



POLITECNICO DI TORINO

Master Degree in Computer Engineering

Master Degree Thesis

**DigitalCertiAnalytics: A tool for  
collection and analysis of X.509v3  
digital certificates**

**Supervisor**

Prof.ssa Diana Gratiela Berbecaru

**Candidate**

Anuar Elio MAGLIARI

APRIL 2025



# Summary

The progressive evolution of digital communications requires ever more accurate and sophisticated protection, which is why this thesis focuses on the in-depth analysis of digital certificates, fundamental elements for ensuring the integrity and protection of information.

The work is divided into two main macro sections: on the one hand, it offers a theoretical examination that describes how certificates work, illustrating the various types of certificates (EV, OV, DV), the chain structure and control mechanisms such as the Signed Certificate Timestamp (SCT) and Certificate Transparency (CT), with particular attention to the implications in terms of security and privacy.

On the other hand, the original contribution of the thesis takes the form of the implementation of DigitalCertiAnalytics, a software designed ad hoc for the collection, verification and analysis of certificates, capable of operating on large-scale datasets, as demonstrated by the study carried out on 20 million domains. The software has been developed to operate on data from heterogeneous sources, such as DomCop and Google CrUX, to assess the validity, distribution and anomalies present in certificate chains, highlighting differences in the presence of SCT, performance and privacy impacts, and validation processes.

Furthermore, by means of a critical comparison supported by graphs and statistical analysis, the requirements of the CA/Browser Forum and the standards defined by the RFCs are compared, with the aim of identifying possible areas of improvement in the current standards.

Finally, the thesis concludes with a reflection on possible future developments of DigitalCertiAnalytics and future prospects in the field of digital certificates, including a brief analysis of post-quantum technologies. The results underline the importance of accurate and transparent certificate management to ensure resilient and secure information systems, contributing significantly to the protection of digital infrastructures.

# Acknowledgements

I sincerely thank Professor Diana Berbecaru for her unwavering support and guidance throughout this research. Her mentorship from the very beginning made all the difference.

My heartfelt thanks to my family, whose affection and support, as well as financial support, have been significant in reaching this milestone.

I would like to express my deep gratitude to my wonderful and beloved fiancée Francesca, whose unconditional love, support and positivity have enlightened and made every step of my journey possible. Thank you for sharing the joys and laughter with me, for being a pillar of moral support and for helping me find happiness and motivation in the most difficult moments. Your unwavering trust and constant presence allowed me to overcome every obstacle, giving me hope and determination. For this I will be forever grateful to you.

My sincere thanks also go to my fiancée's family, who in times of discouragement as well as in times of joy, were present, making us feel part of a large and affectionate family.

I thank all my university colleagues who in their own way have contributed to this achievement and enriched this journey.

And I thank everyone who believed even a little in me.

# Contents

<b>List of Figures</b>	10
<b>List of Tables</b>	14
<b>1 Introduction</b>	15
<b>2 Background</b>	17
2.1 Understanding PKI: Definition and Its Role in Digital Security . . .	17
2.2 Fundamentals of Public and Private Key Cryptography . . . . .	19
2.2.1 Key Generation and Secure Key Management . . . . .	20
2.3 Certification Authorities and PKI Hierarchical Structures . . . . .	20
2.3.1 Certificate Validation Process . . . . .	22
2.3.2 How the Root CA is pre-installed and managed . . . . .	23
2.4 Trust Model in PKI Architectures . . . . .	23
2.4.1 Hierarchical Trust Models . . . . .	23
2.4.2 Peer-to-Peer Trust Model . . . . .	27
2.4.3 Network Trust Model . . . . .	28
2.4.4 Hybrid Trust Model . . . . .	29
2.5 PKI Types: Public vs. Private . . . . .	30
2.5.1 Public PKI: Global Certification Framework . . . . .	31
2.5.2 Private PKI: Enterprise Certification Infrastructure . . . . .	31
2.5.3 Fundamental Differences between Public and Private PKI . . . . .	32
2.6 Public Key Certificate . . . . .	33
2.6.1 General Structure of X.509 Certificates . . . . .	33
2.6.2 Version . . . . .	33
2.6.3 Serial Number . . . . .	34
2.6.4 Signature Algorithm . . . . .	34

2.6.5	Issuer	34
2.6.6	Validity	34
2.6.7	Subject	35
2.6.8	Subject Public Key Info	35
2.6.9	Certificate Extensions	35
2.7	Certificate Revocation Mechanisms	39
2.7.1	Overview of Revocation Mechanisms	39
2.7.2	Online Certificate Status Protocol (OCSP)	39
2.7.3	OCSP Stapling	40
2.7.4	OCSP Must Staple	40
<b>3</b>	<b>Certificate Transparency version 2.0</b>	<b>41</b>
3.1	Rationale Behind Certificate Transparency v2.0	42
3.2	Merkle Tree Hash	42
3.3	Submitters	44
3.3.1	TLS servers	45
3.4	Algorithm Agility	46
3.5	TLS Client Validation and Verification	46
3.5.1	Receiving and Validating SCTs	46
3.5.2	Requesting Inclusion Proofs	46
3.5.3	Validation of Inclusion Proofs	46
3.6	Monitor	47
3.7	Auditing	47
3.8	Understanding Certificate Transparency in Real-World Applications	48
3.8.1	TLS Connection Process with Certificate Transparency	48
3.8.2	Exploring Google's Certificate Transparency Verification Tool	54
3.8.3	Observations	55
<b>4</b>	<b>DigitalCertiAnalytics: Software Architecture and Analysis</b>	<b>56</b>
4.1	DigitalCertiAnalytics: System Architecture and Key Features	58
4.1.1	Core Components of DigitalCertiAnalytics	59
4.1.2	Integration of ZGrab2 for Advanced Network Scanning	60
4.1.3	Relational Database Persistence Layer	61
4.1.4	Zlint: Compliance Verification Tool	61
4.1.5	Non-Relational Database Persistence Layer	64

4.1.6	SSLyze: Certificate Chain Verification . . . . .	65
4.1.7	Certificate Revocation Status Checking . . . . .	66
4.1.8	Data Visualization and Chart Generation . . . . .	68
4.2	User Interaction with DigitalCertiAnalytics . . . . .	68
4.3	System Specifications for Zgrab2 Execution . . . . .	70
<b>5</b>	<b>Digital Certificate Analysis Using the DomCop Dataset</b>	<b>71</b>
5.1	Overview of Leaf Certificates . . . . .	73
5.1.1	Distribution of Certificate Validation Levels . . . . .	75
5.1.2	Validity Duration Distribution and Expiry Trends . . . . .	75
5.1.3	Leaf Certificate Serial Number Analysis . . . . .	78
5.1.4	Leaf Certificate Chain Length Analysis . . . . .	79
5.2	Analysis of X.509 Leaf Certificate Extensions . . . . .	80
5.2.1	Analysis of Basic Constraints Extension . . . . .	80
5.2.2	Analysis of Authority Key Identifier Extension . . . . .	81
5.2.3	Analysis of Subject Key Identifier Extension . . . . .	81
5.2.4	Analysis of Key Usage Extension . . . . .	82
5.2.5	Analysis of Extended Key Usage Extension . . . . .	83
5.2.6	Analysis of Certificate Policies Extension . . . . .	85
5.2.7	Analysis of Subject Alternative Name (SAN) Extension . . . . .	87
5.3	Analysis of Algorithms and Cryptographic Security of Leaf Certificates	89
5.3.1	Distribution of Signature and Key Algorithms in Leaf Certificates . . . . .	89
5.3.2	Comparison of Self-Signed and CA-Signed Certificates . . . . .	91
5.3.3	Analysis of Certificate Signature Validity . . . . .	92
5.4	Revocation Status Checking in Leaf Certificates . . . . .	94
5.4.1	Analysis of Authority Information Access (AIA) . . . . .	94
5.4.2	Certificate Revocation List (CRL) Checking . . . . .	96
5.4.3	Online Certificate Status Protocol (OCSP) Checking . . . . .	97
5.4.4	Analysis of OCSP Stapling . . . . .	98
5.4.5	Analysis of OCSP Must-Staple . . . . .	99
5.5	Analysis of Signed Certificate Timestamps (SCTs) . . . . .	99
5.6	Overview of Intermediate Certificates . . . . .	102
5.6.1	Validity Duration Distribution and Expiry Trends . . . . .	103

5.7	Analysis of X.509 Extensions in Intermediate Certificates . . . . .	105
5.7.1	Analysis of Basic Constraints Extension . . . . .	105
5.7.2	Analysis of Key Usage Extension . . . . .	106
5.7.3	Analysis of Authority Key Identifier Extension . . . . .	107
5.7.4	Analysis of Certificate Policies Extension . . . . .	107
5.7.5	Analysis of Extended Key Usage Extension . . . . .	109
5.7.6	Analysis of Subject Key Identifier Extension . . . . .	110
5.7.7	Analysis of Authority Information Access . . . . .	110
5.7.8	Analysis of Certificate Revocation List . . . . .	111
5.8	Distribution of Signature and Key Algorithms in Intermediate Certificates . . . . .	112
5.9	Overview of Root Certificates . . . . .	113
5.9.1	Distribution of Validity Duration and Expiry Trend . . . . .	116
5.10	Analysis of X.509 Extensions in Root Certificates . . . . .	117
5.10.1	Analysis of Basic Constraints Extension . . . . .	117
5.10.2	Analysis of Key Usage Extension . . . . .	118
5.10.3	Analysis of Extended Key Usage . . . . .	119
5.10.4	Analysis of Subject Key Identifier . . . . .	120
5.10.5	Analysis of Authority Key Identifier . . . . .	120
5.10.6	Analysis of Certificate Policies . . . . .	121
5.11	Distribution of Signature and Key Algorithms in Root Certificates .	121
5.12	Zlint Analysis for Leaf Certificates . . . . .	122
5.13	Compliance Analysis Against CA/Browser Forum Requirements . .	125
5.13.1	Comparison of Leaf TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.7.6) . . . . .	125
5.13.2	Comparison of Intermediate CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.6.1) . . . . .	131
5.13.3	Comparison of Root CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.1.2) . . . . .	133
<b>6</b>	<b>Digital Certificate Analysis Using the Google CrUX Dataset</b>	<b>135</b>
6.1	Overview of Leaf Certificates . . . . .	137
6.1.1	Validity Duration Distribution and Expiry Trends . . . . .	138
6.1.2	Leaf Certificate Serial Number Analysis . . . . .	140
6.2	Compliance Analysis of Google CrUX Certificates Against CA/Browser Forum Requirements . . . . .	141



6.2.1	Comparison of Leaf TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.7.6) . . . . .	141
6.2.2	Comparison of Intermediate CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.6.1) . . . . .	147
6.2.3	Comparison of Root CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.1.2) . . . . .	149
6.3	Comparison Analysis Between the DomCop and Google Datasets . . . . .	150
<b>7</b>	<b>Conclusions and Future Directions</b>	<b>152</b>
	<b>Bibliography</b>	<b>154</b>
<b>A</b>	<b>User's Manual</b>	<b>157</b>
A.1	Installation Guide for DigitalCertiAnalytics . . . . .	157
A.1.1	Data Analysis Using ZGrab2 . . . . .	159
A.2	Launching DigitalCertiAnalytics . . . . .	159
A.2.1	Usage Examples . . . . .	160
A.3	ZGrab2 Installation Guide . . . . .	161
A.3.1	Installation on macOS . . . . .	161
A.3.2	Installation on Windows . . . . .	161
A.3.3	Installation on Ubuntu . . . . .	162
A.4	Zlint Installation Guide . . . . .	162
A.4.1	Installation on macOS . . . . .	163
A.4.2	Installation on Windows . . . . .	163
A.4.3	Installation on Ubuntu . . . . .	164
A.5	SSLyze Installation Guide . . . . .	164
A.5.1	General prerequisites . . . . .	164
A.5.2	Installation on macOS . . . . .	164
A.5.3	Installation on Windows . . . . .	165
A.5.4	Installation on Ubuntu . . . . .	165
<b>B</b>	<b>Developer's Reference Guide</b>	<b>166</b>
B.1	Directory and File Structure . . . . .	166
B.2	Log Management . . . . .	167
B.3	MongoDB Database Dumping . . . . .	167
B.4	Final Developer Observations . . . . .	167

# List of Figures

2.1	Diagram of the architecture of a One-Tier PKI with a single root CA. Taken from [1]. . . . .	24
2.2	Diagram of the architecture of a Two-Tier PKI with Root CA and Issuing CAs. Taken from [1]. . . . .	25
2.3	Diagram of the architecture of a Three-Tier PKI with Root CA, Intermediate CAs, and Issuing CAs. Taken from [1]. . . . .	26
2.4	Illustration of the Peer-to-Peer Trust Model in a PKI system. . . . .	28
2.5	Trust List (Network Trust) Model overview. . . . .	29
2.6	Hybrid Trust Model overview. . . . .	30
2.7	Comparison between Public and Private PKI, adapted from [1]. . . . .	32
3.1	Representation of a Merkle Hash Tree, where each leaf is a certificate, taken from the article [2]. . . . .	43
3.2	Proof of inclusion in a Merkle Hash Tree, illustrating the process of verifying a certificate's inclusion using hashes, taken from the article [3]. . . . .	44
3.3	TLS Connection Establishment Process: The client sends a ClientHello message requesting CT information, followed by the server certificate with Signed Certificate Timestamps (SCTs). . . . .	49
3.4	Extracting SCTs from the Certificate: Python code parses the certificate to identify and extract the SCTs as shown. . . . .	49
3.5	SCT Extraction Result: The extracted SCTs from the certificate of www.polito.it are displayed, confirming the certificate transparency information. . . . .	50
3.6	Step 1 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key. . . . .	51
3.7	Step 2 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key. . . . .	52

3.8	Step 3 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key. . . . .	52
3.9	Requesting Inclusion Proof: If not provided by the server, the client interacts with the public log to request the inclusion proof using the <code>get-proof-by-hash</code> API. . . . .	53
3.10	Verifying Inclusion Proof: The client reconstructs the Merkle tree and verifies the inclusion proof by comparing the root hash from the Signed Tree Head (STH). . . . .	54
3.11	SCT Verification Result: The results from using the Google Certificate Transparency tool show that all three SCTs in the certificate of <code>www.polito.it</code> are valid. . . . .	55
4.1	Distribution of issued certificates per country in the DomCop dataset.	58
4.2	Architecture of DigitalCertiAnalytics Tool for Large-Scale Certificate Analysis. . . . .	59
4.3	Illustration of the ZMap project, of which ZGrab2 and ZLint are components. . . . .	60
4.4	Code for sending an OCSP request to check the revocation status of a digital certificate. . . . .	67
4.5	Command line interface for interacting with DigitalCertiAnalytics. . . . .	69
5.1	ZGrab2 scan results for domains in the DomCop dataset. . . . .	71
5.2	Certificate Status Analysis. . . . .	72
5.3	Distribution of issued certificates per country. . . . .	73
5.4	Distribution of leaf certificates issued by various certification authorities. . . . .	74
5.5	Distribution of Validation Levels across Leaf Certificates: DV, OV, and EV. . . . .	75
5.6	Distribution of the period of validity of certificates. . . . .	77
5.7	Deadlines for the distribution of leaf certificates. . . . .	78
5.8	Distribution of serial numbers in leaf certificates, showing compliance with length and uniqueness requirements. . . . .	79
5.9	Distribution of Basic Constraint Extension. . . . .	80
5.10	Key Usage in Extensions Field. . . . .	82
5.11	Critical vs Not Critical Key Usage in Extensions Field. . . . .	83
5.12	Critical vs Not Critical Extended Key Usage in Extensions Field. . . . .	84
5.13	Results related to the Certificate Policies (CP) extension in leaf certificates, showing the distribution of policy identifiers for different validation levels (DV, OV, EV) as required by the CA/Browser Forum Baseline Requirements. . . . .	85

5.14	Critical vs Not Critical Certificate Policies Extension. . . . .	86
5.15	Critical vs Not Critical Extensions of Subject Alternative Names. . . . .	88
5.16	Distribution of Key and Length Algorithms. . . . .	89
5.17	Signature Algorithm Used. . . . .	90
5.18	Representation of the percentage of self-signed certificates compared to certificates signed by a certification authority (CA). . . . .	92
5.19	SSLyze result for www.polito.it - Part 1 . . . . .	93
5.20	SSLyze result for www.polito.it - Part 2 . . . . .	93
5.21	Distribution of AIA URLs in the leaf certificates. . . . .	95
5.22	Critical vs Not Critical Extensions of Authority Information Access. . . . .	96
5.23	Critical vs Not Critical Extensions of Certificate Revocation Lists. . . . .	97
5.24	Distribution of OCSP response statuses across certificates, highlighting the proportion of valid, revoked, and unknown statuses. . . . .	98
5.25	SCT Count per Certificate. . . . .	100
5.26	Top Signed Certificate Transparency (SCT) Logs. . . . .	101
5.27	Top SCT Log Operators. . . . .	101
5.28	Distribution of various issuers for Leaf certificates across the analyzed dataset. . . . .	102
5.29	Distribution of intermediate certificates by country, with the majority issued by the United States and Belgium. . . . .	103
5.30	Distribution of the period of validity of intermediate certificates. . . . .	104
5.31	Deadlines for the distribution of intermediate certificates. . . . .	104
5.32	Distribution of Basic Constraint Extension. . . . .	105
5.33	Key Usage in Extensions Field. . . . .	106
5.34	Critical vs Not Critical Key Usage in Extensions Field. . . . .	107
5.35	Comparison of Certificate Policies Extension: Critical vs Non-Critical in Intermediate Certificates. . . . .	108
5.36	Critical vs Not Critical Certificate Policies Extension. . . . .	108
5.37	Extended Key Usage in Extensions Field. . . . .	109
5.38	Critical vs Not Critical Extended Key Usage in Extensions Field. . . . .	110
5.39	Distribution of Authority Information Access (AIA) in Intermediate Certificates. . . . .	111
5.40	Distribution of Key and Length Algorithms. . . . .	112
5.41	Signature Algorithm Used. . . . .	113
5.42	Overview of the distribution of root certificates in the dataset. . . . .	114

5.43	Distribution of root certificates by country, with the majority issued by the United States and Great Britain. . . . .	115
5.44	Distribution of the period of validity of root certificates. . . . .	116
5.45	Deadlines for the distribution of root certificates. . . . .	117
5.46	Key Usage in Extensions Field. . . . .	118
5.47	Critical vs Not Critical Key Usage in Extensions Field. . . . .	119
5.48	Distribution of Extended Key Usage (EKU) in Root Certificates. . .	120
5.49	Distribution of Key and Length Algorithms. . . . .	121
5.50	Signature Algorithm Used. . . . .	122
5.51	Distribution of Lints results. . . . .	123
5.52	Most Common Lints Found in Digital Certificates. . . . .	124
5.53	Distribution of Issuers with the Highest Number of Lint Errors. . .	125
6.1	Google BigQuery query for extracting European domains based on popularity ranking. . . . .	136
6.2	ZGrab2 scan results for domains in the Google dataset. . . . .	136
6.3	Status analysis of certificate extraction from European domains. . .	137
6.4	Distribution of certificates by issuer country in the dataset, showing a dominance of U.S. entities, with Let's Encrypt leading the distribution. . . . .	138
6.5	Validity duration distribution of certificates issued before September 2020, with a portion exceeding the 39-month validity period. . . . .	139
6.6	Validity duration distribution of certificates issued after September 2020, with almost all certificates adhering to the 398-day validity limit. . . . .	139
6.7	Expiration trend of certificates, with significant peaks observed in December 2024 and January 2025, suggesting a cascading issuance schedule. . . . .	140
6.8	Analysis of Serial Number field, highlighting compliance and anomalies such as negative serial numbers and duplicates. . . . .	141

# List of Tables

2.1 Comparison between One-Tier, Two-Tier, and Three-Tier Certificate Authorities. . . . .	27
--	----

# Chapter 1

## Introduction

In a rapidly digitizing world and global interconnection, the security of internet communications is of paramount importance for data security, and the integrity of information exchanged daily. Infrastructures based on cryptographic protocols, in particular TLS, play a key role in ensuring the confidentiality of communications and their protection against external attacks. The heart of this security system is formed by digital certificates, compliant with the X.509 standard, which allow authentication of the identity of servers and users, generating a chain of trust that starts from a Root Certificate Authority (CA) recognized and terminated with certificates issued to the final entities.

In recent years, due to the increasing traffic and massive adoption of online services, the exponential increase in traffic has brought to light problems related to the validation process and revocation of certificates. Events such as the compromise of authoritative CAs, which led to the exchange of fraudulent certificates, thus marking the end of trust in safety ecosystems, make clear the need to improve and make more transparent the verification mechanisms adopted by browsers and operating systems. The management of revocation is a crucial point, if on the one hand you require the speed in reporting the compromise of a CA to prevent and avoid man-in-the-middle attacks, on the other hand the complexity and fragmentation of verification methods (as the use of CRL and OCSP) still represent a weak point in the system.

This thesis is part of a research framework aimed at understanding in depth the dynamics of digital certificate validation and to analyse, on a large scale, the certificates in circulation. Particular attention is devoted to the implementation of DigitalCertiAnalytics, an innovative tool developed for the collection, verification and analysis of X.509 certificates from millions of domains. The tool not only allows you to examine the technical properties of certificates, such as extensions and signature algorithms, but also to check that they meet the minimum requirements set by international standards and the CA/Browser Forum [4] [5] [6].

The work of this thesis is divided into two main branches. The first part is dedicated to the theoretical analysis of the foundations of the Public Key Infrastructure (PKI), deepening the hierarchy of Certificate Authorities, the mechanisms

of authentication and revocation, as well as the trust model that supports the entire architecture. In particular, the methods of certification of public keys and revocation processes are examined, with a specific focus on the phenomenon of Certificate Transparency. The latter introduces a public monitoring and verification dimension, making it possible to control in real time the issuance of certificates through public append-only logs. This approach represents a significant advance in preventing the distribution of fraudulent certificates and rebuilding the chain of trust.

The second part of the thesis describes the practical implementation of the DigitalCertiAnalytics tool, developed specifically to conduct analyses on large datasets. By integrating advanced tools such as ZGrab2 [7] for certificate retrieval and specific modules for the analysis of information contained in certification chains [8] [9]. The tool was tested on two separate datasets, one international and one European, each composed of 10 million domains, allowing a thorough comparison of the different panoramas and configurations adopted by the issuers.

The organization of the thesis is created to guide the reader following the line which starts with the theoretical bases on PKI and mechanisms of validation and revocation, continues with real implementation of the tool, and ends with analysis of experimental results.

The ultimate objective of this work is twofold: on the one hand, to provide an in-depth analysis of the current state of digital certification infrastructures, highlighting their gaps and on the other hand, presenting DigitalCertiAnalytics as a support tool for network administrators and developers, capable of operating on very large datasets and providing data useful for the optimization of security policies.



# Chapter 2

## Background

### 2.1 Understanding PKI: Definition and Its Role in Digital Security

In this digital age, the degree of security of online communications is one of the issues that is becoming increasingly important, especially due to the rapid spread of business via internet and the growing need to turn to online services. The assurance of security and integrity of operations is supported by encryption systems based on the mechanism of so-called public key infrastructures (PKIs). A complex infrastructure coordinated by the simultaneous cooperation of several units in order to provide security services such as authentication, integrity, confidentiality and non-repudiation through the use of public key encryption techniques. The focal point on which it lays down the foundations is a two-key model, a public key, freely disclosed and a private key, which must remain secret and closely linked to the public key. Key elements of PKI are a trusted entity, the Certification Authority (CA), which issues digital certificates, and the digital certificate, also called Public Key Certificate (PKC), which gives authenticity to an entity's identity (website, organisation, individual), by linking a public cryptographic key to that entity which it can use either to encrypt data (messages, documents etc.) or to verify a digital signature affixed to a given document. Through the digital certificate, the Certification Authority digitally signs the content, through its private key and ensuring that the public key has not been replaced and that the identity of the certified entity is that declared by it.

In communication between two entities that wish to communicate securely, PKI takes over to ensure that the data exchanged is confidential and comes from the entity actually declared. For example, if a user wishes to send a secret message to another user, he can encrypt it with a public key of that other user, only the recipient, having the private key corresponding to that public key, will be able to decrypt the message. The process establishes the tacit agreement that only the legitimate recipient can read messages sent to him. In addition, if the sender wants the recipient to be able to verify the message sent to him, he can digitally sign it with his private key. This way, anyone who has the public key of the sender

suitable for verification can verify that the signature is authentic and that the message has been signed by him.

From the above, it is understood that a digital certificate is valid if issued by a certification authority and if the data contained in it attest to the integrity and authenticity of the message by ensuring immutability from the creation of the message itself to its delivery. But, note, all this is only possible if the public key and private key association always keep secret, since the subject itself is no longer free to protect the confidentiality of its secret key, Each body may use the certificate to impersonate it. Therefore, efficient management of certificate revocation is necessary, the Certification Authority publishes a Certification Revocation List (CRL) that each entity can consult when it can to verify the validity of the certificate, or there are protocols such as the Online Certificate Status Protocol (OCSP), which allow you to know in real time the status of a given certificate.

The completion of the PKI authentication process requires the certificate store, which is a database that stores all digital certificates issued by the CA and other information. Repositories allow anyone to search for valid certificates and are typically accessed via standard protocols such as LDAP or HTTP. In a PKI, trust is constituted by a system of Certification Authorities that are linked together in a hierarchy, with the top "root CA" which constitutes the main trusted entity. The other CAs, called intermediate CAs, are subordinate to "root" and can issue certificates on behalf of the "root". The hierarchical system ensures a management of trust that is in its way scalable and secure, since the "root" could issue certificates to subordinate CAs and these, in turn, release them to the final entities. A PKI has applications in the most varied fields.

For example, on secure websites such as those using HTTPS, the site's digital certificate contains the server's public key, browsers check the validity of the certificate to establish a secure connection with the site. Similarly, PKI is used for electronic signatures, for the authentication of devices in corporate networks, to protect data during communications.

Another very important entity of the PKI infrastructure is the Registration Authority (RA), which has the task of verifying the identity of the applicant and approving or rejecting the application for certification, acting as an intermediary between the certificate applicant and the Certification Authority (CA). Thorough identity checks before the generation of certificates increase trust between stakeholders [10].

Therefore, the main purpose of PKI is to allow new certificates to be issued and end users to check the validity of digital certificates during communications to establish a secure connection in the network. For this reason, the PKI incorporates various functions such as user registration, key generation, key registration, key recovery, key update, management of CA root certificates, cross certification, initialization, support of different trust models, certificate revocation and distribution of revoked certificate information [6], [11].

The RFC 5280 [6] describes all the processes crucial for registration, initialization, certification and other important steps in managing a PKI. In particular, the

registration process requires that the interested party present itself to the Certification Authority (CA) for the issuance of the certificate, which can be performed directly by the CA or the Registration Authority. Next, the initialization phase aims to configure the client with the public keys of the respective certified CA and its own key pair. Certification is an operation in which a CA issues the certificate for the benefit of your public key and makes it available to the client or in a repository. While, the update of the key pair, which must be renewed periodically, replacing the old pair with a new one, of which a new certificate must be issued, to mark the change. This mode provides increased key security over time, thus avoiding the system being exposed to possible attacks that may result from outdated keys. Another fundamental step is the one that takes place with the revocation request, which consists in the fact that an authorized person informs the Certification Authority that the revocation of a certificate must be made by motivating its cause.

Finally, the process called cross certification which consists in the reciprocal exchange of information between two CAs with the issue of a cross certificate. A cross-certificate is a certificate that one CA issues to the other in order to ensure mutual recognition of certificates issued by each Authority. This process is particularly useful in complex environments where many different entities have to cooperate with each other in a secure manner [6].

## 2.2 Fundamentals of Public and Private Key Cryptography

Asymmetric encryption, also known as public key encryption, is a basic concept for the protection of digital communications and the security of information resources such as the Public Key Infrastructure (PKI), as it is based on the use of two distinct keys, the public key and the private key, which are mathematically related but not coincident. Each user or participant in this system has a pair of such keys, both of them perform a particular and well-defined task.

- **Public key:** It is a key that can be freely distributed and shared by anyone. Its main function is to encrypt data directed to the holder of the corresponding private key. Being accessible to all, it is possible to use it whenever there is a need to send an encrypted message in a secure way, while only those who will be in possession of the private key involved will have the right to decrypt the encrypted message.
- **Private key:** It is a key that must be kept secret and guarded. It is used to decrypt messages that have been encrypted with the corresponding public key and to digitally sign the message itself, to prove its authenticity and integrity.

### 2.2.1 Key Generation and Secure Key Management

The key generation mechanism involves the use of complex mathematical algorithms whose characteristic is to link the two keys definitively to each other, making it impractical to execute the second one from the first.

