closer to the sensors and control loops. E.g., driverless vehicles, coping with lots of sensor data, can send data to low latency 5G base-stations representing the edge of the internet. Therefore, in such a case, Edge Computing would imply the processing of sensor data directly at the base stations, relieving network load [Woo+17].

Middleboxes Sometimes it is not possible placing all functionality at the endpoints, but the need to augment the session along its communication path arises. Such functionality can include intrusion detection, parental filtering, content optimization (e.g., compression, transcoding), or compliance with corporate practices. Generally, middleboxes are bad news for privacy since they allow third parties to peek into data or even modify it during transport. However, with TLS end-to-end secured connections becoming standard, sustaining middlebox functionality has become difficult. [Nay+15].

Of course, one can argue that the mentioned services should be implemented at corresponding endpoints rather than in between. However, that often leads to a suboptimal solution. E.g., an ISP cache for thousands of users is much more effective at the proxy-side than at each endpoint. Another example is intrusion detection or content-based routing that fundamentally needs network-wide visibility [Nay+15].

In Section 2.5.3 (Multi-Context TLS), we summarize an approach by Naylor et al. [Nay+15] trying to solve the problem of incorporating middleboxes into networks despite TLS connections yet respecting fine-grained permissions.

Note that CDN caching servers in literature are often referred to as edge servers, or proxy servers but should not be confused with terms outside of the CDN's literature field. In this thesis, we will always call them CDN server or Point of Presence (PoP).

3.3 Cryptography

In this section, we discuss cryptographic primitives and constructions essential to the thesis.

3.3.1 Forward and Backward Security

We distinguish between forward and backward: The former indicates that secured data in the past will not be compromised in the future by the compromise of key material. On the other hand, the latter indicates the vice versa security assumption: Data secured in the future, will not be compromised by any compromise of key material of the past [BH05; BM99; Gün89].

3.3.2 Basic Complexity Assumptions

Given a multiplicative, finite cyclic group $\mathbb{G}$ of order q and a generator $1 \neq g \in \mathbb{G}$, we assume following problems to be hard in a cryptographic scheme:

The Discrete Logarithm Assumption It is computationally hard to find x given the generator g and a value g^x. That is why lots of algorithms in cryptography base their complexity assumptions of the discrete-log problem [Kat10, Section 2.2.3].

The Computational Diffie-Hellman Assumption Given the generator g and group elements g^x, g^y, Computational Diffie-Hellman (CDH) states that it is hard to calculate g^{xy} [Kat10, Section 5.3]:

$$g, g^x, g^y \Rightarrow g^{xy} \tag{3.1}$$

The Decisional Diffie-Hellman Assumption Given the generator g and some group elements g^x and g^y (whereas x, y are some randomly chosen integers), Decisional Diffie-Hellman (DDH) states that it is hard to distinguish g^{xy} from the other group elements [Kat10, Section 5.3]:

$$g, g^x, g^y, g^z \Rightarrow \begin{cases} 0 & \text{if } z = xy \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

Note: In some groups, there is no efficient way to verify a valid solution to CDH. Distinguishing between a correct and an incorrect solution boils down to DDH. However, efficiently computable bilinear maps (like we will use) cannot hold DDH, since a solution is efficiently verifiable [Kat10, Section 5.3].

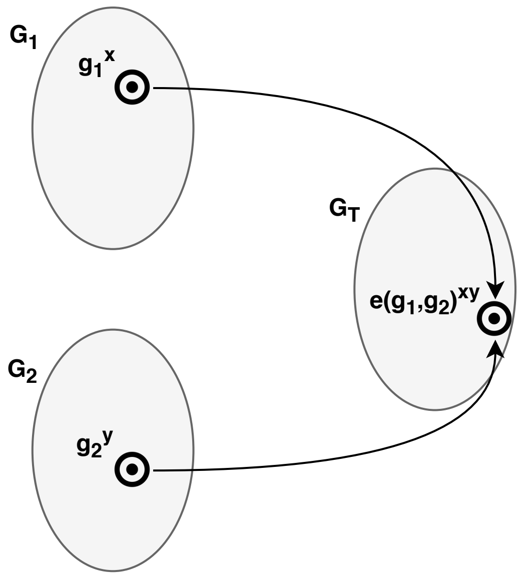


Figure 3.4: Pairing scheme visualized: shows mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ [Bon].

3.3.3 Bilinear Pairing

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ represent two multiplicative, finite cyclic groups of prime order p. Furthermore let $\mathbb{G}_T$ be a multiplicative, finite cyclic group of order p. Then a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ (see Figure 3.4) for pairing is defined by 3 properties [BF01; GPS08; Kat10]:

Bilinearity It needs to hold [BF01, Sec. 2]:

$$\forall x, y \in \mathbb{Z},\ \forall g_1 \in \mathbb{G}_1,\ \forall g_2 \in \mathbb{G}_2 :\ e(g_1^x, g_2^y) = e(g_1, g_2)^{xy} = e(g_1^y, g_2^x) \tag{3.3}$$

Non-degeneracy If $g_1 \neq 1$ and $g_2 \neq 1$, then also $e(g_1, g_2) \neq 1$.
In other words, if one pairs a generator g with itself, it should yield a valid element $e(g, g)$ in the target group G_T; otherwise, it is not a useful bilinear mapping [Bon].

Computability It has to exist an algorithm that efficiently computes the mapping e in polynomial time.

As a consequence of the pairing's properties, we observe an easy Decisional Diffie-Hellman problem in $\mathbb{G}$. Given $g, g^x, g^y, g^z \in \mathbb{G}$, we can easily test if $z = x \cdot y$ [Bon]:

$$e(g, g^z) \stackrel{?}{=} e(g^x, g^y) \tag{3.4}$$

Therefore, we have to consider a Bilinear Decision Diffie-Hellman complexity assumption. Given $h, g, g^x, g^y, g^z, e(h, g)^z \in \mathbb{G}$, it is computationally hard again to find out if $z = x \cdot y$[Bon].

Different bilinear pairing froms can be classified into different basic types of pairings [Bon; GPS08]:

Type 1 $\mathbb{G}_1 = \mathbb{G}_2$, symmetric using supersingular curves.
Type 2 $\mathbb{G}_1 \neq \mathbb{G}_2$, but a computable trace map exists ($\phi : \mathbb{G}_2 \to \mathbb{G}_1$).
Type 3 $\mathbb{G}_1 \neq \mathbb{G}_2$, with no efficient way to compute a homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. It also gets called asymmetric pairing.

For all three types, there exists a non-trivial homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$, since they are all cyclic groups of the same order. However, the non-trivial homomorphism can be computation-wise as hard as the discrete logarithm. Type 2 is defined to have only a computable, non-trivial homomorphism $\mathbb{G}_2$ to $\mathbb{G}_1$. In general, homomorphisms are useful to prove the security of many pairing-based primitives. Since type 3 does not have trivial homomorphisms, primitives have been hard to prove. However, in 2016, Abe et al. [AHO16] presented an efficient conversion method into the asymmetric setting, making security proving easier [GPS08].

Pairing Elliptic Curves was introduced by Weil [Wei40] in 1940 and was formative for Elliptic Curve theory [Sil86]. In a second paper, Miller et al.

[Mil+86] presented an efficient algorithm for pairing calculations. Subsequently, an approach was discovered that reduced the elliptic curve discrete logarithm problem to the multiplicative group discrete logarithm problem in finite fields [MVO91].

When using Weil pairings in elliptic curve cryptography, the basic idea is to solve a problem, impossible in an elliptic curve group, by equating in the target group, making use of the unique bilinear mapping properties [BF01].

3.3.4 Identity-Based Encryption and Signature

Identity-based cryptography compared to traditional public-key cryptography has the apparent difference to use an arbitrary string as its public key. While traditional public-key cryptography generates a key-pair for the following operations, identity-based cryptography uses a Private Key Generator to extract a private key from a given public-key identity and a previously setup master secret.

Identity The identity consists of a string or a byte array and serves as a public identifier for a specific subject.

Shamir [Sha84] introduced first notions for Identity-Based encryption (IBE) and Identity-Based signature (IBS) in 1984. While for IBS researchers found satisfactory solutions in the same century [FS86; SP88], IBE received several proposals all having major downsides [CS98; MY91; Tan87; TI89]. However, in 2001 Boneh et al. [BF01] introduced a first fully functional version of IBE derived from Weil Pairings. In 2004 a construction without Random Oracles was presented [BB04].

We note that we use multi-instance identity-based cryptography in this thesis, i.e., we assume multiple possible setups of identity-based public parameters instead of a single, global instance as often presumed in identity-based schemes. Therefore, we reintroduce a Key Pair Generator instead of a Private Key Generator. Consequently, a public key exists separately from the public parameters. Two distinct algorithms yield the public parameters and the public key, Setup respectively Gen. That allows us to equip the server's certificate with a typical public key.

3.3.5 Hierarchical Identity-Based Encryption

HIBE generalizes IBE enabling to delegate multiple, fine-grained rights for decryption by generating specific private keys corresponding to the rights represented as identities. The encryption rights belong to different identity groups residing in several hierarchical levels. In other words, a master secret key gets derived several times by different identities (rights) depending on the number of levels present. A private key derived for multiple levels by specific identities can not decrypt a message encrypted on the same level using different identities. A different level of delegations also impedes decryption. Data encrypted with two levels of IDs can not be decrypted by private key with three levels, although the first two IDs would match.

In 2002 Gentry et al. [GS02] presented a first hierarchical identity-based scheme with random oracle. In 2005 Boneh et al. [BBG05] presented a Hierarchical Identity-Based Encryption (HIBE) scheme without random oracles but with constant ciphertext and depth-independent decryption costs. They based the security of their system on the BDH Inversion assumption [BBG05, p. 2.3]. Generally, HIBE implies forward secure encryption as shown by Canetti et al. [CHK03].

To show a HIBE, we assume the following pairing setup: Let BGen be an algorithm that, on input of a security parameter 1^κ, outputs $\mathsf{B}G = (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \hat{g}) \leftarrow \mathsf{BGen}(1^\kappa)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p with bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators $g, \hat{g}$ of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

Below (Definition 3.1) we show a HIBE following Boneh et al. [BBG05]:

Definition 3.1 (Hierarchical Identity-Based Encryption)**.** An Hierarchical Identity-Based Encryption (HIBE) scheme with message space $\mathcal{M}$ and identity space $\mathcal{ID}^{\leq \ell}$, for some $\ell \in \mathbb{N}$, consists of the probabilistic, polynomial-time (PPT) algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Del}, \mathsf{Enc}, \mathsf{Dec})$:

$\mathsf{Setup}(1^\kappa, \ell)$: On input of security parameter κ and hierarchy parameter ℓ, outputs public parameters pp.

$\mathsf{Gen}(\mathsf{pp})$: On input of public parameters pp, outputs a keypair $(\mathsf{pk}, \mathsf{sk}_\varepsilon)$.

$\mathsf{Del}(\mathsf{sk}_{id'}, id)$: On input secret key $\mathsf{sk}_{id'}$ and $id \in \mathcal{ID}^{\leq \ell}$, outputs a secret key sk_{id} for id iff id' is a prefix of id, otherwise $\mathsf{sk}_{id'}$.

$\mathsf{Enc}(\mathsf{pk}, id, m)$: On input of a public key pk, a message $m \in \mathcal{M}$ and an identity $id \in \mathcal{ID}^{\leq \ell}$, outputs a ciphertext c_{id} for id.

$\mathsf{Dec}(\mathsf{sk}_{id'}, c_{id})$: On input of a secret key $\mathsf{sk}_{id'}$ and a ciphertext c_{id}, outputs $m \in \mathcal{M} \cup \{\perp\}$.

For the scheme to be correct, it is required that every derived secret key delegated for specific identities enables to decrypt a ciphertext encrypted with precisely the same identities.

More formally, we require for all $\kappa, \ell \in \mathbb{N}$, all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, \ell)$, all $(\mathsf{pk}, \mathsf{sk}_\varepsilon) \leftarrow \mathsf{Gen}(\mathsf{pp})$, all $m \in \mathcal{M}$, all $id, id' \in \mathcal{ID}^{\leq \ell}$ where id' is a prefix of id, all $\mathsf{sk}_{id} \leftarrow \mathsf{Del}(\mathsf{sk}_{id'}, id)$, all $c_{id} \leftarrow \mathsf{Enc}(\mathsf{pk}, id, m)$, $\mathsf{Dec}(\mathsf{sk}_{id}, c_{id}) = m$ holds [BBG05].

We provide the experiment with following oracles:

$\mathsf{Del}^1(\mathsf{sk}_\varepsilon, id)$: Stores id in $\mathcal{Q}$ and returns $\mathsf{Del}(\mathsf{sk}_\varepsilon, id)$.

$\mathsf{Del}^2(\mathsf{sk}_\varepsilon, id)$: This oracle checks whether id is a prefix of id^* and returns $\perp$ if so. Otherwise it returns $\mathsf{Del}(\mathsf{sk}_\varepsilon, id)$.

```
Experiment $\mathsf{Exp}^{\text{hibe-ind-cpa}}_{\mathsf{HIBE},A}(\kappa, \ell)$
    $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, \ell)$, $(\mathsf{pk}, \mathsf{sk}_\varepsilon) \leftarrow \mathsf{Gen}(\mathsf{pp})$
    $b \xleftarrow{\$} \{0,1\}$
    $(id^*, m_0, m_1, \mathsf{st}) \leftarrow A^{\mathsf{Del}^1(\mathsf{sk}_\varepsilon, \cdot)}(\mathsf{pk})$
    if $\exists id \in \mathcal{Q}$ such that $id$ is a prefix of $id^*$, return 0
    if $m_0, m_1 \notin \mathcal{M}$ or $|M_0| \neq |M_1|$, return 0
    $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, id_*, m_b)$
    $b^* \leftarrow A^{\mathsf{Del}^2(\mathsf{sk}_\varepsilon, \cdot)}(\mathsf{st}, c^*)$
    if $b = b^*$ return then 1, else return 0
```

Experiment 3.1: The HIBE-IND-CPA experiment for a HIBE scheme [BBG05].

Definition 3.2. For any PPT adversary A, we define the advantage in the HIBE-IND-CPA experiment $\mathsf{Exp}^{\text{hibe-ind-cpa}}_{\mathsf{HIBE},A}$ (cf. Experiment 3.1) as

$$\mathbf{Adv}^{\text{hibe-ind-cpa}}_{\mathsf{HIBE},A}(\kappa, \ell) := \left| \Pr\left[\mathsf{Exp}^{\text{hibe-ind-cpa}}_{\mathsf{HIBE},A}(\kappa, \ell) = 1 \right] - \frac{1}{2} \right| \tag{3.5}$$

for an integer $\ell \in \mathbb{N}$. A HIBE is HIBE-IND-CPA-secure, if $\mathbf{Adv}^{\mathsf{hibe\text{-}ind\text{-}cpa}}_{\mathsf{HIBE},A}(\kappa, \ell)$ is a negligible function in κ for all PPT adversaries A [BBG05].

Subsequently, we recall Boneh-Boyen-Goh (BBG) [BBG05]. It explicitly uses a Setup algorithm to generate the public parameters (see Scheme 3.1) to separate it from Gen to support multiple instances. Further, we selected asymmetric bilinear groups (type 3).

$\underline{\mathsf{Setup}(1^\kappa, \ell)}$: Generate a bilinear group $(p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \hat{g}) \leftarrow \mathsf{BGen}(1^\kappa)$ and $g_2, g_3, h_1, \dots, h_\ell \xleftarrow{\$} \mathbb{G}_1$, fix a hash function $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$ and return $\mathsf{pp} \leftarrow (H, p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \hat{g}, g_2, g_3, h_1, \dots, h_\ell)$.

$\underline{\mathsf{Gen}(\mathsf{pp})}$: Choose $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and return $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\hat{g}^\alpha, g_2^\alpha)$.

$\underline{\mathsf{Del}(\mathsf{sk}_{id}, id)}$: Parse id as $(I_1, \dots, I_k)$ with $k < \ell$,

- If sk_{id} is the master secret g_2^α, sample $v \xleftarrow{\$} \mathbb{Z}_p$ and return $(g_2^\alpha \cdot (h_1^{H(I_1)} \cdots h_k^{H(I_k)} \cdot g_3)^v, \hat{g}^v, h_{k+1}^v, \dots, h_\ell^v)$,
- else assume that sk_{id} is $(g_2^\alpha \cdot (h_1^{H(I_1)} \cdots h_{k-1}^{H(I_{k-1})} \cdot g_3)^{v'}, \hat{g}^{v'}, h_k^{v'}, \dots, h_\ell^{v'}) = (a_0, a_1, b_k, \dots, b_\ell)$, sample $w \xleftarrow{\$} \mathbb{Z}_p$ and output $(a_0 \cdot b_k^{H(I_k)} \cdot (h_1^{H(I_1)} \cdots h_k^{H(I_k)} \cdot g_3)^w, a_1 \cdot \hat{g}^w, b_{k+1} \cdot h_{k+1}^w, \dots, b_\ell \cdot h_\ell^w)$.