- **RSA**: a popular algorithm for generating public and private keys, employing complex mathematical operations based on the choice of very large primes; the public key is the product of two prime numbers  $p$  and  $q$ ; The private one depends on the calculation of an exponent that is related by a mathematical relation to the exponent itself. This mechanism ensures that only the holder of the private key is able to decipher data encrypted with the public key [12].
- **ECC** (Elliptic Curve Cryptography): this algorithm is one of the latest alternatives to RSA encryption and is based on the use of elliptical curves for public and private key generation; The security of ECC is based on the difficulty of the discrete logarithm problem in elliptic curves and is peculiar for its adaptability to devices with limited resources, such as smartphones and IoT devices [13].

To preserve the secrecy of private keys, it is advisable that wherever they are kept, you take the greatest precautions, since whoever comes into possession of them can use them to decrypt all encrypted messages. Tools such as Hardware Security Modules (HSMs) and devices like smart cards and USB tokens are generally used.

HSMs are hardware devices that store and generate keys, preventing unauthorized access. Similar properties have the smart card and USB token, which generate and store keys and submit them to authentication before they can be used.

## 2.3 Certification Authorities and PKI Hierarchical Structures

The primary task of a Certification Authority (CA) is to ensure that public keys are securely associated with the identities declared. It is the practice for the Certification Authority to verify the identity of the applicant, issue a certificate to that public key, and place its digital signature on the certificate, and such signature constitutes proof of authenticity and integrity for the issued certificate.

The CA is therefore an entity that behaves as a "trusted third party", that is, a party in which users place their trust to determine whether a public key actually belongs to the declared entity. A certificate signed by the CA is considered valid and therefore authentic if the issuing CA has good reputation.

The Public Key Infrastructure (PKI) is based on a hierarchical structure for the management and distribution of digital certificates. The Hierarchical Trust Model is the most common structure, but later we will also analyze different types of Trust

Models present. The hierarchy of Certificate Authorities (CA) is a particularly important element for this infrastructure, since by ensuring a system of trust, it makes possible the generation of certificates, their verification and/or revocation.

The CA hierarchy is organized so that each lower level can rely on the reliability of the issuer above itself, in such a way that a chain of trust is established from the CA Root to the final certificates, which are issued to subjects such as, for example, web servers, devices or end users.

The main levels of a CA hierarchy are:

1. **Root CA** (Root Certificate Authority): The Root CA is the highest layer of the hierarchy, which constitutes a master CA whose certificate is self-signed (that is signed by the same CA). The CA Root is particularly important because due to its self-signature it is the starting point of the chain, becoming essential to establish trust in the whole system to allow validation of subsequent emissions in the certification chain. Issues certificates for Intermediate CA and sometimes for end-entity (final certificates, such as those for servers). However, the practice prohibits final certificates from being issued directly by the Root CA, but through other intermediate CAs. Also, being the root CA the main key that seals all trust in the whole system, it is subject to the most stringent security requirements.
2. **Intermediate CA** (Intermediate Certificate Authority): The Intermediate CA occupies a lower position in the hierarchy than the Root CA from which it was issued. The use of Intermediate CA represents a best security practice since it involves a reduction in the risks related to the compromise of the Root CA. This, in fact, does not suit the direct issuance of certificates for the final entities, and therefore the Intermediate CA acts as "bridge" between the Root CA and the Issuing CA (or other certificates placed in a lower level). The Intermediate CA issues certificates for other Intermediate CAs (if necessary) or directly for final certificates (such as server, user, etc.). The Intermediate CA is indeed well guarded, but does not enjoy the same complete protection of the Root CA, and in case of compromise the resulting damage is limited compared to what would result from an action directed against the Root CA.
3. **Issuing CA** (Issuing Certificate Authority): Issuing CAs are a particular kind of CA Intermediary solely responsible for issuing certificates to end entities, such as web server, e-mail or end-user certificates. These CAs are required to verify the identity of the entities for which they issue certificates, for example they may issue an SSL certificate for a website. Although, the lower sensitivity compared to the Root CA, their security is of particular importance, since their compromise could, however, lead to a fraudulent issuance of certificates.
4. **Leaf Certificates** (End-Entity Certificates): Leaf certificates, the final certificates, are issued to actual end users or end systems (such as web servers, devices, or users) and are signed by the issuing CA or one of the intermediate CAs in the PKI. They are presented to the guarantee of authenticity and security of digital communications and are used in authentication, encryption,

digital signature, etc. Unlike service certificates, a compromise of the final certificates, even if serious, would not result in a threat to the entire PKI infrastructure, but it can nevertheless cause serious damage, for example the interception of secure communications.

In the current context, the Root CA is pre-installed in many widely used operating systems and browsers, in such a way that it facilitates the user in the validation of the digital certification and immediately returns to him the advantage of these certifications without resorting to manual configurations. It is an approach necessary to promote and facilitate the use of secure technologies, such as HTTPS, authentication of digital identities or privacy in online communications.

In practice, most modern operating systems and most commonly used browsers (for example Chrome, Firefox, Safari, Internet Explorer, etc.) already include a list of trusted CA Root. When a client receives a digital certificate, their operating system or browser will immediately start searching their list of trusted CA Root to verify that the root CA obtained by tracing the chain of certificates is authentic and valid.

This list is usually automatically updated by the manufacturers of operating systems or applications, for example by using applications such as Windows Update, macOS Update, or other forms of regular system updates. This helps to keep digital connections safe and usable at all times, as the pre-installation of Root CA reduces the dangers of tampering and unreliable CA.

For example, Windows uses the Windows Certificate Store, where it keeps the certificates of the Root. Each system update can import new root certificates or delete compromised or obsolete ones. MacOS also includes a preconfigured set of Root CA which can be viewed and adjusted via the certification management portal. Whereas, Linux, keeps the root certificates in the system certificate package, which you can easily update by using the `update-ca-certificates` command or through update package managers (e.g. `yum`, `apt`, `rpm`, `pacman`).

### 2.3.1 Certificate Validation Process

For the process of validation of digital certificates, it is essential that a Root CA is present in the operating system, only then you can perform the verification of authenticity of the certificates that arrive at the system. In the most common situation, that of an HTTPS connection with a website server, the system must ensure that the certificate has been issued by a CA to which it must give its full confidence. To do this, it starts by tracing the whole chain of certificates, from the leaf up to the Root CA (root). If the certificate is valid and the chain of trust is intact, the certificate can be considered authentic, and the verification is a success. This process ensures, with the power of encryption, that the entity issuing the certificate is the one it declares itself to be, and the user is saved from a possible man-in-the-middle. If the system does not see its root CA, or if the chain of trust is broken, the certificate is refused, and the user is pointed out to the possibility of communication insecurity, or in some cases it is not even allowed [14].

### 2.3.2 How the Root CA is pre-installed and managed

The management of Root certificates within the operating system is pursued through a centralized distribution and update system, formally structured in various phases, such as the following.

- **Initial deployment and configuration:** Upon installation of the operating system, a pre-configured package of root certificates is distributed, which includes widely recognized and trusted worldwide root certifications, and which has been selected on the basis of reliability and safety criteria, often verified by independent authorities. Examples of pre-installed root certifications include DigiCert, GlobalSign and VeriSign.
- **Periodic updates:** Preconfigured CA root is periodically renewed by operating system updates, where unusable or compromised ones can be removed (as well as new certifications added). This is done automatically, without any direct user intervention, so that the digital certificates they sign can continue to maintain a high degree of validity and security over time.
- **Removal of compromised CAs:** When a CA Root has been compromised, or no longer corresponding to the required security standards, it should be immediately removed from the trusted root certification records and not allow other certifications to be disseminated or authenticated [13].

## 2.4 Trust Model in PKI Architectures

In the design of a Public Key Infrastructure (PKI), one of the most delicate points is the definition of the structure of the trust model, which governs the issuance of digital certificates and the security conditions of the keys. The different PKI architectures can be organized in a more or less complex way depending on the level, complexity and security requirements of the environment in which they are to operate.

### 2.4.1 Hierarchical Trust Models

The Hierarchical Trust Model is the most common type of model implemented in PKI, where the main configurations are one-tier, two-tier and threeTier and have as many different combinations related to different degrees of separation between the entities involved in the process of issuing certificates [15].

#### One-Tier Hierarchy

One-Tier PKI involves a single Certificate Authority (CA) capable of issuing certificates for end users or devices (clients, web servers, etc.). The root CA, as the certificate issuing center for all end users or devices, forwards requests and directly signs the certificates for all entities using the PKI system. It is a simple and easily implementable structure, but it involves significant security risks. First, the main flaw of this type of PKI is that the root CA is constantly in operation

and, therefore, is always exposed to the risk of its private key being compromised. Additionally, since all certificates are issued directly by the root CA, the entire system can be considered poorly scalable, making it suitable only for small-scale scenarios where the protection of the root CA is not a primary concern [1].

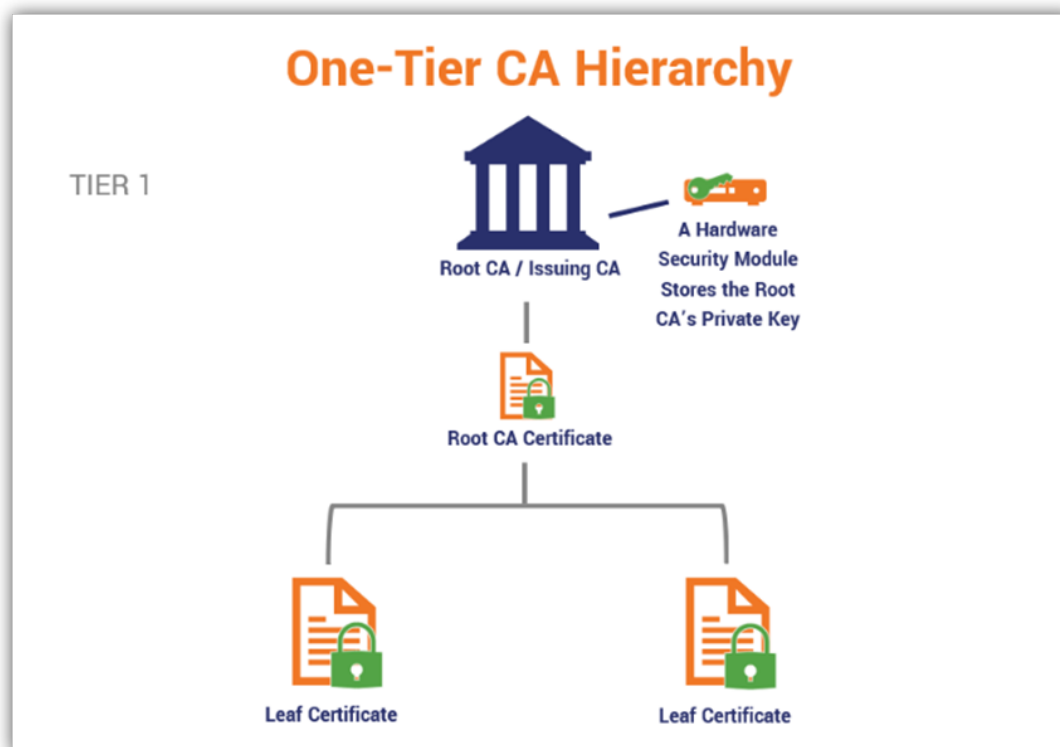


Figure 2.1. Diagram of the architecture of a One-Tier PKI with a single root CA. Taken from [1].

### Two-Tier Hierarchy

The Two-Tier model proposes a two-level structure with the Root CA and the Intermediate CAs, which are responsible for issuing the final certificates. With this approach, the Root CA is typically offline, meaning it is not directly connected to the Internet and is not directly involved in the certificate issuing processes, but is limited to signing the certificates of the Intermediate CAs, which are responsible for distributing certificates to the end entities. The Intermediate CAs can be online and thus able to issue certificates while maintaining the security of the Root CA. This setup therefore provides a good compromise between security and scalability, as it separates the Root CA from all daily operations and reduces the risk of exposing the private root key. This architecture also allows for more flexible certificate management, as the Intermediate CAs can be distributed across multiple levels, enabling the management of certificates on a large scale [1].



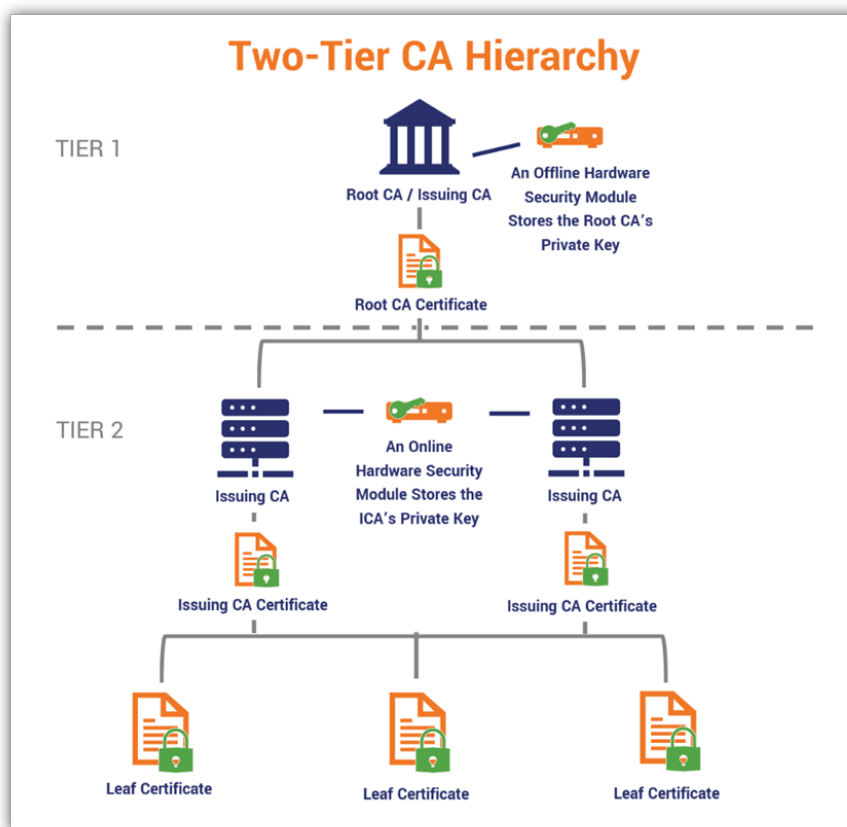


Figure 2.2. Diagram of the architecture of a Two-Tier PKI with Root CA and Issuing CAs. Taken from [1].

### Three-Tier Hierarchy

The Three-Tier model is the most complex and, precisely because of that, the most secure. In this model, the Root CA exclusively signs the Intermediate CAs, which in turn sign the Issuing CAs, which are therefore the only ones authorized to issue certificates to the public. In this structure, roles are strictly defined, as the Root CA only signs the Intermediate CAs, which impose their signature on the Issuing CAs, which, finally, are responsible for issuing the final certificates (users). As in the Two-Tier architecture, it is immediately apparent that the Root CA remains isolated from the network, thus protecting it from compromise. Furthermore, this model stands out for its maximum level of protection, resulting from the complete separation between the Root CA and the certificates issued to end users. In the case of a compromise of an Issuing CA or an Intermediate CA, the loss of trust would not affect the entire PKI system, as the Root CA is isolated and free from all daily operations. The Three-Tier PKI is, in short, suitable for environments where maximum security is required, such as large enterprises or government offices, which require advanced key protection and complex certificate management [1].

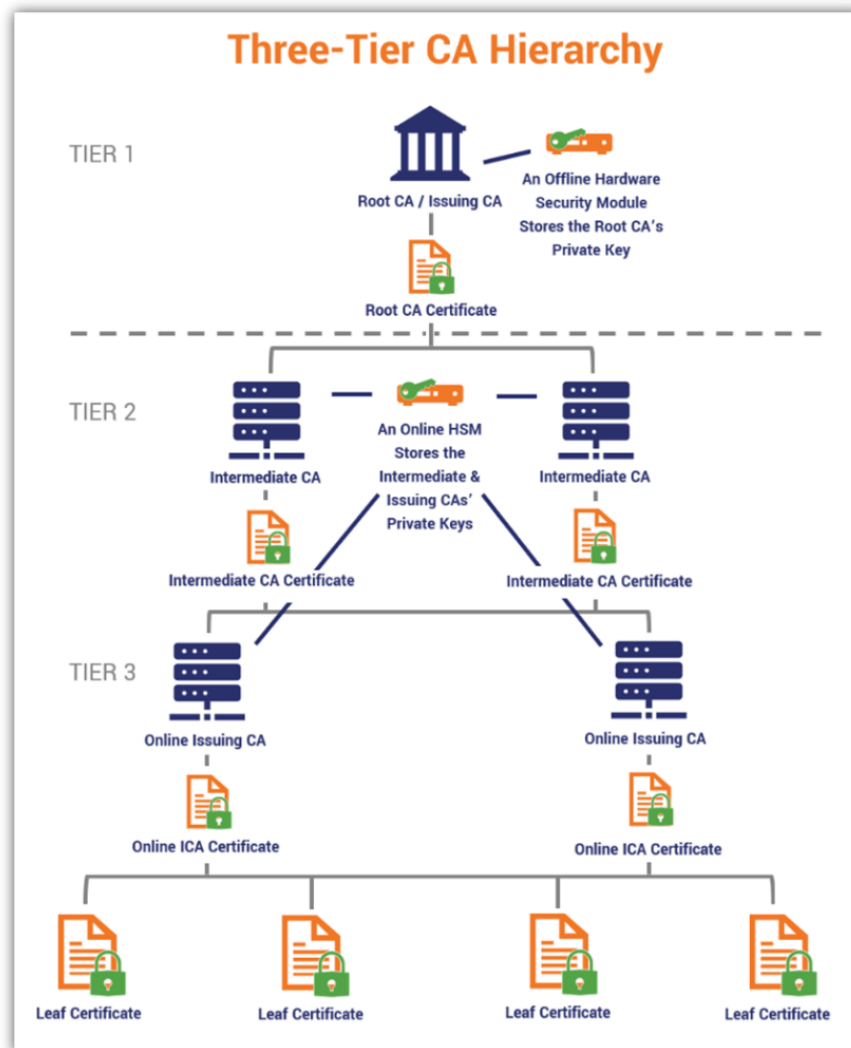


Figure 2.3. Diagram of the architecture of a Three-Tier PKI with Root CA, Intermediate CAs, and Issuing CAs. Taken from [1].

### Main differences and considerations

The primary distinguishing features of the two PKI models are that the Root CA is removed from the remaining system parts in the first model. In other words, it is not very secure, and even less complex, that the One-Tier PKI provides a rather poor protection mechanism for the Root CA on large-scale production sites. The Two-Tier PKI offers, on the other hand, a balance between security, flexibility, and scalability since it keeps the Root CA apart from daily operations. Three-Tier PKI introduces the highest possible security since upstream of the Root CA exists some rather complex structures that guarantee private key protection and great scalability.

There is great variability between Three-Tier, Two-Tier, and One-Tier models depending upon the security and scalability requirements of any particular case. Simpler architectures, for example the One-Tier, would come handy when

moderate security is sufficient. Three-Tier model is more or less meant for large enterprises or government offices requiring a certain degree of control over trust and protection of CA private keys.

Characteristics	One-Tier	Two-Tier	Three-Tier
CA Levels	Single level (Root CA)	Two levels (Root CA + Intermediate CA)	Three levels (Root CA + Intermediate CA + Issuing CA)
Root CA Security	Low (Root CA is online)	High (Root CA is offline)	Maximum (complete separation and protection of Root CA)
Scalability	Limited	Good	Excellent
Complexity	Low	Medium	High
Common Use	Small organizations	Medium-sized organizations	Government organizations or large enterprises

Table 2.1. Comparison between One-Tier, Two-Tier, and Three-Tier Certificate Authorities.

## 2.4.2 Peer-to-Peer Trust Model

The Peer-to-Peer Trust Model is based on the assumption that the two involved Certification Authorities are not in a subordinate or dependent relationship, but are considered entities of equal level. In this model, there is no root CA from which trust derives; instead, each CA can be considered independent, and users can rely on their local CA as a reference point for trust. Although the two CAs operate in separate trust domains, they can mutually certify each other through a process called cross-certification, which establishes a trust relationship between these distinct domains [16] [17].

Cross-certification is the process by which two CAs, operating in two separate trust domains, agree to extend mutual trust between them. There are two types of cross-certification

- Cross-certification within the same domain: When two Certification Authorities belong to the same domain, for example, both are internal CAs within an organization.
- Inter-domain cross-certification: Two Certification Authorities belonging to completely different domains.

Cross-certification can be unidirectional or bidirectional. In particular, in the case of unidirectional cross-certification, one CA certifies another CA, without the reverse occurring. In this case, there is a unilateral trust relationship where the

users of the certifying CA can trust the certified CA, but not vice versa. This structure is more commonly used in CA-level PKI implementations.

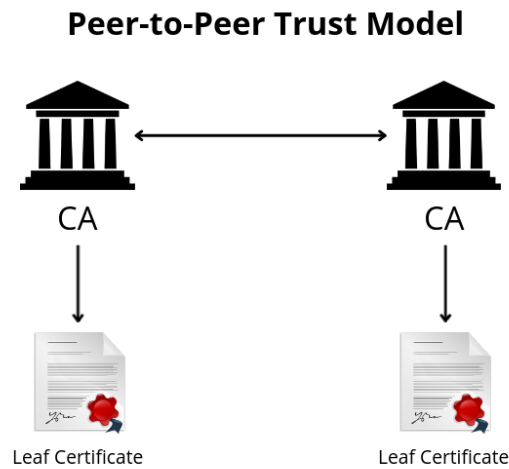


Figure 2.4. Illustration of the Peer-to-Peer Trust Model in a PKI system.

This Trust Model is characterized by a high level of flexibility, allowing for the extension of trust domains without the need for a centralized Root CA schema. This model is particularly suited to environments where multiple entities establish mutual trust relationships, such as in autonomous organizations. However, its management becomes challenging when numerous CAs are involved, making it difficult to control certification paths, which can lead to infinite loops or incorrect certification paths. Consequently, although this model is well-suited to small organizations, its limited scalability and management complexity make it less suitable for larger implementations [16] [17].

### 2.4.3 Network Trust Model

The model of Trust List, also known as Network Trust Model, is when users form trust in multiple Certification Authorities based on the pre-configured list of trusted public keys. It is more commonly applied in large-scale environments where there are multiple certificate issuers, and the user wants to set a trust with these entities in rapid and secure ways. Rather than involving a central trust authority in the system, users have a list of public keys from different CAs that are already put into their systems. The signature version of different CAs, when put into the verification list of the digital certificates presented while secure communications, can be accessed by each user in a system before trust is allocated to a CA. This gives the system simplicity since users will be easily able to trust more than one CA without the burden of convincing each one separately [16] [17].

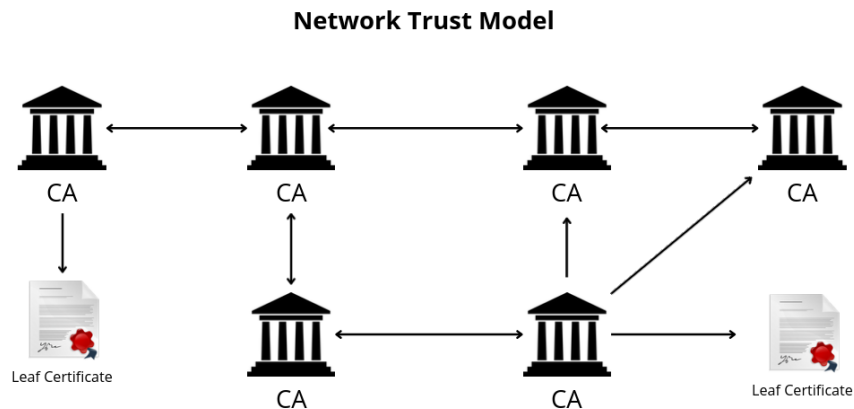


Figure 2.5. Trust List (Network Trust) Model overview.

However, despite its many advantages in terms of scalability and ease of implementation, the model hides some pitfalls and dangers. The primary concern is directly related to the counterfeiting attack. If an attacker were given the opportunity to introduce a fake certificate into the trust list, or to take control of a trusted CA in any way, it would follow that the security of the system would be jeopardized. This would pose a serious threat, especially because a public key, once placed in the trust list, offers no other possibility for revocation in the event it is compromised. Therefore, the lack of a practical mechanism for revocation increases the risk of long-term vulnerabilities.

Moreover, since the model requires the user to manage, potentially, a large number of public keys, it can become a significant burden for those managing the systems, requiring advanced technical knowledge to ensure the proper maintenance of the keys. If users do not manage the keys properly or configure the systems incorrectly, they expose themselves to greater possibilities of attacks. Therefore, the model is more suited to environments where simplicity and large-scale interoperability are prioritized over a strict level of security, as the complexity of managing multiple public keys can pose a challenge [16] [17].

#### 2.4.4 Hybrid Trust Model

The Hybrid Trust Model combines multiple PKI trust models, such as the hierarchical model and the cross-certified model, offering a more elastic and expandable solution to the problem of maintaining trust in the most complex PKI infrastructures. This type of model is particularly suitable for identifying conditions in which different organisations or departments need to establish selective relationships of trust between them, and shall, however, take care to keep separate other sections of their infrastructure [16] [17].

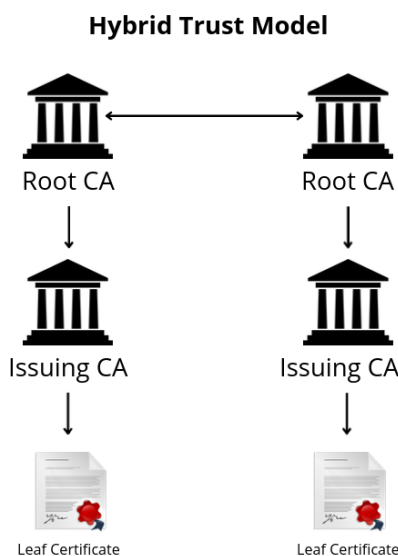


Figure 2.6. Hybrid Trust Model overview.

In this model it is no longer necessary for the Root CAs to be all located in the same domain. There may be, in fact, a link between the Root CAs belonging to different domains based on the reciprocity of a cross-certification, as shown in figure 2.6. Intermediate CAs, located right in the middle between root CAs and end users, place their trust only on the directly connected root CA, thus limiting the trust of certificates to the sector of the infrastructure adjoining it.

One of the main advantages is the greater flexibility, the possibility that it gives to organizations to manage with selection the various entities that will enter the network of trust, narrowing the field of confidence. Scalability is another advantage, where at the moment when an organization grows or at the same time it establishes new relationships with other external entities, through the process of cross-certification is possible to add new trusted domains without compromising the overall security of the system.

However, this model has some operational disadvantages. The initial configuration of the system can be complicated, managing many different CAs and determining how cross-certification between them. Also, as the number of CAs involved increases, managing certificates and mapping relationships tends to become more difficult than with other models. There is also the risk of inconsistencies between security policies of different CAs, which if not properly managed could introduce vulnerabilities in the system [16] [17].

## 2.5 PKI Types: Public vs. Private

Although the most common use of PKI as a support for authentication, confidentiality and data integrity, each implementation has its own particular reasons that can be traced back to the requirements of the environment in which it is to

be accepted. In particular, the two main types of PKI, public and private, differ greatly in the way certificates are handled, the trust of the entities involved and the scope.

### **2.5.1 Public PKI: Global Certification Framework**

The public PKI is based on an external and globally accepted certification scheme, whereby Certification Authorities issue certificates valid for a wide range of users and services. This authority infrastructure is regulated by a set of regulations and standards, which are established by a number of specific bodies such as the CA/Browser Forum, a federation which is set up to lay down the rules for issuing and managing certificates. When an organization or individual requests a certificate from a public CA, the CA performs the identity verification of the applicant (domain or organizational entity), where upon confirmation of authenticity it issues the certificate. This PKI is usually implemented with a two- or three-level hierarchy.

### **2.5.2 Private PKI: Enterprise Certification Infrastructure**

Private PKI, on the other hand, is a solution used by organizations to protect their resources and communications within controlled environments. In an organization that adopts this model, the same acts as owner and guarantor of the Certification Authority (internal CA) that issues certificates valid only for devices or users present within the limits of the corporate LAN.

These certificates are therefore not reliable in the eyes of external systems at the time that the internal CA root to which they connect is not registered in the global trust store of the browsers or operating systems in use. It is obvious that any new device intended to recognize the subscriptions issued by the internal CA root must be prepared in advance, for example by adding the certificate of the internal CA root to your settings.

This solution offers greater security protection and control, where the responsibility for deciding which entities can issue certificates, and how these should be governed, remains entirely in the hands of the organization. Therefore, there are a number of applications for private PKI, ranging from VPN protection to user authentication, from the distribution of certificates in favor of business IoT devices to safeguarding internal communications, Made possible with the help of technologies such as S/MIME for mail. It should also be borne in mind that a private PKI allows to dictate and follow more freely their own guidelines in the matter of security, as well as to enforce access rights in the field of containerized systems or the use of corporate cloud services [18].