$\underline{\mathsf{Enc}(\mathsf{pk}, id, m)}$: Parse id as $(I_1, \dots, I_k) \in (\mathbb{Z}_p^*)^k$ with $k < \ell$, compute $id' \leftarrow (H(I_1), \dots, H(I_k)) \in (\mathbb{Z}_p^*)^k$, and:

- Sample $s \xleftarrow{\$} \mathbb{Z}_p$ and compute $(C_1, C_2, C_3) \leftarrow (e(g_2, \mathsf{pk})^s \cdot m, \hat{g}^s, (h_1^{id'[1]} \cdot \ldots \cdot h_k^{id'[k]} \cdot g_3)^s)$.
- Return (C_1, C_2, C_3).

$\underline{\mathsf{Dec}(\mathsf{sk}_{id}, c_{id})}$: Consider $id = (I_1, \dots, I_k)$ with $k < \ell$, parse sk_{id} as $(a_0, a_1, b_{k+1}, \dots, b_\ell)$, c_{id} as (C_1, C_2, C_3). Return $m \leftarrow C_1 \cdot e(C_3, a_1) \cdot e(a_0, C_2)^{-1}$.

Scheme 3.1: HIBE-IND-CPA-secure version of the BBG HIBE [BBG05].

3.3.6 Signature Schemes

Since in this thesis, we require a signature scheme for our goal, we swiftly recall the definition of traditional signature schemes [Kat10].

Definition 3.3. A signature scheme Σ consists of the probabilistic, polynomial-time (PPT) algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, which are defined as follows [Kat10]:

$\mathsf{Setup}(1^\kappa)$: On input security parameter κ, outputs public parameters pp.
$\mathsf{Gen}(\mathsf{pp})$: On input public parameters pp, outputs a signing key sk and a verification key pk with associated message space $\mathcal{M}$.
$\mathsf{Sign}(\mathsf{sk}, m)$: On input, a secret key sk and a message $m \in \mathcal{M}$, outputs a signature σ.
$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: On input a public key pk, a message $m \in \mathcal{M}$ and a signature σ, outputs a bit $b \in \{0, 1\}$.

At this point we remark that it is common to generate multiple, independent public keys with the same public parameters pp. In our case we reuse parameters. We need to separate the key-pair generation from the generation of the public parameters. Therefore, we define an algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$, whose output is then passed to Gen.

It is assumed that a signature scheme satisfies the usual (perfect) correctness notion [Kat10], i.e., for all security parameters $\kappa \in \mathbb{N}$, for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$, for all $m \in \mathcal{M}$, we have that

$$\Pr[\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1.$$

The standard *existential unforgeability under adaptively chosen message attacks* (EUF-CMA security) notion is recalled beneath. It states that an adversary who has access to a signing oracle is still unable to forge signatures on unqueried messages [Kat10].

Definition 3.4 (EUF-CMA)**.** For any PPT adversary A, we define the advantage in the EUF-CMA experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}$ (cf. Experiment 3.2) as

$$\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}(\kappa) := \Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}(\kappa) = 1\right]. \tag{3.6}$$

A signature scheme Σ is EUF-CMA-secure, if $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}(\kappa)$ is a negligible function in κ for all PPT adversaries A [Kat10].

We provide the experiment with the following oracle:

$\mathsf{Sign}'(\mathsf{sk}, m)$: This oracle computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, adds m to $\mathcal{Q}$, and returns σ.

```
Experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}(\kappa)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$
  $(m^*, \sigma^*) \leftarrow A^{\mathsf{Sign}'(\mathsf{sk},\cdot)}(\mathsf{pk})$
  if $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 0$, return 0
  if $m^* \in \mathcal{Q}$, return 0
  return 1
```

Experiment 3.2: The EUF-CMA experiment for a signature scheme Σ [Kat10].

3.3.7 Hierarchical Identity-Based Signature

Similarly to HIBE (cf. Section 3.3.5), we consider the (hierarchical) identity-based signature scheme by Gentry et al. [GS02]. It is based on a standard notion of signature schemes (cf. Section 3.3.6). However, an additional key delegation algorithm Del similar to HIBE (cf. Scheme 3.1) will be added. The Verify algorithm takes the identities as additional arguments. It only succeeds if the key used for the signature has been delegated for those particular identities. We briefly recall the formal definition by Gentry et al. [GS02] adapted to a multi-instance setting.

Definition 3.5. A hierarchical identity-based signature scheme (HIBS) consists of PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Del}, \mathsf{Sign}, \mathsf{Verify})$ such that [GS02]:

$\mathsf{Setup}(1^\kappa, \ell)$: On input of security parameter κ and hierarchy parameter ℓ, outputs public parameters pp.

$\mathsf{Gen}(\mathsf{pp})$: On input of public parameters pp, outputs a master signing key sk_ε and a verification key pk with message space $\mathcal{M}$.

$\mathsf{Del}(\mathsf{sk}_{id'}, id)$: On input of secret key $\mathsf{sk}_{id'}$ and $id \in \mathcal{ID}^{\leq \ell}$, outputs a secret key sk_{id} for id iff id' is a prefix of id, otherwise $\mathsf{sk}_{id'}$.

$\mathsf{Sign}(\mathsf{sk}_{id}, m)$: On input of a secret key sk_{id} and a message $m \in \mathcal{M}$, outputs a signature σ.

$\mathsf{Verify}(\mathsf{pk}, id, m, \sigma)$: On input of a public key pk, an identity $id \in \mathcal{ID}^{\leq \ell}$, a message $m \in \mathcal{M}$ and a signature σ, outputs a bit $b \in \{0, 1\}$.

The corresponding *standard existential unforgeability under adaptively chosen message attacks* (EUF-CMA security) notion we recall beneath:

We provide the experiment with following oracles:

$\mathsf{Del}'(\mathsf{sk}_\varepsilon, id)$: Stores id in $\mathcal{Q}^{\mathcal{ID}}$ and returns $\mathsf{Del}(\mathsf{sk}_\varepsilon, id)$.
$\mathsf{Sign}'(\mathsf{sk}, id, m)$: This oracle computes $\mathsf{sk}_{id} \leftarrow \mathsf{Del}(\mathsf{sk}_\varepsilon, id)$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{id}, m)$, adds m to $\mathcal{Q}$, and returns σ.

```
Experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathsf{HIBS},A}(\kappa, \ell)$
    $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, \ell)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$
    $(m^*, id^*, \sigma^*) \leftarrow A^{\mathsf{Del}'(\mathsf{sk}_\varepsilon, \cdot), \mathsf{Sign}'(\mathsf{sk}_\varepsilon, \cdot, \cdot)}(\mathsf{pk})$
    if $\mathsf{Verify}(\mathsf{pk}, id^*, m^*, \sigma^*) = 0$, return 0
    if $m^* \in \mathcal{Q}$, return 0
    if $\exists id \in \mathcal{Q}^{\mathcal{ID}}$ such that $id$ is a prefix of $id^*$, return 0
    return 1
```

Experiment 3.3: The EUF-CMA experiment for a hierarchical identity-base signature scheme HIBS [GS02].

Definition 3.6 (EUF-CMA)**.** For any PPT adversary A, we define the advantage in the EUF-CMA experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathsf{HIBS},A}$ (cf. Experiment 3.3) as

$$\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HIBS},A}(\kappa) := \Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathsf{HIBS},A}(\kappa) = 1\right]. \tag{3.7}$$

A signature scheme Σ is EUF-CMA-secure, if $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma,A}(\kappa)$ is a negligible function in κ for all PPT adversaries A [GS02].

Naor Transform

To convert the previously discussed Hierarchical Identity-Based Encryption (HIBE) (cf. Section 3.3.5) to a signature, we map it to the HIBS definition (cf. Section 3.3.7). That can be done using the IBE-to-signature transformation from Naor [BF01]. It converts any IND-CPA secure IBE scheme into an EUF-CMA secure signature scheme. More specifically, it converts any level $\ell + 1$ HIBE to a level ℓ HIBS (cf. [KN09]) as shown in Scheme 3.2. The resulting signature of this special case would look much like a traditional signature, since no level for delegation is available.

```
$\underline{\mathsf{Setup}(1^\kappa, \ell)}$: Let $\mathsf{pp} \leftarrow \mathsf{HIBE.Setup}(1^\kappa, \ell + 1)$ and return pp.
$\underline{\mathsf{Gen}(\mathsf{pp})}$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HIBE.Gen}(\mathsf{pp})$ and return $(\mathsf{pk}, \mathsf{sk})$.
$\underline{\mathsf{Del}(\mathsf{sk}_{id'}, id)}$: Return $\mathsf{sk}_{id} \leftarrow \mathsf{HIBE.Del}(\mathsf{sk}_{id'}, id)$.
$\underline{\mathsf{Sign}(\mathsf{sk}_{id}, m)}$: Parse $id$ as $id_1, \ldots, id_l$ and
  return $\sigma \leftarrow \mathsf{HIBE.Del}(\mathsf{sk}, (id_1, \ldots, id_\ell, m))$.
$\underline{\mathsf{Verify}(\mathsf{pk}, (id_i)_{i=1}^{\ell}, m, \sigma)}$: Choose $m' \xleftarrow{\$} \mathsf{HIBE}.\mathcal{M}$ and
  compute $c \leftarrow \mathsf{HIBE.Enc}(\mathsf{pk}, (id_1, \ldots, id_\ell, m), m')$.
  Return 1 if $\mathsf{HIBE.Dec}(\sigma, c) = m'$, otherwise return 0.
```

Scheme 3.2: (Hierarchical identity-based) Signature scheme obtained by applying Naor-transform to HIBE [GS02].

Theorem 3.1. If the HIBE scheme provides HIBE-IND-CPA-security and has a message space that is exponentially large in the security parameter, then the HIBS scheme from Scheme 3.2 provides EUF-CMA-security [Cui+07].

3.3.8 Time-Bound Identity-Based Signature

Alber et al. [AMR20] introduce a special case of HIBS: Time-Bound Identity-Based Signature (TBIBΣ). The novel scheme augments the identity-based delegation mechanism by a time-bound component called epoch. This epoch is designed to bind the signature to a specific time period. Consequently, Alber et al. [AMR20] introduce this time-boundness by delegating the secret key additionally by the corresponding epoch. Thus, a resulting delegated key is not usable before or after the specified epoch.

Epoch With TBIBΣ, every level of the identity-based hierarchy gets a time-boundness added. The underlying idea ensures that the delegated key subsequent to this level is not valid before and after the specified validity. Practically, the epoch is defined by an integer, low-resolution timestamp of choice, represented by an identity pointing to the validity time-span on a time-grid of predefined cell lengths (see Figure 3.5).

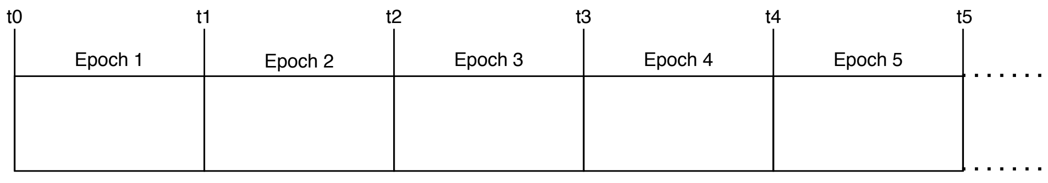


Figure 3.5: Graphical representation of the epochs and its corresponding time grid.

TBIBΣ **Definition**

Alber et al. [AMR20] also define the syntax of time-bound identity-based signatures. The algorithms Sign, Verify and Del from our previous HIBS definition (cf. Section 3.3.7) are extended by the epoch variable.

Definition 3.7 (Time-bound identity-Based Signatures)**.** A Time-Bound Identity-Based Signature (TBIBΣ) scheme with identity-space $\mathcal{ID}$ consists of the PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Del})$, which are defined as follows [AMR20]:

$\mathsf{Setup}(1^\kappa, n)$: On input of security parameter κ and the maximal number of epochs n, outputs public parameters pp.[3]

$\mathsf{Gen}(\mathsf{pp})$: On input of the public parameters pp, outputs a master signing key $\mathsf{sk}_{0,\varepsilon}$ and a verification key pk with associated message space $\mathcal{M}$.

$\mathsf{Del}(\mathsf{sk}_{i-1,id'}, i, id)$: On input of a secret key $\mathsf{sk}_{i-1,id'}$, an identity $id \in \mathcal{ID}^{\leq n}$ and an epoch $i \in [n]$, outputs a secret key $\mathsf{sk}_{i,id}$ for id iff id' is a prefix of id, otherwise $\mathsf{sk}_{id'}$.

$\mathsf{Sign}(\mathsf{sk}_{i,id}, m)$: On input of a secret key $\mathsf{sk}_{i,id}$ for an identity $id \in \mathcal{ID}^{\leq n}$ and an epoch $i \in [n]$ and a message $m \in \mathcal{M}$, outputs a signature σ.

$\mathsf{Verify}(\mathsf{pk}, i, id, m, \sigma)$: On input of a public key pk, an identity $id \in \mathcal{ID}^{\leq n}$, an epoch $i \in [n]$, a message $m \in \mathcal{M}$ and a signature σ, outputs a bit $b \in \{0, 1\}$.

[3]We allow $n = \infty$ to denote an unbounded number of epochs.

Alber et al. [AMR20] regard the scheme to be correct if for all security parameters $\kappa \in \mathbb{N}$ and $n = n(\kappa) \in \mathbb{N}$, for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa, n)$, for all $id \in \mathcal{ID}$, for all $i \in [n]$, for all extracted keys $\mathsf{sk}_{i,id} \leftarrow \mathsf{Del}(\mathsf{sk}, i, id)$, for all $m \in \mathcal{M}$, we have that

$$\Pr[\mathsf{Verify}(\mathsf{pk}, i, id, m, \mathsf{Sign}(\mathsf{sk}_{i,id}, m)) = 1] = 1.$$

TBIBΣ **Security Considerations**

Alber et al. [AMR20] further want to extend the unforgeability security notion EUF-CMA to cover epochs for TBIBΣ delegation keys. They aim to make the adversary select not only the target identity but also some target epoch. Consequently, the notion states that a scheme is considered unforgeable as long as the adversary can not forge a signature for both, a selected target epoch and identity. Thus, it must hold even though an adversary gains access to a delegated key derived from the same master key using the same identity, but a different epoch. Vice versa, it must also hold for a delegated key derived from the same master key for the same epoch but using a different identity.

We provide the experiment with access to following oracles:

$\mathsf{Sign}'(\mathsf{sk}, i, id, m)$: This oracle computes $\mathsf{sk}_{i,id} \leftarrow \mathsf{Del}(\mathsf{sk}, i, id)$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{id,i}, m)$, adds m to $\mathcal{Q}$, and returns σ.
$\mathsf{Del}'(\mathsf{sk}, i, id)$: Stores (i, id) in $\mathcal{Q}^{\mathcal{ID}}$ and returns $\mathsf{Del}(\mathsf{sk}, i, id)$.

Experiment $\mathsf{Exp}^{\text{euf-cma}}_{\mathsf{TBIB\Sigma},A}(\kappa, n)$

```
  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, n)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$
  $(m^*, i^*, id^*, \sigma^*) \leftarrow A^{\mathsf{Del}'(\mathsf{sk},\cdot),\mathsf{Sign}'(\mathsf{sk},\cdot,\cdot,\cdot)}(\mathsf{pk})$
  if $\mathsf{Verify}(\mathsf{pk}, i^*, id^*, m^*, \sigma^*) = 0$, return 0
  if $m^* \in \mathcal{Q}$, return 0
  if $(i^*, id^*) \in \mathcal{Q}^{\mathcal{ID}}$, return 0
  return 1
```

Experiment 3.4: The EUF-CMA experiment for a Time-Bound Identity-Based Signature (TBIBΣ).

Definition 3.8 (EUF-CMA)**.** For any PPT adversary A, we define the advantage in the EUF-CMA experiment $\mathsf{Exp}^{\text{euf-cma}}_{\mathsf{TBIB\Sigma},A}$ (cf. Experiment 3.4) as

$$\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{TBIB\Sigma},A}(\kappa, n) := \Pr\left[\mathsf{Exp}^{\text{euf-cma}}_{\mathsf{TBIB\Sigma},A}(\kappa, n) = 1\right], \tag{3.8}$$

for an integer $n \in \mathbb{N}$. A time-bound identity-based signature scheme TBIBΣ is EUF-CMA-secure, if $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{TBIB\Sigma},A}(\kappa, n)$ is a negligible function in κ for all PPT adversaries A.

3.3.9 TBIBΣ Construction

In this section, we recapitulate the direct construction of TBIBΣ from a HIBS by Alber et al. [AMR20]. The underlying idea is a mapping from TBIBΣ's time and identity components to identities of an underlying identity-based signature scheme of choice (cf. Section 3.3.7). We note that we can use the epoch as a separate identity (instead of concatenating it with the actual identity) and introduce it with an additional call to Del. However, in our context, we want every delegation to be time-bound by an epoch, and thus we use the following compact construction by Alber et al. [AMR20] (see Scheme 3.3).

$\underline{\mathsf{Setup}(1^\kappa, n)}$: : Let $\mathsf{pp} \leftarrow \mathsf{IBS.Setup}(1^\kappa)$ with $\mathsf{IBS}.\mathcal{ID} = \{0,1\}^{\lceil \log_2(n) \rceil} \times \mathcal{ID}$ and return pp.
$\underline{\mathsf{Gen}(\mathsf{pp})}$: Return $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{IBS.Gen}(\mathsf{pp})$.
$\underline{\mathsf{Del}(\mathsf{sk}, i, id)}$: Return $\mathsf{sk}_{i,id} \leftarrow \mathsf{HIBE.Del}(sk, i \| id)$
$\underline{\mathsf{Sign}(\mathsf{sk}_{i,id}, m)}$: Return $\sigma \leftarrow \mathsf{IBS.Sign}(\mathsf{sk}_{i,id}, m)$.
$\underline{\mathsf{Verify}(\mathsf{pk}, i, id, m, \sigma)}$: Return $\mathsf{IBS.Verify}(\mathsf{pk}, i \| id, m, \sigma)$.

Scheme 3.3: TBIBΣ scheme from any IBS.

From the security of the underlying IBS scheme (cf. Definition 3.6), Alber et al. [AMR20] show that the TBIBΣ scheme is secure with the following Theorem 3.2.

Theorem 3.2. If IBS is EUF-CMA-secure, then TBIBΣ from Scheme 3.3 is EUF-CMA-secure, i.e. if there is an EUF-CMA-adversary A against TBIBΣ, then there exists an EUF-CMA-adversary against IBS with

$$\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{TBIB\Sigma},A}(1^\kappa, n) = \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{IBS},B}(1^\kappa).$$

3.3.10 Practical Considerations

Alber et al. [AMR20] discuss some optimizations for efficient signing and verification. They apply when implementing the TBIBΣ scheme from Scheme 3.3 based on the BBG HIBE from Scheme 3.1. We summarize them in the following subsections.

Deterministic Verification

Alber et al. [AMR20] mention that verification of the Naor Transform (testing encryption and decryption to ensure the signature is correctly derived from corresponding identities) is probabilistic. However, standard signature schemes, such as EdDSA and ECDSA, are deterministic. Therefore the latter have better performance since they do not need access to a secure randomness source, which consequently adds computational complexity to the TBIBΣ verification approach. On prudent selection of the underlying HIBS scheme, Alber et al. [AMR20] show that it is possible to check the correctness of the signature using a deterministic method based on Boneh et al. [BLS01] and Waters [Wat05].

On the one hand, they attach a non-interactive zero-knowledge proof that will be verified instead of testing the decryption. On the other hand, is the chosen HIBE BBG-like, it allows to check for correct key derivation by equating the following: Assuming a derived key for the last level, $\mathsf{sk}_{id_1,\ldots,id_\ell} = (\mathsf{sk}_1, \mathsf{sk}_2)$ we have that

$$e\left(h_1^{H(id_1)} \cdots h_\ell^{H(id_\ell)} \cdot g_3, \mathsf{sk}_2\right) \cdot e(g_2, \mathsf{pk}) = e(\mathsf{sk}_1, \hat{g}).$$

In other words, a key satisfying the equation above can also decrypt any ciphertext. Therefore, it is a valid deterministic and faster verification alternative.

Faster Signing with Precomputation

In parts of the signature lifecycle id_1 to $id_{\ell-1}$ stay constant. Therefore, it can be advantageous to precompute all $h_i^{H(id_i)}$, so that every time they are needed, we can fetch them from a cache. That can be useful when verifying, delegating, or respectively, signing multiple times using a similar sequence of identities. Alber et al. [AMR20] describe the signing case the following:

> *Indeed, given the secret key $\mathsf{sk}_{i,id} = (a_0, a_1, b_\ell)$, the signature can be computed as*
>
> $$\left(a_0 \cdot b_\ell^{H(m)} \cdot \left(t \cdot h_k^{H(m)}\right)^w, a_1 \cdot \hat{g}^w\right)$$
>
> *where $t \leftarrow h_1^{H(id_1)} \cdots h_{\ell-1}^{H(id_2)} \cdot g_3$. This t can be either computed when deserializing the secret key or directly be stored in the serialized key. It can be reused for all subsequent signing operations, thus saving $O(\ell)$ group operations and rendering signing runtime complexity independent of ℓ.*

4 Integration of TBIBΣ into TLS

The Time-Bound Identity-Based Signature (TBIBΣ) Scheme introduced by Alber et al. [AMR20] has been described in detail in the last section (see Section 3.3.8). Now, we present an integration of the TBIBΣ scheme into Transport Layer Security (TLS).

We note that our approach is meant for protocol version 1.3, but TBIBΣ *can also be integrated into older versions of the protocol since the approach does not require any changes to the TLS handshake itself.*

Further, as TBIBΣ *is just a variation of* HIBS, *we wrote a* HIBS *provider for* TBIBΣ *signature operations in the TLS stack.*

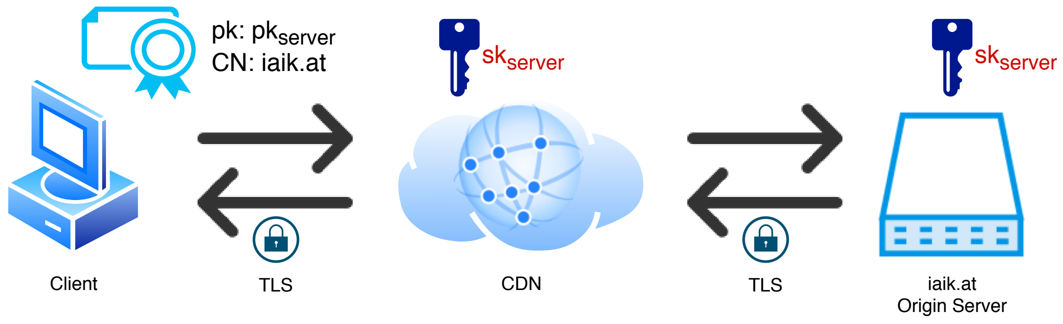


Figure 4.1: Shows a typical Full Delegation scenario on the internet involving Content Delivery Network (CDN) services.

Scenario When we talk about a typical CDN setup, we suppose an interplay between a client, a CDN, and an origin server. We present such a setup graphically in Figure 4.1. A typical client is usually a web browser. It would like to start a connection to an origin server to retrieve some content. As the origin server often is not in the network-wise vicinity, in terms of distances on the internet, latency can be quite high or throughput quite low. Furthermore, the origin server may not have the power to offer computational demanding services to all users, and therefore seeks to offload demanding tasks. Thus, the origin server can contract a CDN service to delegate some or all content or work-load. This measure would improve the connection's performance and enhance the scalability of a service.

The actual application of such a CDN service may vary depending on the needs of the origin server. One way is to upload static data to the CDN service via an uplink. Then the CDN distributes all data to its nodes. Another way is a cache like behavior buffering often requested data from the origin server. Consequently, the CDN node only retrieves data on demand and keeps it in storage for some time, depending on the eviction strategy in use, and serves similar data requests from the storage.

The latter practices show that CDNs, while serving data, act in the origin server's name. Therefore it constitutes what is formally known as a delegation. Further, to make a client connect to the CDN, we need to take control of the name resolution (see Section 3.2.6 for more details). In short, we use the authoritative approach, for which we edit the DNS entry for the origin server's domain name to point to the CDN node in proximity to the client.

Furthermore, TLS secures all connections between client and CDN, and between CDN and origin server. To serve data in an authenticated way, one party signs the TLS handshake, and the other verifies using the internet's Public Key Infrastructure (PKI) (in a server-to-server scenario also vice versa). Full Delegation practices shown in Section 1.1 are widely in use and need to be replaced. We will present a promising replacement option (cf. Chapter 2 for other options).

4.1 Concept

To eliminate Full Delegation practices in TLS, we propose to apply the TBIBΣ scheme introduced in Section 3.3.8. As shown in Figure 4.2, an origin server may decide to delegate the right to deliver content by sending a short-lived, derived version of the private key (also known as delegated key) to a CDN service. Such a provider does not have to be highly trusted as it only receives a key valid for a short time. In case it acts deliberately malicious or in negligence, the harm that it may cause is limited.

The CDN distributes the delegated key to all its nodes enabling temporary delivery of correctly authenticated content over TLS to the requesting clients. A CDN node does so by signing the TLS handshake using the delegated key received from the origin server. Consequently, thanks to the properties of TBIBΣ, the client can verify the authenticity of a connection and its corresponding delegation by using the TBIBΣ public key belonging to the origin server's certificate (see Figure 4.2). Since this implies only changes to the signature algorithm, no modification of the TLS protocol is needed, and therefore implementation effort remains minimal.

Like in TLS with conventional signatures, the server's public key pk_{server} and secret sk_{server} key get generated (see Section 3.3.8). Then a Certificate Signing Requests (CSR) is sent to the CA for signing. Since the CSR is self-signed the CA needs to deal with a TBIBΣ signature. If the CA approves the CSR and verifies the signature, it signs the requested certificate and sends it back to the server.

The server now begins to derive multiple delegated keys sk_{deleg}, one for each upcoming epoch (and each subdomain if needed). Those delegated keys are then sent to the corresponding CDN over a mutually authenticated channel (e.g., a standard TLS connection). If needed, the server's certificate is communicated too.

Coming back to the delegated key, we note that we derive it by the origin server's domain name and a validity epoch (see Section 3.3.8). Including the domain name into the delegation can restrict its validity to a specific subdomain. Furthermore, including the epoch limits the validity in terms of time. Generally, a delegation can be arbitrary short-lived (cf. Section 4.1.3). Then the `delegate` algorithm converts both (domain name and epoch) to identities and concatenates them so that they may serve as a single identity. Next, the secret key gets derived by the concatenated identities yielding the delegated key sk_{deleg}. For further details, please revisit Section 3.3.8.

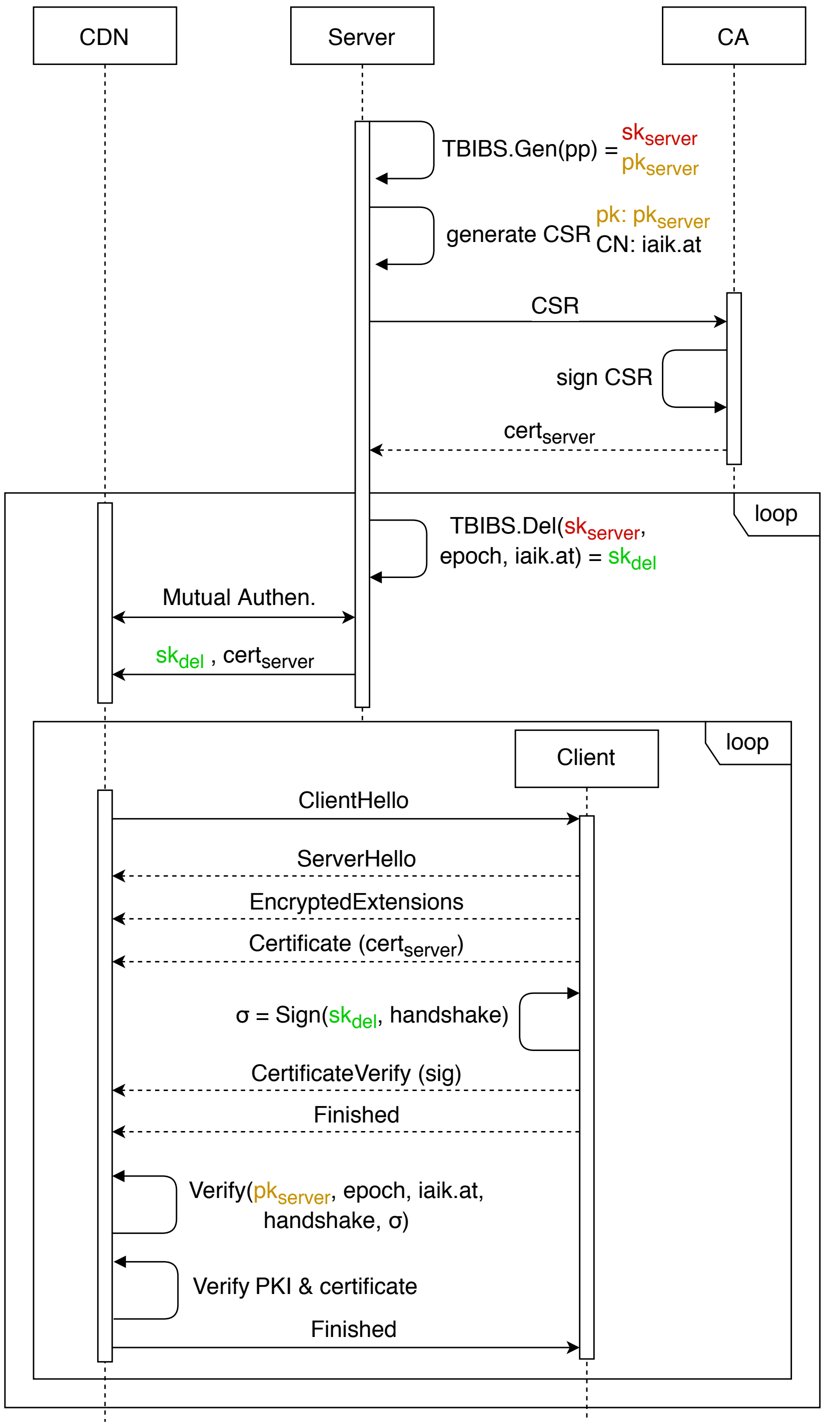

CDN
Server
CA
TBIBS.Gen(pp) = sk_server pk_server
generate CSR pk: pk_server CN: iaik.at
CSR
sign CSR
cert_server
loop
TBIBS.Del(sk_server, epoch, iaik.at) = sk_del
Mutual Authen.
sk_del , cert_server
loop
Client
ClientHello
ServerHello
EncryptedExtensions
Certificate (cert_server)
σ = Sign(sk_del, handshake)
CertificateVerify (sig)
Finished
Verify(pk_server, epoch, iaik.at, handshake, σ)
Verify PKI & certificate
Finished

Figure 4.2: The communications of the TBIBΣ TLS solution as a Sequence Diagram.

On a new connection of a client to the CDN, the handshake proceeds as usual. However, when negotiating the signature scheme, the client and the server will compromise on the newly introduced TBIBΣ. For signing the handshake's hash, the server's TLS stack will then utilize the TBIBΣ signature scheme and apply the currently valid delegated key sk_{deleg}.

After the client has received the server's certificate and the signature over the handshake's hash, it uses the TBIBΣ's `verify` algorithm to prove the subdomain, the current epoch, and the handshake's hash valid. To do so, the client gathers its own values for the identities (subdomain from the SNI extension, the current epoch derived from the client's clock, and the hash from its TLS stack's handshake recordings) and submits them to the verification process. If the algorithm finds that those input parameters will produce the same signature as given and the remaining PKI verification passes, the authentication's correctness is confirmed.

The approach has the following advantages:

Control over private key The CDN services only obtain a derived key, while the secret key never leaves the origin server. Thus, the origin server stays in control of the private key at all times.

Limited validity A derived key is only valid for a limited amount of time. In other words, this limitation in terms of time provided via an epoch offers a cheap alternative to standard revocation mechanisms, since an additional classical revocation checking is redundant. We discuss further details in the upcoming Section 4.1.2.

Epoch-wise forward security According to HIBE's EUF-CMA, knowing delegated keys of some identities does not help the adversary find a specific identity's delegated key. Since epochs are nothing else than identities (see Section 3.3.8), TBIBΣ delegated keys are forward- and backward-secure. In the case of TBIBΣ in TLS, assuming a delegated key of a future epoch is compromised, there will be no effect on the validity of the signatures issued with the current epoch's delegated key.

No protocol changes Often, TLS libraries support the integration of additional signature algorithms. If this is the case, an implementation of our approach is straight forward, since no changes to the TLS stack itself are necessary.

TBIBΣ enables the implementation of additional protection mechanisms. The secret key can be moved to a dedicated key server handling only the key delegation. That reduces potential attack vectors. Such a server would keep key material separate from directly exposed content servers. Therefore, we can achieve tighter network protection since the key server does not need to handle any incoming connections. It only pushes a delegated private key to the CDN periodically. The push takes place before the key enters its validity period. So no delay due to delegation or transport is induced.