### 2.5.3 Fundamental Differences between Public and Private PKI

One of the distinguishing features of both architectures is their reliability. With a public PKI, certificates are generally recognized by browsers and devices, which facilitates secure communication between users and systems within the Internet. This infrastructure is essential for the authentication of websites via HTTPS, for the digital signature of documents and for client authentication at public web services. Otherwise, with a private PKI, certificates have limited validity within the issuing organization and in order to use them externally you must explicitly install the private CA root certificate on the devices involved. As a result, private PKI is less suitable for initiatives to engage with the public but very useful in ensuring internal security and allowing full control over valuable assets such as business data and applications [10] [18].

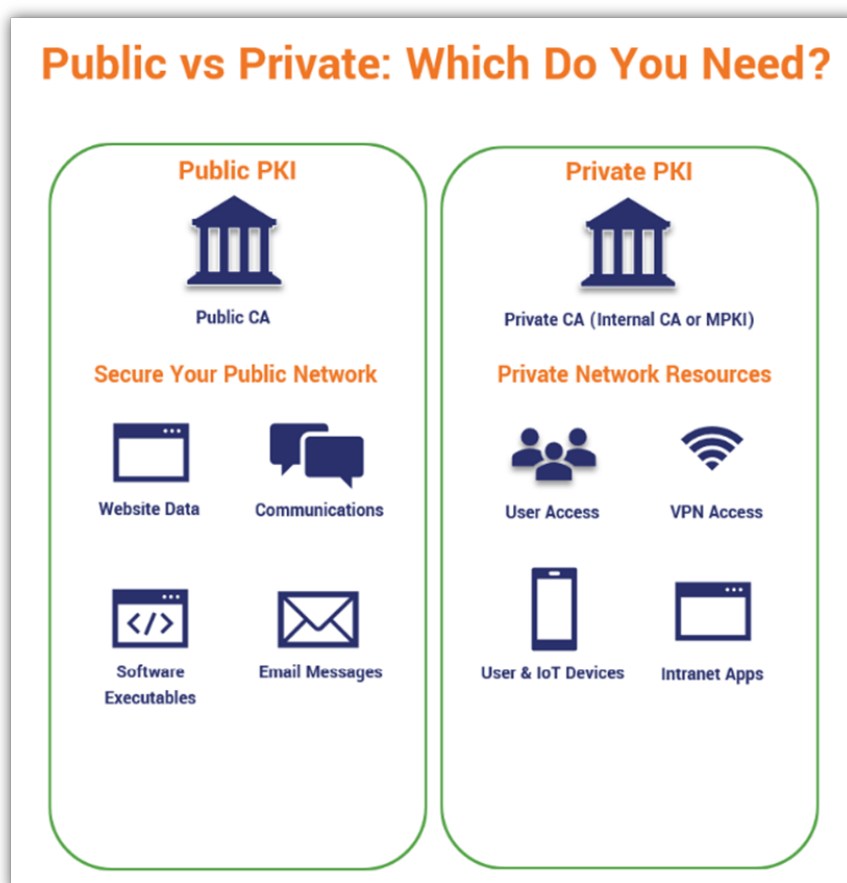


Figure 2.7. Comparison between Public and Private PKI, adapted from [1].

Another important aspect concerns safety management. The public PKI, being under the responsibility of external CAs, involves a degree of dependency on third parties, making infrastructure security dependent on each CA's policies. Public



CAs have strict validation procedures and high security standards, but the confidence placed in them is not entirely safe from risks, as shown by the various attacks on CAs that have occurred in recent years, DigiNotar (2011) [19] and Comodo (2011) [20]. Conversely, in a private PKI the user entity exercises direct control over the certificates and private keys, thus being able to implement the security policy directly. As a result, special attention must be paid to the security of the system and regular compliance with the certificate management policy.

## 2.6 Public Key Certificate

A public key certificate (PKC) is a digital document that securely establishes the link between a public key and the identity of an entity. Issued by a trusted Certificate Authority (CA), the PKC guarantees that the public key actually belongs to the declared entity, thus facilitating authentication, encryption and verification of digital signatures.

The X.509 v3 certificate structure follows a specific precise scheme which gets its encoding from the RFC 5280 standard [6]. The certificate contains distinct fields which serve separate functions to secure communication processes.

### 2.6.1 General Structure of X.509 Certificates

The X.509v3 certificate, generally encoded in DER or PEM formats, consists of two main parts:

- **TBSCertificate (To Be Signed Certificate):** This section contains the essential data of the certificate and is the one that will be submitted to the digital signature of the certifying authority.
- **Signature Algorithm & Signature:** This part includes the signing algorithm and the digital signature itself, applied by the certification authority to guarantee the integrity and authenticity of the data contained in the TBSC certificate.

### 2.6.2 Version

The *Version* field defines the specific iteration of the X.509 format adopted, and depending on its setting, different possibilities and restrictions are configured in the certificate content. According to the RFC 5280 standard [6], this field is encoded as an integer, with the values 0, 1 and 2 corresponding to the V1, V2 and V3 versions respectively.

- **V1:** It is the initial version of the X.509 format, defined in 1988.
- **V2:** This version extends the original format by introducing `issuerUniqueID` and `subjectUniqueID` fields. These fields allow unambiguous identification of the sender and the subject, particularly in situations where name ambiguities

may occur. As stated in RFC 5280 [6], the presence of at least one of these fields requires that the certificate is not version 1.

- **V3:** Version V3 represents the most significant evolution of the format, introducing support for *extensions*. These allow the inclusion of additional, customisable information, such as constraints on key usage, certification policies and other attributes, in order to enhance the security and functionality of the certificate. In accordance with RFC 5280 [6], the inclusion of the `extensions` section forces the use of version 3.

### 2.6.3 Serial Number

The *Serial Number* is a unique identifier assigned by the Certificate Authority to distinguish each certificate issued by the same entity. It must be a non-negative number and have an appropriate length to prevent collisions, thus guaranteeing the uniqueness and traceability of the certificate.

### 2.6.4 Signature Algorithm

The *Signature Algorithm* field explains by which algorithm the certificate authority signed the certificate. Examples include `SHA256withRSA` or `ECDSA with SHA-256`. This field, besides identifying the algorithm, contains any optional parameters used in generating the digital signature that are available to the verifier for the reasoning of integrity and authenticity.

This field has to be encoded according to RFC 5280 [6], so that the signing algorithm can be properly identified. The CA/Browser Forum [4] requirements will also demand the use of secure and standard algorithms to encode high security and interoperability into the issued certificates.

### 2.6.5 Issuer

The *Issuer* field represents the Distinguished Name (DN) of the certificate authority that issued the certificate. The DN is composed of a series of attributes that uniquely identify the issuer, commonly including:

- **Common Name (CN)**
- **Organization (O)**
- **Organization Unit (OU)**
- **Country (C)**

### 2.6.6 Validity

The *Validity* field defines the time interval in which the certificate is considered valid. It consists of two basic parameters:

- **Not Before:** Date and time when the certificate becomes operational.
- **Not After:** Date and time after which the certificate is no longer valid.

The correct setting of these values must comply with the requirements of the CA/Browser Forum [4], which include specific limits on the maximum duration of certificates.

### 2.6.7 Subject

The *Subject* field indicates the Distinguished Name (DN) of the certificate's entity, uniquely identifying its holder. The DN generally contains one or more of the following attributes:

- **Common Name (CN)**
- **Organization (O)**
- **Other attributes such as Location (L) and Country (C)**

### 2.6.8 Subject Public Key Info

The *Subject Public Key Info* field contains the public key of the owner entity and the algorithm associated with it. In particular, it consists of:

- **Key algorithm:** it specifies the cryptographic method used, e.g. RSA or ECC.
- **Public key:** the actual key value, used in encryption operations and digital signature verification.

The requirements of the CA/Browser Forum [4] require the adoption of algorithms and public keys that comply with current security standards. For example, for TLS certificates, the use of RSA keys with a minimum length of 2048 bits or ECC keys with approved curves, such as P-256 or P-384, is required.

### 2.6.9 Certificate Extensions

The main innovation introduced with version 3 of X.509 certificates is the *Extensions* field, which allows the inclusion of additional information to extend the certificate functionality. Extensions may be classified as critical or non-critical:

- **Critical:** If an extension is marked as critical, applications that do not recognize it or cannot process it must consider the certificate invalid.
- **Not Critical:** Not Critical extensions can be ignored by applications that do not recognize them, without affecting the validity of the certificate.

## Basic Constraints

The *Basic Constraints* extension is crucial in X.509v3 certificates, as it determines whether the certificate subject is a Certificate Authority (CA) and defines any limitations on the depth of the certificate chain. This extension includes the following components:

- **CA (boolean):** Indicates whether the entity is a CA. If set to **TRUE**, the entity is authorised to issue other certificates, if **FALSE**, the entity is an end entity and cannot issue certificates.
- **pathLenConstraint (optional):** Specifies the maximum number of certificates that may follow in the certificate chain issued by the current CA. For instance, a value of 0 indicates that the CA may only issue certificates for end entities.

According to RFC 5280 [6], the extension *Basic Constraints* must be present and marked as critical in CA certificates. In the absence of this extension or if the extension is present but the **CA** value is set to **FALSE**, the certificate public key must not be used to verify signatures on other certificates. Furthermore, the *pathLenConstraint*, if present, limits the maximum length of certificate paths that include this certificate.

The CA/Browser Forum [4], in its *Baseline Requirements*, states that certificates of intermediate CAs must contain the extension *Basic Constraints* with the value **CA** set to **TRUE** and, if applicable, an appropriate *pathLenConstraint*.

## Key Usage

The *Key Usage* extension limits the encryption operations that may be performed with the public key in the certificate. Key Usage commonly has the following values:

- **Digital Signature:** It allows the use of the key to generate digital signatures, guaranteeing the authenticity and integrity of the data.
- **Key Encipherment:** Allows the use of the key to encrypt session keys, typically used during handshaking in protocols such as TLS.
- **Certificate Signing (keyCertSign):** Authorises the key to sign digital certificates, an essential feature for Certification Authorities (CAs).

According to the RFC 5280 standard [6], the *Key Usage* extension is optional but, if present, must be marked as critical. This implies that applications that do not recognize or cannot process this extension must reject the certificate. Furthermore, RFC 5280 specifies that the extension must be used to limit the use of the key when necessary, for example, by preventing a key intended for digital signature from being used to encrypt data.

The CA/Browser Forum [4], in its *Baseline Requirements*, states that the *Key Usage* extension must be present and marked as critical in CA certificates. In particular, for root and subordinate CA certificates, the **keyCertSign** and **cRLSign** bits must be set. If the CA private key is used to sign OCSP responses, the **digitalSignature** bit must also be set.

## Extended Key Usage

The *Extended Key Usage* (EKU) extension specifies further purposes for which the public key of the certificate may be used, in addition to those defined in the *Key Usage* extension. This extension allows the use of the key to be restricted to specific functions, such as server authentication or e-mail protection.

The CA/Browser Forum [4], in its *Baseline Requirements*, states that certificates should not contain EKU extensions that are not relevant to the intended use of the certificate. For instance, a certificate intended for authentication of a public server should not include EKU scopes related to internal or private services. Furthermore, the EKU extension, if present, should be marked as non-critical, allowing applications that do not recognize it to still use the certificate for the stated purposes.

## Subject Alternative Name (SAN)

The **Subject Alternative Name (SAN)** extension allows a number of alternative names to be associated with the certificate, such as additional domain names, email addresses and IP addresses. This functionality is particularly crucial in multi-domain certificates, as it allows a single entity to protect multiple identities without the need to issue separate certificates.

According to RFC 5280 [6], if the *Subject* field is empty, the SAN extension must be present and contain at least one value, thus ensuring that there is always at least one alternative identifier for the certified entity.

## Authority Key Identifier (AKI)

The **Authority Key Identifier (AKI)** extension is mostly used to identify the CA's public key that signed the certificate and, thus, to build the chain of trust more easily.

Under the RFC 5280 standard [6], at a minimum, the AKI extension is supposed to include a Key Identifier that links a certificate back to a certain CA's public key. This condition is necessary so that verifiers can identify the CA's certificate correctly and authenticate the certification chain. The CA/Browser Forum guidelines [4], meanwhile, require that for SSL/TLS application certificates, the AKI extension be put in place.

## Authority Information Access (AIA)

The **Authority Information Access (AIA)** extension provides the information needed to access resources related to the Certificate Authority (CA) that issued the certificate. In practice, this field typically includes URLs that allow:

- Obtaining the CA certificate, essential for building the chain of trust;
- Check the status of the certificate via the OCSP protocol.

The CA/Browser Forum guidelines [4] require the inclusion of the AIA extension in certificates intended for SSL/TLS contexts, so that verifiers can easily find up-to-date information on the issuing CA and revocation status.

## CRL Distribution Points (CDP)

The **CRL Distribution Points** extension provides URLs from which the Certificate Revocation List (CRL), an essential document for verifying whether a certificate has been revoked by the Certificate Authority (CA), can be retrieved. This field allows verifiers to access official lists of revoked certificates, thus ensuring that only valid and trusted certificates are used within the chain of trust.

According to the RFC 5280 standard [6], if the CDP extension is present, it must be encoded as a sequence of *DistributionPoint* containing at least one *fullName* field with URLs conforming to the defined format. The specification also requires that if the certificate is used in critical contexts, this extension must not be omitted. Therefore, the key fields of a CRL include:

- **thisUpdate**: indicates the date and time when the CRL was issued, representing the time reference point of the list.
- **nextUpdate**: specifies the date and time by which the next update of the CRL is to be published. This field is essential to ensure that verifiers know when the list may have changed and, therefore, when to perform a new consultation.
- **CRLNumber**: an incremental identifier to distinguish different versions of the CRL, useful for checking the update history.
- **DistributionPoint** (with the associated *fullName* field): where one or more URLs conforming to the defined format are specified, from which the CRL may be retrieved.
- **reasons** (optional): a set of flags indicating the reasons why certificates have been revoked.
- **CRLIssuer** (optional): indicates the identity of the CRL issuer, if this does not match the certificate *Issuer* field

The CA/Browser Forum [4] guidelines, in turn, require the inclusion of the CDP extension in SSL/TLS certificates, ensuring that at least one valid URL is provided to allow timely consultation of the CRL.

## Subject Key Identifier (SKI)

The **Subject Key Identifier** extension uniquely identifies the subject public key, commonly derived from a hash of the public key itself. This identifier will further ease the processes of searching, comparing, and managing the key in the certification chain so that the efficiency of building and validating the trust path is increased.

According to the RFC 5280 standard [6] and the CA/Browser Forum guidelines [4], the SKI extension is recommended for all certificates, as it allows better identification of keys.

## Certificate Policies

The **Certificate Policies** extension specifies the policies and rules governing the use of the certificate, using object identifiers (OIDs) to indicate the rules adopted

by the Certificate Authority (CA). This extension informs the relying parties of the conditions and procedures by which the certificate was issued, providing essential information on the operational context and security requirements applied.

As specified in RFC 5280 [6], the Certificate Policies extension must contain at least one policy OID, but policy qualifiers, like the URI of the Certification Practice Statement or user information notes, may be included if desired. In addition, the CA/Browser Forum guidelines [4] mandate that publicly trusted certificates must contain the Certificate Policies extension, so that subject to interpretation, the issuance conditions and usage restrictions are as clear as possible.

## 2.7 Certificate Revocation Mechanisms

The effectiveness of PKI depends not only on the correct issuance of certificates, but also on the ability to invalidate compromised or erroneously issued certificates in a timely manner. In this context, the certificate revocation mechanism plays a key role, as it allows certificates that are no longer trusted to be removed from the ‘chain of trust’, reducing the risk of attacks such as impersonation or man-in-the-middle.

### 2.7.1 Overview of Revocation Mechanisms

The Certificate Revocation List (CRL) is a list, periodically updated and digitally signed by the Certification Authority (CA), containing the serial numbers of certificates revoked before their natural expiry date. This list is digitally signed by the CA to guarantee the authenticity and integrity of the list. This signature prevents unauthorised parties from modifying the content of the CRL, e.g. by removing revoked certificates or inserting new ones with the intention of compromising trust in authentication systems. A crucial aspect of the CRL is the possibility of checking the validity of a certificate against the time of its issuance. For example, if a digitally signed document three months ago is verified today, it is necessary to ensure that the key used was still valid at the time of signing. In this way, even if the certificate was revoked later, this does not affect the validity of the signature affixed to the document, since the verification refers to the state of the certificate at the time of signing. In spite of its essential role, the adoption of the CRL also presents certain criticalities. In particular, in order to check the status of an individual certificate, it is necessary to download the entire list of revoked certificates, an operation that can be burdensome in terms of bandwidth and response time, especially in environments with a large number of certificates.

### 2.7.2 Online Certificate Status Protocol (OCSP)

The OCSP is the preferred mechanism for real-time checks and was developed to overcome some of the limitations of CRLs (standardised in RFC 6960 [21]). It involves a ‘pull’ mode in which the client sends a request to an OCSP responder,

usually managed by CAs, containing the serial number of the certificate to be checked, receiving in reply a signed message indicating whether the certificate is ‘Good’, ‘Revoked’ or ‘Unknown’. OCSP responses thus allow real-time verification without having to download entire revocation lists. In addition, they allow for more timely verification, reducing the window of vulnerability between the time of revocation and its propagation to clients.

### **2.7.3 OCSP Stapling**

In order to mitigate some of the critical issues of OCSP, especially those related to latency and privacy, OCSP stapling was introduced. In this solution, the server presenting the certificate is responsible for attaching a signed and timed OCSP response, obtained periodically from the responder, within the TLS handshake [11].

### **2.7.4 OCSP Must Staple**

OCSP Must Staple is an extension to the TLS protocol that obligates the server to attach to an ordinary handshake an up-to-date OCSP response signed by the certificate authority. If the client does not get this response, the connection is rejected, thus assuring that the certificate validity is always verifiable in real time.



# Chapter 3

## Certificate Transparency version 2.0

In recent years, the web ecosystem has witnessed numerous incidents that have exposed the inherent vulnerabilities of the traditional trust model based on the Public Key Infrastructure (PKI). A case in point is the DigiNotar incident of 2011, in which a malicious actor managed to compromise the Dutch CA DigiNotar and issue fraudulent certificates for several high-profile domains, including Google's, thus exposing hundreds of thousands of users to potential man-in-the-middle attacks. Symantec's certificate issuing practices also raised concerns, highlighting how errors and failures in management can undermine trust in the PKI system. Such incidents show that, although web clients are able to detect anomalies in certificates, they are not always able to identify in a timely manner when a certificate has been issued by a compromised CA or has followed practices that do not comply with security standards.

In response to these critical issues, the concept of Certificate Transparency (CT) was developed. CT is an open framework that aims to improve transparency and accountability in the process of issuing digital certificates. Based on appendix-only public registries, CT records every certificate issued by a CA, making it accessible for real-time monitoring and auditing. In this way, domain owners and security experts can quickly detect unauthorized or suspicious certificates, effectively combating fraudulent issuance. RFC 6962 [22] defines the mechanisms behind CT, which later evolved into RFC 9162 [23], which rendered the previous standard obsolete and made significant improvements. Furthermore, the CA/Browser Forum guidelines [4] emphasize the importance as a complementary measure to strengthen the security of the entire PKI system.

A key aspect is the active and coordinated participation of various actors who contribute to ensuring the transparency and security of the entire PKI ecosystem, among them are the Submitters, Log Servers, Monitors and Auditors.

## 3.1 Rationale Behind Certificate Transparency v2.0

The transition from version 1.0 to version 2.0 of Certificate Transparency (CT) was driven by the need to overcome certain limitations noted in the first implementation and to respond to the growing security needs of the PKI ecosystem, replacing the previous protocol defined by RFC 6962 [22].

The protocol now supports hash and signature algorithm agility, i.e., the possibility of specifying permitted algorithms via IANA registers, making the system more adaptable to the evolution of cryptographic techniques. Another important change concerns the format of pre-certificates: these have been transformed into CMS objects instead of X.509 certificates, thus avoiding violations of the serial number uniqueness requirement defined in RFC 5280 [6].

Furthermore, with CT 2.0, precertificate signing certificates and the precertificate poison extension were eliminated, as they are no longer needed due to the change in the format of precertificates. A further significant change is the introduction of Log IDs, where each log is identified by an OID rather than the public key hash, simplifying the management of identifications via an IANA registry.

The data structure used to encapsulate most of CT's information has been replaced by the new **TransItem**, which standardises the format for both certificates and pre-certificates, removing a level of abstraction present in CT 1.0. This change, together with the introduction of Log artefact extensions managed via an IANA registry, has simplified and made the format of logged entries more extensible.

In the context of the API, new features are enabled by CT 2.0, such as the `get-all-by-hash` and `submit-entry` APIs that replace the old `add-chain` and `add-pre-chain` commands, enabling simultaneous inclusion and consistency proofs. Moreover, presenting SCTs (Signed Certificate Timestamps) is enriched now, since they can attach their corresponding inclusion proofs by virtue of the new TLS extension `transparency_info`, which replaced the old `signed_certificate_timestamp`.

Finally, CT 2.0 introduces more detailed verification algorithms for inclusion proofs and for consistency between Signed Tree Heads (STH), improving the auditing capability and the control of the correct evolution of the log.

## 3.2 Merkle Tree Hash

In the field of cryptography, Merkle trees represent an elegant and powerful solution to guarantee data integrity in an efficient and verifiable manner. The basic idea at the root of this structure is to organize the data in a hierarchical form: each element, or leaf, is subjected to a specific hash function, and the results are progressively combined, through further hashing operations, to form a single root value. This value, known as the Merkle Tree Hash, summarizes the entire data set and makes it possible to detect any alterations that may have occurred in any part of the structure.

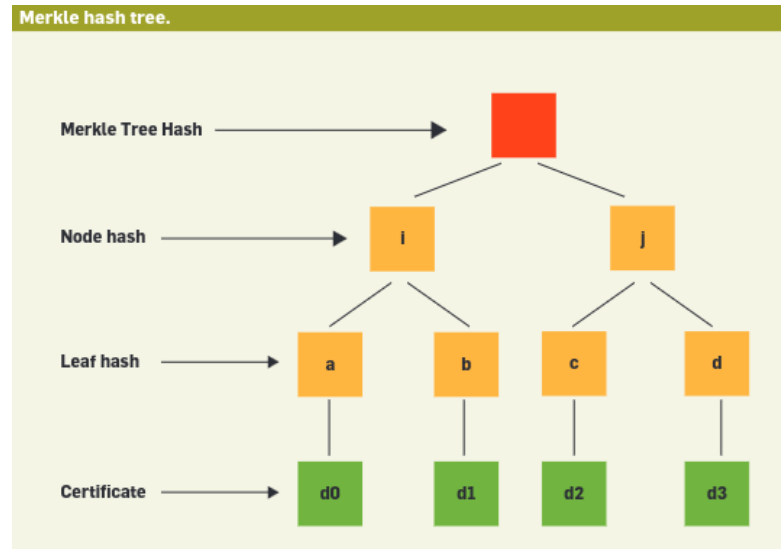


Figure 3.1. Representation of a Merkle Hash Tree, where each leaf is a certificate, taken from the article [2].

The procedure for constructing a Merkle tree starts by calculating the hash for each data element in the list, using a prefix (typically 0x00) that separates the context of the leaves from that of the internal nodes. When the list contains only one element, the hash is calculated directly on that data. If, on the other hand, there are more elements, the list is split into two parts, choosing the greater power of two below the total number of elements as the threshold. In this way, each of the two sublists is processed recursively to generate the hashes of the subtrees, which are then combined together using an additional prefix (usually 0x01). This mechanism, in addition to guaranteeing the correctness of the aggregation, introduces a clear separation of domains, an essential element to prevent attacks based on the second prefix.

A particularly interesting aspect of Merkle trees lies in the possibility of compactly proving the inclusion of a specific datum without having to recompute the entire tree. This feature is realised in the so-called **proof of inclusion**, which consists of a minimum sequence of hashes from the missing nodes along the path connecting the concerned leaf to the root, as shown in Figure 3.2. Then, starting from the hash calculated for the certificate in question, it is possible to successively combine the hashes provided by the proof and verify, at the end of the process, whether the result matches the root value of the tree. If this is the case, it can be concluded that the data is indeed included in the structure, making the whole mechanism extremely useful for applications where verification efficiency and security are of primary importance.

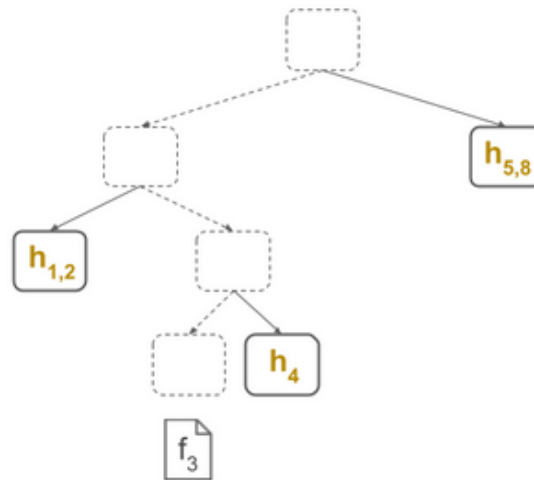


Figure 3.2. Proof of inclusion in a Merkle Hash Tree, illustrating the process of verifying a certificate’s inclusion using hashes, taken from the article [3].

Therefore, in addition to an inclusion test, Merkle trees achieve security with the use of so-called **consistency tests**. Besides convincing the fact that the evolution of the tree happened in **append-only** mode, that is the addition of new data would never change previously present data, the consistency test is made on a series of intermediate hashes that show that a certain root, calculated at an earlier stage, is still present within the updated structure. A procedure that is also recursive in nature exploits a combination of operations, which by comparing and aggregating hash values enables the reconstruction of the path leading from historic to current through hash values. The consistency test thus proves the preservation of integrity with respect to previous data, hence the immutability and consistency of the tree at each moment in time.

### 3.3 Submitters

In this context, Certification Authorities (CAs) are responsible for issuing certificates and must record each certificate issued in public CT logs, thus acting as **submitters**. These logs, managed by independent operators including names such as Google, DigiCert and Let’s Encrypt, act as **append-only** logs using Merkle Hash Tree cryptographic data structures. Each log is uniquely identified by an OID, assigned through a process that conforms to the CA/Browser Forum [4] guidelines, which allows the logs to be differentiated and ensures efficient information management.

Thus, submitters send certificates or pre-certificates (i.e. pre-announcements of certificates) to public logs prior to final issuance in order to make them available for public monitoring and auditing. To ensure proper attribution of each certificate or pre-certificate to the issuer, each submission must be accompanied by any additional certificates necessary to verify the chain up to a recognised trust anchor

(typically a root or intermediate certificate), although the trust anchor may be omitted under certain circumstances [23].

The log evaluates submissions based on minimum acceptance criteria, which include conformity to the required inputs, e.g. the correct arrangement of certificates in the chain without order errors and the presence of essential flags (such as the `cA` flag in Basic Constraints or the `keyCertSign` bit in Key Usage). In addition, the chain must respect the limits imposed by the `pathLenConstraint` fields, if present. Only if all these minimum requirements are fulfilled will the log accept the submission.

Once the log has accepted the submission, it returns a Signed Certificate Timestamp (SCT), which constitutes signed proof of the inclusion of the certificate in the log. Submitters, depending on their needs, should validate the SCT received: if the objective is to use it directly during the TLS handshake or for the construction of a certificate, validation becomes an essential step to ensure compliance and security. Alternatively, if the certificate is simply sent to the log to make it public, validation of the SCT may be considered optional.

In addition, any entity, not just CAs, can submit certificates directly to the logs, since TLS clients are designed to reject certificates not registered in a public log. This mechanism provides a strong incentive for issuers to proceed with certificate submission, thus ensuring continuous and transparent issuance control. In the case of pre-certificates, CAs can pre-submit a certificate before formal issuance, obtaining in response SCTs that can be incorporated into the final certificate.

### 3.3.1 TLS servers

TLS servers implementing Certificate Transparency (CT) must adopt at least one of the mechanisms provided to present one or more Signed Certificate Timestamps (SCTs) to end users during the complete TLS handshake when requested by the client.

For example, a TLS server may present SCTs via the TLS 1.3 extension called `transparency_info`. This method allows the server to participate in CT without needing the direct cooperation of the CA, and also allows SCTs and inclusion proofs to be updated in real time. Alternatively, the server may use **OCSP stapling**, in which the OCSP response provided during the handshake also includes SCT and inclusion proofs, or finally **embedding CT information** as an extension in the X.509 v3 certificate, although this approach is less flexible as SCTs cannot be updated over time and the certificate may be considered non-compliant in the future if the log does not remain reliable.

The presentation of CT data is done via the `TransItem` structure, which neatly aggregates SCTs, inclusion proofs and Signed Tree Heads (STH). Thus, if the server has this information, it is recommended that it be sent together in the TLS message in order to protect the client's privacy and reduce the load on the logs.

## 3.4 Algorithm Agility

Managing the agility of algorithms in a CT log is essential to ensure consistency and backward compatibility. A log cannot change its signature or hash algorithm mid-life, as this would compromise the validity of already issued SCTs. Therefore, should it be necessary to deprecate an algorithm, the log must be frozen and a new log started, so that certificates that are still valid can obtain new SCTs from the new log. This strategy, imposed by the specifications and guidelines of the CA/Browser Forum [4], keeps the system simple and secure, avoiding the complexity that would result from supporting several algorithms simultaneously.

## 3.5 TLS Client Validation and Verification

TLS clients play a really important role in performing Certificate Transparency (CT) by providing a mechanism to verify in real time that each certificate submitted has been registered in one or more public logs. The clients, which support the CT extension, include a request for CT information, generally via the `transparency_info` extension, in the ClientHello message sprinkled into the TLS handshake. With the request, the server can attach, along with the certificate, Signed Certificate Timestamps (SCT) and, if available, inclusion proofs.

### 3.5.1 Receiving and Validating SCTs

When the client receives one or more SCTs, the first step is to verify their authenticity. To do this, the client reconstructs the signed part of the certificate (TBSCertificate) and uses the public key of the log, identified by the `log_id`, to check that the signature of the SCT is valid. This process ensures that the certificate has indeed been recorded in the log and that no tampering has occurred.

### 3.5.2 Requesting Inclusion Proofs

In the event that the client does not directly receive an inclusion proof with the SCT, he may request it from the log using special APIs (such as `get-proof-by-hash` or `get-all-by-hash`). However, this operation may entail a partial loss of privacy, since it reveals to the log the domain that the client is accessing. For this reason, it is preferable that the server already provides inclusion proofs during the handshake, thus avoiding additional requests that could compromise the client privacy.

### 3.5.3 Validation of Inclusion Proofs

After verifying the SCTs and, if necessary, obtaining inclusion proofs, the TLS client assesses the conformity of the certificate. Only if the CT information is

consistent and valid, that is, if the signature of the SCTs is correct and the inclusion proofs confirm the inclusion of the certificate in the log, is the certificate considered trustworthy. This evaluation step is crucial to protect the end user, as it ensures that the certificate has been registered transparently and that there are no fraudulent certificates in circulation.

## 3.6 Monitor

In parallel, the role of monitor plays a crucial role, actively helping to verify the correctness of the logs and to promptly identify any unauthorised certificates. Thus, monitors are in charge of periodically inspecting public logs to ensure that the log behaviour complies with specifications and that each issued certificate can be traced back to a valid entry in the log.

To proceed with supervision, monitors are supposed to follow through a structured set of steps. They first get hold of the corresponding **Signed Tree Head** (STH), which is a cryptographic header of the log, representing exactly the snapshot of the current state of the log. They check the signature of the STH to make sure that the log is unwisely substituted. Monitors proceed to retrieve all the entries in the tree corresponding to STH and check each of them for which certificates should have special sense for them, like domain-related ones. It is finally checked that the tree built from the entries retrieved matches exactly with the hash in the STH, thus ensuring log consistency and integrity [23].

With regard to the inspection of new entries, monitors perform a series of operations in a repeated manner: they retrieve the updated STH, re-verify its signature and collect the new log entries. If the monitor does not maintain a complete copy of all entries, a consistency proof is used to verify that the new entries have been correctly integrated from their previous state. This procedure ensures that any changes to the log are verifiable and that any discrepancies can be detected in a timely manner.

## 3.7 Auditing

The auditing of Certificate Transparency (CT) logs ensures that the current state of a log can be reliably traced back to previously published states, verified as correct. In this way, it can be ensured that the promises made by the log, expressed through Signed Certificate Timestamps (SCT), have actually been fulfilled. Audit operations can be performed by both specialized monitors and TLS clients, which verify the consistency and integrity of the information recorded.

In particular, during an audit, it is crucial to verify four behavioural properties of the log:

- **Maximum Merge Delay** (MMD): represents the maximum time allowed for changes made to the log to be propagated and made visible in full form.

- **STH Frequency Count:** defines the frequency with which Signed Tree Heads (STH) are published.
- **Append-only property:** ensures that items entered in the log once cannot be removed or modified, thus ensuring the immutability of the log.
- **Consistency of the log view:** requires the display of the log to be uniform and consistent across all sources that consult it, so as to avoid discrepancies between different stakeholders.

A compliant log publishes a series of STHs over time, each derived from the previous and subsequent entries, and this behaviour can be demonstrated by auditing the STHs. In addition, SCTs provided to TLS clients can be compared with the attached certificate and verified through Merkle inclusion tests, which attest to the inclusion of the certificate in the Merkle tree of the log [23].

If an audit fails, the action to be taken by the auditor is not explicitly defined, but it is essential that the auditor collects signed evidence of the abnormal behaviour of the log. A monitor, for example, can check the consistency of received STHs, making sure that each entry is accessible and that the STH matches the structure of the log tree. Similarly, a TLS client can perform an audit by comparing an SCT with an STH published after the timestamp of the SCT plus the Maximum Merge Delay, requesting a Merkle inclusion proof, and verifying that the SCT exactly matches the certificate or precertificate presented by the server.

## 3.8 Understanding Certificate Transparency in Real-World Applications

### 3.8.1 TLS Connection Process with Certificate Transparency

The following example illustrates a practical application of Certificate Transparency (CT) according to the RFC 6962 standard, which is currently still prevalent in the digital certificate issuance landscape compared to RFC 9162, as confirmed in the analysis Chapter 5.5. In this scenario, the operational flow followed by a TLS client when establishing a secure connection with a server supporting CT is described. The example, developed in Python, shows how the client interacts with the server to receive and validate CT information, in order to ensure that the certificate has been correctly registered in a public registry.



## Establishing the TLS Connection

```
def get_certificate(host, port=443):
    """
    Connects via TLS to a host and retrieves its certificate in DER format.

    Parameters:
        host (str): The server's hostname.
        port (int): The server's port (default: 443).

    Returns:
        bytes: DER-encoded certificate.
    """
    context = ssl.create_default_context()
    sock = socket.socket(socket.AF_INET)
    conn = context.wrap_socket(sock, server_hostname=host)
    conn.connect((host, port))
    cert_bin = conn.getpeercert(binary_form=True)
    conn.close()
    return cert_bin
```

Figure 3.3. TLS Connection Establishment Process: The client sends a ClientHello message requesting CT information, followed by the server certificate with Signed Certificate Timestamps (SCTs).

At the start of a TLS session, the client sends a ClientHello message including, among other extensions, a request for CT information via the `transparency_info` extension. The server, in response, sends its certificate, encoded in DER format, and attaches one or more Signed Certificate Timestamps (SCTs). Figure 3.3 shows how the client establishes the TLS connection and receives the certificate from the server.

## Extraction of SCTs

```
def extract_scts(cert_bin):
    """
    Extracts and parses Signed Certificate Timestamps (SCTs) from the certificate.

    SCTs are embedded in the CT extension identified by OID "1.3.6.1.4.1.11129.2.4.2".

    Parameters:
        cert_bin (bytes): DER-encoded certificate.

    Returns:
        list: A list of dictionaries, each representing an SCT.
    """
    cert = x509.load_der_x509_certificate(cert_bin, default_backend())
    sct_list = []
    try:
        ct_ext = cert.extensions.get_extension_for_oid(
            x509.ObjectIdentifier("1.3.6.1.4.1.11129.2.4.2")
        ).value

        # Helper lambdas to get human-readable entry type and extension info.
        print_entry_type = lambda s: {"PRE_CERTIFICATE": "Pre Certificate", "X509_CERTIFICATE": "Certificate"}.get(s, "")
        print_extension = lambda s: None if s == "" else s

        if ct_ext:
            for idx, sct in enumerate(ct_ext, start=1):
                sct_info = {
                    'version': sct.version,
                    'log_id': sct.log_id.hex(),
                    'timestamp': sct.timestamp, # datetime object
                    'entry_type': sct.entry_type,
                    'extensions': sct.extension_bytes.hex(),
                    'hash_alg': sct.signature_hash_algorithm,
                    'sig_alg': sct.signature_algorithm,
                    'signature': sct.signature.hex()
                }
                sct_list.append(sct_info)
```

Figure 3.4. Extracting SCTs from the Certificate: Python code parses the certificate to identify and extract the SCTs as shown.

```

(ct-check) ct1sh-man@P15en-Pan-on-RedHat:~$ python code.py
Retrieving issuer certificate from: http://GEANT.crt.sectigo.com/GEANTOVRSA4.crt
SCT 1:
Version: 0
Log ID: dddcca3495d7e11605e79532fac79ff83d1c50fdb003a1412760a2cacbbc82a
Timestamp: 2024-11-22 09:56:37.697000
Entry Type:
Extension: None
Signature Hash Algorithm: sha256
Signature Algorithm: SignatureAlgorithm.ECDSA
Signature: 304502206d4f59e0eac7ee91ea574a8e65b57b715815125c850908fe67874adef3715d022100ce73b471f65d3fcf419949dc74f98ea378adb7c046b526af2f29bd0d164f328
SCT 2:
Version: 0
Log ID: ccfb0f6a85710965fe959b53cee9b27c22e9855c0d978db6a97e54c0fe4c0db0
Timestamp: 2024-11-22 09:56:37.664000
Entry Type:
Extension: None
Signature Hash Algorithm: sha256
Signature Algorithm: SignatureAlgorithm.ECDSA
Signature: 3044022089da037794e363aa2dbedd246770879242f009abcaa46ef2665bf044cc26962402201c236d1a788723750be7cb2a691881667e638ff50607950ac0f86fae7a8d5e72
SCT 3:
Version: 0
Log ID: 12f14e34bd53724c840619c38f3f7a13f8e7b56287889c6d300584ebe586263a
Timestamp: 2024-11-22 09:56:37.735000
Entry Type:
Extension: None
Signature Hash Algorithm: sha256
Signature Algorithm: SignatureAlgorithm.ECDSA
Signature: 3045022057b0cfe500984ba7ed885fd2056a7d303d8e3cf65a1bfe533a08f0c02f1ed2b02210095b857abd3092e2bec0e663b94600637c92a9e1aa18b7db36ac742ee57838499

```

Figure 3.5. SCT Extraction Result: The extracted SCTs from the certificate of `www.polito.it` are displayed, confirming the certificate transparency information.

Once the certificate is received, the client extracts the SCTs present in the certificate CT extension. This extraction process is illustrated in Figure 3.4, where Python code parses the certificate to identify and isolate the SCTs. Figure 3.5 shows the result obtained, based on the digital certificate of `www.polito.it`. In this way, the client acquires the necessary information to proceed with validation.

## Verification of SCTs

```
def verify_sct(sct, cert_bin, log_pub_key_pem, issuer_cert_bin=None):
    """
    Verify the Signed Certificate Timestamp (SCT) using the CT log's public key.

    Depending on whether the SCT refers to a precertificate or a normal certificate,
    the function constructs the appropriate signature input and then uses the provided
    public key to verify the signature.

    Parameters:
        sct (dict): SCT information extracted from the certificate.
        cert_bin (bytes): DER-encoded certificate.
        log_pub_key_pem (str): CT log's public key in PEM format.
        issuer_cert_bin (bytes, optional): DER-encoded issuer certificate (required for precerts).

    Returns:
        bool: True if the SCT signature is valid, False otherwise.
    """
    cert = x509.load_der_x509_certificate(cert_bin, default_backend())
    tbs_cert = cert.tbs_certificate_bytes

    # Convert the timestamp to milliseconds if needed.
    if isinstance(sct['timestamp'], datetime):
        timestamp_int = int(sct['timestamp'].timestamp() * 1000)
    else:
        timestamp_int = sct['timestamp']

    # Parse any SCT extensions as bytes.
    ext_bytes = bytes.fromhex(sct['extensions']) if sct['extensions'] else b""

    if sct['entry_type'] == LogEntryType.PRE_CERTIFICATE:
        if issuer_cert_bin is None:
            print("Issuer certificate required for SCT verification (PreCertificate).")
            return False
        # Compute issuer key hash from the issuer certificate.
        issuer_cert = x509.load_der_x509_certificate(issuer_cert_bin, default_backend())
        issuer_pub_bytes = issuer_cert.public_key().public_bytes(
            encoding=serialization.Encoding.DER,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
        issuer_key_hash = sha256(issuer_pub_bytes).digest() # 32 bytes
```

Figure 3.6. Step 1 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key.

```

# Build signature input for precertificates.
signature_input = bytes([sct['version'].value])
signature_input += b"\x00" # Signature type: certificate_timestamp
signature_input += struct.pack(">Q", timestamp_int)
signature_input += b"\x00\x01" # Entry type: 1 for precertificate
signature_input += issuer_key_hash
tbs_clean = remove_ct_poison(tbs_cert)
signature_input += tbs_clean
signature_input += len(ext_bytes).to_bytes(2, "big")
signature_input += ext_bytes
else:
# Build signature input for normal X509 certificates.
# The full certificate DER is prefixed by a 3-byte length field.
signature_input = bytes([sct['version'].value])
signature_input += b"\x00" # Signature type: certificate_timestamp
signature_input += struct.pack(">Q", timestamp_int)
signature_input += b"\x00\x00" # Entry type: 0 for X509 certificate
cert_length = len(cert_bin)
signature_input += cert_length.to_bytes(3, "big")
signature_input += cert_bin
signature_input += len(ext_bytes).to_bytes(2, "big")
signature_input += ext_bytes

# Load the CT log's public key from its PEM representation.
public_key = serialization.load_pem_public_key(
    log_pub_key_pem.encode("utf-8"), backend=default_backend()
)

# Select the correct hash algorithm based on its name (case-insensitive).
hash_alg_name = sct['hash_alg'].name.lower()
if hash_alg_name == "sha256":
    chosen_hash = hashes.SHA256()
elif hash_alg_name == "sha1":
    chosen_hash = hashes.SHA1()
else:
    raise ValueError("Unsupported hash algorithm in SCT: {}".format(sct['hash_alg'].name))

```

Figure 3.7. Step 2 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key.

```

try:
    if sct['sig_alg'] == SignatureAlgorithm.ECDSA:
        # Verify using ECDSA.
        public_key.verify(
            bytes.fromhex(sct['signature']),
            signature_input,
            ec.ECDSA(chosen_hash)
        )
    else:
        # Verify using RSA (PKCS1v15 padding).
        public_key.verify(
            bytes.fromhex(sct['signature']),
            signature_input,
            padding.PKCS1v15(),
            chosen_hash
        )
    print("SCT signature is valid.")
    return True
except Exception as e:
    print("SCT signature verification failed:", e)
    return False

```

Figure 3.8. Step 3 - Verifying SCT Signature: The client reconstructs the certificate TBSCertificate and verifies the signature of the first SCT using the log's public key.

The first task of the client is the validation of SCTs. This operation involves the reconstruction of the part of the certificate subject to signature (TBSCertificate), with the removal of any unnecessary extensions, such as the CT poison extension. Next, the client verifies the signature of each SCT using the public key of the corresponding log, identified by the `log_id`. Figures 3.6, 3.7 and 3.8 illustrate the validation process, which makes it possible to ensure that the certificate has indeed been recorded in the public log and that the information has not been tampered with. These operations follow what is specified in the RFC 6962 standard, section on the construction and verification of the Merkle Tree Leaf [22].

## Requesting and Verifying Inclusion Proofs

```
def get_inclusion_proof(cert_bin, log_url, is_precert=False, issuer_cert_bin=None, timestamp_value=None):
    """
    Retrieve an inclusion proof from a Certificate Transparency (CT) log via its REST API.
    The function first computes the leaf hash for the given certificate and then queries
    the CT log's REST API to obtain the inclusion proof (audit path).

    Parameters:
        cert_bin (bytes): DER-encoded certificate.
        log_url (str): Base URL of the CT log.
        is_precert (bool): True if the certificate is a precertificate.
        issuer_cert_bin (bytes, optional): DER-encoded issuer certificate (required for precerts).
        timestamp_value (int): Timestamp in milliseconds associated with the SCT.

    Returns:
        tuple: (proof_data (dict or None), leaf_hash (str))
    """
    # Compute the leaf hash for the certificate.
    leaf_hash = compute_leaf_hash(cert_bin, is_precert, issuer_cert_bin, timestamp_value)

    # Retrieve the current tree size from the Signed Tree Head (STH).
    sth_response = requests.get(f"{log_url}/ct/v1/get-sth")
    if sth_response.status_code != 200:
        print("Error retrieving Signed Tree Head (STH):", sth_response.status_code, sth_response.text)
        return None, leaf_hash

    try:
        tree_size = sth_response.json().get("tree_size")
    except Exception as e:
        return None, leaf_hash

    # Query the CT log for the inclusion proof using the leaf hash and tree size.
    api_url = f"{log_url}/ct/v1/get-proof-by-hash?hash={leaf_hash}&tree_size={tree_size}"
    response = requests.get(api_url)
    if response.status_code != 200:
        return None, leaf_hash

    try:
        proof_data = response.json()
    except Exception as e:
        return None, leaf_hash

    return proof_data, leaf_hash
```

Figure 3.9. Requesting Inclusion Proof: If not provided by the server, the client interacts with the public log to request the inclusion proof using the `get-proof-by-hash` API.

```

def verify_inclusion_proof(proof, sth):
    """
    Verify an inclusion proof by recomputing the Merkle root from the audit path and
    comparing it with the root hash provided in the Signed Tree Head (STH).

    Parameters:
        proof (dict): Inclusion proof containing 'leaf_hash', 'leaf_index', and 'audit_path'.
        sth (dict): STH data containing 'sha256_root_hash'.

    Returns:
        bool: True if the computed Merkle root matches the STH's root hash, False otherwise.
    """
    computed_hash = bytes.fromhex(proof["leaf_hash"])
    leaf_index = proof["leaf_index"]

    # Iterate over each node in the audit path to reconstruct the Merkle root.
    for node_hex in proof["audit_path"]:
        node = bytes.fromhex(node_hex)
        # Depending on the leaf index, the node order is chosen accordingly.
        if leaf_index % 2 == 0:
            computed_hash = sha256(b'\x01' + computed_hash + node).digest()
        else:
            computed_hash = sha256(b'\x01' + node + computed_hash).digest()
        leaf_index //= 2

    if computed_hash.hex() == sth["sha256_root_hash"]:
        print("Inclusion proof is valid.")
        return True
    else:
        print("Inclusion proof is invalid.")
        return False

```

Figure 3.10. Verifying Inclusion Proof: The client reconstructs the Merkle tree and verifies the inclusion proof by comparing the root hash from the Signed Tree Head (STH).

In case the client did not receive the inclusion proofs from the server directly, it can interact with the public log to request it. The process in this case begins by calculating the leaf hash of the certificate and requesting via API `get-sth` the current size of the tree. The client uses the API `get-proof-by-hash` to retrieve the inclusion proof, as shown in Figure 3.9. Finally, the client reconstructs the Merkle tree to verify the inclusion proof and compares the root hash thus obtained with that found in the signed tree head (STH) provided by the log (Figure 3.10). This step is important to assert that not only the certificate has been included in the log but also that the log itself is functioning correctly.

### 3.8.2 Exploring Google’s Certificate Transparency Verification Tool

A further example of the application of the Certificate Transparency system was conducted using the official tool developed by Google, found in the repository `certificate-transparency-go` [24]. This tool, written in Go, performs similar operations to those illustrated in the previous example, but works directly with certificates in `.pem` format. In this test, the tool was applied to the digital certificate of `www.polito.it`.

```

[ct-check]
10225 18:08:56.737889 48784 sctcheck.go:129] No issuer in chain; attempting online retrieval
10225 18:08:56.792664 48784 sctcheck.go:232] Examine embedded SCT[0] with timestamp: 1732269397697 (2024-11-22 10:56:37.697 +0100 CET) from logID: ddc6ca3495d7e11605e79532fac79ff83d1c50df0b003a1412760a2c6cbcb8
24
10225 18:08:56.792700 48784 sctcheck.go:246] Validate embedded SCT[0] against log "Google 'Xenon2025h2' log"...
10225 18:08:56.792800 48784 sctcheck.go:251] Validate embedded SCT[0] against log "Google 'Xenon2025h2' log"... validated
10225 18:08:56.793006 48784 sctcheck.go:255] Check embedded SCT[0] inclusion against log "Google 'Xenon2025h2' log"...
10225 18:08:57.000106 48784 sctcheck.go:266] Check embedded SCT[0] inclusion against log "Google 'Xenon2025h2' log"... included at 148334469
10225 18:08:57.000176 48784 sctcheck.go:233] Examine embedded SCT[1] with timestamp: 1732269397664 (2024-11-22 10:56:37.664 +0100 CET) from logID: ccfb0f6a05710965fe959b53cee9b27c22e9855c0d978d06a97e54c0fe4c0d
b0
10225 18:08:57.000245 48784 sctcheck.go:246] Validate embedded SCT[1] against log "Cloudflare 'Nimbus2025'"...
10225 18:08:57.000551 48784 sctcheck.go:251] Validate embedded SCT[1] against log "Cloudflare 'Nimbus2025'"... validated
10225 18:08:57.000575 48784 sctcheck.go:255] Check embedded SCT[1] inclusion against log "Cloudflare 'Nimbus2025'"...
10225 18:08:57.841984 48784 sctcheck.go:266] Check embedded SCT[1] inclusion against log "Cloudflare 'Nimbus2025'"... included at 488975089
10225 18:08:57.841995 48784 sctcheck.go:233] Examine embedded SCT[2] with timestamp: 1732269397735 (2024-11-22 10:56:37.735 +0100 CET) from logID: 12f14e34bd53724c840619c38f3f7a13f8e7b56297889cd308584ebe5862b
5a
10225 18:08:57.842035 48784 sctcheck.go:246] Validate embedded SCT[2] against log "Google 'Argon2025h2' log"...
10225 18:08:57.842179 48784 sctcheck.go:251] Validate embedded SCT[2] against log "Google 'Argon2025h2' log"... validated
10225 18:08:57.842192 48784 sctcheck.go:255] Check embedded SCT[2] inclusion against log "Google 'Argon2025h2' log"...
10225 18:08:58.259638 48784 sctcheck.go:266] Check embedded SCT[2] inclusion against log "Google 'Argon2025h2' log"... included at 183197642
10225 18:08:58.259682 48784 sctcheck.go:100] Found 3 embedded SCTs for "polito.certificate.pem", of which 3 were validated

```

Figure 3.11. SCT Verification Result: The results from using the Google Certificate Transparency tool show that all three SCTs in the certificate of `www.polito.it` are valid.

The result, illustrated in Figure 3.11, shows that all three SCTs in the certificate of `www.polito.it` were verified as valid.

### 3.8.3 Observations

The adoption of Certificate Transparency (CT) offers significant advantages in terms of security and transparency in monitoring the issuance of digital certificates. However, the integration of inclusion proofs into the validation process of SCTs inevitably introduces a privacy issue. If SCTs do not automatically include the inclusion path, the TLS client is faced with the need to send a direct request to the log to obtain such proofs, thus exposing the domain that the client is accessing. This process may compromise the client's confidentiality, as the log is informed of the connection, potentially detecting sensitive information about the user's behaviour.

An alternative solution proposed is the Certificate Transparency model with enhanced privacy, described in the article [25]. The authors propose an enhanced version of the CT system that addresses the issue of client privacy by reducing the need for direct communication between the client and the logs. The suggested approach aims at balancing the security ensured by the public monitoring of certificates with the respect for user privacy by preventing their interactions from being traced by logs.

## Chapter 4

# DigitalCertiAnalytics: Software Architecture and Analysis

This research thesis is intended to continue the previous work [26], in which the author has deepened the security of navigation in the World Wide Web, with particular attention to the behavior of web browsers regarding revoked certificates. In particular, the previous thesis examined TLS, which ensures secure connections between entities, but highlighted that the protocol may not be sufficient if browsers accept connections from web servers using revoked certificates. The study analyzed a dataset of X.509 certificates from Alexa Top 1M sites, and found that over 55% of the certificates were issued by Let's Encrypt, with approximately 4054 end-entity certificates lacking a revocation status verification mechanism. In addition, it has been observed how six web browsers handle revocation information in different situations and on different operating systems, with some browsers always applying a `soft fail` approach when revocation information is not available, and others that check the revocation status only for EV certificates. Finally, we tested several TLS clients, emulating the connection with servers belonging to the same list, to compare the TLS implementations and find that some of them do not validate the certificate chain correctly, ignoring the revocation status.

In this thesis, the analysis focused on a much larger dataset of digital certificates (20 million domains), compared to the previous work which included 1 million certificates [26], to see how the situation has changed over the years with the introduction of Baseline Requirements by the CAB Forum [4]. This forum is a global collaborative organization that includes Certificate Authorities and web browser manufacturers, and is responsible for setting minimum standards for CAs issuing SSL/TLS certificates recognized by major browsers. The objective of the Baseline Requirements is to improve the overall security of secure online communications by addressing various aspects such as certificate validity, verification methods and revocation management. Therefore, these requirements were taken as a reference to attest the violations presented during the analysis conducted on the X.509 digital certificates.

In addition, in the thesis [26] were used domains provided by the list Alexa Top 1M that is now a list discontinued. In this case, the analyses were carried out on



two lists of different domains.

The first is DomCop Top10M [27] where DomCop is a platform specialized in searching and purchasing expired domains, designed to facilitate the identification of digital assets with high SEO metrics. In addition, DomCop provides on its site a list of 10 million domains taken from the Open PageRank initiative, an initiative created by DomCop, which reintroduces metrics similar to Google's PageRank using open source data provided by Common Crawl [28] and Common Search. The aim is to enable cross-domain comparisons through free authority metrics that are accessible to everyone. This project was born in response to the removal of Google's PageRank toolbar in 2016, which left a void in the online marketing industry. Open PageRank aims to fill this gap by offering a free alternative based on data collected from Common Crawl [28], which includes billions of indexed web pages.

The second list was generated through BigQuery [29], a fully managed platform for analyzing large amounts of data, available on Google Cloud Studio. BigQuery enables you to perform SQL queries on large datasets in a scalable and fast way. In particular, the CrUX (Chrome User Experience Report) data set was used, a real data set that measures the user experience on websites using performance metrics such as Largest Contentful Paint (LCP), First Input Delay (FID) and Cumulative Layout Shift (CLS). This data, collected from users of the Chrome browser, provides a real perspective on website performance from the end-user's point of view.

The BigQuery and CrUX integration enables advanced queries to real-world website performance data, allowing for detailed analysis at scale. You can segment data based on variables such as device, geolocation and operating system. For this research, a query was processed to extract a list of domains located in Europe, based on the column `experimental.popularity.rank`, which represents the ranking of popularity of sites. The objective was to select the most visited European domains in order to analyse the panorama of digital certificates issued within the European context.

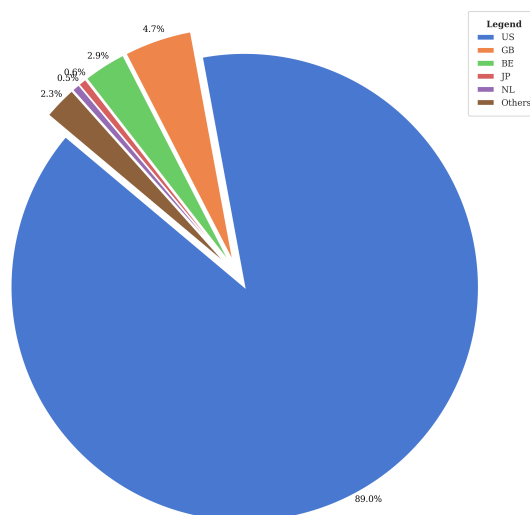


Figure 4.1. Distribution of issued certificates per country in the DomCop dataset.

This choice is motivated by the observation that in the DomCop Top10M list 89% of domains have a digital certificate issued by a US Certificate Authority (CA). The specific situation of digital certificates issued by European CAs has therefore been examined in order to compare the two geographical contexts.

## 4.1 DigitalCertiAnalytics: System Architecture and Key Features

To carry out the analyses described, it was decided to take the scripts developed by the author of the thesis [26], in order to reuse and optimize the work previously carried out. However, as the same thesis points out, the preparation of a list of 1 million domains had taken about 20 days, which is not acceptable for an analysis of a significantly larger dataset containing 20 million domains (10 million from the DomCop Top10M list and 10 million from Google’s CrUX list).

At first, parallelization across the original Python scripts was implemented to improve their execution performance through a use of multiple cores and multi-threading/multiprocessing. This is, in itself, was a rather challenging task and has yielded very little in terms of output because of the different performance of available computing systems. It is binding since one is in need of high-performance hardware and enormous processing capacity.