Besides, a separate key server can enhance local security too. Content servers may periodically receive a delegated private key, just like CDNs. If one of the more exposed content servers is compromised, a new delegated key will help solve the problem with the beginning of the upcoming epoch.

Furthermore, when operating a key server, it might make sense to deploy a Hardware Security Module (HSM) or similar measures. It allows us to store the key material sandboxed outside of the execution environment. Having only to maintain the key server highly secure, while content servers work with the delegated key supplied by the former, may reduce administration costs aside from reducing surface for attacks.

On the performance side, one should make use of the deterministic `verify` algorithm (see Section 3.3.10) as it performs noticeably faster than normal probabilistic `verify` directly obtained by the Naor-Transform. Further, precomputations (see Section 3.3.10) can become pretty useful in the application of TLS. A CDN node might need to sign lots of handshakes in one epoch to deliver a domain content to several clients. Thus, it will only have to recompute the precomputations once per epoch but will benefit the rest of the time.

4.1.1 Epochs Limit a Delegation

We defined the term epoch in Section 3.3.8. However, there are different ways to apply an epoch as an identifier in TBIBΣ. An integer counter called timestamp represents the number of seconds since some start time T_0 module the epoch length. However, it could also represent the number of epochs since T_0. T_0 might be defined by, e.g., a $NotBefore$ field from a certificate or the Unix start time (January 1st in 1970). Another way can be to use a UTC date-time and flatten it to hours or days. Then one can encode the string representation as identifier. Alternatively, one could use more sophisticated other popular time cycle approaches. That can, for example, be a standardized algorithm from the Time-based One-time Password algorithm (TOTP) protocol [MRa+11]. Looking at TOPT (see Equation (4.1)), we can observe the similarity to the first possibility we described.

$$epoch = \left\lfloor \frac{timestamp_{UTC}}{epochlength} \right\rfloor \tag{4.1}$$

The length of an epoch is a parameter that needs to be agreed on before any communication can occur. One way is to standardize it in a corresponding document. Neither the TLS handshake nor the delegation cryptography offers the possibility to negotiate a certain epoch length without introducing changes to the protocol. However, we can use the public key structure to transport it as one of the public parameters. Since the origin server generates the key pair, the origin server could decide on the epoch length. On the other hand, we could introduce a TLS extension for negotiation or X.509 certificate extension for specification, but we do not consider this an option since it would introduce implementation changes that we aim to avoid.

However, the length of an epoch should be well-grounded. Different applications may need different epoch lengths. Possible lengths may range from a few minutes up to a week, similar to validity periods of an Online Certificate Status Protocol (OCSP) response (see Figure 4.3). Nevertheless, one should keep in mind to set neither too long epochs nor too short epochs for one's specific use case. Setting it to a long epoch can put one out of control on a key material's compromise since no classical revocation on a delegation is possible. That may force one to revoke the certificate itself. Consequently, the delegation would lose its trust since the basis of trust represented by the certificate loses its validity. However, revoking the certificate comes with operational costs [Chu+18; Sta+12; ZAH16], which may not be desirable. Therefore we suggest limiting the epoch length to a relatively low value.

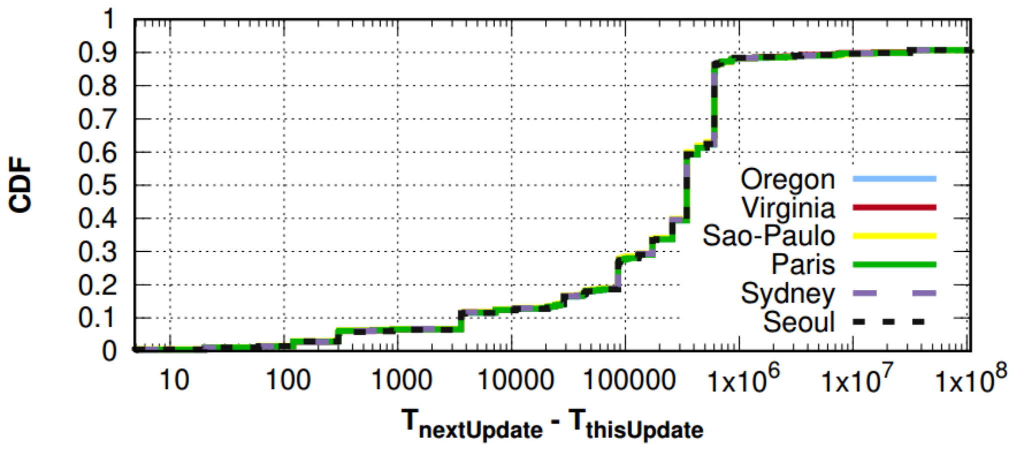


Figure 4.3: Shows the cumulative distribution of validity periods of web certificates. Note that x is represented in seconds [Chu+18, Figure 8].

On the other hand, setting an epoch too short is also counterproductive and may cause operational problems, as discussed in Nir et al. [Nir+18, pp. 4.2, 4.3, 5.2]. Generally, one must consider how long the recovery of the origin server's delegation mechanism takes since its failure is an entirely possible occurrence. On that basis, one determines the epoch's minimal length so that failures may not lead to expiration and outage. Before discussing revocation (see Section 4.1.2) and proposing an epoch length (see Section 4.1.3), we will talk about clock skew and its implications on short-lived solutions.

Often clients lack precise clock systems, despite of NTP (Newtork Time Protocol) being a standard for over thirty years [Ace+17; Mil+10]. So-called clock skew can lead to different problems [Nir+18]:

1. The skewed system time is earlier than the beginning of the epoch. A valid signed target would be seen invalid. We can also mitigate that by checking with the previous epoch as long as the maximal time skew threshold is fulfilled (skewed time plus maximally allowed time skew falls into the current epoch).
2. The skewed system time is later than the end of the epoch. Again, a valid signed target would be seen invalid. We can mitigate that by

also checking with the next epoch, as long as the maximal time skew threshold is fulfilled (skewed time subtracted maximally allowed time skew falls into the current epoch). A sensible renewal policy can also solve it (see the paragraph after the next).

3. The skewed system time is inside the old epoch, and the target is signed with the old epoch. A target is seen valid, although it should not. This is a security-relevant issue and is discussed in Section 4.1.3.

To summarize the measures, we consider the epoch before and after the actual one as long as they are inside the maximal skew threshold. Validation would, therefore, be performed worst case for all three possible epochs. As soon as one of the three validations passes, the verification process can move on. The two additional signature verifications we need are, however, an expensive trade-off.

Alternatively, we propose an approach that can save the effort of at least one `verify` call. However, it does not come without other extra costs. In short, we operate two epoch cycles in parallel, which implies the transmission of an additional signature. Assume an epoch length of a day. Now, one cycle begins and ends at midnight, the other at noon. Both epochs yield corresponding delegated keys, which get pushed to the CDN. The CDN will now sign for both cycles and send both signatures to the client. Latter, will first try to verify the signature of the primary cycle. If it fails, it will check the secondary signature. That means, if the signature is signed by the past epoch of the primary cycle, while the client tries to verify with the actual epoch, verification will not immediately fail. However, it will switch to the second cycle for further verification. It only fails when the second one fails too. That helps to mitigate failures provoked by such epoch-discontinuities and imprecise clocks.

4.1.2 Revocation

While a delegated private key allows the signing in the certificate owner's name, it does not remove any PKI mechanisms currently standardized. A client, verifying the validity, does not only make the usual certificate checks but implicitly also the delegation manifested in the signature. Consequently, we can still use revocation mechanisms such as CRL and OCSP, since certificates are still standard X.509s.

However, to revoke a delegation, there is no such method. The standard revocation mechanism would revoke the whole certificate. Therefore we use the time-bound feature. It ensures that delegations can be rendered rapidly unusable for attackers. Hence it offers an equal benefit as conventional revocation. In other words, delegation in TBIBΣ for TLS works similarly to Short-Term Automatically Renewed (STAR) [Nir+18], but instead of using short-lived certificates, we use hierarchical identity-based cryptography. In fact, both approaches can be used for delegation in TLS, but TBIBΣ offers minimal changes to existing TLS stacks. On the other hand, both share the same idea of superseding the traditional revocation practice. The latter needs the CA to sign the CRL or OCSP data to inform the relying party that the certificate is still valid. Further, issuing a certificate used to require human intervention. Revocation checks, on the other hand, are issued frequently and automatically. So, while certificates were a human burden, they would have long-lasting validity, while revocation data was renewed frequently. However, this scenario changed, since automated renewal protocols exist (Automated Certificate Management Environment (ACME), e.g., Let's Encrypt), just like automated revocation data renewal existed. Now it is easy to see that the extra revocation layer is no longer needed, and a reduction of complexity is at hand. To go more into detail, let us imagine a relying party caching revocation. For that party, it makes sense to only update on every *nextUpdate* time window specified by the current revocation response. So, if epochs have the same length as revocation data updates, TBIBΣ delegations offer a similar security level. Besides, we have to mention that although an epoch provides a way to limit the validity period further, an epoch can never expand the validity period of the certificate itself.

A potentially subverting factor for short-lived delegations is session resumption because a session may outlive the delegation's validity. Interestingly, TLS in version 1.3 proposes to limit session longevity by the certificate's validity period; however, it does not require it [Res18, p. 4.6.1]. Therefore, Chuat et al. [Chu+20] argue that few browser implementations will limit session lifetimes to the certificate's validity period. That reflects the situation for short-lived delegations, also not limiting the session's validity. Consequently, a rogue CDN node could chain sessions using pre-shared key (PSK) resumption without being noticed by the client. Allowing such practice could weaken TBIBΣ's security. We recommend limiting the session lifetime to the epoch length to ensure that a discontinuation of the delegation by the origin server is apprehended. Such checks involve modifications of the TLS stack, but are necessary if one plans to use session resumption.

Alternatively, Chuat et al. [Chu+20] propose to disallow session resumption. This naive approach, though, would have a harsh impact on TLS performance. Nevertheless, to protect users from user tracking, browsers such as Tor Browser or JonDoBrowser turn session resumption entirely off by default [Sy+18].

Again, we can observe an analogy to short-lived certificates [Chu+20]. Nir et al. [Nir+18] and Topalovic et al. [Top+12] had previously neglected to discuss session resumption, although it has a significant impact on approaches based on the short-lived paradigm. In the context of proxy certificates, Chuat et al. [Chu+20] also propose a domain-based on-off session resumption policy, which they plan to achieve through a certificate extension. It would allow a fine-grained differentiation between security-critical and performance-dependent subdomains.

Barnes et al. [Bar+20] discuss session resumption for DeC and recommend caching the credential for re-validation if the client also caches the certificate chain for the same purpose. We described TLS's session resumption in Section 3.2.3.

4.1.3 Security Consideration on Epoch Lengths

Nir et al. [Nir+18] have a similar discussion in their IETF draft, i.e longevity for STAR using ACME. They argue that the validity of such a short-term certificate should equal the period of CRL or OCSP updates. With modern hardware, powerful and reliable enough to renew certificates for tens of thousands of relying parties, the longevity of 1-2 days should be possible. Further taking into account clock skew, they argue that reasonable security can be ensured by reducing the certificate longevity by twice the upper bound for skew [Nir+18]. Twice because CA, as well as the relying party, can suffer clock skew. If we consider a clock skew caused by a time zone problem (maximal 24 hours), a planed validity of four days should be reduced by two days and, therefore last two days. We can apply these arguments to the longevity of epochs. As discussed in Section 4.1.1, regarding clock skew, we can take into consideration the previous and the next epoch, as long as a threshold of 24 hours of clock skew is not passed.

Now, we want to propose an epoch length by example. Suppose a Let's Encrypt certificate, which is valid for three months. It gets renewed automatically with the mechanisms Let's Encrypt provides. Further, Let's Encrypt's OCSP *nextUpdate* amounts to seven days. Therefore, we recommend a renewal of the TBIBΣ's delegated key every seven days; supposing a clock skew of maximal 24 hours was also already included in the OCSP update period. However, since TBIBΣ resides one level beneath typical PKI, we propose a more fine-grained epoch of one day so that the delegated key is renewed, e.g., every midnight.

4.2 Implementation

In the following section, we will look at changes applied to the TLS library and how the HIBS signature provider (for TBIBΣ) is written and plugged into the TLS stack. The discussion will include library-specific aspects as well as general indications useful for the integration into other libraries than the IAIKs TLS stack (`iSaSiLk`).

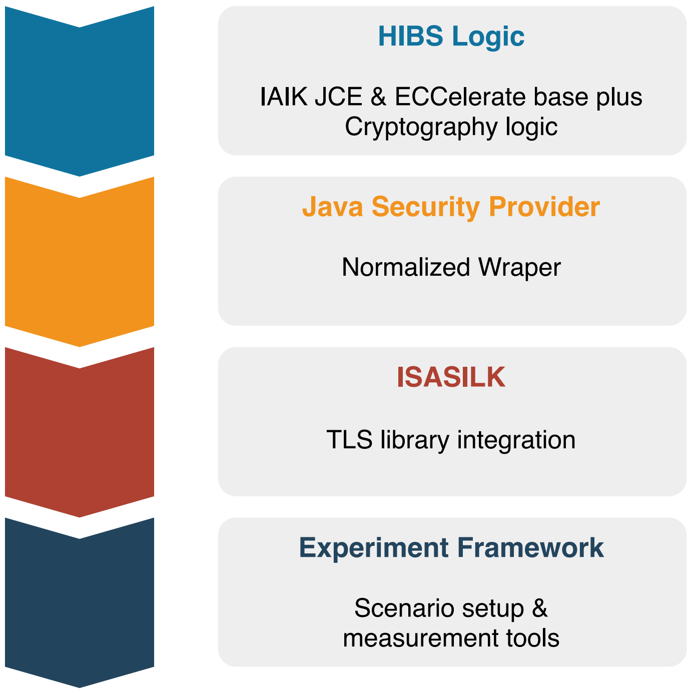


Figure 4.4: An overview of the library layers and how they interact.

We wrote the implementation exclusively in Java because of considerations regarding existing in-house libraries. As shown in Figure 4.4, it consists of multiple parts. The IAIK's `ECCelerate` library offers the basis for Elliptic Curve Pairing operations used for our HIBS framework. The latter we wrap into a Java Cryptography Architecture (JCA) compatible signature provider. We then plug it into a modified TLS stack (`iSaSiLk`), also owned by IAIK, and use all for the TBIBΣ's integration showcase.

4.2.1 HIBS Signature Provider Library

The HIBS signature provider is written in Java and wraps the HIBS framework, based on the IAIK's `ECCelerate`. Since TBIBΣ is just a variation of HIBS we can use the HIBS provider for TBIBΣ signature operations in the TLS stack.

We followed the JCA guidelines when creating the security provider. Therefore it possesses several standardized classes. The `HIBSPublicKey`, the `HIBSPrivateKey`, and the `HIBSDelPrivKey` classes hold key material information and provide de- and encoding functionality. The `HIBSKeyFactory` class en- and decodes X.509 and PKCS8, and exposes a key factory interface according to JCA. The `HIBSKeyPairGenerator` class creates the master key pair consisting of private and public key. It also exposes an interface, according to JCA standards. The `HIBSWithSHA256Signature` class exposes the signature interface offering HIBS in the JCA's standardized form. In addition, the `HIBSProvider` class registers those interfaces and further offers information about the signature algorithms and abilities the provider library provides.

Recalling the TBIBΣ and HIBS explanation from Sections 3.3.7 and 3.3.8, we can see that the work-flow of the scheme differs fundamentally from standard signature schemes operating with `gen`, `sign` and `verify` calls. We note that signing and delegating in HIBS-based signatures are the same thing. So, if we want to delegate in the HIBS library, we call `sign` (see Listing 4.1). Further, to specify which curve should be used, we set, on creation of `HIBSKeyPairParamSpec`, the `SecurityParameters` accordingly. Besides, for setting the identity list, we can call `setParameter` on the `Signature` instance before calling `initSign` respectively `initVerify` (see Listing 4.1).