After months of development and numerous optimization attempts, DigitalCertiAnalytics was created, an advanced tool designed to address the challenges of large-scale analysis of digital certificates. DigitalCertiAnalytics allows you to explore multiple aspects of certificates, including validity, signature algorithms and security policies, while offering a display of the results in intuitive graphs. It also identifies non-compliance with the requirements of the CA/B Forum, as well

as technical regulations such as RFC 5280 and other relevant guidelines. These aspects will be discussed in detail in the following sections.

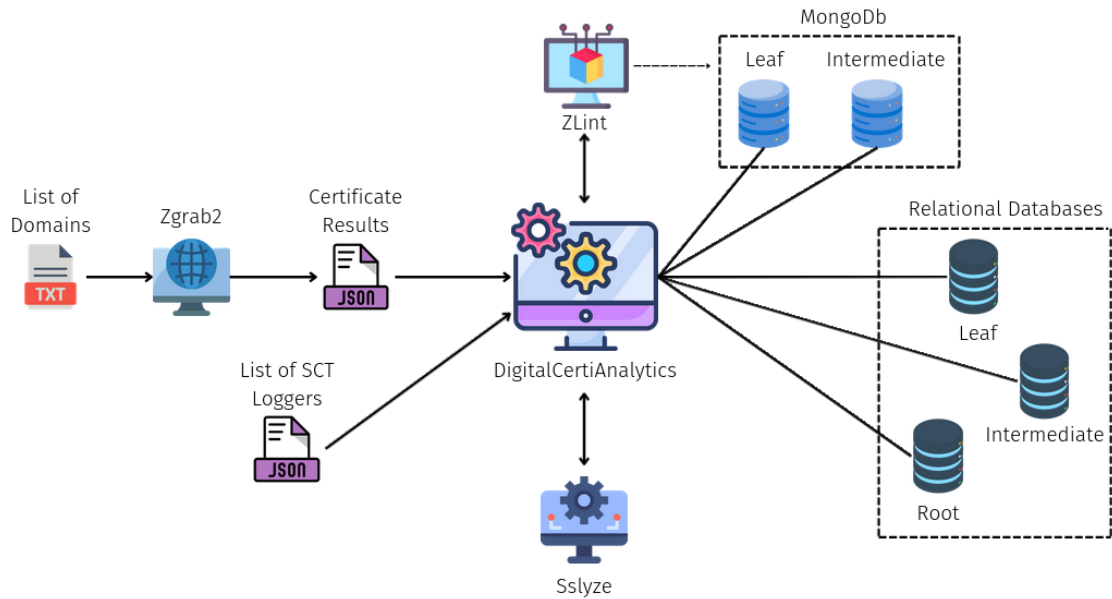


Figure 4.2. Architecture of DigitalCertiAnalytics Tool for Large-Scale Certificate Analysis.

### 4.1.1 Core Components of DigitalCertiAnalytics

The **DigitalCertiAnalytics** software, developed in Python, is composed of basic modules that guarantee its proper functionality. It operates by integrating and analyzing data from various industry tools that have been specifically selected due to their capabilities in performing partial analyses related to digital certificates. This architecture enables the re-use of existing tools thus making the whole process faster and more accurate.


For its operation, the software requires as input a file in JSON format containing the digital certificates related to the list of domains subject to analysis. The starting base is a list of domains, which can be supplied in a simple text file (TXT), where each line represents a distinct domain.

In an early stage of development, following a similar approach to the Python scripts described in [26], we evaluated the use of the OpenSSL library, or the execution of OpenSSL commands through the Python subprocess module (For example, using the `s_client` command). However, this methodology has not been effective for downloading digital certificates. This approach would require individual connections to each server associated with the domains, with the additional need to set a timeout to manage any missed links. This process would have been a significant waste of time, and would not be feasible for datasets consisting of millions of domains.

To overcome this limitation, it was decided to explore tools already available, identifying ZGrab2 as a highly effective solution. ZGrab2 has proven particularly useful in automating and speeding up the process of acquiring digital certificates on a large scale, reducing processing times and enabling faster and scalable analysis.

### 4.1.2 Integration of ZGrab2 for Advanced Network Scanning

ZGrab2 is an advanced network scanning tool, part of the open-source ZMap project [30] and supported by the National Science Foundation [31]. Designed for large-scale data collection and security assessments, ZGrab2 analyses lists of domains from a text file in which each line contains IP, domain and tags; in the absence of IPs, the domain is resolved via DNS. The tool establishes TLS connections and produces a JSON file containing connection details, including errors, protocol and timestamps.



**The ZMap Project**  
The ZMap Project is a collection of open source measurement tools for performing large-scale studies of the hosts and services that compose the public Internet.

<p><b>ZMap</b></p> <p>ZMap is a fast single-packet network scanner optimized for Internet-wide network surveys. On a computer with a gigabit connection, ZMap can scan the entire public IPv4 address space on a single port in under 45 minutes. With a 10GigE connection and PF_RING, ZMap can scan the IPv4 address space in 5 minutes.</p>	<p><b>ZGrab</b></p> <p>ZGrab is a stateful application-layer scanner. ZGrab is written in Go and supports HTTP, HTTPS, SSH, Telnet, FTP, SMTP, POP3, IMAP, Modbus, BACNET, Siemens S7, and Tridium Fox. For example, ZGrab can perform a TLS connection and collect the root HTTP page of all hosts ZMap finds on TCP/443.</p>	<p><b>ZDNS</b></p> <p>ZDNS is a utility for performing fast DNS lookups, such as completing an A lookup for all names in a zone file, or collecting CAA records for a large number of websites. ZDNS contains its own recursive resolver and supports A, AAAA, ANY, AXFR, CAA, CNAME, DMARC, MX, NS, PTR, TXT, SOA, and SPF records.</p>
<p><b>LZR</b></p> <p>LZR is a scanner that efficiently identifies what protocol an Internet service runs. LZR can identify 99% of unexpected Internet services in five handshakes. It runs as shim between ZMap and ZGrab.</p>	<p><b>ZCrypto</b></p> <p>ZCrypto is a TLS and X.509 library for researchers. It is based on Go's standard library, but supports a more extensive set of cipher suites, extra lenient ASN.1 and X.509 parsing, and handshake transcription.</p>	<p><b>ZLint</b></p> <p>ZLint is an X.509 certificate linter that checks for conformity with X.509 RFCs, CA/Browser Forum baseline requirements, root store policies, and ETSI standards.</p>
<p><b>ZCertificate</b></p> <p>ZCertificate is a command-line utility that parses X.509 certificates, performs browser validation and ZLint tests, and produces a JSON encoding of the certificate.</p>	<p><b>ZAnnotate</b></p> <p>ZAnnotate is a utility that annotates IPs with additional metadata, such as Maxmind GeoIP2 locations and routing data from a TABLE_DUMPv2 MRT file.</p>	<p><b>ZIterate</b></p> <p>ZIterate is a utility that will produce random permutations of the IPv4 address space. It supports selecting IPs from a set of networks and sharding across multiple servers.</p>

Figure 4.3. Illustration of the ZMap project, of which ZGrab2 and ZLint are components.

The distinguishing trait of ZGrab2 is automatic parsing of digital certificates. Apart from this, the JSON output includes not only the raw form of the certificate, but also processed information: issuer, subject, extensions present and the complete certificate chain. ZGrab2 also supports OCSP stapling, the server can include an OCSP response confirming the non-revocation of the certificate, passed to the client on a TLS handshake, as an additional layer of security without having to make another request from the client.

In addition, the tool extracts Signed Certificate Timestamps (SCTs), which attest to the certificate inclusion in public Certificate Transparency registers, ensuring transparency and protection against fraudulent certificate issuance. Certificate fingerprints, calculated with algorithms such as SHA-256, MD5 and SHA-1, are also provided for the unique identification of certificates.

The utility ZGrab2 executes a line-by-line processing of the input file, tries to establish a TLS connection for every one of its domains, and writes results into a large JSON file whose size depends solely on the number of domains scanned. This integrated approach, featuring automatic data collection along with a thorough certificate parsing process, comes in very handy for an advanced study of the digital certificate ecosystem, allowing for detailed systematic analyses of Use Cases.

### **4.1.3 Relational Database Persistence Layer**

The persistence of digital certificates was organised through an architecture based on relational databases, divided into three distinct schemas: Leaf, Intermediate and Root. This separation reflects the typical hierarchy of PKI and allows targeted analysis for each type of certificate. Although Root certificates are usually pre-installed in the client's trust stores, it was decided to create a dedicated schema for Root certificates that are transmitted during the TLS process, in order to comprehensively analyse them.

The database was created using SQLite3, a serverless solution known for its lightness, speed and ease of management, which allows all data to be stored in a single, easily transferable file. For data management and visualisation, the DB Browser for SQLite programme was used. This relational structure guarantees a fixed and consistent schema, which is essential for the accurate analysis of certificates, and allows the creation of simple backups, which is particularly relevant given the considerable amount of data (JSON files for Leaf and Intermediate certificates reach an average size of about 80 GB).

In addition, submission of the detailed public log information, including the log IDs, is necessary to analyze the Signed Certificate Timestamps (SCTs), important in giving assurance to the transparency and integrity of certificates. SCTs provide for proof of registering a certificate in one or in many Certificate Transparency logs, and information concerning those logs is stored within the database in special tables. All this provides the possibilities to better assess the distribution of certificates and activities of key players as a means to monitor the evolution of the digital certificate landscape.

### **4.1.4 Zlint: Compliance Verification Tool**

In computer science, a lint (or linter) is a programme that performs a static analysis of code, detecting possible errors and suggesting improvements. In the context of DigitalCertiAnalytics, the detection of non-conformities with standards, such as those of the CA/Browser Forum, RFCs and Mozilla PKI specifications, is a key

component of digital certificate analysis. An initial approach could have been to collect the rules of the main standards and implement them directly in the software, assigning different severities according to the type of violation. However, such a solution would have entailed rigidity and less flexibility, making any update or modification of the rules a matter of direct intervention in the source code. Although DigitalCertiAnalytics is open source, this methodology would have limited the accessibility of the programme to experienced users only, also exposing it to potential bugs resulting from manual modifications.

To overcome these limitations, it was decided not to reinvent the wheel, but to integrate existing solutions that perform this task more efficiently and can be updated automatically. To this end, it was decided to incorporate Zlint, a key tool of the ZMap [30], which offers several key functionalities, such as verifying compliance with standards (CA/Browser Forum, RFC, Mozilla PKI), checking the validity and correctness of TLS/SSL certificates, and detecting violations of policies and best practices.

Zlint [8] is an open source tool aimed at identifying errors and vulnerabilities in X.509 format certificates - the standard used in PKIs. Its series of automated checks (known as linters) that can be applied to certificates allows for the detection of various misconfigurations, insecure practices and inconsistencies, which could detract from the security of SSL/TLS-protected communications. With the integration of Zlint into DigitalCertiAnalytics, the system enjoys the benefits of having a current tool that supports many security standards, thus rendering implementation easier and minimizing the risks arising from manual code changes.

Zlint's version v3.6.4-9-g989baefd was used for the analysis, which takes into account different security standards, thus ensuring that the certificates analysed comply with the latest and most relevant industry standards. Zlint examines various aspects of certificates, such as the correct setting of fields, the safe use of keys and compliance with requirements set by major international standards, taking into account different standards, which include:

- **Apple:** Apple has its own guidelines for digital certificates, especially for security and compatibility with its operating systems (macOS, iOS). Zlint considers these requirements to ensure that certificates issued for Apple devices comply with the highest security standards.
- **CABF\_BR** (Certificate Authority Browser Forum – Baseline Requirements): These are the basic requirements set by **CAB Forum** [4], a consortium of certification authorities (CAs) and browser manufacturers. The **CA/B Forum Baseline Requirements** are designed to ensure that digital certificates are issued in a secure and reliable manner, with specifications regarding key management, authentication, validity and revocation.
- **CABF\_CS\_BR** (CABF Code Signing Baseline Requirements): Similar to the Baseline Requirements, they differ mainly in the code signing certificate, which implements them for any software or application to be signed. These are standards meant to guarantee the integrity of digitally signed software from an authentic source and has not been tampered with.

- **CABF\_EV** (Extended Validation): The Extended Validation (EV) certificates are a type of certificate that requires more rigorous validation of the entity requesting the certificate. These certificates are recognized by many browsers and provide additional security for websites, showing the organization name in the address bar.
- **CABF\_SMIME\_BR** (CABF S/MIME Baseline Requirements): This S/MIME certificate stands for Secure/Multipurpose Internet Mail Extensions and is used to sign and encrypt email. The basic requirements established by CABF about S/MIME are somewhat similar to those of SSL/TLS certificates, but they will apply itself to the scope of email communications.
- **Community**: This is a set of community-defined standards and guidelines that could cover a variety of common practices in the area of digital certificate security and management.
- **ETSI\_ESI** (European Telecommunications Standards Institute – Electronic Signatures and Trust Services): This standard includes the use of electronic signatures and trust services over Europe. ETSI sets out guidelines for ensuring that electronic signatures shall be legally accepted and secure in line with eIDAS (Electronic Identification and Trust Services) regulations.
- **Mozilla**: Mozilla, the founder of Firefox, has its own compliance requirements for issued digital certificates. These requirements are used to ensure that the certificates are compatible with the Firefox browser and meet Mozilla’s security standards.
- **RFC3279**: The RFC 3279 specifies signature algorithms and certificate formats for X.509 digital certificates, especially for use in digital signature environments and secure communications. These standards refer to the specification of how encryption algorithms should be configured.
- **RFC5280**: The RFC 5280 [6] defines the specifications for X.509 certificates, which are the basis for most digital certificates. Contains detailed requirements for the structure, management and validity of certificates, as well as establishing best practices for their creation and management.
- **RFC5480**: The RFC 5480 describes guidelines for using elliptic curve signature (ECC) algorithms in X.509 digital certificates. It is particularly useful for issuing certificates based on ECC encryption, which is more secure and less expensive in terms of resources than other types of cryptography.
- **RFC5891**: The RFC 5891 is the specification describing international domains for SSL/TLS certificates, including domains with non-ASCII characters, such as those containing special characters from non-Latin languages.
- **RFC6962**: The RFC 6962 describes the Certificate Transparency system (as discussed in Chapter 3), a mechanism designed to ensure transparency in issuing SSL/TLS certificates. This standard was developed to prevent fraudulent certificate issuance.
- **RFC8813**: The RFC 8813 focuses on key management, authentication practices and long-term validity of certificates.

It is clear that Zlint, taking into account a wide range of international standards and policies, is able to conduct very rigorous audits of digital certificates. These standards ensure that certificates comply with security best practices, legal regulations and technical specifications, contributing to transparent and secure management of the entire certificate infrastructure.

Initially, Zlint focused mainly on supporting the RFC 5280 standard [6] and the Baseline Requirements of the CA/Browser Forum [4], which were the main references for issuing certificates, especially those used in SSL/TLS connections. In its initial version, Zlint included some 220 linters, capable of verifying 95 per cent of the clauses related to the Baseline Requirements and 90 per cent of those in RFC 5280, thus guaranteeing that certificates complied with the established guidelines for security and reliability, as highlighted in the reference article [32].

Over time, the need for greater transparency and the increase in the complexity of regulations led to an expansion in the number of linters. In the current version, used within DigitalCertiAnalytics, the number of linters has increased to 370. This expansion has enabled the inclusion of additional international standards and regulations, including those relating to **Apple**, **Mozilla**, **ETSI** and other industry regulations, making Zlint an even more versatile tool capable of detecting a wider range of errors or inconsistencies in digital certificates.

Zlint classifies the linters it uses to verify certificates into different levels of severity, depending on the extent of non-conformance with the requirements in the documents of the CA/Browser Forum and the Baseline Requirements:

- **INFO**: Indicates situations that do not represent critical violations, but could be improved in the future, such as sub-optimal completion of optional fields.
- **WARNING**: Indicates deviations from recommendations (SHOULD) that do not invalidate the certificate, but could lead to risks or non-optimal optimisation.
- **ERROR**: Denotes violations of mandatory requirements (MUST) that compromise the validity or security of the certificate.
- **FATAL**: This is a violation that is so serious that it renders the certificate unusable and goes beyond the ERROR level.

Zlint also uses two further results: **NE** (Not Evaluated) when the check has not been applied, and **NA** (Not Applicable) when the rule is not relevant for the certificate under consideration. These categories allow a modular and precise assessment of certificate compliance with international regulations and best practices, facilitating the early detection of any vulnerabilities.

#### **4.1.5 Non-Relational Database Persistence Layer**

The choice of a non-relational database for managing the results provided by Zlint was dictated by the nature of the data produced. Zlint, in fact, generates an output in JSON format, which includes a large number of fields, each containing the result of the checks performed (as discussed above). This rich and complex



structure makes it more difficult to manage in a traditional relational database, which is characterised by a rigid schema and less suited to easily manage changes or additions of new fields in the future.

For these reasons, MongoDB was chosen as the storage solution. MongoDB, being a non-relational document-based database, offers greater flexibility thanks to the absence of a fixed schema and the use of collections and documents, which naturally adapt to the JSON structure of Zlint's results. This flexibility allows you to easily store every output of Zlint without having to restructure the database in case of updates or changes in the data structure. In addition to flexibility, MongoDB offers high performance in terms of write and read speed. This is particularly useful for quick queries of data and immediate answers during subsequent analyses or verifications.

The Zlint analysis was strictly limited to leaf and intermediate certificates because these are by far the most important and the most numerous entities in the context of the verifications. Root certificates, in the smallest quantity and with static characteristics, did not demand that same attention.

The software operates in a well-defined flow:

1. Scrolls through each certificate in the relational database.
2. Invokes Zlint to perform certificate checks.
3. Receives results in JSON format.
4. Stores those results in the non-relational database MongoDB.

To make communication between the local database and the MongoDB database efficient, pymongo version 4.10.1 was used, which provides a fairly clear and comprehensive interface to MongoDB itself.

#### **4.1.6 SSLyze: Certificate Chain Verification**

A digital certificate is only considered valid if its certification chain is valid, i.e. if every signature along the chain can be successfully verified. In practice, each certificate is signed by an authority that makes its certificate available containing the public key needed to verify the integrity and legitimacy of the certificate. This verification process makes it possible to trace the chain back to the root certificate, which signs itself. However, the validity of the chain is not enough: it is essential to verify that the root certificate is present in the local trust store, so as to guarantee trust in the entire chain.

In some cases, the root certificate is included in the transmitted chain, but more frequently it is omitted, necessitating direct retrieval from the trust store based on the issuer data contained in the last intermediate certificate.

To meet this challenge without having to develop a complex new system, SSLyze, an open source tool [9] written in Python and designed to analyse the security

of SSL/TLS configurations, was integrated. SSLyze performs an in-depth analysis of the SSL/TLS versions supported by the servers and verifies the validity of the certificate chains, ensuring compliance with security standards. Thanks to its integrated trust store, SSLyze can automatically check chain validity and periodically update trusted root certificates [33].

The integration of SSLyze was realised via the Python module `subprocess`, which allows SSLyze to be executed as an external process. The developed system uses the hosts stored in the relational database to query SSLyze and obtain chain validation. The results, provided in JSON format, are analysed and stored in the database, keeping only the data strictly necessary for analysis.

It is important to note that SSLyze allows the verification of certification chains against different trust stores:

- Android CA Store (15.0.0.r5)
- Apple CA Store (iOS 17, iPadOS 17, macOS 14, tvOS 17, and watchOS 10)
- Java CA Store (jdk-13.0.2)
- Mozilla CA Store (2024-09-01)
- Windows CA Store (2023-12-11)

So, it is enough that one of the trust stores used by SSLyze validate the certificate to consider the whole chain as trusted. This logic allows for a wide coverage in the audits, since it relies on multiple trust stores to determine the chain's trust. In addition, SSLyze provides results related to the control of additional features such as OCSP Must-Staple and Certificate Transparency. For the latter, the instrument indicates the number of SCT (Signed Certificate Timestamps) present in the certificate. However, this information has not been taken into account at this stage as the DigitalCertiAnalytics software already implements such checks independently.

#### **4.1.7 Certificate Revocation Status Checking**

One key aspect to be analysed is the verification of the revocation status of a digital certificate. As explained in the Section 2.7, this process is essential to determine whether a certificate is still valid or has been revoked before its natural expiration, thus affecting the security and reliability of communication. In the DigitalCertiAnalytics system, this verification is carried out by two methods: consultation of CRL lists and use of the OCSP protocol.

In the first case, when the certificate is inserted into the database, the CRL lists are consulted to check whether the serial number of the certificate is among those revoked. Before proceeding with the verification, the validity of the CRL list is checked by checking its expiration date and if the list has expired, it is not considered reliable. In this process, the requests library (version 2.28.1) was used to make HTTP requests to CRLs. This is an improvement over the [26] work, where such control was not carried out, which could compromise the accuracy of the revocation status verification.

In the second case, OCSP verification starts by checking for the presence of the AIA field in the certificate. If the field is absent, the impossibility of performing verification via OCSP is reported. If it is present and contains a valid URL for the Online Certificate Status Protocol (OCSP), a request to the corresponding OCSP server is built and executed. The implementation uses aiohttp (version 3.10.10) to make asynchronous requests. First, an OCSP request is built using the OCSPRequestBuilder class, to which the certificate to be verified, the issuer certificate and the required hashing algorithm are added, thus generating the request. The request is then executed with a 20-second timeout. If the OCSP server response was HTTP 200 OK, the returned data is decoded and analyzed.

```

async with aiohttp.ClientSession() as session:
    builder = ocsplib.OCSPRequestBuilder()
    builder = builder.add_certificate(current_certificate, issuer_certificate, alg)
    req = builder.build()
    req_path = base64.b64encode(req.public_bytes(serialization.Encoding.DER))
    final_url = urljoin(ocsp_link + '/', req_path.decode('ascii'))
    async with session.get(final_url, timeout=20) as response:
        try:
            result = "Impossible Retrieve OCSP Information"

            if response.status == 200:
                ocsplib_resp = await response.read()
                ocsplib_decoded = ocsplib.load_der_ocsp_response(ocsp_resp)
                if ocsplib_decoded.response_status == OCSPResponseStatus.SUCCESSFUL:
                    if ocsplib_decoded.certificate_status == OCSPCertStatus.GOOD:
                        result = "Good"
                    elif ocsplib_decoded.certificate_status == OCSPCertStatus.REVOKED:
                        result = "Revoked"
                    elif ocsplib_decoded.certificate_status == OCSPCertStatus.UNKNOWN:
                        result = "Unknown"
                else:
                    result = "Impossible Retrieve OCSP Information"
            else:
                result = "Not Ok OCSP Response"

            return result
        except asyncio.TimeoutError:
            return "Impossible Retrieve OCSP Information"
        except aiohttp.ClientTimeout:
            return "Impossible Retrieve OCSP Information"
        except aiohttp.ClientResponseError as e:
            logging.error(f"Errore durante la richiesta OCSP per {final_url}: {e}")
            return "Impossible Retrieve OCSP Information"
        except Exception:
            logging.error(f"Errore durante la richiesta OCSP per {final_url}: {e}")
            return "Impossible Retrieve OCSP Information"

```

Figure 4.4. Code for sending an OCSP request to check the revocation status of a digital certificate.

At this point, as shown in figure 4.4, depending on the certificate status contained in the OCSP response, one of the following results can be obtained:

- **Good:** The certificate is valid and not revoked.
- **Revoked:** The certificate has been revoked and is no longer reliable.
- **Unknown:** The OCSP server is unable to determine certificate status.

A status code other than 200 for the OCSP response means that the result returned is **Not Ok OCSP Response**, which indicates an error in communication with the OCSP server.

The advantage of using both CRL and OCSP is that if you have problems with one of these two methods, you can use the other to ensure a more reliable and resilient verification of the certificate revocation status, reducing the risk of false negatives.

#### **4.1.8 Data Visualization and Chart Generation**

DigitalCertiAnalytics software includes advanced methods for generating graphs related to the collected data, focusing on the most relevant and significant aspects. These charts are saved as images in .png format within a dedicated directory structure, located in the `analysis` folder. Each certificate category has its own subfolder: `leaf` for leaf certificates, `intermediate` for intermediate certificates and `root` for root certificates.

Matplotlib (version 3.9.2) and seaborn (version 0.13.2) libraries were used in the making of the graphs due to their flexibility and clear aesthetic professional representation. As such, it was introduced to highlight data and make it understandable at first glance, especially in dealing with large volumes of information.

The graphs provide a concise and effective overview, making it easier to identify trends, anomalies or specific problems than simply reading numbers or descriptions. For example, they can visually represent the percentage distribution of valid, expired or revoked certificates, giving a clear perception of the overall status of the certificates.

The OCSP analysis, which checks the revocation status of certificates, is also represented graphically, showing whether the certificates have been validated correctly or if revocation problems have occurred. This visual representation facilitates the understanding of validity states. And many other features and functionalities that will be discussed later.

## **4.2 User Interaction with DigitalCertiAnalytics**

Interaction with the **DigitalCertiAnalytics** software is exclusively via command line, providing a simple and direct interface for performing different types of analysis. As shown in figure 4.5, the user has a set of dedicated commands to access all the features discussed above.

The operation modes are designed for maximum flexibility and control: they enable users to specify precisely what parameters of the analyses should be taken into account. Each command has been assigned to an individual function of the software, making the interface self-explanatory even for users with only basic technical knowledge.

The available commands allow, for example, to start the analysis of digital certificates, generate detailed reports, perform OCSP verifications and obtain visual graphs of the collected data. The control line is therefore the central point of interaction, allowing efficient use and adaptable to different operational scenarios.

```

DigitalCertiAnalytics

usage: python -m analysis.main [-h] [--delete_all_db] [--delete_leaf_db] [--delete_intermediate_db] [--delete_root_db] [--leaf_analysis] [--leaf_ocsp_analysis] [--leaf_zlint_check] [--leaf_chain_validation]
                             [--intermediate_analysis] [--intermediate_ocsp_analysis] [--intermediate_zlint_check] [--root_analysis] [--root_ocsp_analysis] [--plot_all_results] [--plot_leaf_results]
                             [--plot_intermediate_results] [--plot_root_results] [-v]

Strumento per l'analisi dei certificati digitali.

options:
  -h, --help                Mostra le opzioni disponibili per l'analisi dei certificati digitali e la generazione di grafici.
  --delete_all_db           Se presente, elimina tutti i database prima di iniziare.
  --delete_leaf_db         Se presente, elimina il database leaf prima di iniziare.
  --delete_intermediate_db Se presente, elimina il database intermediate prima di iniziare.
  --delete_root_db         Se presente, elimina il database root prima di iniziare.
  --leaf_analysis           Rimuove il database esistente e analizza i certificati leaf nel file JSON generato da zgrab2.
  --leaf_ocsp_analysis      Esegue l'analisi OCSP per i certificati leaf presenti nel database.
  --leaf_zlint_check        Esegue l'analisi Zlint sui certificati leaf per verificare eventuali vulnerabilità e configurazioni errate secondo determinati requisiti ufficiali.
  --leaf_chain_validation   Esegue la validazione della catena dei certificati leaf per verificare la conformità e l'affidabilità della catena di trust.
  --intermediate_analysis   Rimuove il database esistente e analizza ed analizza i certificati intermediate nel file JSON generato da zgrab2.
  --intermediate_ocsp_analysis Esegue l'analisi OCSP per i certificati intermediate presenti nel database.
  --intermediate_zlint_check Esegue l'analisi Zlint sui certificati intermediate per verificare eventuali vulnerabilità e configurazioni errate secondo determinati requisiti ufficiali.
  --root_analysis           Rimuove il database esistente e analizza i certificati root nel file JSON generato da zgrab2.
  --root_ocsp_analysis      Esegue l'analisi OCSP per i certificati root presenti nel database.
  --plot_all_results        Genera e visualizza i grafici per tutti i dati analizzati sui certificati.
  --plot_leaf_results       Genera e visualizza i grafici per i risultati dell'analisi dei certificati leaf.
  --plot_intermediate_results Genera e visualizza i grafici per i risultati dell'analisi dei certificati intermedi.
  --plot_root_results       Genera e visualizza i grafici per i risultati dell'analisi dei certificati root.
  -v, --verbose            Attiva la modalità verbose per una registrazione dettagliata.

Utilizza le opzioni disponibili per eseguire diverse analisi sui certificati e generare report visivi. Usa -v per attivare la modalità verbose per ulteriori dettagli.
(base) elio@manifium-vm-01:~/src/~/TEST_TESI/src/

```

Figure 4.5. Command line interface for interacting with DigitalCertiAnalytics.

- Options for deleting databases:** The options `--delete_all_db`, `--delete_leaf_db`, `--delete_intermediate_db`, and `--delete_root_db` are provided to remove certain databases or all the databases before embarking on the analysis. Particularly useful when is necessary to delete data from a previous analysis and start again.
- Certificate analysis options:** Options like `--leaf_analysis`, `--intermediate_analysis` and `--root_analysis` are essential for starting the analysis of certificates in their respective categories (leaf, intermediate, root). These categories refer to the various levels of the certification chain:
  - **Leaf:** the final certificate, directly linked to the server.
  - **Intermediate:** certificates that are intermediates between the root certificate and the leaf certificate.
  - **Root:** the main certificate that signs the authority certificate. These certificates are also sent within the chain.
- Each analysis involves deleting the previous databases to ensure a new and accurate data collection.
- OCSP and Zlint analysis options:** The OCSP analysis (`--leaf_ocsp_analysis`, `--intermediate_ocsp_analysis`, `--root_ocsp_analysis`) checks the revocation status of certificates using the OCSP (Online Certificate Status Protocol). This analysis is useful for confirming that the certificates have not been revoked and are still valid. The Zlint analysis (`--leaf_zlint_check`, `--intermediate_zlint_check`) performs a check on certificates using Zlint, verifying the certificates' compliance with security policies and validity standards.
- Certificate chain validation:** The option `--leaf_chain_validation` validates the leaf certificate chain. This means that all certificates in the chain are valid and from a trusted set of certificates.

- **Chart generation options:** The options `--plot_all_results`, `--plot_leaf_results`, `--plot_intermediate_results`, and `--plot_root_results` allow you to generate visual charts based on the analysis results. Charts can be used to display metrics such as the distribution of vulnerabilities, certificate validity, and other important information.
- **Verbose mode:** The options `-v` or `--verbose` to enable verbose mode provides more information on how commands are being executed. Hence, if you want more information on the process or to debug some issue, use this.

### 4.3 System Specifications for Zgrab2 Execution

During the analysis using the tool Zgrab2 (4.1.2), a system with Intel Core i7-8550U (quad-core, 1.80 GHz - 2.00 GHz), 8 GB of DDR4 RAM at 2400 MHz and a 510 GB SSD was used. From a connectivity point of view, the system had a Qualcomm Atheros QCA9377 wireless interface with a theoretical maximum connection speed of about 650 Mbps. The quality of the network played a key role in the speed with which requests to servers and data collection were made. In addition, there was a VirtualBox Host-Only Ethernet Adapter virtual network adapter, which, while not directly involved in the scan, is part of the system software configuration and may have affected the interactions with the virtualized environment.

## Chapter 5

# Digital Certificate Analysis Using the DomCop Dataset

Starting with the dataset provided by **DomCop** containing 10 million domains, the process of collecting X.509 digital certificates was started on 10 October 2024 using ZGrab2 (as shown in Figure 4.2).

```
C:\Users\franc\Desktop\zgrab2>zgrab2 tls --port 443 -f C:/Users/franc/Desktop/top-10m-websites.txt --output-file=certs_p
olito.json --timeout=30
time="2024-10-18T11:08:05+02:00" level=info msg="started grab at 2024-10-18T11:08:05+02:00"
time="2024-10-18T18:41:28+02:00" level=info msg="finished grab at 2024-10-18T18:41:28+02:00"
{"statuses":{"tls":{"successes":7045024,"failures":2954976}}, "start":"2024-10-18T11:08:05+02:00", "end":"2024-10-18T18:41
:28+02:00", "duration":"7h33m22.8423222s"}
```

Figure 5.1. ZGrab2 scan results for domains in the DomCop dataset.

The execution time of the operation took **7 hours and 33 minutes**, generating an 82 GB JSON file. Each line of the file consisted of a domain result, and there was a status field within such a line to represent **success** or **error** in that operation. The list of types of errors was encountered:

- **Connection-timeout**: Connection timeout, probably due to latency issues or unreachable domains.
- **I/O-timeout (14.61%)**: Timeout during input/output operations, suggesting possible data transfer issues.
- **Unknown-error (2.04%)**: Errors of unknown nature, which could result from anomalies in the protocol or unforeseen situations.

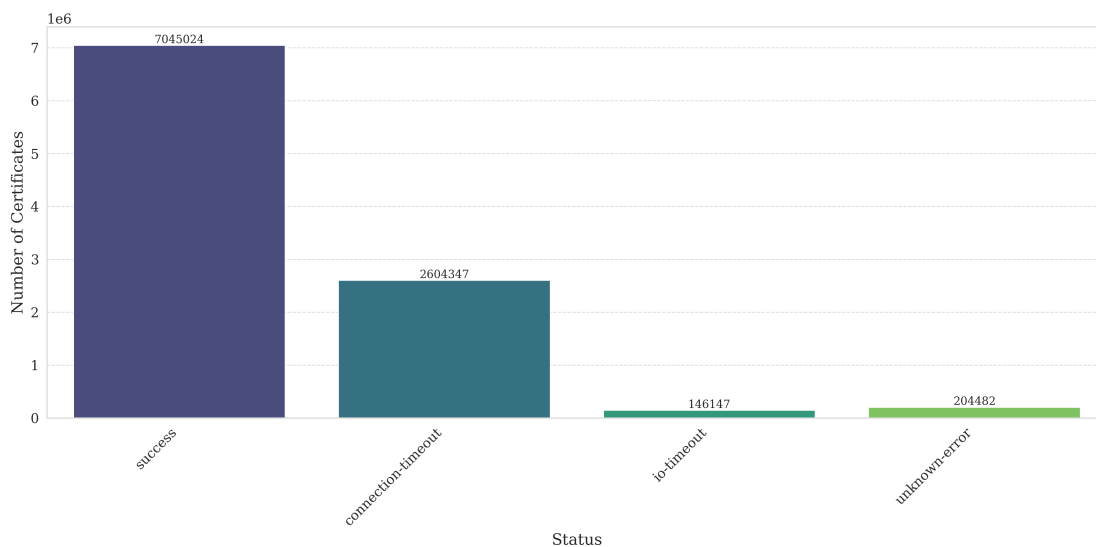


Figure 5.2. Certificate Status Analysis.

As shown in Figure 5.2, the process returned 7,045,024 valid X.509 leaf certificates (**70,45%**), while 2,954,976 domains (**29,55%**) failed due to connection errors or other anomalies.

In this study, the analysis of certificates was divided into two main phases. In a first step, leaf certificates were examined in depth, analysing all aspects and fields to understand the configurations adopted in the final part of the chain of trust. In subsequent sections, intermediate and root certificates (as distinct from the root trusts pre-installed in the systems) transmitted during TLS/SSL negotiation were evaluated, in order to obtain a complete view of the entire structure of the certificate chain.

The following certificates were then imported and analyzed with the **Digital-CertiAnalytics** a process that took 19 hours and 46 minutes. Compared to the previous study [26], where it took **21 days** to analyze **645,332 domains**, this huge improvement allows for further scaling analyses to even occur on a daily basis, therefore, the constantly evolving digital certificate landscape can be monitored. The primary limit here is, however, needed memory space for the great amounts of generated data.

In addition, it is important to note that the total number of leaf certificates (equal to 7,045,024) also includes duplicate certificates, which may result from the presence of multiple Subject Alternative Names (SANs) within the same certificate. Therefore, the number of unique leaf certificates is **5,000,264**. This value is significantly higher than the number of distinct domains collected, which amounted to **442,331**, so this allows a more qualitative analysis to be conducted, considering the study as a benchmark at a general level given the availability of a limited number of data to be examined.



## 5.1 Overview of Leaf Certificates

A further important aspect concerns the distribution of leaf certificate versions. The analysis revealed that 99% of the certificates belong to X.509 version 3, while only the remaining 1% are distributed between versions 1 and 2, which is negligible overall. This result is not surprising, since version 3 introduces several key extensions, such as Key Usage, Extended Key Usage and Subject Alternative Name, which are not present in previous versions. Moreover, the RFC 5280 standard [6] profiles public key infrastructure (PKI) certificates based on version 3 itself, which is why most Certificate Authorities only issue X.509v3 certificates.

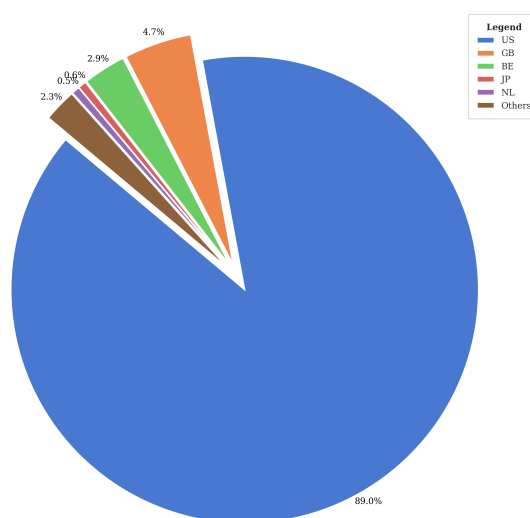


Figure 5.3. Distribution of issued certificates per country.

While, the geographical analysis of the leaf certificates (Figure 5.3) showed that 89% of the valid certificates were issued by US certification authorities. The remaining percentages were distributed as follows: 4.7% from UK authorities, 2.9% from Belgium, 0.6% from Japan, 0.5% from the Netherlands, and the remaining 2.3% from other authorities with smaller percentages. This distribution probably reflects the predominance of the US market in the area of digital certifications, as well as differences in policies and infrastructure adopted in other countries.

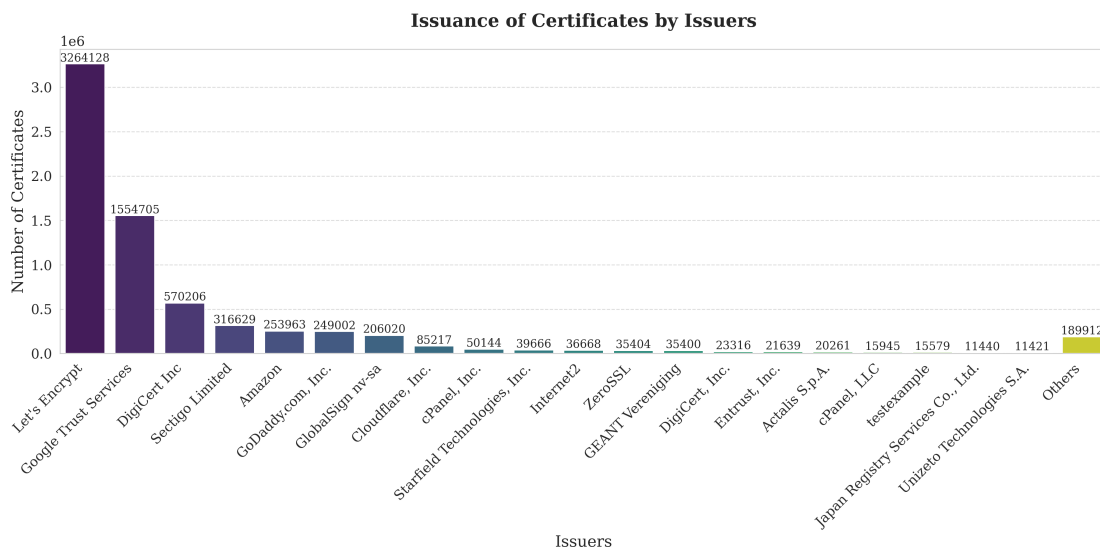


Figure 5.4. Distribution of leaf certificates issued by various certification authorities.

Looking at Figure 5.4, it is clear that the issuer landscape is strongly dominated by a few major players. In particular, Let's Encrypt is confirmed as the most popular Certificate Authority, responsible for issuing 46.33% of the certificates in the analysed dataset. This figure not only highlights the popularity of Let's Encrypt, but also reflects its strategy of free and automated issuance, which has made the adoption of TLS/SSL certificates accessible to a large number of sites, fostering the spread of secure connections on the Internet. Where market dominance has remained unchanged since 2021, it still holds the majority share of digital certificates issued.

Google Trust Services follows with 22.06% of certificates, indicative of the growing trust of web operators and applications towards issuers integrated in established ecosystems such as Google's. This significant presence may also be attributable to the close integration with Google Services, which supports the adoption of certificates in a transparent manner for domain owners and web platforms.

A further significant player is DigiCert Inc, which issued 8.09% of the certificates. DigiCert is known for its advanced security solutions and support for certificates with high validation standards, which makes it particularly popular in areas where security is a priority. Sectigo Limited and Amazon, with 4.49% and 3.60% respectively, complete the picture of major issuers. Sectigo, the result of the merger of Comodo CA with others, continues to maintain a solid market share, while the presence of Amazon highlights how cloud operators are also making inroads into issuing certificates, integrating their service offerings to guarantee secure connections.

The remaining issuers, with less significant percentages, complete an ecosystem in which a few players dominate the market. This concentration is particularly interesting from the point of view of security and interoperability: a market dominated by a few major players may guarantee high and uniform standards, but at

the same time it may present risks related to centralisation, where any vulnerability or compromise of an issuer may have significant impacts on a large scale.

### 5.1.1 Distribution of Certificate Validation Levels

The distribution of validation levels shows that **95.35%** of leaf certificates are issued with **DV** validation (Domain Validation), **3.62%** with **OV** validation (Organisation Validation) and only **0.28%** with **EV** validation (Extended Validation). These figures are in line with expectations, since DV certificates, which require exclusive verification of domain control, are generally easier and quicker to obtain, which is why they represent the majority. OV and EV certificates, which involve additional checks on the identity of the organisation, are in fact much less common, with EV, which follows particularly stringent verification procedures, present in a marginal percentage.

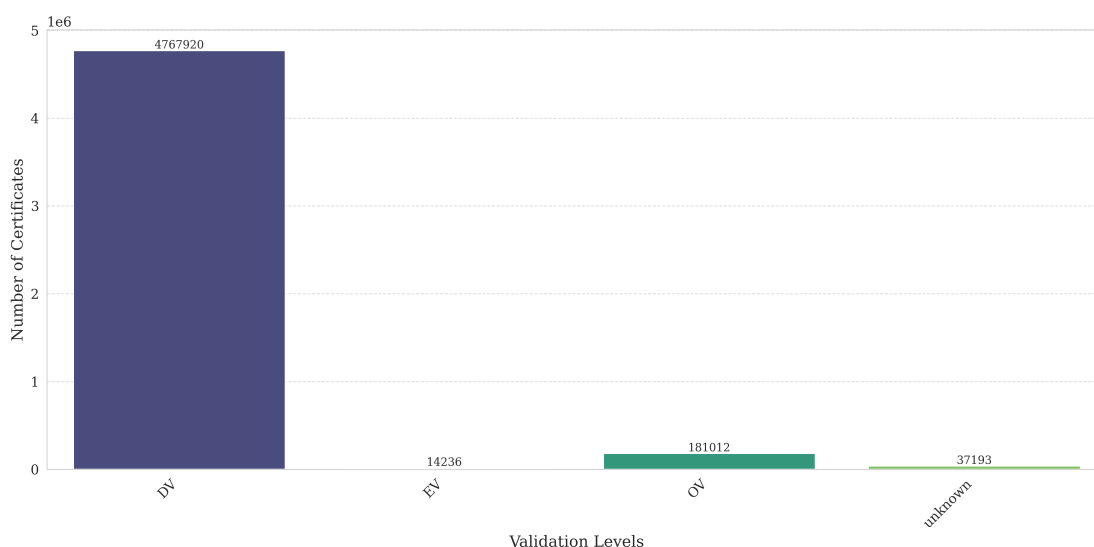


Figure 5.5. Distribution of Validation Levels across Leaf Certificates: DV, OV, and EV.

It is also interesting to note from the Figure 5.5 that 0.74% of certificates have no recognised validation level. This anomaly may result from errors in the issuing processes or from configurations that do not comply with the directives of the CA/Browser Forum, which, in its Baseline Requirements, clearly defines the validation criteria for publicly trusted certificates [4]. Whereas, the RFC 5280 standard does not explicitly define validation levels, these being the result of policies adopted by issuers.

### 5.1.2 Validity Duration Distribution and Expiry Trends

The distribution of certificate validity periods in the dataset reveals considerable heterogeneity that offers interesting insights both in terms of compliance with

current best practices and in relation to possible anomalies in the issuance and management processes. In this study, the validity of certificates was considered by analysing the fields `Not Before` and `Not After`, which indicate the start and expiry dates of the certificate respectively, integrating this analysis with the validation level of the certificate.

The level of validation is of particular importance as it reflects the evolution of issuing policies over time. Prior to April 2015, certificates for websites could be issued with significantly longer validity periods than the current standard. With the April 2015 update, the CA/Browser Forum recommended that certificates for websites should not exceed 39 months (about 3 years and 3 months), in order to ensure more frequent renewal and reduce the risk of prolonged use of outdated certificates. Subsequently, with the update of 1 September 2020, the maximum period of validity was further reduced: for all public certificates issued after that date, the validity cannot exceed 398 days (approx. 13 months). This reduction is intended to limit the time window during which a key can be exploited in the event of compromise, as well as to encourage regular checks on the identity of the subject. This policy also applies to EV certificates, which, although requiring a more rigorous validation process, were subject to the same limit of 398 days [4] [11].

The analysis shows that the majority of certificates, 81.76%, are valid for less than one year. Of these, 98% are DV certificates and 1.04% OV certificates, while the remainder are certificates with a validation level of less than 1%. This figure indicates the pervasive uptake of the CA/Browser Forum directives in the sense that the issuers have implemented a procedure to periodically reissue the certificates which-almost in factual terms-imply security for the entire PKI infrastructure. That in this sense is an obvious sign of compliance with contemporary standards, whereby exposure to risk is sought to be confined in the event of a key compromise.

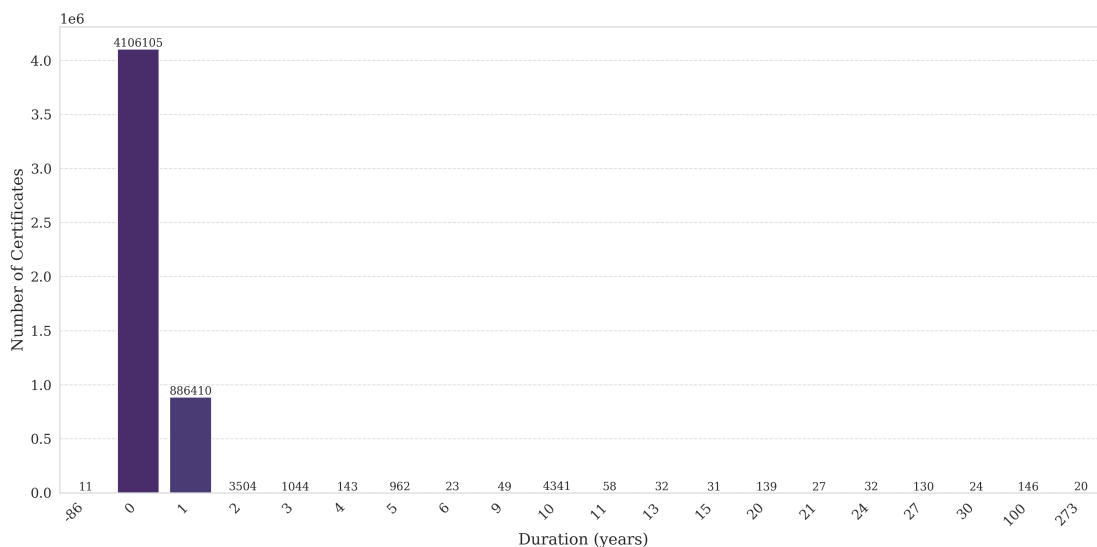


Figure 5.6. Distribution of the period of validity of certificates.

While, 16.80% of the certificates have a validity of around one year or slightly more, suggesting that there is some uniformity in the issuance of annual certificates, probably the result of company or regulatory policies that require periodic renewal to ensure that identification information is up-to-date. Within this group, 79.90% are DV certificates, 15.51% OV and the remaining 3.29% do not fall into any recognised validation category.

Less common, but negligible, is the 0.13% of certificates with a validity period of up to 10 years. Of these, 9.63% are DV certificates, 3.92% OV and 86.43% do not fall into any recognised validation category. Although these issuers are in the minority, the issuance of certificates with such extended validity is generally discouraged, as too long periods can increase the risk in the event of compromise and reduce the frequency with which the identity of the subject is verified.

Of particular concern, however, are the anomalies found: about 639 certificates report a duration of between 10 and 273 years, and about 11 certificates show negative validity values, or where the date in the `Not After` field precedes that in the `Not Before` field. These extreme values suggest probable configuration errors, a hypothesis that is further reinforced by the fact that these anomalies are found in certificates without a recognised validation category.

Furthermore, for certificates issued after 1 September 2020, 4,971,980 certificates (**99.43%**) have an overall validity of 398 days, in line with current requirements, while 3,444 certificates (**0.07%**) violate these limits, indicating that they exceed the permitted validity. Similarly, for certificates issued between April 2015 and 1 September 2020, 19,903 certificates (**86.60%**) comply with the 39-month limit, while 2,309 certificates (**10.40%**) exceed this threshold.

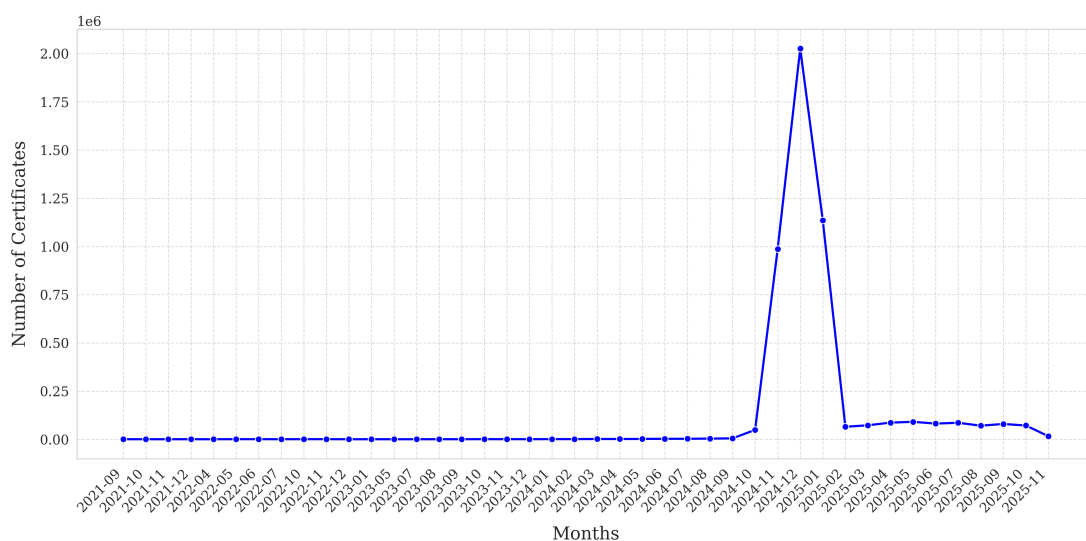


Figure 5.7. Deadlines for the distribution of leaf certificates.

A further significant trend, illustrated in Figure 5.7, indicates that a large proportion of expiring certificates are concentrated in the period between October 2025 and February 2025, in line with what has been observed in the distribution of validity terms. This uniformity could result from renewal policies set up to facilitate automation and centralised management of certificates. However, this concentration of expiry dates may result in significant simultaneous demand for renewal and revocation operations, which requires a proactive and robust management system to prevent any delays or inefficiencies in update procedures.

### 5.1.3 Leaf Certificate Serial Number Analysis

As defined in the RFC 5280 standard [6], the serial number represents a unique positive integer for each certificate issued by the same Certification Authority (CA). The standard does not impose a fixed minimum length, making even a single byte value valid, but it must be greater than 0. At the same time, to ensure compatibility and uniformity, CA/Browser Forum best practices recommend a maximum length of 20 bytes (160 bits) [4].

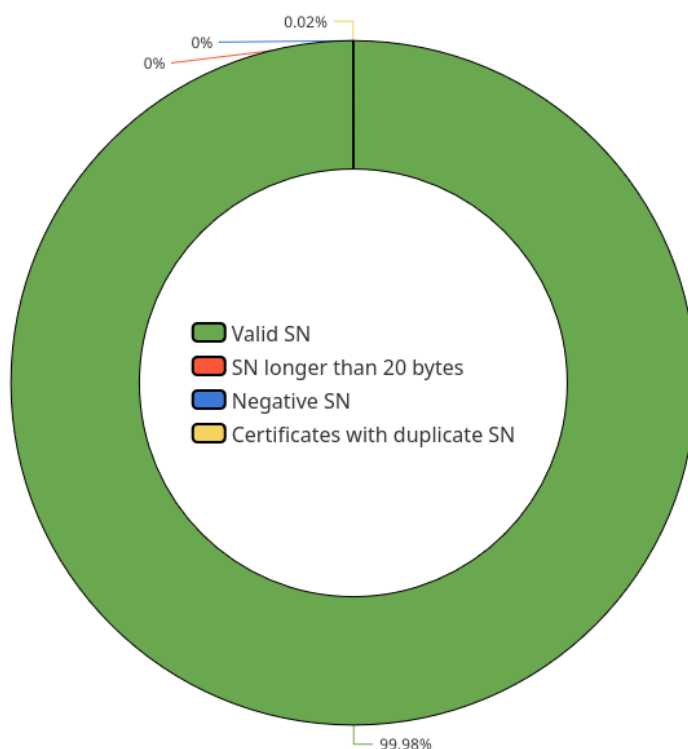


Figure 5.8. Distribution of serial numbers in leaf certificates, showing compliance with length and uniqueness requirements.

From the analysis of the dataset, it can be seen that all the certificates have a serial number, but only **99.98%** (4,999,241 certificates) conform to the above specifications. However, cases were identified where the serial number does not meet these standards: 34 certificates have a negative serial number, and 989 certificates have duplicate serial numbers where 23.66% have a value of 0. Interestingly, no certificate analysed has a serial number greater than 20 bytes. These results highlight the importance of strictly following the technical specifications to guarantee unambiguousness.

#### 5.1.4 Leaf Certificate Chain Length Analysis

The distribution of the length of the certificate chain in the dataset provides interesting insights both in terms of common configurations and compliance with the best practices defined by the CA/Browser Forum. In particular, the analysis shows that 69.67% of leaf certificates (3,483,611 certificates) are accompanied by a chain composed of a leaf certificate, an intermediate certificate and a root certificate that completes the chain. This configuration, which is the most common solution, is generally preferred as it offers an optimal balance between security and simplicity in managing the chain of trust.

Approximately 26.96% of the certificates (1,348,129 certificates) have a chain with two intermediate certificates, while 2.03% (101,260 certificates) have a chain

consisting of three intermediates. A further 1.34% of the certificates (67,264 certificates) are issued directly from a root certificate, without intermediates, a configuration that is less common but nevertheless adopted in some particular cases.

The CA/Browser Forum guidelines recommend keeping the certificate chain as simple as possible to facilitate validation and reduce the risks associated with complex configurations. The Baseline Requirements suggest the use of a limited number of intermediate certificates, since excessively long chains may introduce additional points of failure, increase latency and complicate the management of revocation [4].

## 5.2 Analysis of X.509 Leaf Certificate Extensions

### 5.2.1 Analysis of Basic Constraints Extension

The Basic Constraints Extension is a fundamental field in X.509 certificates, as it defines whether a certificate can be used to sign other certificates, that is, whether it is enabled as a Certificate Authority (CA), or whether it is an end-entity certificate. In this study, the analysis of the leaf certificates showed that 99.46% of them correctly had the `is_ca` flag set to false, thus confirming that these certificates are not intended to act as issuers of other certificates. However, 0.085% of the leaf certificates had the `is_ca` value set to true, incorrectly identifying themselves as certificate authorities, while 0.45% of the certificates had no value in this extension. These results indicate that, although the majority of certificates follow best practices, there are configuration anomalies that could compromise the robustness of the chain of trust.

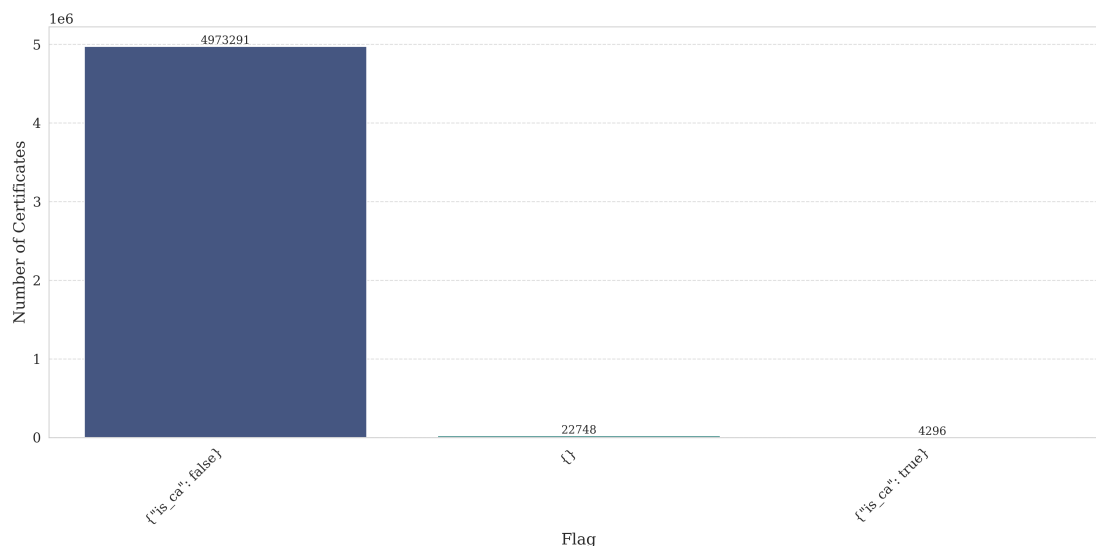


Figure 5.9. Distribution of Basic Constraint Extension.