Listing 4.1: Java Code showing the usage of the HIBS Signature library for key generation, delegating, signing and verifying.

```java
// Key Generation
KeyPairGenerator kg = KeyPairGenerator.getInstance("HIBS");
HIBSKeyPairParamSpec params = HIBSKeyPairParamSpec
  .create(2, new SecurityParams(HIBScurve.BN_P461));
kg.initialize(params);
KeyPair kp = kg.generateKeyPair();

// Initializing
Signature sig = Signature.getInstance("HIBS");

// Delegating
sig.initSign(kp.getPrivate());
sig.update(delData);
byte[] signature = sig.sign();
HIBSDelPrivKey delPrivKey = new HIBSDelPrivKey(signature);

// Signing
sig.setParameter(new HIBSAlgorithmParameterSpec()
  .addDelegateIDs(delData));
sig.initSign(delPrivKey);
sig.update(signData);
byte[] signature2 = sig.sign();

// Verifying
sig.setParameter(new HIBSAlgorithmParameterSpec()
  .addDelegateIDs(delData));
sig.initVerify(kp.getPublic());
sig.update(signData);
boolean isValid = sig.verify(signature2);
```

4.2.2 TLS Stack

To plugin our JCA conform provider into IAIK's `iSaSiLk`, we had to subclass a wrapper (class using the wrapper pattern). In more detail, we derived the `ECCelerateProvider` class and overrode only necessary methods, with a fallback to the `ECCelerate` functionality to keep the remaining elliptic curve algorithms operational. However, the major changes are the additions to the `sign` and `verify` code of the handshake. As seen in Section 4.2.1, we need to set additional parameters. These parameters are the identities: epoch and domain name. Both have to be provided by TLS stack, introducing some minor changes: The epoch can be retrieved by consulting the client's clock, and the domain name can be obtained by checking the SNI-extension of TLS. Further, also an option for TBIBΣ needs to be added in the handshake's `signature_algorithms` extension, but that remains a minor addition.

Let us now go through the main components of the system and discuss the impacts of our approach on them:

Certificate In general, the X.509 structure is compatible with our TBIBΣ approach, and we do not need any fundamental changes to the certificate. Practically speaking, the certificate's public key is replaced by a TBIBΣ one, which we explained in detail in Section 3.3.8. Like any other public key, the TBIBΣ's one will be serialized using ASN.1 encoding and stored as bytes in the X.509 certificate. Nevertheless, we have to make sure that all parties involved in the communication (i.e., client, CDN, origin server, and Certificate Authority (CA)) will understand how to decode and verify a certificate containing this public key.

Therefore, for our demo implementation, we used the IAIK's JCE security library to simulate a CA and the Certificate Signing Requests (CSR) process.

Certificate Authority The PKI based system used on the web requires a certificate issued by a trusted CA for a specific domain to be valid. For using our scheme in this system, the certificate's signature does not need to be a TBIBΣ signature. On the other hand, the CA needs to allow the scheme and consequently a TBIBΣ public key in the certificate. Consequently, it needs to validate the incoming CSR and therefore requires the ability to check a TBIBΣ self-signature on the CSR. The CAs acceptance of new schemes is discussed in Section 5.3.

Origin Server The origin server requests a certificate from the CA using a CSR and periodically renews it. Therefore the origin server self-signs its CSR with its TBIBΣ private key. In return, it receives a certificate containing its TBIBΣ public key.

Further, it uses its secret key to derive a delegated key using the `delegate` algorithm periodically from the TBIBΣ's scheme implementation. For delegation in the demo implementation, we used a domain name and a flattened timestamp as an identity to derive the delegated key. In more detail, flattened timestamp means UTC date-time modulo the epoch (which is a simplified version of what we discussed in Section 4.1.1).

Next, it pushes the delegated key to the CDN, who distributes it to its nodes (the first push also includes the certificate). That happens already before the key's epoch starts. Besides, pushing the delegated key to the CDN can be conducted over an arbitrary, end-to-end secure channel, e.g., using standard HTTPS. The push is a simple transfer of data and can be pretty high-level.

To summarize, the origin server needs the HIBS signature provider for TBIBΣ key generation and delegation, the IAIK's JCE for the CSR, and the means for "out-of-band" communication with the CDN.

Content Delivery Network The CDN service receives a delegated key from the origin server before every start of a new epoch using an end-to-end secure push. Furthermore, the CDN service receives the origin server's certificate on the first push, and whenever the certificate is updated. The CDN service then updates all its nodes with the new authentication information.

As soon as a client is connecting to the CDN, the corresponding node signs the handshake with the delegated key. The signature is sent within the context of the `CertificateVerify` message back to the requesting client. The latter can now verify the authenticity of the handshake and consequently endorse the connection.

To pave the way for the described modification, we extended our CDN's TLS stack by the HIBS signature provider as described in the beginning of this section. Note that we did not change the TLS stack's fundamental workings.

Client The client (e.g., a web browser) needs the HIBS signature library to be added to its TLS stack. Further, similar to the additional parameters for signing, there is also the need for additional ones for the verification (see the first paragraph of this section). Consequently, the client has to provide not only the certificate's public key, signature, and handshake's hash but also the requested domain and the current epoch. Again, the epoch is a flattened timestamp, as described for the origin server above. The domain is retrieved from the SNI-extension. Is the validation using the additional parameters successful, the TLS stack can continue with the remaining validation process.

Standard PKI checks are performed as usual: certificate chain validation, revocation check, hostname validation, time validation, and other possible checks from extensions. In other words, the rest of the TLS protocol remains the same also for the client.

4.3 Integration into Major TLS-Libraries and Browsers

Cryptographic libraries supporting pairing operations are widely available nowadays. In this thesis, we used IAIK's `ECCelerate` library written in Java. Alber et al. [AMR20] used Relic [AG] for C and languages supporting C bindings. Both libraries are well equipped and suite well for $\mathsf{TBIB\Sigma}$ implementations. Since a C implementation exist, integrations into OpenSSL and its forks are easily possible. The same applies to GnuTLS and major browsers like Chrome and Edge.

Further, besides C pairing libraries, there exists also one for Rust [zkc], Java Script [jor], Haskell [ary], and many more. One can find a full list of available libraries on Github [art]. Thus, even for Firefox, slowly transitioning to Rust, an integration is possible. Further, there are ongoing standardization efforts for pairing-friendly curves [Sak+20]. It will ensure compatibility between implementations in the future.

4.4 Other Applications

In this section, we present ideas to use or extend $\mathsf{TBIB\Sigma}$ or HIBS in applications in- or out-side of the classic TLS, respectively, CDN context.

4.4.1 $\mathsf{TBIB\Sigma}$ in TLS Restricting Connection and Content Types

In Section 3.3.8, we described the $\mathsf{TBIB\Sigma}$ scheme, which is a time-based variation of Hierarchical Identity-Based Signature (HIBS) to limit longevity. Alternatively, it is also possible to use other information than time as an identity. Therefore, derivations of delegated keys respective signatures requiring other constraints are possible. As an example, we want to discuss the connection and content types a TLS session is encapsulating.

Connection types mean to limit the use of a delegation to specific application layer protocols such as HTTPS, SPYD, or HTTP/2. ALPN[1] is used to negotiate the protocol. Further, some servers associate different certificates with a specific protocol [Key18]. We propose that in a CDN setup, the origin server may derive a delegated key for each specific protocol. So, instead of different certificates, the origin server only needs one; however, it can ensure that the CDN only operates with the protocols it specified (i.e., it sent the CDN the specific delegated key). On the other hand, this would not easily be possible without significant changes to the client's TLS stack, since it would have to retrieve the specific application layer protocol identifier for signature verification.

Further, also restrictions on content types are possible. We could, e.g., restrict content the CDN might be allowed to serve. Imagine, if a web service should cache only pictures, it might push only delegated keys to the CDN, which restrict the content allowed to deliver to JPEGs.

Other identity types are imaginable for constraining a delegated key, similar to constrain possibilities specified in the Extended Key Usage extension of X.509 [Coo+08, Section 4.2.1.12]. All these additional constrains would need similar modifications, as we presented in Section 3.3.8 for time constraints. We would need to add an identity with standardized type specifiers.

4.4.2 HIBS and TBIBΣ in Other CDN Constellations

In this section, we discuss several ideas for TBIBΣ helping with CDN Interconnections (CDNI) [PDB14] similar to the discussions held by Sheffer et al. [She+20a, Section 4.1] regarding STAR for delegations in CDNI.

[1]Since Google introduced SPYD, an alternative to HTTPS, clients used a TLS extension called Next Protocol Negotiation (NPN) to discover the availability of the application protocol. With RFC 7301 [Fri+14] a successor for NPN was specified: Application-Layer Protocol Negotiation (ALPN). In contrast to NPN, ALPN includes a list of supported protocols already in the *ClientHello* message. Thus the negotiation takes only a single round. Further, the server can associate a specific certificate to a specific protocol.

Parallel Sometimes, the origin owner has multiple agreements with CDN providers in place. That can have various reasons: On the one hand, the owner prefers different CDNs for different geographical regions. On the other hand, the owner wants to keep a "backup" CDN to manage temporary traffic peaks. TBIBΣ can facilitate such practices, as the origin server can distribute arbitrarily many short-lived delegated keys for specific subdomains to different parties.

Chained Larger CDNs can have regional contractors for certain regions. Those regional subcontractor-CDNs usually do not have a contractual relationship with the origin owner. Latter might not even be aware of them. Thus, it makes sense for the primary CDN to restrict namespaces further or introduce additional restrictions when delegating to a subcontractor. TBIBΣ can support arbitrary many delegation levels with the possibility of introducing restrictions on each level via identities.

4.4.3 HIBS and TBIBΣ in Other Applications

The potential of Hierarchical Identity-Based Signature (HIBS) is much bigger than its application presented for TLS. Numerous other areas require verification of multiple delegations or hierarchical signature structures. Here we shortly discuss potential applications for future work:

Smart Contracts Smart contracts [Rös+98; Sza97] are immutable contracts that automatically execute on a distributed ledger. Imagine an arbitrary scenario where a smart contract needs to redirect money. It receives input from one wallet and distributes it according to its contract logic to other private wallets or smart contracts. Transferring the money first to the smart contracts wallet before processing incurs extra effort and consequently extra fees. With TBIBΣ one could delegate the smart contract privileges to directly gain short-lived access to the source wallet and therefore extend the contracts capabilities, while lowering execution costs.

IoT alternative to D2TLS We shall repeat the scenario of D2TLS [Cho+19] mentioned in Section 2.5.1: A security agent sets up a session for a low-power Internet of Things (IoT) device, so that it may communicate securely on demand evading relatively high computational efforts. While the D2TLS approach lets the device sign every handshake's hash for the security agent, we propose to push a time-bound identity-constraint delegated TBIBΣ key in an "out-of-bound" manner to the agent. The latter can then use it once or multiple times to set up a connection to a cloud domain specified by an identity. The remaining steps take place the same as in D2TLS.

In other words, instead of splitting the TLS connection, we would only exchange the signature algorithm and perform an out-of-bound communication to push a delegated key. Besides, delegating in TBIBΣ has comparable computational costs as signing in other elliptic curve based algorithms (EdDSA, ECDSA). However, when establishing connection, no additional latency is introduced since splitting is avoided.

Further, we could introduce an efficient revocation mechanism by letting the IoT device send a nonce to the cloud server in plain. This nonce is then used to derive the delegated key. The cloud server knowledgable of the nonce identity can use it to verify the security agent's signature. On needing to revoke the delegated key, the device pushes a new nonce to the cloud. Nevertheless, the epoch should remain a second revocation factor to limit damage when a compromise is not detected or an attacker prevents the smart device from accessing the network to issue the revocation.

Delegation in STIR Delegations in STIR [PST14] ecosystem were already proposed by Sheffer et al. [She+20a, p. 4.2] using STAR certificates. However, also TBIBΣ is suitable to delegate telephone numbers from one service provider to another, supporting restrictions of TNAuthList as an additional identity on deriving the delegated key. Similar to Section 4.4.2 (subcontracting service providers seen as CDNs), CDNI practices are also supported.

5 Evaluation

In this chapter, we will evaluate the implementation's performance of the scheme and the TLS integration. Further, we discuss the resulting implications. Next, we evaluate the benefits and downsides of the approach and compare it to other proven or promising solutions (see Section 5.3). Finally, we will pursue a characterization of our approach into the 19-criteria-framework of Chuat et al. [Chu+20] (see Section 5.3.2).

5.1 Measurements

In this first section, we will take a look at the performance of the Time-Bound Identity-Based Signature (TBIBΣ) scheme's implemenation. We discuss performance differences compared to other typical signature algorithms and give a theoretical explaination for performance and key sizes. Subsequently, we will evaluate the performance of the TBIBΣ integrated into a TLS stack and discuss its implications for operational use.

5.1.1 TBIBΣ Setup

As mentioned in Section 4.2.2 our implementation of the TBIBΣ scheme is based on cryptographic primitive from the IAIK's JAVA Security libraries. Further, the implemented scheme is then integrated into the IAIK's TLS library to evaluate our claims concerning modest modifications to TLS stacks. Besides, Alber et al. [AMR20] implemented a second version of the TBIBΣ scheme based on Relic [AG] and written in C. We will use those result for comparing to our Java implementation and discuss implications for performance sensitive applications.

Generally, our TBIBΣ implementation offers Barreto-Naehring curves and we target a security of 128 bits. Using recent security estimations by Barbulescu et al. [BD19] and Menezes et al. [MSS16], we chose curves with 256 and 461 bits. The former certainly only offering 100 bits, not 128 bits of security, should also suffice for key material renewed every three months [Aas+19].

Alber et al. [AMR20], on the other hand, chose a pairing-friendly Barreto-Lynn-Scott curve [BLS02] with 381 bits (BLS-381). It also corresponds to 128 bits of security.

5.1.2 TBIBΣ Evaluation

Algorithm	Sign	Verify	pk	σ
EdDSA	$E_{\mathbb{G}}$	$2\ E_{\mathbb{G}}$	$\mathbb{G}$	$2\ \mathbb{Z}_p$
ECDSA	$E_{\mathbb{G}}$	$2\ E_{\mathbb{G}}$	$\mathbb{G}$	$2\ \mathbb{Z}_p$
TBIBΣ	$2\ E_{\mathbb{G}_1}, E_{\mathbb{G}_2}$	$3\ P$	$\mathbb{G}_2$	$\mathbb{G}_1, \mathbb{G}_2$

Table 5.1: Operations of `sign` and `verify`, sizes of public keys (pk) and signatures (σ) for EdDSA, ECDSA and TBIBΣ. $E_{\mathbb{G}}$ denotes an exponentiation in group $\mathbb{G}$ and P a pairing operation. Further, $\mathbb{G}$ symbolizes the size of a group element, while $\mathbb{Z}_p$ represents an integer (cf. Scott [Sco20]).

We present an estimation of performance and payload sizes in Table 5.1. We base these on the most expensive element performed, respectively contained. We found that TBIBΣ needs two exponentiations in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ for signing, which is relatively cheap compared to verify needing three pairing operations. The latter, we found the most expensive. We can observe that the competing signature algorithms have much cheaper verify and sign calculations. Even a TBIBΣ sign is more expensive than a verify of the competition. On the other hand, these latter do not offer identity-based features.

Regarding the size of the payload, we estimated for the public key an element in $\mathbb{G}_2$ and for the signature an element of each group, $\mathbb{G}_1$ and $\mathbb{G}_2$. Compared to the competitors, we can see that the public key is of a similar size, and only the signature is predicted to be much bigger for TBIBΣ.

Algorithm	Sign	Verify
ECDSA (secp256r1)	0.80	1.63
Ed25519	0.11	1.13
TBIBΣ (BN-461)	15.86	79.29
TBIBΣ (BN-512)	19.72	96.67
TBIBΣ (BN-638)	33.53	157.80

Table 5.2: Runtime benchmarks of the Java implementation of TBIBΣ (see Section 3.3.8) compared with the optimized standard signatures from the `ECCelerate` library in milliseconds (ms).

In Table 5.2 and Figure 5.1, we compare the performance of our TBIBΣ implementation in Java to its competitor signing algorithms ECDSA and EdDSA, also implemented in the IAIK's `ECCelerate`. We can observe a significant difference between our TBIBΣ prototype implementation and the well-optimized competitors. BN-461 is 20 times slower than ECDSA when singing and nearly 50 times slower when verifying.

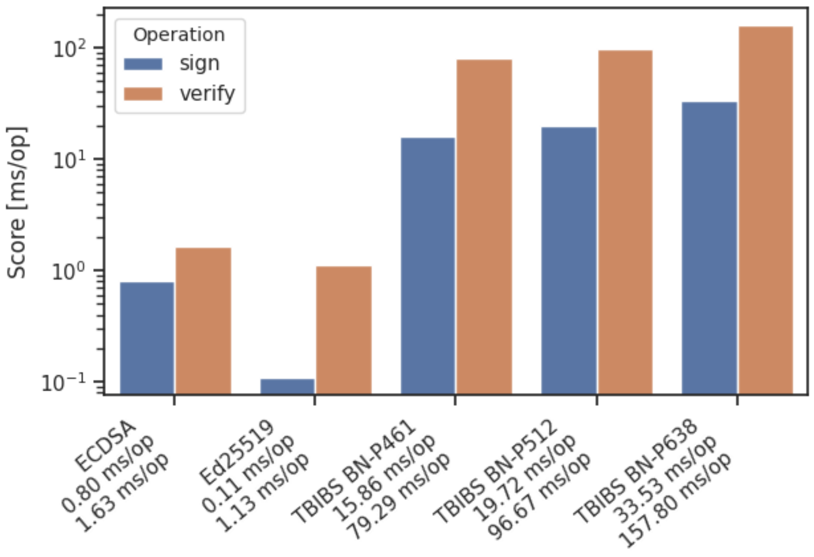


Figure 5.1: Plot of the Java runtime benchmarks of TBIBΣ from Section 3.3.8 compared with the optimized standard signatures from the `ECCelerate` library in milliseconds per operation (ms/op). Note the logarithmic scale.