The CA/Browser Forum guidelines specify the **Basic Constraints** to be inserted in the certificates, and therefore ensure that the configuration is well done. This, however, is optional in the case of end-entity (leaf) certificates. The `is_ca` flag has to be set to **FALSE** if the **Basic Constraints** are provided. That requirement was put in place so that no confusion about the certificate hierarchy would be caused. If there is a certificate intended solely for the purposes of authentication of end entities, it should not be able to sign other certificates, since that functionality can easily be exploited in an attack aimed at undermining the integrity of the trust chain. From that point of view, a few leaf certificates that have the `is_ca` flag set to true break from guidelines and could lead to issues related to certificate chain validity.

Comparison with previous [26] studies, such as the one conducted in 2021, shows a slight improvement: in the previous sample, 99.34% of the end-entity certificates had the `is_ca` flag set to false, only 0.25% were set to true, and 0.41% were without the Basic Constraints field. This trend suggests a relative increase in errors or omissions in certificate configuration, which could be due to changes in issuer procedures or technical issues in the certificate generation automation processes.

### 5.2.2 Analysis of Authority Key Identifier Extension

The Authority Key Identifier (AKI) is one of the fundamental extensions for identifying the public key of the issuer that signed the certificate. This extension facilitates the construction of the chain of trust, as it allows clients to uniquely associate the current certificate with the certificate of the issuer, especially in scenarios where the issuer may have multiple certificates with similar names.

The analysis revealed that only **80%** of the certificates have the Authority Key Identifier extension, while the remaining 19.73% do not. This result represents a notable deviation from the data reported in previous studies, as evidenced in [26], where almost 99.74% of the certificates (leaf) correctly presented this extension and only 0.23% lacked this information.

The CA/Browser Forum guidelines explicitly require the inclusion of AKI in publicly trusted certificates, to ensure robust issuer identification and reduce the risk of errors in validation [4], as also required by the RFC 5280 standard [6]. The lack of this extension in such a high percentage in the dataset suggests a potential looseness in quality controls or variations in the release policies adopted by some issuers.

### 5.2.3 Analysis of Subject Key Identifier Extension

The Subject Key Identifier (SKI) extension is essential for the unique identification of the public key associated with a certificate, thus facilitating the construction of certificate chains and the verification of digital signatures. The data show that 99.81% of leaf certificates include this extension, while only 0.19% lack it, a slightly

better result than the previous study, in which 99.74% of leaf certificates carried it.

The CA/Browser Forum guidelines do not recommend the inclusion of the Subject Key Identifier in all leaf certificates, as this extension does not provide a substantial benefit in terms of security or the efficiency of the validation process, since they do not issue any other certificates [4].

## 5.2.4 Analysis of Key Usage Extension

The Key Usage extension defines, through a series of flags, the purposes for which the public key associated to the certificate may be used. In particular, the presence of certain flags establishes whether the key can be used to digitally sign data, to encrypt keys in the handshake phase or to conduct key exchange operations. The CA/Browser Forum guidelines require that the content of this extension be configured differentially according to the type of public key present in the SubjectPublicKeyInfo field, distinguishing between RSA and ECDSA certificates.

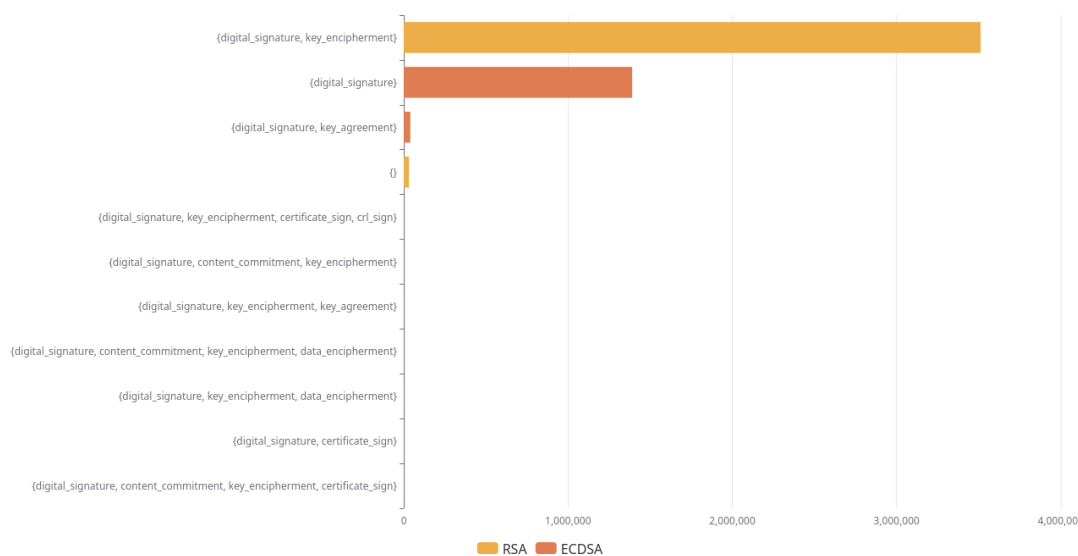


Figure 5.10. Key Usage in Extensions Field.

According to best practice for RSA certificates, the `digitalSignature` flag (mandatory according to the directive) must be selected with the optional inclusion of the `keyEncipherment` flag and the `dataEncipherment` flag, but the latter should not be used. Different combinations of encryption flags do not guarantee that a single key will only function within its designated scope. On the other hand, for ECDSA-based certificates, the directives require the inclusion of the `digitalSignature` flag, while other flags, such as `keyAgreement`, are optional but not commonly recommended.

The results of the analysis, illustrated in Figure 5.10, show that certificates are divided into two main groups: 71.24% of certificates use an RSA key, while the remaining 28.76% are ECDSA-based. For RSA certificates, 99.09% (or 3,512,056 certificates) contain only the `digitalSignature` and `keyEncipherment` flags, in full compliance with requirements. Only 0.88% of the certificates do not have any flag at all, while an even smaller percentage (135 certificates) have a configuration that also includes the `keyCertSign` and `cRLSign` flags, a configuration that does not comply with the required specifications. A total of 586 certificates (0.02%) do not comply with the guidelines.

With regard to ECDSA certificates, 97.21% (or 1,390,871 certificates) include only the `digitalSignature` flag, while 2.78% (39,765 certificates) have the combination of `digitalSignature` and `keyAgreement`, a configuration that is considered acceptable. On the other hand, a very small number of ECDSA certificates (141, or about 0.01%) violate the requirements.

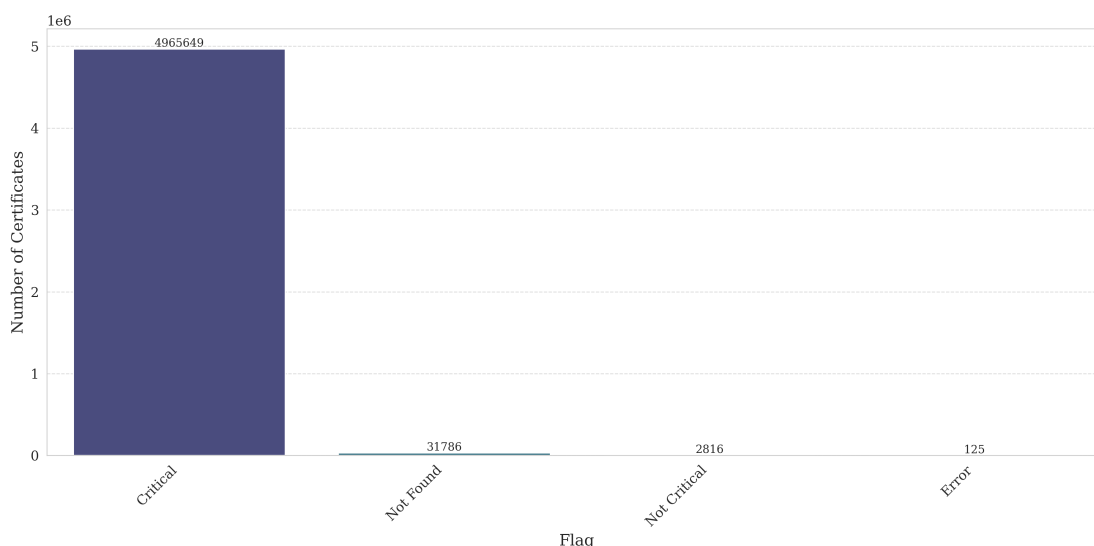


Figure 5.11. Critical vs Not Critical Key Usage in Extensions Field.

A new angle revealed from the study relates to the flag for criticality related to the Key Usage extension. In the dataset, a very huge 99.31% of the leaf certificates classify it as Critical, meaning that the clients have to interpret and adhere to the values indicated. Only 0.056% will label it as Not Critical, while a tiny 0.63% do not clearly mark Critical or Not Critical. This falls in line with the recommendations of the CA/Browser Forum, which suggests classifying the extension as Critical in order to avert ambiguity and restrict its usage to its stated function [4].

### 5.2.5 Analysis of Extended Key Usage Extension

The Extended Key Usage (EKU) extension specifically defines the purposes for which the public key associated with a certificate may be used, going beyond

the basic functionalities guaranteed by the Key Usage field. Hence, it specifies particular applications such as server authentication (`server_auth`) and client authentication (`client_auth`), and possibly other purposes such as e-mail protection or code signing, depending on the needs of the application environment.

The analysis shows that only 0.34% of the leaf certificates have no value for the EKU field, while the remaining 99.65% include one or more specifications in this field. In particular, 78.92% of the leaf certificates report the combination `server_auth` and `client_auth`, signalling the intention to be used for both server and client authentication in TLS/SSL communications. This configuration allows for greater flexibility and interoperability in environments where both inbound and outbound connections need to be supported, for example. A further 20.81% of the certificates carry only the value `server_auth`, which suggests that these certificates are primarily intended for server authentication and do not include functionality for client authentication. Other configurations, present in a negligible number of certificates (less than 200 cases), indicate less common usage choices.

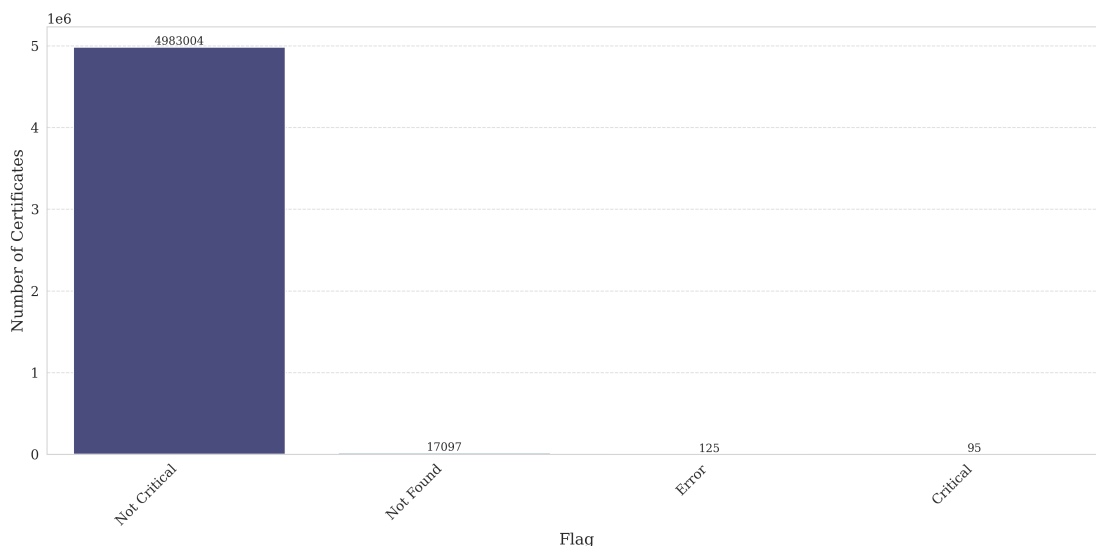


Figure 5.12. Critical vs Not Critical Extended Key Usage in Extensions Field.

With regard to the criticality of the extension, the majority of leaf certificates (99.65%) state EKU as Not Critical, while only an extremely small percentage (0.001%) set it as Critical. A small fraction (0.34%) do not specify Critical. This setting is consistent with the indications of the CA/Browser Forum, which recommends not making the EKU a critical extension in TLS certificates, in order to preserve compatibility and interoperability with clients, avoiding the possibility that failure to interpret the extension may block certificate validation [4].

The values relative to the criticality flag show consistency with what was observed in the study of [26], as regards both the Key Usage extension and the Extended Key Usage extension.

## 5.2.6 Analysis of Certificate Policies Extension

The Certificate Policies extension, as defined by RFC 5280 [6], is used to communicate to PKI users the policies according to which the certificate was issued. It allows, for example, a link (typically in http or https format) to be associated with a Certification Practice Statement (CPS) or other compliance statement, thus providing additional information on the level of assurance provided by the certificate.

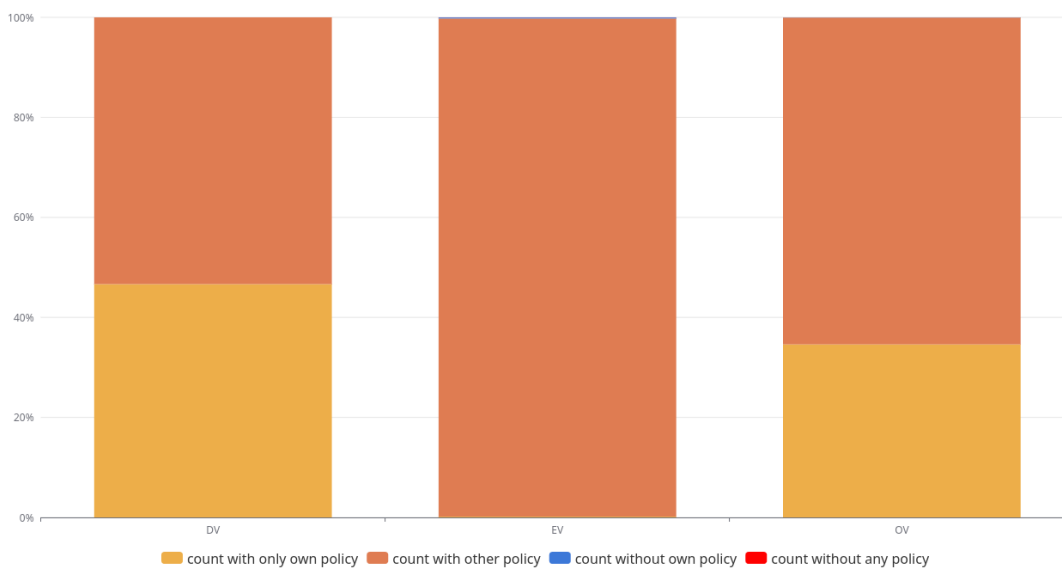


Figure 5.13. Results related to the Certificate Policies (CP) extension in leaf certificates, showing the distribution of policy identifiers for different validation levels (DV, OV, EV) as required by the CA/Browser Forum Baseline Requirements.

Figure 5.13 shows the results for the Certificate Policies (CP) extension in leaf certificates, which must include the appropriate **Reserved Certificate Policy Identifier** for their validation level, or 2.23.140.1.2.1 for DV certificates, 2.23.140.1.2 for OV certificates and 2.23.140.1.1 for EV certificates, as required by the Baseline Requirements of the CA/Browser Forum [4].

For the sake of clarity, `count_with_only_own_policy` refers to certificates that carry only the policy identifier reserved for their type (DV, OV or EV), while `count_with_other_policy` refers to those that, in addition to this policy, include other policies. `count_without_other_policy` identifies those certificates that, although having the extension, do not carry the specific policy identifier, while `count_without_any_policy` refers to those that do not present any policy information.

In the analysed dataset of 5,000,264 unique leaf certificates, 99.27% (4,963,598 certificates) include the CP extension, a slightly higher percentage than in the previous study ([26]).

For the certificates **DV**, the data show that:

- 87.45% (4,169,667 certificates) of the DV certificates have their own policy only.
- 99.99% (4,767,624 certificates) include their policy together with other policies.
- Only 0.004% (198 certificates) were found to be without their own policy.
- And 0.002% (98 certificates) have no policy.

For **EV** certificates, on the other hand, the results indicate that:

- A very limited number, 0.16% (23 certificates) of the EV certificates, contain only their own policy.
- 99.77% (14,203 certificates) include their policy together with others.
- In addition, 30 EV certificates (0.21%) do not have their own policy, while 3 certificates (0.02%) have no policy.

Finally, for certificates **OV**, it is noted that:

- 52.90% (95,747 certificates) present their policy exclusively.
- 99.95% (180,921 certificates) include their own policy as well as others.
- Only 89 OV certificates (0.05%) do not report their policy, while in 2 certificates (0.001%) the CP extension is completely absent.

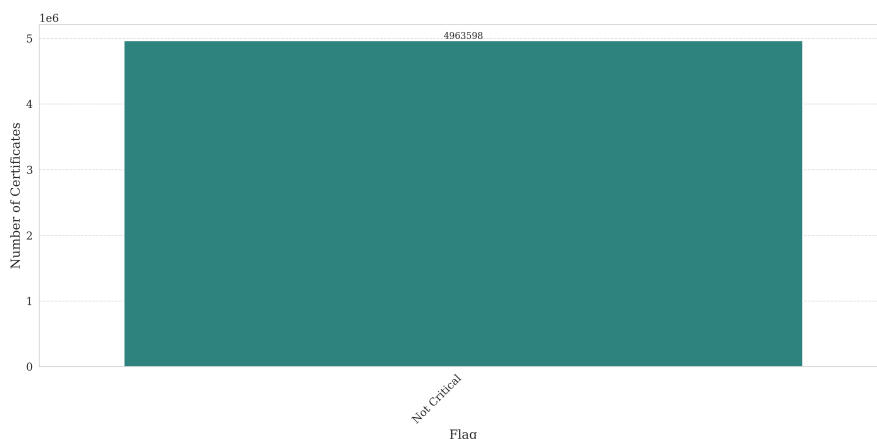


Figure 5.14. Critical vs Not Critical Certificate Policies Extension.

The fact that 100% of the certificates in which the extension is present are configured as Not Critical is in accordance with the CA/Browser Forum guidelines. These guidelines, in fact, recommend that the Certificate Policies extension should not be set as critical, so as not to hinder the validation process by clients, should it not be interpreted or be relevant to the trust assessment [4].

These results indicate that, in most cases, issuers include the Certificate Policies extension to comply with CA/Browser Forum requirements, although they often

do not specify further details. This approach makes it possible to maintain a high standard of security and transparency without introducing unnecessary complexity in leaf certificates. Furthermore, the slight improvement over the previous study [26] may reflect a gradual adaptation to best practices or simply as a consequence of the greater availability of data, considering that the sample analysed in this study includes a significantly larger number of certificates than the approximately 400,000 previously examined.

### **5.2.7 Analysis of Subject Alternative Name (SAN) Extension**

The Subject Alternative Name (SAN) extension is crucial to ensure that the certificate is issued for the correct domains and to facilitate the chain of trust validation process. According to the guidelines of the CA/Browser Forum, publicly trusted certificates must include in the SAN field all domain names for which the certificate is valid, thus ensuring a precise match between the requested domain and the data contained in the certificate [4]. The RFC 5280 standard, while defining the SAN field as optional, strongly recommends its use, as this improves the robustness of subject identification and verification of digital signatures [6].

The analysis carried out on the dataset shows that about 2,375 certificates do not have any values listed in the SAN field, with another group of about 130 certificates sharing generic values like `localhost.localdomain` or `example.com` which do not actually refer to the domain for which the certificate was issued. Overall, out of 4,910,515 leaf certificates (98.20%) show a proper match between the domain and SAN field, with 115,081 certificates (1.80%) not having any such match. These anomalies are reflective of a condition whereby most certificates do, indeed, conform to recommended best practices, however, certain glaring misses still exist which would altogether undermine the efficacy of the chain of trust.

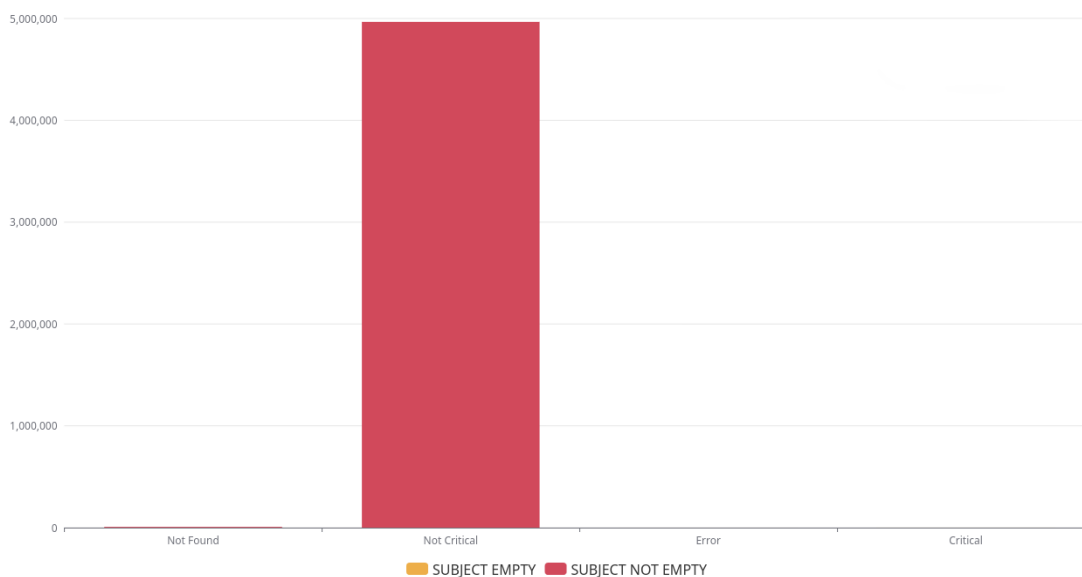


Figure 5.15. Critical vs Not Critical Extensions of Subject Alternative Names.

According to the CA/Browser Forum Baseline Requirements for certificate subscribers (leaf), the Subject Alternative Name (SAN) extension must be configured to clearly reflect the identity of the entity. In particular, if the Subject field is populated, the SAN extension must be marked as Not Critical, since the information contained in the Subject already provides a valid identification of the entity. Conversely, if the Subject field is empty, the SAN extension takes on an even greater role in the identification and must therefore be set as Critical to ensure that the TLS client does not ignore this essential field during the validation process.

From the analysis of the certificates, the data show that, among the certificates in which the Subject field is populated, 99.84% present the SAN extension as Not Critical, in full compliance with the CA/Browser Forum guidelines. Only in 6 cases was the marking set as Critical, while in a further small percentage (0.15%) the criticality information was not found. In contrast, for certificates in which the Subject field is empty, the correct configuration requires the SAN extension to be marked as Critical. In this group, 233 out of 246 certificates (approx. 94.72%) were compliant, while in 13 certificates the extension was not found.

The results validate, in general, the thought that the issuers of subscriber certificates dutifully observe the Best Practices advanced by the CA/Browser Forum with appropriate use of the SAN extension to assure subject identity, marginally improvement over the study conducted last year in which 99.79% semantically put a leaf about certificates contains the SAN field correctly. The presence of some cases in which the configuration did not meet expectations indicates further potential examinations and refinements required by the issuing processes.



## 5.3 Analysis of Algorithms and Cryptographic Security of Leaf Certificates

This section analyses in detail the distribution of signature algorithms and related key lengths adopted in leaf digital certificates, with the aim of assessing compliance with the security standards recommended by the CA/Browser Forum and highlighting the implications in terms of cryptographic performance and robustness.

### 5.3.1 Distribution of Signature and Key Algorithms in Leaf Certificates

The distribution of key algorithms studied in this dataset is very much in favor of the RSA algorithm, which was used by 73.23% of the leaf certificates and is actually even more common. ECDSA is only responsible for the other 27.77%. The intended purpose of both algorithms is the same: to protect cryptographic communications, but indeed they differ in speed and how much computational processing power they may require. Being a public-key algorithm, RSA has been in use for decades, so it must invariably rely on longer keys if good levels of security are to be provided. In other words, while the RSA algorithm usually depends on a longer-length key to provide acceptable assurance of security, ECDSA is based on elliptic curve techniques that afford a very high degree of security with much shorter keys, making them more efficient and faster to process.

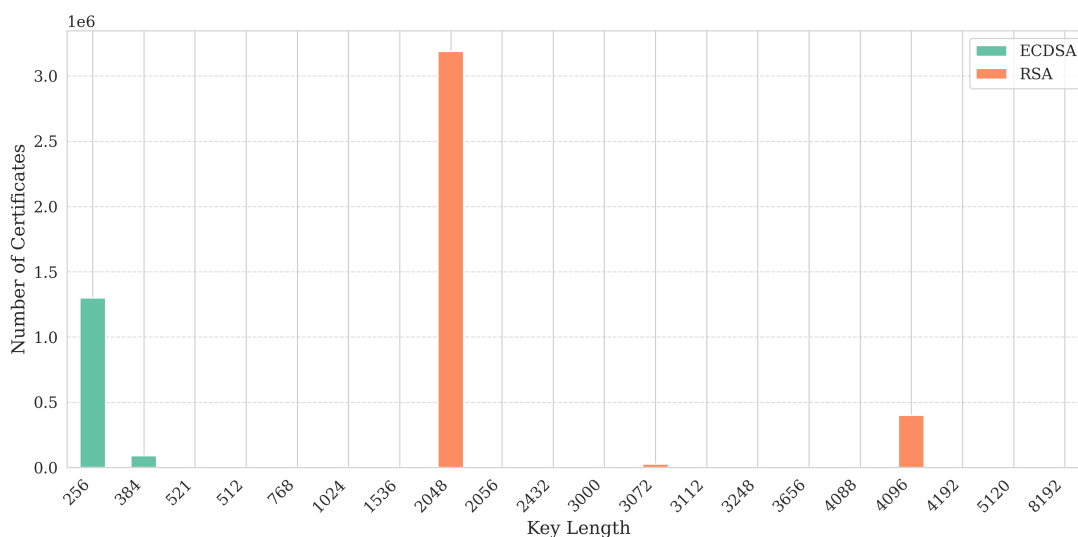


Figure 5.16. Distribution of Key and Length Algorithms.

Figure 5.16 shows that, as far as ECDSA is concerned, 93.56% of the certificates adopting this algorithm employ a 256-bit key, typically corresponding to the P-256 curve, the standard recommended by the CA/Browser Forum for TLS certificates.

Only a small percentage (6.43%) uses 384-bit keys, while the use of 521-bit keys is practically negligible (0.01%).

In the case of RSA, the most common configurations are 2048-bit keys, used by 88.15% of RSA certificates, followed by 4096-bit keys (11.07%) and, to a lesser extent, 3072-bit keys (0.72%). These values are in line with CA/Browser Forum guidelines, which require a minimum of 2048 bits for RSA keys and recommend the adoption of standard curves for ECDSA (typically P-256), in order to balance security and performance [4].

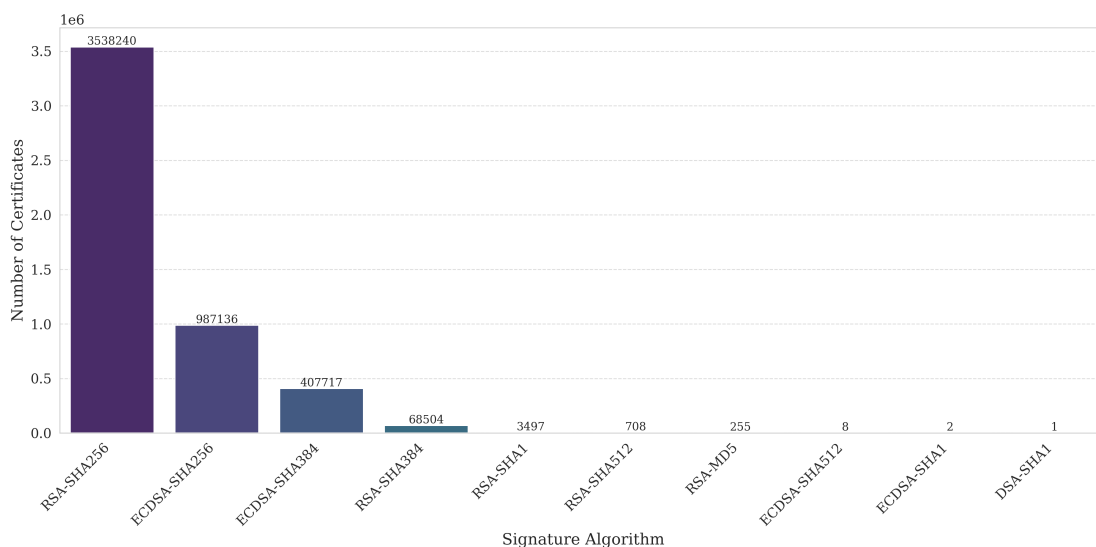


Figure 5.17. Signature Algorithm Used.