In Table 5.3 by Alber et al. [AMR20], we can see similar differences. They also note that ECDSA and EdDSA implementations (OpenSSL) are highly optimized for the specific architecture. Alber et al. [AMR20] attest that disabling optimizations brings the performance of ECDSA verification close to the signing of TBIBΣ. We expected that since we saw the same number of group operations for those prior in this chapter.

Both benchmarks were run on a Thinkpad T450s with an Intel Core i7-5600U CPU. While OpenSSL results were obtained by running the command `openssl speed` on Linux, numbers of TBIBΣ and Java implementations were gathered by averaging over 1000 runs.

Algorithm	Sign	Verify
EdDSA (Ed25519)	0.05	0.14
ECDSA (sepc256r1)	0.02	0.08
TBIBΣ (BLS-381)	0.58	1.94

Table 5.3: Runtime benchmarks in ms of ECDSA, EdDSA, and TBIBΣ (see Section 3.3.8). The numbers for Ed25519 and ECDSA are from OpenSSL 1.1.1 with enabled optimizations, while TBIBΣ is from the Relic implementation of Alber et al. [AMR20]

5.2 Scheme Performance

5.2.1 TLS Setup

As mentioned in Section 4.2.2, we use the IAIK's `iSaSiLk` TLS library to integrate our TBIBΣ implementation. Subsequently, we implemented a client-server demo. We use this setup, together with Linux's `Queueing Disciplines`, to simulate a client-CDN context. We performed the TLS benchmarks using OpenJDK's Java Microbenchmark Harness (JMH) library for ten iterations each of 20 seconds, including a warmup of five iterations each of ten seconds. Besides, we used the kernel extension `haveged` as an unpredictable random generator to prevent drainage of the secure randomness pool, slowing down the benchmark. It harvests the indirect effects of hardware events [die].

Further, we used the standard Linux command `ping` to find a round trip time representing a standard home with a wireless router and glass fiber connection connecting to `google.com`. It resulted in a round trip time of circa 20ms. To simulate such a typical delay, we introduced a network latency of 10ms, including a normal-distributed variance of one ms. Again, the benchmarks were all performed on an office notebook (Thinkpad T450s with i7-5600U), and thus absolute numbers might not be comparable with server performances.

Signature	Full Handshakes		Server time
Algorithm	ops/s	s/ops	ms
RSA (2048 bit)	8.942	0.112	7.314
ECDSA (secp256r1)	9.254	0.108	4.106
EdDSA (ed25519)	9.198	0.109	4.013
TBIBΣ (BN-256)	7.703	0.130	11.324
TBIBΣ (BN-461)	4.598	0.217	26.504
TBIBΣ (BN-638)	3.275	0.305	51.338

Table 5.4: Benchmarks of the TLS 1.3's handshake for ECDSA, EdDSA, and TBIBΣ (one op is a full handshake).

5.2.2 TLS Evaluation

Observing Table 5.4 and Figure 5.2, showing results of the TLS handshake benchmarks for ECDSA, EdDSA, and TBIBΣ, we can see that the cryptographic performance plays a much smaller role, although TLS 1.3 reduced the round trips to one. The 20ms latency introduced by that single round trip lead to a slowdown for BN-461 compared to ECDSA around factor 2. On the other hand, BN-256, with 100 bits of security, has only a trade-off of 20%, which we find sound compared to the benefits the solution brings. Further, we assumed reasonable, continental latency. For latencies higher than 10ms, the performance gap would close even further. Besides, an assembler optimization, like OpenSSL offers, and some precomputation (cf. Section 3.3.10) may further curb performance drawbacks. As the Relic implementation of

Alber et al. [AMR20] showed, verification pace can be far beneath typical network latencies. Also, further handshakes can use session resumption. Therefore, signature verification can be avoided by session resumption until a new epoch starts.

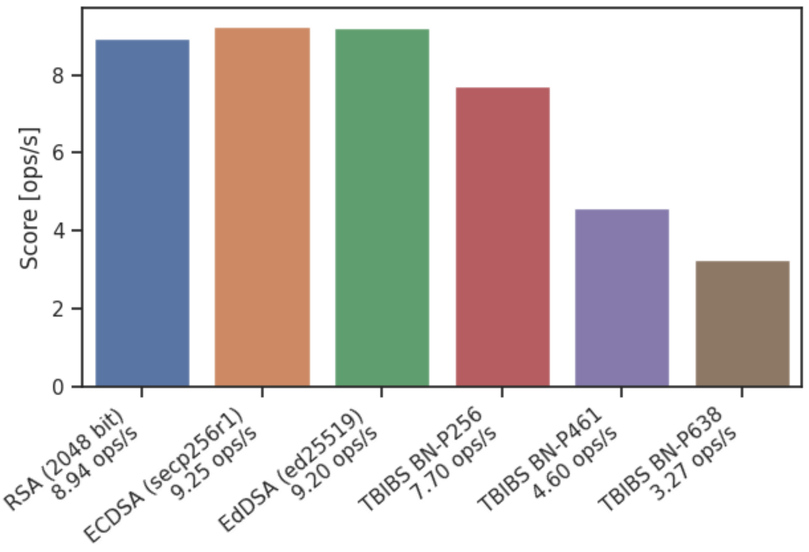


Figure 5.2: Plot of TLS 1.3 benchmarks in `iSaSiLk`. Shows the duration of a handshake for ECDSA, EdDSA, and TBIBΣ (one op is a full handshake).

In terms of CPU server time (also see Table 5.4), we can see that TBIBΣ is more than double as expensive as the standard elliptic curve algorithms, whereas the server's CPU time of RSA (2048 bit) comes near BN-256. Another observation is how short computation time is, compared to a full handshake operation (even though the Java implementation is far slower than the Relic one).

In Table 5.1, we saw that a TBIBΣ public key approximates to an element's size in $\mathbb{G}_2$, which equals 64, 96, or 116 bytes for BN-256, BLS-381, or BN-461 (without point compression: 128, 191, 230 bytes). On the other hand, an EdDSA or ECDSA public key only uses 32 bytes, respectively 64 bytes without point compression. Likewise, signatures only take 64 bytes with the latter signature algorithms (two scalars in $\mathbb{Z}_p$), while TBIBΣ signatures even take one element of each group, $\mathbb{G}_1$ and $\mathbb{G}_2$. That leads to 96, 143, or 173 bytes for BN-256, BLS-381, or BN-461 (without point compression even 192 bytes, 286 bytes, or 345 bytes). Comparing with the number of bytes recorded during a transaction mentioned in Table 5.5, we can see clear effects of the huge sized public key and signature of TBIBΣ: the records sent by the server are much bigger for TBIBΣ than for ECDSA. Not even RSA, suffering from big key sizes, has similar record sizes. The difference is only noticeably on the bytes sent by the server since the `Certificate` message contains the public key and the `CertificateVerify` message the signature.

In contrast to KeylessSSL, we do not need to establish a connection from the CDN to the key server, which introduces additional latency. An additional round trip needs to be made. While between client and CDN, the network-wise distance typically is rather low, it is not between CDN and key (origin) server. Consequently, it results in a higher latency for the latter. Our approach avoids that by pushing the delegated key a priori to the CDN over an "out-of-band" communication channel.

Compared to Delegated Credentials, our approach does not have any performance-wise advantage, but also no significant drawback. We showed that the computational delay is neglectable compared to the network's one. Besides, DeC needs to validate the blob's signature, which also costs an additional signature check.

Sent by	**Client**		**Server**	
TLS 1.3 with	bytes	records	bytes	records
RSA (2048 bit)	643	3	1773	5
ECDSA (secp256r1)	643	3	1385	5
EdDSA (ed25519)	643	3	1330	5
TBIBΣ (BN-256)	645	3	2769	5
TBIBΣ (BN-461)	645	3	3915	5

Table 5.5: Bytes and record number sent by the endpoints during a TLS 1.3's handshake for RSA, ECDSA, EdDSA, and TBIBΣ.

5.3 Qualitative Evaluation

In this section, we will discuss the qualitative properties of our TBIBΣ solution for TLS. In contrast to the previous section, we will focus less on numbers and measurements, but more on the advantages and disadvantages of our approach when in use.

5.3.1 Discussion

TBIBΣ A striking advantage of our Time-Bound Identity-Based Signature scheme is already prominently mentioned in the signature scheme's name, time-bound. It refers to the delegated key's restriction to a specific time-span, we described as an epoch in 3.3.8. To recognize the benefit, we have to imagine a key compromise regarding the delegated private key. Such a compromise would in TBIBΣ only affect security (authenticity, non-repudiation, and integrity) in a specific epoch. For all subsequent epochs, new and unrelated delegated keys are securely distributed, and thus unforgeability guarantees are restored. We can see that proper authenticity, non-repudiation, and integrity are made possible without revoking the origin server's original key material.

Besides using identities to restrict the delegated key to a specific time-span, the origin server can also restrict it by a specific subdomain. That enables the origin server to make fine-grained restrictions on what content is outsourced to the CDN. E.g., the subdomain `images.iaik.at` only can be for images used by a website. Thus, the origin server delegates only that address to the CDN to let only the image content being served. That can help to reduce the attack surface in the event of a CDN going rogue.

To add on top, Alber et al. [AMR20] showed that we could equip TBIBΣ with forward security features, i.e., we can make the master private key held by the origin server forward secure. Consequently, even on a master secret compromise, we can offer mitigation measures. The master secret itself gets updated for each epoch. In fact, if the key for the next epoch is compromised, that has no negative consequences for delegation, respectively, signature validity in the current epoch. For more details about the forward security feature, see Alber et al. [AMR20].

TLS integration A beneficial aspect of the TBIBΣ's TLS integration is that it works with very few modifications to the TLS stack. The short-lived delegation structure TBIBΣ features exclusively takes place in the cryptographic part of the signature scheme. Therefore, we need to plug the TBIBΣ scheme into the TLS stack, like any other cryptography library. Only calls like `sign` and `verify` require additional arguments such as all delegated identities. However, that is nothing exceptional since several signature algorithms need special parameters, too (e.g., RSASSA-PSS's mask generation algorithm [IAI]). Further, consolidated mechanisms like Public Key Infrastructure (PKI) and consequently, revocation are not touched in any manner by our TBIBΣ integration. They still coexist and retain their original function. While solutions like KeylessSSL open an additional channel to the keyserver and, consequently, constitute a multi-party connection, our approach does not need any multi-party security assumptions. In fact, our scheme still works within the boundaries of the same two-party connection as regular TLS does. Thus, proofs of (S)ACCE in TLS [Jag+12] are still valid and do not need any further inspection.

For TBIBΣ to find broad adoption on the internet, we would need the CA/Browser Forum's approval of our signature scheme. Only then, major browsers and certificate authorities would consider an implementation for their product. Browsers like Chrome, Safari, and Firefox would add the scheme to the supported algorithms for their TLS stack, and Certificate Authorities would accept TBIBΣ's public key-based certificates for signing. However, getting new schemes approved by the forum is very hard since they are known for being very conservative. That might be one of the significant downsides of our approach. While for TBIBΣ and DeC standard PKI mechanisms remain unchanged, both require Certificate Authorities involvement. While TBIBΣ needs the CA to accept its algorithm, DeC needs the CA to issue end-entity certificates with its special `DelegationUsage` extension.

A further aspect of TBIBΣ in TLS is the anonymity of the CDN. The client can verify the signature and the delegation but can not identify the Content Delivery Network (CDN) the delegation was warranted. However, that can be a desired feature not only in other applications (discussed in Section 4.4.3), but also for TLS. We can evade this property of TBIBΣ by including the CDN's identity in the delegation process. Similar to restricting content types, like discussed in 4.4.1, the origin server also derives by the identifier to whom it delegates. The identifier is then added to the identity list, including epoch and domain. The client then verifies the CDN's supposed identity using the signature to be sure about the correctness. Like DeC, anonymity is given, but revealing the identity (i.e., mentioning it in the blob) is also possible [Bar+20].

Besides, compared to proxy certificates, TBIBΣ (and other solutions such as DeC) does not exercise any additional pressure on Certificate Transparency logs (see Section 3.2.2). Delegations do not produce extra certificates that would be logged for Certificate Transparency needs. Also the Short-Term Automatically Renewed (STAR) (see Section 2.2.5) delegation solution puts pressure on the logs. However, that is because of its short-lived approach, which applies to all Automated Certificate Management Environment (ACME) systems.

Our approach offers another advantage: the CDN can directly perform key management actions if necessary. Cangialosi et al. [Can+16] found that third parties' key management is more thorough. As CDN providers might detect key material compromises early, with TBIBΣ they can quickly revoke a certificate by signing the revocation request with their delegated key. The CA can verify the signature with the origin's private key.

> "If you did not originally issue the certificate, but you have a copy of the corresponding private key, you can revoke by using that private key to sign the revocation request [Enc20]."

Last but not least, TBIBΣ and HIBS, in general, can verify a steep hierarchy of delegations without a linear increase of effort. In most standardized delegation solutions such as certificate chains in the PKI of the internet, every delegation-level is represented by a certificate, respectively, one signature verification. For a chain of h levels, this results in h verifications. HIBS, on the other hand, would execute delegate operation for each level and verify once at the cost of an additional group operation per level (base cost is three pairings, see 5.1). So, when it comes to scenarios with steep delegation hierarchies (e.g., CDNI, see Section 4.4.2) –steeper than a standard PKI certificate chain–, HIBS schemes are advantageous.

5.3.2 Systematic Characterization

In the following section, we will characterize our approach according to the 19-criteria framework of Chuat et al. [Chu+20]. Please see Table 5.6 for a summary of the benefits supported by different competing approaches. Further, we extend the table by an analysis of our solution presented in this thesis. Note that we reduced the table's criteria to 16 since (damage-free) CA revocation is not relevant for delegation approaches, and all of them support delegation by definition. For a detailed description of each benefit, see Chuat et al. [Chu+20].

Supports leaf revocation Delegated Credentials and TBIBΣ technically do not offer the benefit of revoking a credential or certificate at the chain of trust's end. However, thanks to their short-lived characteristic domain owners, they allow invalidating a delegation on key compromise or similar eventualities. Chuat et al. [Chu+20] give partial points to proxy certificates since they can be short-lived too.

Supports autonomous revocation With both, Delegated Credentials and TBIBΣ it is possible to perform revocation autonomously (i.e., independent from a CA, browser vendor, or log). With both, the domain owner can decide independently to stop the short-lived delegation by ending the distribution of credentials, respectively delegated keys. Again, partial points are given for proxy certificates, since they can be short-lived. This benefit corresponds partially to requirements R4 and R7 (cf. Section 1.2).

Avoids Full Delegation The benefit is basic for all approaches mentioned in related work since it distinguishes it from bad practice, Full Delegation approaches. This corresponds to requirement R1 (cf. Section 1.2).

Support domain-based policies While name constraint and proxy certificates can specify policies per domain in the certificate, Delegated Credentials and TBIBΣ only get partial points because their semantics are limited (only a time component is supported in its standard versions). The benefit corresponds to requirement R6 (cf. Section 1.2).

No trust-on-first-use required None of the approaches requires trust-on-first-use.

Preserves user privacy All of the delegation approaches do not give any domain-related data to a third party.

Scheme	Reference	Supports leaf revocation	Supports autonomous leaf rev.	Avoids Full Delegation	Supports domain-based policies	No trust-on-first-use required	Preserves user privacy	No increased page-load delay	Low burden on CAs	Reasonable logging overhead	Non-proprietary	No special hardware required	No extra CA involvement	No browser-vendor involvement	Server compatible	Browser compatible	No out-of-band communication
Priv. key shar.	[Chu+20]	○	○	○	○	●	●	●	●	●	●	●	●	●	●	●	●
Cruise-liner c.	[Chu+20]	○	○	○	○	●	●	●	●	●	●	●	○	●	●	●	●
Name const. c.	[Lia+14]	○	○	●	●	●	●	●	●	○	●	●	○	●	●	◑	●
DANE-based	[Lia+14]	○	○	●	○	●	●	○	●	●	●	●	●	●	●	○	●
SSL splitting	[LK05]	○	○	●	○	●	●	○	●	●	●	●	●	●	○	●	●
KeylessSSL	[SN14]	○	○	●	○	●	●	○	●	●	●	●	●	●	○	●	●
STAR Del.	[She+20a]	●	○	●	●	●	●	●	○	○	●	●	○	●	○	●	●
Proxy cert.	[Chu+19]	◑	◑	●	●	●	●	●	●	●	●	●	●	●	◑	○	●
DeC	[Bar+20]	●	●	●	◑	●	●	●	●	●	●	●	◑	●	○	○	●
TBIBΣ		●	●	●	◑	●	●	●	●	●	●	●	◑	●	◑	◑	●

Table 5.6: The table presents 16 of the 19 criteria by Chuat et al. [Chu+20] relevant for delegation schemes. While the assessment of the approaches discussed in Chapter 2 is adopted from Chuat et al. [Chu+20], we added a characterization of TBIBΣ.

● offers the benefit; ◑ partially offers benefit; ○ does not offer the benefit.

No increased page-load delay Chuat et al. [Chu+20] give full points for this benefit if none or small processing delays arise. Approaches with additional network latencies get no points. SSL splitting, KeylessSSL, and the DANE-based approach all need an additional round-trip leading to zero points.

Low burden on CAs Only one of the approaches imposes additional operational effort for the Certificate Authority. A CA hardly adopts approaches limiting financial benefits. Only STAR delegations get no points because the CA has to support the ACME protocol and make sure the CDN is trustworthy.

Reasonable logging overhead The approaches putting heavy pressure on certificate logs are name constraint certificates and STAR delegations. For name constraint certificates, every domain owner may issue an arbitrary number of certificates. STAR delegations produce a growing number of short-lived certificates for each delegation. Neither Delegated Credentials nor TBIBΣ is in doubt since they delegate by credential, respectively delegated key.

Non-proprietary All of the approaches are open and neither restricted nor controlled by a third party.

No special hardware is required Although pairing operations (TBIBΣ) may benefit from specialized hardware, it is unnecessary. None of the approaches necessarily needs specialized hardware.

No extra CA involvement Chuat et al. [Chu+20] give partial points to Delegated Credentials since the CA needs to include an extension in the end-entity certificate. We give TBIBΣ partial points too since the CA needs to accept the TBIBΣ signature scheme. No points are given to name constraint, Cruise-liner and STAR delegation certificates, as a significant part of the delegation process is managed by the CA.

No browser-vendor involvement None of the approaches needs active browser-vendor participation.

Server compatibility We give partial points to TBIBΣ since the signature scheme must be added to supported schemes of the TLS stack (mostly a matter of plugging in the signature library). Other approaches like Delegated Credentials and KeylessSSL need changes in the TLS stack logic to be supported. STAR delegations get no points since it requires the origin server to approve and pass CSR to the ACME CA while authenticating as the domain owner.

Browser compatibility Again, we give partial points to TBIBΣ since the signature scheme must be added to the TLS stack to ensure support for the approach.

No out-of-band communication None of the approaches uses a separate channel or communicates with a third party server. Neither Delegated Credentials nor TBIBΣ gets the point deduction for pushing the credential, respectively, the delegated key to the CDN, since this channel needs to be established for the content anyways.

6 Conclusion

Finally, we want to conclude with a summary of the thesis and give an outlook on future work.

6.1 Summary

Full Delegation is an open issue for ecosystems not conceptualized for supporting a delegation system such as TLS. Nevertheless, we showed that proper authentication of delegated parties in TLS is possible using an identity-based signing algorithm. By only exchanging the signature algorithm, we showed how we solve this problem without introducing fundamental changes to TLS.

From the requirements we set at the beginning of this paper (cf. Section 1.2, we could fulfill the most: The master secret is kept secure, and a delegation is unforgeable and non-repudiable. On the other hand, the origin server is only aware of the delegations given out for a particular epoch, but not of CDN's actions or further sharing of the delegated key. Also, offering transparency on the CDN's identity needs extra modifications. Nonetheless, the issuance and revocation can be conducted independently and efficiently, and the origin server has control over the validity period. Also, privileges delegated to the Content Delivery Network (CDN) can be fine-grained by introducing additional identities.

Further, our approach offers additional advantages like limited key management, little pressure on Certificate Transparency logs, and optionally forward-security. However, Delegated Credentials (DeC) is backed by several major internet companies, which makes its push through easier.

6.2 Future Work

The next step would be to put the approach into practice. After receiving the support of the CA/Browser Forum, clients and CAs would need to add the Time-Bound Identity-Based Signature (TBIBΣ) algorithm to their libraries and ensure the verify algorithm has access to the correct domain name (SNI) and epoch. One would need to write standards for the origin server and CDN specifications. The origin server would benefit from an automated management environment for delegating keys similar to Automated Certificate Management Environment (ACME). Also, the delegated key distribution and the epoch length would need standardization.

However, Full Delegation not only exists in the context of TLS. In Section 4.4, we explained multiple other ideas that might be interesting to explore in the future. Further, a combined solution of DeC and TBIBΣ could be interesting for CDN Interconnections (CDNI) (see Section 4.4.2) since DeC offers the possibility of custom signature schemes. Therefore, we hope to see TBIBΣ in productive use in various applications in the future.

7 Bibliography

Literature

[Aas+19] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth D. Schoen, and Brad Warren. "Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web." In: *ACM CCS*. ACM, 2019, pp. 2473–2487 (cit. on p. 75).

[AHO16] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. "Design in Type-I, Run in Type-III: Fast and Scalable Bilinear-Type Conversion Using Integer Programming." In: *CRYPTO (3)*. Vol. 9816. LNCS. Springer, 2016, pp. 387–415 (cit. on p. 39).

[Ace+17] Mustafa Emre Acer, Emily Stark, Adrienne Porter Felt, Sascha Fahl, Radhika Bhargava, Bhanu Dev, Matt Braithwaite, Ryan Sleevi, and Parisa Tabriz. "Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors." In: *ACM CCS*. ACM, 2017, pp. 1407–1420 (cit. on p. 60).

[Adr+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice." In: *ACM CCS*. ACM, 2015, pp. 5–17 (cit. on p. 29).

[AMR20] Lukas Alber, Stefan More, and Sebastian Ramacher. "Short-Lived Forward-Secure Delegation for TLS." In: *CCSW*. to appear. ACM. 2020 (cit. on pp. 6, 47–53, 70, 74, 75, 77, 79, 81).

[Bae+97] Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, and Peter Sturm. "Enhancing the Web's Infrastructure: From Caching to Replication." In: *IEEE Internet Comput.* 1.2 (1997), pp. 18–27 (cit. on pp. 1, 34, 35).

[BH05] Boaz Barak and Shai Halevi. "A model and architecture for pseudo-random generation with applications to /dev/random." In: *ACM CCS*. ACM, 2005, pp. 203–212 (cit. on p. 37).

[BD19] Razvan Barbulescu and Sylvain Duquesne. "Updating Key Size Estimations for Pairings." In: *J. Cryptol.* 32.4 (2019), pp. 1298–1336 (cit. on p. 75).

[Bar+19] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. "Automatic Certificate Management Environment (ACME)." In: *RFC* 8555 (2019), pp. 1–95 (cit. on p. 33).

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees." In: *SCN*. Vol. 2576. LNCS. Springer, 2002, pp. 257–267 (cit. on p. 75).

[BM99] Mihir Bellare and Sara K. Miner. "A Forward-Secure Digital Signature Scheme." In: *CRYPTO*. Vol. 1666. LNCS. Springer, 1999, pp. 431–448 (cit. on p. 37).

[Beu+15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. "A Messy State of the Union: Taming the Composite State Machines of TLS." In: *IEEE S&P*. IEEE Computer Society, 2015, pp. 535–552 (cit. on p. 29).

[Bha+18] Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. "A Formal Treatment of Accountable Proxying Over TLS." In: *IEEE S&P*. IEEE Computer Society, 2018, pp. 799–816 (cit. on p. 21).

[Bha+17] Karthikeyan Bhargavan, Ioana Boureanu, Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. "Content delivery over TLS: a cryptographic analysis of keyless SSL." In: *EuroS&P*. IEEE, 2017, pp. 1–6 (cit. on p. 12).

[Bla+03] Simon Blake-Wilson, Magnus Nyström, David Hopwood, Jan Mikkelsen, and Tim Wright. "Transport Layer Security (TLS) Extensions." In: *RFC* 3546 (2003), pp. 1–29 (cit. on p. 28).

[BPW12] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. "Secure Proxy Signature Schemes for Delegation of Signing Rights." In: *J. Cryptol.* 25.1 (2012), pp. 57–115 (cit. on pp. 2, 17).

[BB04] Dan Boneh and Xavier Boyen. "Secure Identity Based Encryption Without Random Oracles." In: *CRYPTO*. Vol. 3152. LNCS. Springer, 2004, pp. 443–459 (cit. on p. 40).

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. "Hierarchical Identity Based Encryption with Constant Size Ciphertext." In: *EUROCRYPT*. Vol. 3494. LNCS. Springer, 2005, pp. 440–456 (cit. on pp. 41–43).

[BF01] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing." In: *CRYPTO*. Vol. 2139. LNCS. Springer, 2001, pp. 213–229 (cit. on pp. 18, 38, 40, 46).

[Bon+03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps." In: *EUROCRYPT*. Vol. 2656. LNCS. Springer, 2003, pp. 416–432 (cit. on p. 17).

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing." In: *ASIACRYPT*. Vol. 2248. LNCS. Springer, 2001, pp. 514–532 (cit. on p. 51).

[BC95] Hans-Werner Braun and Kimberly C. Claffy. "Web Traffic Characterization: An Assesment of the Impact of Caching Documents from NCSA's Web Server." In: *Comput. Networks ISDN Syst.* 28.1&2 (1995), pp. 37–51 (cit. on p. 34).

[Bul+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution." In: *USENIX*. USENIX Association, 2018, pp. 991–1008 (cit. on p. 20).

[Bul+20] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection." In: *IEEE S&P*. IEEE, 2020, pp. 54–72 (cit. on p. 20).

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. "A Forward-Secure Public-Key Encryption Scheme." In: *EUROCRYPT*. Vol. 2656. LNCS. Springer, 2003, pp. 255–271 (cit. on p. 41).

[Can+16] Frank Cangialosi, Taejoong Chung, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. "Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem." In: *ACM CCS*. ACM, 2016, pp. 628–640 (cit. on pp. 1, 3, 7, 83).

[Cho+19] Eunsang Cho, Minkyung Park, Hyunwoo Lee, Junhyeok Choi, and Ted Taekyoung Kwon. "D2TLS: delegation-based DTLS for cloud-based IoT services." In: *IoTDI*. ACM, 2019, pp. 190–201 (cit. on pp. 19, 20, 72).

[Chu+20] Laurent Chuat, Abdel Rahman Abdou, Ralf Sasse, Christoph Sprenger, David Basin, and Adrian Perrig. "SoK: Delegation and Revocation, the Missing Links in the Web's Chain of Trust." In: *EuroS&P*. IEEE, 2020. arXiv: `1906.10775v2` (cit. on pp. 6, 7, 18, 26, 35, 62, 63, 74, 84–86).

[Chu+19] Laurent Chuat, AbdelRahman Abdou, Ralf Sasse, Christoph Sprenger, David A. Basin, and Adrian Perrig. "Proxy Certificates: The Missing Link in the Web's Chain of Trust." In: *CoRR* abs/1906.10775 (2019) (cit. on pp. 14–16, 85).

[Chu+18] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John P. Rula, Nick Sullivan, and Christo Wilson. "Is the Web Ready for OCSP Must-Staple?" In: *Internet Measurement Conference*. ACM, 2018, pp. 105–118 (cit. on pp. 59, 60).

[Coo+08] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and W. Timothy Polk. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation

List (CRL) Profile." In: *RFC* 5280 (2008), pp. 1–151 (cit. on pp. 2, 8, 23, 25, 26, 31, 71).

[CS98] Ronald Cramer and Victor Shoup. "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack." In: *CRYPTO*. Vol. 1462. LNCS. Springer, 1998, pp. 13–25 (cit. on p. 40).

[Cui+07] Yang Cui, Eiichiro Fujisaki, Goichiro Hanaoka, Hideki Imai, and Rui Zhang. "Formal Security Treatments for Signatures from Identity-Based Encryption." In: *ProvSec*. Vol. 4784. LNCS. Springer, 2007, pp. 218–227 (cit. on p. 47).

[Dil+02] John Dilley, Bruce M. Maggs, Jay Parikh, Harald Prokop, Ramesh K. Sitaraman, and William E. Weihl. "Globally Distributed Content Delivery." In: *IEEE Internet Comput.* 6.5 (2002), pp. 50–58 (cit. on pp. 1, 34, 35).

[FS86] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems." In: *CRYPTO*. Vol. 263. LNCS. Springer, 1986, pp. 186–194 (cit. on p. 40).

[Fri+14] Stephan Friedl, Andrei Popov, Adam Langley, and Emile Stephan. "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension." In: *RFC* 7301 (2014), pp. 1–9 (cit. on p. 71).

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. "Pairings for cryptographers." In: *Discret. Appl. Math.* 156.16 (2008), pp. 3113–3121 (cit. on pp. 38, 39).

[GS02] Craig Gentry and Alice Silverberg. "Hierarchical ID-Based Cryptography." In: *ASIACRYPT*. Vol. 2501. LNCS. Springer, 2002, pp. 548–566 (cit. on pp. 41, 45–47).

[Gün89] Christoph G. Günther. "An Identity-Based Key-Exchange Protocol." In: *EUROCRYPT*. Vol. 434. LNCS. Springer, 1989, pp. 29–37 (cit. on p. 37).

[HGL20] Stephen Herwig, Christina Garman, and Dave Levin. "Achieving Keyless CDNs with Conclaves." In: *USENIX*. USENIX Association, 2020, pp. 735–751 (cit. on p. 20).

[HS12] Paul E. Hoffman and Jakob Schlyter. "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA." In: *RFC* 6698 (2012), pp. 1–37 (cit. on pp. 8, 9).

[Hoo+12] Hans Hoogstraaten, Ronald Prins, Daniël Niggebrugge, Danny Heppener, Frank Groenewegen, Janna Wettink, Kevin Strooy, Pascal Arends, Paul Pols, Robbert Kouprie, et al. "Black Tulip: Report of the investigation into the DigiNotar certificate authority breach." In: *Fox-IT, Tech. Rep* (2012) (cit. on p. 26).

[III11] Donald E. Eastlake III. "Transport Layer Security (TLS) Extensions: Extension Definitions." In: *RFC* 6066 (2011), pp. 1–25 (cit. on p. 32).

[Jag+12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. "On the Security of TLS-DHE in the Standard Model." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 273–293 (cit. on pp. 12, 82).

[Joh+01] Kirk L. Johnson, John F. Carr, Mark S. Day, and M. Frans Kaashoek. "The measured performance of content distribution networks." In: *Comput. Commun.* 24.2 (2001), pp. 202–206 (cit. on p. 1).

[Kat10] Jonathan Katz. *Digital Signatures*. Springer, 2010 (cit. on pp. 37, 38, 44, 45).

[Ken93] Steve Kent. "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management." In: *RFC* 1422 (1993), pp. 1–32 (cit. on p. 23).

[KN09] Eike Kiltz and Gregory Neven. "Identity-Based Signatures." In: *Identity-Based Cryptography*. Vol. 2. Cryptology and Information Security Series. IOS Press, 2009, pp. 31–44 (cit. on p. 46).

[KPW97] Seungjoo Kim, Sangjoon Park, and Dongho Won. "Proxy signatures, Revisited." In: *ICICS*. Vol. 1334. LNCS. Springer, 1997, pp. 223–232 (cit. on p. 17).

[KWZ01] Balachander Krishnamurthy, Craig E. Wills, and Yin Zhang. "On the use and performance of content distribution networks." In: *Internet Measurement Workshop*. ACM, 2001, pp. 169–182 (cit. on pp. 1, 34, 35).

[LLK13] Ben Laurie, Adam Langley, and Emilia Käsper. "Certificate Transparency." In: *RFC* 6962 (2013), pp. 1–27 (cit. on p. 26).

[LK05] Chris Lesniewski-Laas and M. Frans Kaashoek. "SSL splitting: Securely serving data from untrusted caches." In: *Comput. Networks* 48.5 (2005), pp. 763–779 (cit. on pp. 10, 85).

[Lia+14] Jinjin Liang, Jian Jiang, Hai-Xin Duan, Kang Li, Tao Wan, and Jianping Wu. "When HTTPS Meets CDN: A Case of Authentication in Delegated Service." In: *IEEE S&P*. IEEE Computer Society, 2014, pp. 67–82 (cit. on pp. 2, 7–9, 31, 32, 85).

[MRa+11] David M'Raïhi, Salah Machani, Mingliang Pei, and Johan Rydell. "TOTP: Time-Based One-Time Password Algorithm." In: *RFC* 6238 (2011), pp. 1–16 (cit. on p. 59).

[MUO96] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. "Proxy Signatures for Delegating Signing Operation." In: *ACM CCS*. ACM, 1996, pp. 48–57 (cit. on pp. 2, 17).

[MY91] Ueli M. Maurer and Yacov Yacobi. "Non-interactive Public-Key Cryptography." In: *EUROCRYPT*. Vol. 547. LNCS. Springer, 1991, pp. 498–507 (cit. on p. 40).

[McG08] David A. McGrew. "An Interface and Algorithms for Authenticated Encryption." In: *RFC* 5116 (2008), pp. 1–22 (cit. on p. 28).

[MSS16] Alfred Menezes, Palash Sarkar, and Shashank Singh. "Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-Based Cryptography." In: *Mycrypt*. Vol. 10311. LNCS. Springer, 2016, pp. 83–108 (cit. on p. 75).

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field." In: *STOC*. ACM, 1991, pp. 80–89 (cit. on p. 40).

[Mil+86] Victor Miller et al. "Short programs for functions on curves." In: *Unpublished manuscript* 97.101-102 (1986), p. 44 (cit. on p. 39).

[Mil+10] David L. Mills, Jim Martin, Jack L. Burbank, and William T. Kasch. "Network Time Protocol Version 4: Protocol and Algorithms Specification." In: *RFC* 5905 (2010), pp. 1–110 (cit. on p. 60).

[Nay+15] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodríguez Rodríguez, and Peter Steenkiste. "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS." In: *SIGCOMM*. ACM, 2015, pp. 199–212 (cit. on pp. 20, 21, 36).

[NJP18] Yoav Nir, Simon Josefsson, and Manuel Pegourie-Gonnard. "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier." In: *RFC* 8422 (2018), pp. 1–34 (cit. on p. 28).

[PST14] Jon Peterson, Henning Schulzrinne, and Hannes Tschofenig. "Secure Telephone Identity Problem Statement and Requirements." In: *RFC* 7340 (2014), pp. 1–25 (cit. on p. 73).

[PDB14] Larry L. Peterson, Bruce Davie, and Ray van Brandenburg. "Framework for Content Distribution Network Interconnection (CDNI)." In: *RFC* 7336 (2014), pp. 1–58 (cit. on p. 71).

[Pet13] Yngve N. Pettersen. "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension." In: *RFC* 6961 (2013), pp. 1–10 (cit. on p. 32).

[Res18] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3." In: *RFC* 8446 (2018), pp. 1–160 (cit. on pp. 1, 27–31, 62).

[Rös+98] Martin Röscheisen, Michelle Q. Wang Baldonado, Kevin Chen-Chuan Chang, Luis Gravano, Steven P. Ketchpel, and Andreas Paepcke. "The Stanford InfoBus and Its Service Layers: Augmenting the Internet with High-Level Information Management Protocols." In: *The MeDoc Approach*. Vol. 1392. LNCS. Springer, 1998, pp. 213–230 (cit. on p. 72).

[Sal+08] Joseph Salowey, Hao Zhou, Pasi Eronen, and Hannes Tschofenig. "Transport Layer Security (TLS) Session Resumption without Server-Side State." In: *RFC* 5077 (2008), pp. 1–20 (cit. on p. 29).

[San+13] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP." In: *RFC* 6960 (2013), pp. 1–41 (cit. on p. 32).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards." In: *J. Cryptol.* 4.3 (1991), pp. 161–174 (cit. on p. 17).

[Sch+19] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. "ZombieLoad: Cross-Privilege-Boundary Data Sampling." In: *ACM CCS*. ACM, 2019, pp. 753–768 (cit. on p. 20).

[Sco20] Michael Scott. "On the Deployment of curve based cryptography for the Internet of Things." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 514 (cit. on p. 75).

[Sha84] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes." In: *CRYPTO*. Vol. 196. LNCS. Springer, 1984, pp. 47–53 (cit. on p. 40).

[Shb+15] Wazen M. Shbair, Thibault Cholez, Antoine Goichot, and Isabelle Chrisment. "Efficiently bypassing SNI-based HTTPS filtering." In: *IM*. IEEE, 2015, pp. 990–995 (cit. on p. 28).

[She+20b] Yaron Sheffer, Diego R. López, Óscar González de Dios, Antonio Pastor Perales, and Thomas Fossati. "Support for Short-Term, Automatically Renewed (STAR) Certificates in the Automated Certificate Management Environment (ACME)." In: *RFC* 8739 (2020), pp. 1–22 (cit. on pp. 13, 32, 33).

[Sil86] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Graduate texts in mathematics. Springer, 1986 (cit. on p. 39).

[SP88] Gustavus J. Simmons and George B. Purdy. "Zero-Knowledge Proofs of Identity And Veracity of Transaction Receipts." In: *EUROCRYPT*. Vol. 330. LNCS. Springer, 1988, pp. 35–49 (cit. on p. 40).

[Sta+12] Emily Stark, Lin-Shung Huang, Dinesh Israni, Collin Jackson, and Dan Boneh. "The Case for Prefetching and Prevalidating TLS Server Certificates." In: *NDSS*. The Internet Society, 2012 (cit. on p. 59).

[SS15] Douglas Stebila and Nick Sullivan. "An Analysis of TLS Handshake Proxying." In: *TrustCom/BigDataSE/ISPA (1)*. IEEE, 2015, pp. 279–286 (cit. on p. 10).

[Sy+18] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. "Tracking Users across the Web via TLS Session Resumption." In: *ACSAC*. ACM, 2018, pp. 289–299 (cit. on pp. 29, 63).

[Sza97] Nick Szabo. "Formalizing and Securing Relationships on Public Networks." In: *First Monday* 2.9 (1997) (cit. on p. 72).

[SCP16] Pawel Szalachowski, Laurent Chuat, and Adrian Perrig. "PKI Safety Net (PKISN): Addressing the Too-Big-to-Be-Revoked Problem of the TLS Ecosystem." In: *EuroS&P*. IEEE, 2016, pp. 407–422 (cit. on p. 18).

[Tan87] Hatsukazu Tanaka. "A Realization Scheme for the Identity-Based Cryptosystem." In: *CRYPTO*. Vol. 293. LNCS. Springer, 1987, pp. 340–349 (cit. on p. 40).

[Top+12] Emin Topalovic, Brennan Saeta, Lin-Shung Huang, Collin Jackson, and Dan Boneh. "Towards short-lived certificates." In: *Web 2.0 Security and Privacy* (2012), pp. 1–9 (cit. on p. 63).

[TI89] Shigeo Tsujii and Toshiya Itoh. "An ID-based cryptosystem based on the discrete logarithm problem." In: *IEEE J. Sel. Areas Commun.* 7.4 (1989), pp. 467–473 (cit. on p. 40).

[Tue+04] Steven Tuecke, Von Welch, Douglas Engert, Laura Pearlman, and Mary R. Thompson. "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile." In: *RFC* 3820 (2004), pp. 1–37 (cit. on p. 14).

[VP03] Athena Vakali and George Pallis. "Content Delivery Networks: Status and Trends." In: *IEEE Internet Comput.* 7.6 (2003), pp. 68–74 (cit. on pp. 1, 34).

[Val+18] Luke Valenta, Nick Sullivan, Antonio Sanso, and Nadia Heninger. "In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild." In: *EuroS&P*. IEEE, 2018, pp. 384–398 (cit. on p. 29).

[WOM17] Georg Wagner, Olamide Omolola, and Stefan More. "Harmonizing Delegation Data Formats." In: *Open Identity Summit*. Vol. P-277. LNI. Gesellschaft für Informatik, Bonn, 2017, pp. 25–34 (cit. on p. 21).

[Wat05] Brent Waters. "Efficient Identity-Based Encryption Without Random Oracles." In: *EUROCRYPT*. Vol. 3494. LNCS. Springer, 2005, pp. 114–127 (cit. on p. 51).

[Wei+17] Changzheng Wei, Jian Li, Weigang Li, Ping Yu, and Haibing Guan. "STYX: a trusted and accelerated hierarchical SSL key management and distribution system for cloud based CDN application." In: *SoCC*. ACM, 2017, pp. 201–213 (cit. on p. 20).

[Wei40] André Weil. "Sur les fonctions algébriquesa corps de constantes fini." In: *CR Acad. Sci. Paris* 210.592-594 (1940), p. 149 (cit. on p. 39).

[Wel+04] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist. "X. 509 proxy certificates for dynamic delegation." In: *3rd annual PKI R&D workshop*. Vol. 14. 2004 (cit. on p. 14).

[Woo+17] Paul Wood, Heng Zhang, Muhammad-Bilal Siddiqui, and Saurabh Bagchi. "Dependability in Edge Computing." In: *CoRR* abs/1710.11222 (2017) (cit. on p. 36).

[ZAH16] Liang Zhu, Johanna Amann, and John S. Heidemann. "Measuring the Latency and Pervasiveness of TLS Certificate Revocation." In: *PAM*. Vol. 9631. LNCS. Springer, 2016, pp. 16–29 (cit. on p. 59).

Online Resources

[AMR] Lukas Alber, Stefan More, and Sebastian Ramacher. *TBIBS*. URL: `https://github.com/IAIK/TBIBS` (visited on 09/11/2020) (cit. on p. 6).

[AG] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient LIbrary for Cryptography*. URL: `https://github.com/relic-toolkit/relic` (visited on 04/21/2020) (cit. on pp. 70, 74).

[art] artjomb. *PairingLibs.md*. URL: `https://gist.github.com/artjomb/f2d720010506569d3a39` (visited on 05/21/2020) (cit. on p. 70).

[ary] arybczak. *haskell-mcl*. URL: `https://github.com/arybczak/haskell-mcl` (visited on 05/21/2020) (cit. on p. 70).

[Bar+20] R. Barnes, S. Iyengar, N. Sullivan, and E. Rescorla. *Delegated Credentials for TLS (draft-ietf-tls-subcerts-08)*. Apr. 2020. URL: `https://tools.ietf.org/html/draft-ietf-tls-subcerts-08` (visited on 07/20/2020) (cit. on pp. 16, 32, 63, 83, 85).

[Bon] Dan Boneh. *3rd BIU Winter School on Cryptography: The Basics of Pairings*. February 4th-7th, 2013. Bar-Ilan University, Youtube. URL: `https://www.youtube.com/watch?v=F4x2kQTKYFY` (visited on 03/24/2020) (cit. on pp. 38, 39).

[cer] certificate-transparency.org. *What is Certificate Transparency?* URL: `https://www.certificate-transparency.org/what-is-ct` (visited on 09/11/2020) (cit. on p. 26).

[Cis18] Cisco. *Annual Cybersecurity Report*. 2018. URL: `https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf` (visited on 09/11/2020) (cit. on p. 1).

[Cloa] Cloudflare. *CDN Performance*. URL: `https://www.cloudflare.com/learning/cdn/performance/` (visited on 06/30/2020) (cit. on p. 34).

[Clob] Cloudflare. *What is a CDN? — How do CDNs work?* URL: `https://www.cloudflare.com/learning/cdn/what-is-a-cdn/` (visited on 06/30/2020) (cit. on pp. 1, 34).

[Cloc] Cloudflare. *What is Caching? — How is a Website Cached?* URL: https://www.cloudflare.com/learning/cdn/what-is-caching/ (visited on 07/06/2020) (cit. on pp. 34, 35).

[die] die.net. *haveged(8) - Linux man page*. URL: https://linux.die.net/man/8/haveged (visited on 08/25/2020) (cit. on p. 77).

[Enca] Let's Encrypt. *A nonprofit Certificate Authority providing TLS certificates to 225 million websites.* URL: https://letsencrypt.org/ (visited on 09/20/2020) (cit. on p. 33).

[Encb] Let's Encrypt. *How It Works*. URL: https://letsencrypt.org/how-it-works/ (visited on 09/20/2020) (cit. on p. 33).

[Enc20] Let's Encrypt. *Revoking certificates*. Feb. 2020. URL: https://letsencrypt.org/docs/revoking/ (visited on 08/20/2020) (cit. on p. 83).

[Fis19] Dennis Fisher. *Chrome and Firefox Removing EV Certificate Indicators*. Aug. 2019. URL: https://duo.com/decipher/chrome-and-firefox-removing-ev-certificate-indicators (visited on 07/20/2020) (cit. on p. 3).

[For] CA/Browser Formum. *EV SSL Certificate Guidelines*. URL: https://cabforum.org/extended-validation/ (visited on 07/06/2020) (cit. on p. 3).

[GA18] Ghedini and Alessandro. *Encrypt it or lose it: how encrypted SNI works*. Sept. 2018. URL: https://blog.cloudflare.com/encrypted-sni/ (visited on 03/30/2020) (cit. on p. 28).

[GNI19] Alex Guzman, Kyle Nekritz, and Subodh Iyengar. *Delegated credentials: Improving the security of TLS certificates*. Nov. 2019. URL: https://engineering.fb.com/security/delegated-credentials/ (visited on 07/20/2020) (cit. on pp. 16, 17).

[IAI] IAIK. *JCE Provider API Documentation*. URL: http://javadoc.iaik.tugraz.at/iaik_jce/old/iaik/security/rsa/RSAPssSignature.html (visited on 09/11/2020) (cit. on p. 82).

[imp] imperva. *What is a CDN*. URL: https://www.imperva.com/learn/performance/what-is-cdn-how-it-works/ (visited on 09/11/2020) (cit. on p. 2).

[JJM19] Kevin Jacobs, J.C. Jones, and Thyla van der Merwe. *Validating Delegated Credentials for TLS in Firefox*. Nov. 2019. URL: `https://blog.mozilla.org/security/2019/11/01/validating-delegated-credentials-for-tls-in-firefox/` (visited on 07/20/2020) (cit. on p. 16).

[jor] jorgenhoc. *jspairings*. URL: `https://github.com/jorgenhoc/jspairings` (visited on 05/21/2020) (cit. on p. 70).

[Key18] Keycdn. *ALPN Explained*. Oct. 2018. URL: `https://www.keycdn.com/support/alpn` (visited on 05/25/2020) (cit. on p. 71).

[Lor+14] Salvatore Loreto, John Mattsson, Robert Skog, Hans Spaak, G Gus, Dan Druta, and Mohammad Hafeez. *Explicit trusted proxy in HTTP/2.0*. 2014. URL: `https://tools.ietf.org/html/draft-loreto-httpbis-trusted-proxy20-01` (visited on 09/11/2020) (cit. on p. 21).

[Nir+18] Y Nir, T Fossati, Y Sheffer, and T Eckert. *Considerations For Using Short Term Certificates*. Mar. 2018. URL: `https://tools.ietf.org/id/draft-nir-saag-star-01.html` (visited on 09/11/2020) (cit. on pp. 33, 60, 62, 63).

[PR12] Peon and Roberto. *Explicit proxies for HTTP/2.0*. 2012. URL: `https://tools.ietf.org/html/draft-rpeon-httpbis-exproxy-00` (visited on 09/11/2020) (cit. on p. 21).

[Sak+20] Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby. *Pairing-Friendly Curves*. June 2020. URL: `https://tools.ietf.org/html/draft-irtf-cfrg-pairing-friendly-curves-07` (visited on 07/12/2020) (cit. on p. 70).

[She+20a] Y. Sheffer, D. Lopez, A. Pastor Perales, and T. Fossati. *An ACME Profile for Generating Delegated STAR Certificates*. Mar. 2020. URL: `https://tools.ietf.org/html/draft-ietf-acme-star-delegation-03` (visited on 08/20/2020) (cit. on pp. 13, 14, 71, 73, 85).

[SM19] Michael Shirer and Carrie MacGillivray. *The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. June 2019. URL: `https://www.`

idc.com/getdoc.jsp?containerId=prUS45213219 (visited on 09/11/2020) (cit. on p. 36).

[Sul18] Nick Sullivan. *A Detailed Look at RFC 8446 (a.k.a. TLS 1.3)*. 2018. URL: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/ (visited on 06/25/2020) (cit. on pp. 28–30).

[Sul17] Nick Sullivan. *Introducing Zero Round Trip Time Resumption (0-RTT)*. 2017. URL: https://blog.cloudflare.com/introducing-0-rtt/ (visited on 06/26/2020) (cit. on pp. 29–31).

[SL19] Nick Sullivan and Watson Ladd. *Delegated Credentials for TLS*. Nov. 2019. URL: https://blog.cloudflare.com/keyless-delegation/ (visited on 07/20/2020) (cit. on pp. 12, 16).

[SN14] Sullivan and Nick. *Keyless SSL: The Nitty Gritty Technical Details*. Sept. 2014. URL: https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/ (visited on 03/30/2020) (cit. on pp. 10–12, 85).

[Sys19] Cisco Systems. *Cisco Visual Networking Index 2018*. Feb. 2019. URL: https://cyrekdigital.com/pl/blog/content-marketing-trendy-na-rok-2019/white-paper-c11-741490.pdf (visited on 07/20/2020) (cit. on p. 1).

[Tau17] Tim Taubert. *The future of session resumption*. 2017. URL: https://timtaubert.de/blog/2017/02/the-future-of-session-resumption/ (visited on 06/25/2020) (cit. on p. 30).

[zkc] zkcrypto. *pairing*. URL: https://github.com/zkcrypto/pairing (visited on 05/21/2020) (cit. on p. 70).