Figure 5.17 summarises the distribution of signature algorithms. The data shows that even for the signature, 73.23% of the certificates use RSA, while the remaining 27.77% use ECDSA, only one certificate uses DSA, specifically DSA-SHA1. These results show how, in a context of public certificate issuance, the choice of algorithms is oriented towards ensuring both cryptographic robustness and operational efficiency.

For RSA certificates, that is the 97.97% comprising 70.76% of all leaf certificates using RSA-SHA256. This is the minimum recommended by the CA/Browser Forum guidelines, which call for the use of SHA256-or-better algorithms in order to mitigate the vulnerabilities associated with weaker hash functions [4]. Meanwhile, 1.89% of RSA certificates, or 1.37% of total, use RSA-SHA384, adding extra assurance, where performance can allow. In contrast, only 0.09% of RSA certificates, or 0.07% of total leaf certificates, just use RSA-SHA1 highlighting an almost complete transition to more secure configurations.

Finally, isolated cases of less secure configurations were found: 708 certificates employ RSA-SHA512 and 255 certificates use RSA-MD5. Although representing a tiny fraction of the total, these configurations are in stark contrast to current

best practices, as the use of algorithms based on SHA256 or higher is strongly recommended by the CA/Browser Forum guidelines [4].

As for ECDSA, the data indicate that 70.76% of certificates using it (or 19.74% of total leaf certificates) employ ECDSA-SHA256, typically corresponding to the P-256 curve, the preferred standard for TLS certificates. While 29.22% (or 8.15% of total certificates) use ECDSA-SHA384, a configuration that, while entailing a slight overhead, allows for enhanced security in particularly demanding contexts. Only 8 leaf certificates use ECDSA-SHA512 and 2 ECDSA-SHA1 certificates.

Looking at the results, it clearly emerges that most issuers adopt configurations that conform to the best practices established by the CA/Browser Forum. In particular, the analysis also shows how the predominance of RSA reflects the tradition and broad compatibility of this algorithm, while the significant share of certificates using ECDSA testifies to a progressive transition towards more efficient solutions, particularly in environments where performance and resource management are critical factors. The widespread choice of RSA-SHA256 and ECDSA-SHA256 demonstrates that issuers are oriented towards balancing security and operational efficiency, while minimising the use of obsolete algorithms such as RSA-SHA1 and RSA-MD5.

### **5.3.2 Comparison of Self-Signed and CA-Signed Certificates**

Another fundamental aspect for the security of PKI infrastructures is precisely the distinction between self-signed certificates and certificates signed by a certification authority, where through the analysis conducted on the dataset, it was observed that 99.3% of leaf certificates were signed by a recognised issuer, while only 0.7% were self-signed. This distribution clearly indicates that, in production environments, most issuers comply with the CA/Browser Forum guidelines, which recommend the exclusive use of certificates signed by trusted CAs.

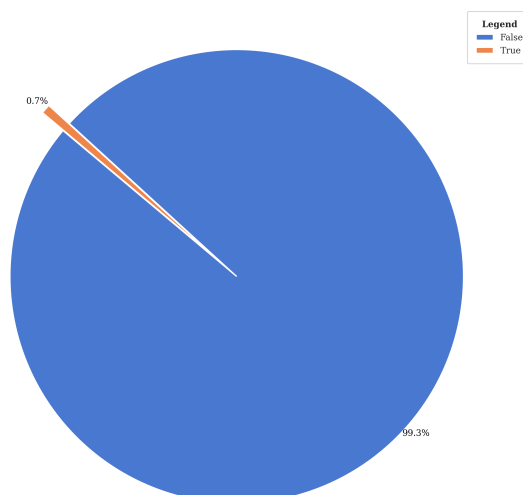


Figure 5.18. Representation of the percentage of self-signed certificates compared to certificates signed by a certification authority (CA).

The CA/Browser Forum guidelines emphasise the importance of a verifiable chain of trust up to the trust anchor, a fundamental requirement to guarantee that the certificate cannot be easily manipulated [4]. Likewise, the RFC 5280 standard requires that each certificate be issued and signed by a certificate authority, thus guaranteeing the authenticity of the public key and the correct validation of the entire chain [6].

Self-signed certificates, which do not allow chain verification through globally recognised trust stores, are generally reserved for test environments or internal networks, where security requirements may be handled differently. Their limited presence (0.7%) in the dataset confirms that, for most critical applications, issuers have adopted configurations that conform to international best practices, thus contributing to a high standard of security.

### 5.3.3 Analysis of Certificate Signature Validity

The analysis of the dataset showed that 99.29% of the leaf certificates had a valid signature, which was obtained by verifying the entire chain. This verification was also conducted using DigitalCertiAnalytics software, with the support of SSlyze.

The CA/Browser Forum guidelines emphasise that each certificate must have a valid signature, obtained through the application of the declared signature algorithms, and that the certificate chain must be verifiable up to the trust anchor [4]. The RFC 5280 standard, in fact, states that the `signatureAlgorithm` field must be checked and that each certificate in the chain must be correctly signed by the respective issuer, thus guaranteeing the consistency and integrity of the entire [6] structure.

```

amirmagliari@MacBook-Air-di-Amir ~ % ssllyze www.polito.it
CHECKING CONNECTIVITY TO SERVER(S)

www.polito.it:443 => 138.192.182.180

SCAN RESULTS FOR WWW.POLITO.IT:443 - 138.192.182.180

* Certificates Information:
  Hostname sent for SNI:      www.polito.it
  Number of certificates detected: 1

Certificate #0 ( _RSAPublicKey )
  SHA1 Fingerprint:          980e512d075508d6e9b99e12fa1b558dbb725f4
  Common Name:              www.polito.it
  Issuer:                    GEANT OV RSA CA 4
  Serial Number:             281c22c80115659871238c2275959919a02572
  Not Before:                2024-11-22
  Not After:                 2028-11-22
  Public Key Algorithm:      _RSAPublicKey
  Signature Algorithm:       sha384
  Key Size:                  3072
  Exponent:                  65537
  SubjAltName - DNS Names:  ['www.polito.it', 'legacy-auth.polito.it', 'legacyprod.polito.it', 'polito.it', 'wwwlegacy.polito.it']

Certificate #0 - Trust
  Hostname Validation:      OK - Certificate matches server hostname
  Android CA Store (83.0.0-9): OK - Certificate is trusted
  Apple CA Store (iOS 16, iPadOS 16, macOS 13, tvOS 16, and watchOS 9):OK - Certificate is trusted
  Java CA Store (jdk-13.0.2): OK - Certificate is trusted
  Mozilla CA Store (2022-12-11): OK - Certificate is trusted
  Windows CA Store (9293-02-19): OK - Certificate is trusted
  Symantec 2018 Deprecation: OK - Not a Symantec-issued certificate
  Received Chain:          www.polito.it -> GEANT OV RSA CA 4 -> USERTrust RSA Certification Authority
  Verified Chain:          www.polito.it -> GEANT OV RSA CA 4 -> USERTrust RSA Certification Authority
  Received Chain Contains Anchor: OK - Anchor certificate not sent
  Received Chain Order:    OK - Order is valid
  Verified Chain contains SHA1: OK - No SHA1-signed certificate in the verified certificate chain

Certificate #0 - Extensions
  OCSP Must-Staple:        NOT SUPPORTED - Extension not found
  Certificate Transparency: OK - 3 SCTs included

Certificate #0 - OCSP Stapling
  NOT SUPPORTED - Server did not send back an OCSP response

* SSL 2.0 Cipher Suites:
  Attempted to connect using 7 cipher suites; the server rejected all cipher suites.

* SSL 3.0 Cipher Suites:
  Attempted to connect using 80 cipher suites; the server rejected all cipher suites.

* TLS 1.0 Cipher Suites:
  Attempted to connect using 80 cipher suites; the server rejected all cipher suites.

* TLS 1.1 Cipher Suites:
  Attempted to connect using 80 cipher suites; the server rejected all cipher suites.

* TLS 1.2 Cipher Suites:
  Attempted to connect using 156 cipher suites.

The server accepted the following 13 cipher suites:
  TLS_RSA_WITH_AES_256_GCM_SHA256      256
  TLS_RSA_WITH_AES_256_CBC_SHA256     256
  TLS_RSA_WITH_AES_256_CBC_SHA        256
  TLS_RSA_WITH_AES_128_GCM_SHA256     128
  TLS_RSA_WITH_AES_128_CBC_SHA256     128

```

Figure 5.19. SSLyze result for www.polito.it - Part 1

```

  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256      128  ECDHE: prime256v1 (256 bits)
  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256     128  ECDHE: prime256v1 (256 bits)
  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA        128  ECDHE: prime256v1 (256 bits)

The group of cipher suites supported by the server has the following properties:
  Forward Secrecy:          OK - Supported
  Legacy RC4 Algorithm:     OK - Not Supported

* TLS 1.3 Cipher Suites:
  Attempted to connect using 5 cipher suites.

The server accepted the following 3 cipher suites:
  TLS_CHACHA20_POLY1305_SHA256             256  ECDHE: X25519 (253 bits)
  TLS_AES_256_GCM_SHA384                   256  ECDHE: X25519 (253 bits)
  TLS_AES_128_GCM_SHA256                   128  ECDHE: X25519 (253 bits)

* Deflate Compression:      OK - Compression disabled

* OpenSSL CCS Injection:    OK - Not vulnerable to OpenSSL CCS injection

* OpenSSL Heartbleed:       OK - Not vulnerable to Heartbleed

* ROBOT Attack:             OK - Not vulnerable.

* Session Renegotiations:
  Client Renegotiation DoS Attack: OK - Not vulnerable
  Secure Renegotiation:          OK - Supported

* Elliptic Curve Key Exchange:
  Supported curves:            X25519, X448, prime256v1, secp384r1, secp224r1,
  prime192v1, secp108k1, secp100r1, secp160r2, secp192k1, secp224k1, secp224r1, secp256k1, sect1
  63k1, sect163r1, sect163r2, sect193r1, sect193r2, sect238k1, sect233r1, sect239k1, sect283k1, sect283r1, sect409k1, sect409r1, sect571k1
  , sect571r1

SCANS COMPLETED IN 33.07783 S

COMPLIANCE AGAINST MOZILLA TLS CONFIGURATION

Checking results against Mozilla's "MozillaTlsConfigurationEnum.INTERMEDIATE" configuration. See https://ssl-config.mozilla.org/ for more details.

www.polito.it:443: FAILED - Not compliant.
* certificate_signatures: Deployed certificate signatures are ('sha384WithRSAEncryption'), should have at least one of ('ecdsa-w
ith-sha512', 'ecdsa-with-sha256', 'sha256WithRSAEncryption', 'ecdsa-with-sha384').
* cipher_suites ('TLS_RSA_WITH_AES_128_CBC_SHA', 'TLS_RSA_WITH_AES_128_CBC_SHA256', 'TLS_RSA_WITH_AES_256_CBC_SHA', 'TL
S_ECDHE_RSA_WITH_AES_256_GCM_SHA384', 'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384', 'TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA256', 'TLS_ECDHE_RSA_WITH_AES_1
28_CBC_SHA256', 'TLS_RSA_WITH_AES_128_GCM_SHA256', 'TLS_RSA_WITH_AES_256_CBC_SHA256', 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA') are supporte
d, but should be rejected.

```

Figure 5.20. SSLyze result for www.polito.it - Part 2

A significant example, illustrated in Figures 5.19 and 5.20, concerns the domain www.polito.it analysed with SSLyze from a terminal. In this case, SSLyze

verified the certificate chain and confirmed its validity, with the chain recognised as trustworthy by several trusted stores (Android, Apple, Java, Mozilla and Windows CA Store). Furthermore, it pointed out that the chain does not contain certificates signed with deprecated algorithms such as SHA1, and that the server, while supporting TLS 1.2 and TLS 1.3, rejects obsolete versions of SSL/TLS, also guaranteeing Forward Secrecy. It is important to point out that these observations are specific to the example of `www.polito.it` and do not necessarily represent the entire dataset.

Overall, the data indicate that only 0.71% of certificates have an invalid signature. Of this percentage, a very small part, 0.01%, is attributable to errors in the generation or configuration of the signature by the issuer. The remaining 0.7% derives instead from the presence of self-signed certificates, which are not subject to the external validation process provided for certificates issued by a recognised certification authority.

These results show that most issuers follow the best practices and guidelines of the CA/Browser Forum, guaranteeing a high standard of security in the PKI infrastructure. At the same time, the analysis of specific cases such as that of `www.polito.it` provides further confirmation of compliance with the guidelines on signing algorithms, proper chain verification, and the adoption of configurations that reject the use of deprecated algorithms.

## **5.4 Revocation Status Checking in Leaf Certificates**

### **5.4.1 Analysis of Authority Information Access (AIA)**

The Authority Information Access (AIA) extension provides crucial information to retrieve data related to the certificate issuer, such as the Certificate Revocation List (CRL) distribution point or the URL for OCSP requests. These fields are defined by the RFC 5280 standard [6] and are strongly recommended by the CA/Browser Forum guidelines [4] to ensure that certificates can be effectively verified.

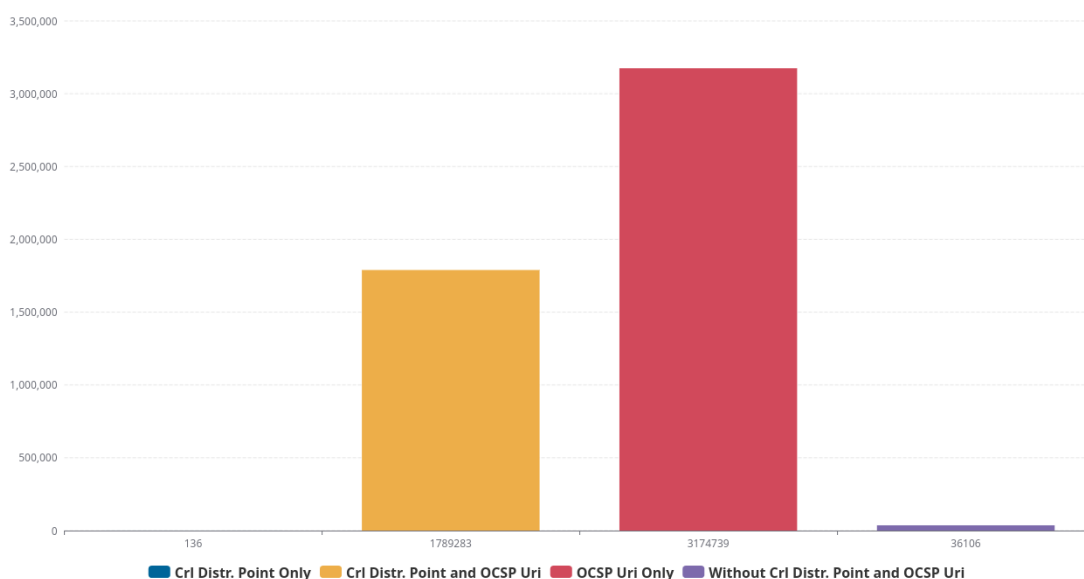


Figure 5.21. Distribution of AIA URLs in the leaf certificates.

An analysis of the dataset, as in Figure 5.21, shows that:

- 136 certificates (0.003%) contain only the CRL Distribution Point.
- 1,789,283 certificates (35.78%) contain both the CRL Distribution Point and the OCSP URI.
- 3,174,739 certificates (63.49%) contain only the OCSP URI.
- 36,106 certificates (0.72%) contain neither the CRL Distribution Point nor the OCSP URI.

This distribution indicates that most certificates include at least one mechanism for verifying revocation status, with a marked preference for the OCSP URI, which allows revocation status to be obtained quickly and reliably via online queries. The RFC 5280 standard [6] prescribes that if the AIA extension is present, it must contain a *accessDescription* field with a *accessMethod* (typically indicating OCSP or CRL) and a *accessLocation* (typically a URL), which must respect the defined format.

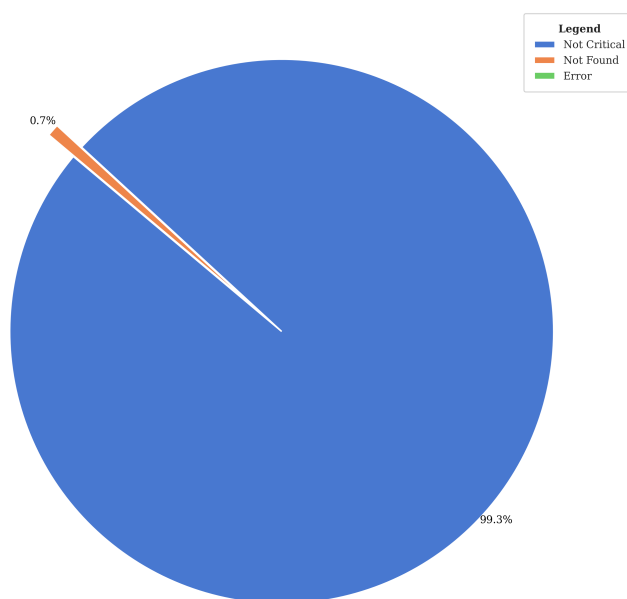


Figure 5.22. Critical vs Not Critical Extensions of Authority Information Access.

The CA/Browser Forum guidelines [4], on the other hand, require the inclusion of the AIA extension in publicly trusted certificates to ensure that clients can verify the revocation status and identity of the issuer efficiently. The data of the present study are consistent with those reported in previous studies [26], showing that in 99.3% of the certificates the AIA extension is configured as Not Critical, while in about 0.7% of the cases the extension is not present and in a further small fraction there were read errors.

The results thus confirm that, by and large, issuers do go for best-practice configurations, fairly having incorporated revocation access mechanisms. One of the more significant features of the OCSP URI-central data points is rapid and reliable verification of the status of certificates, and its adoption-on its own or in combination with CRL Distribution Point is fundamental to ensuring the security of the trust chain.

#### 5.4.2 Certificate Revocation List (CRL) Checking

In the course of the analysis, it was observed that 64.1% (3,205,169 certificates) did not contain any CRL information, resulting in an empty list. Instead, as shown in Figure 5.23, the remaining 35.9% have the extension **CRL Distribution Points** configured as **Not Critical**, in compliance with the guidelines of the CA/B Forum, which prescribe that if revocation information is provided, it must be made reliably accessible without affecting the normal certificate validation process [4]. It was also found that an error occurred in 125 certificates when reading the extension data, indicating potential operational criticalities.



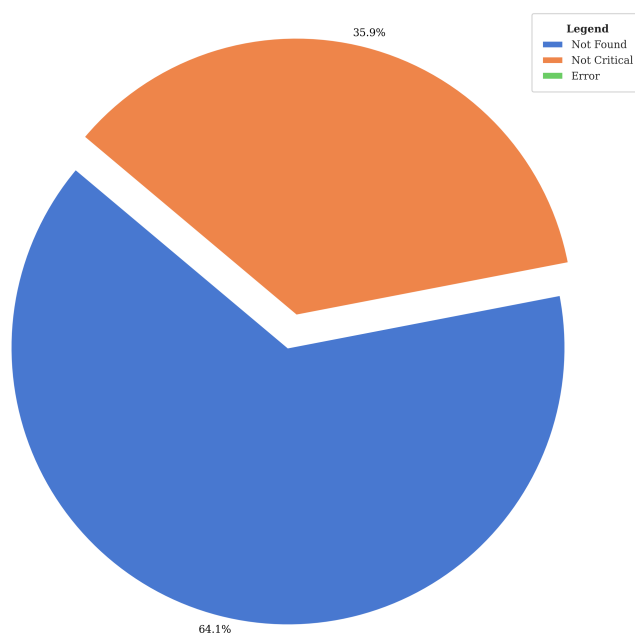


Figure 5.23. Critical vs Not Critical Extensions of Certificate Revocation Lists.

With regard to the general distribution of the status for certificates, it can be seen that almost all of them, or about 99.8% (3,198,759 certificates), are valid, which in turn represents the efficacy in the issuance and management processes. A negligible percentage of certificates, 0.1%, have been revoked, providing some indication to the effectiveness of the appropriate revocation mechanisms being activated should data anomalies or compromises arise. Furthermore, each of these, together with about 0.05% of expired CRLs and certificates with read errors, requires periodic checks and updates to ensure the integrity and timeliness of the revocation information implicit in this context.

These results show that the majority of certificates adhere to the requirements imposed by the CA/B Forum, ensuring that revocation information is handled in a transparent and non-intrusive manner, so as not to compromise the validation process. The **Not Critical** configuration of the extension, where present, is functional in avoiding interruptions in cases where the CRL is not available, while the absence of the extension in cases where it is not provided is also consistent with expectations.

### 5.4.3 Online Certificate Status Protocol (OCSP) Checking

Analysis of the dataset showed that 99.8% of the certificates returned a positive OCSP response (**Good**), amounting to 4,990,263 certificates, as shown in Figure 5.24. Only 0.1% of the certificates were reported as **Revoked**, while a further 0.1% generated errors in the OCSP request. These errors may result from the issuer URL or OCSP URL not being available, or from communication problems with the OCSP server.

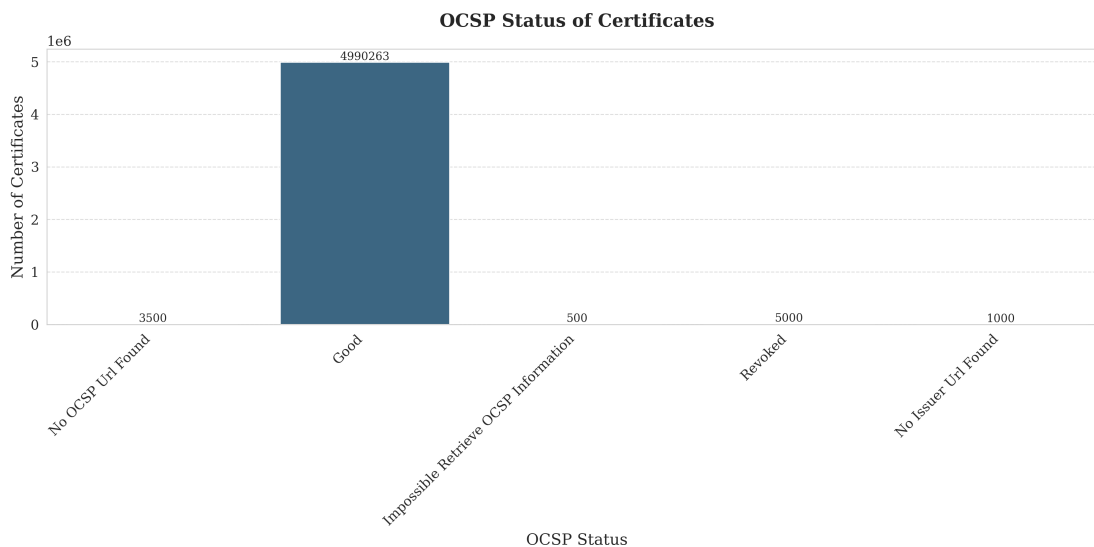


Figure 5.24. Distribution of OCSF response statuses across certificates, highlighting the proportion of valid, revoked, and unknown statuses.

These results indicate a high degree of compliance with the CA/Browser Forum guidelines [4], which require certificates to include up-to-date information on revocation status, obtainable through OCSF or, alternatively, through CRL. The presence of a high percentage of `Good` responses confirms that revocation mechanisms are generally well implemented, however, the small percentage of errors highlights the importance of verifying and properly maintaining OCSF server configurations to avoid potential security risks.

#### 5.4.4 Analysis of OCSF Stapling

As introduced in the previous Section 2.7.3, the OCSF stapling mechanism allows the server to provide, during the TLS handshake, a signed and cached OCSF response, obtained from the CA, attesting to the revocation status of the certificate. This approach avoids the client having to make a direct request to the OCSF server of the CA, thus improving both performance and privacy, and reducing delays in the validation process.

Technical standards, in particular the `status_request` extension defined in RFC 6066 [34], provide for the use of OCSF stapling to make certificate status checking more efficient. The CA/Browser Forum, in its guidelines, strongly encourages the adoption of OCSF stapling for publicly trusted certificates to ensure that clients can quickly and reliably verify revocation status without having to contact the OCSF server directly, thus reducing the risk of verification failures due to connectivity problems or delays in OCSF server response [4].

The analysis of the dataset shows that 2,396,861 certificates (47.93%) have OCSF stapling, while 2,603,403 certificates (52.07%) do not implement it. This data shows a heterogeneous situation: while almost half of the certificates benefit

from OCSP stapling, which according to best practices should be a standard feature to improve the security and efficiency of real-time validation, a slightly higher share does not use this mechanism, which could lead to more delays and problems in verifying the revocation status.

#### **5.4.5 Analysis of OCSP Must-Staple**

The `OCSP Must Stapling` functionality requires the server to provide a stapled OCSP response during the TLS handshake, without which the connection may be rejected by some clients. An analysis of the dataset showed that only 3,987 certificates (0.08% of the total) had enabled the OCSP Must Stapling feature, while 4,996,152 certificates (99.92%) did not report any values for this configuration. In addition, an error was found in 125 certificates when reading the data. These results show that, despite the recognised benefits in terms of performance, reduced client load and increased privacy, the `Must Stapling` configuration is not widely adopted in the analysed certificate landscape.

This finding suggests that many servers, while able to support OCSP stapling, do not mandate that the OCSP response be included in the TLS handshake, probably for reasons of compatibility or poor implementation of the functionality required by the CA/Browser Forum guidelines.

### **5.5 Analysis of Signed Certificate Timestamps (SCTs)**

The Signed Certificate Timestamps mechanism is a key element of the Certificate Transparency system (3), which provides public proof of the issuance of digital certificates and allows for real-time monitoring of any anomalies or unauthorised issuances. In practice, each SCT is a promise, digitally signed by a public log, attesting to the inclusion of the certificate in an append-only register. This system, defined in the RFC 6962 [22] and RFC 9162 [23] standards, is strongly recommended by the CA/Browser Forum [4] to increase the transparency and security of the PKI ecosystem.

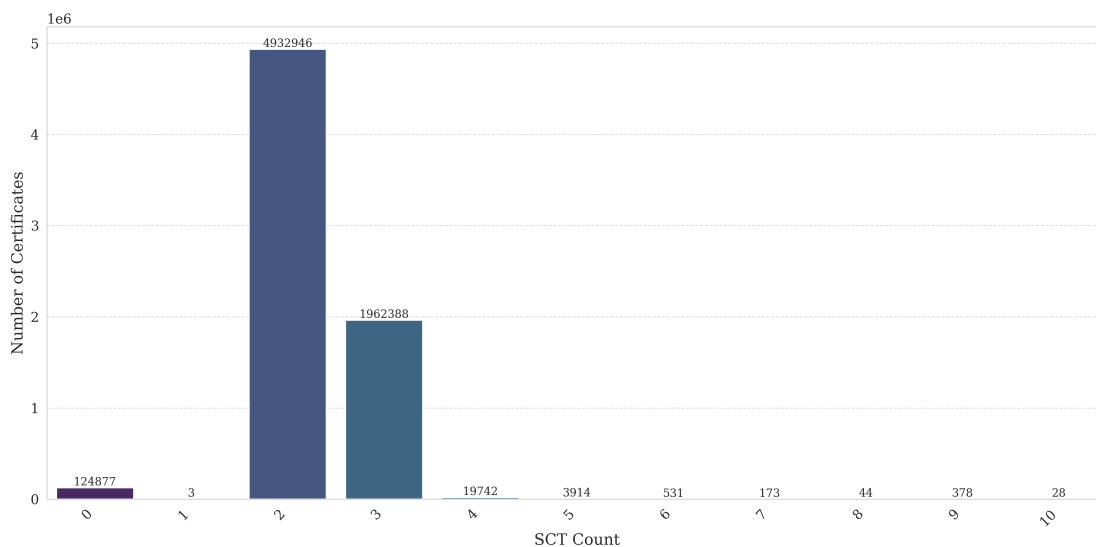


Figure 5.25. SCT Count per Certificate.

From the analysis of the dataset, Figure 5.25 shows that the majority of the certificates have two SCTs (70.02%), followed by certificates with three SCTs (27.85%) and a small percentage with four SCTs (0.3%). Only three certificates recorded only one SCT, while 1.77% of the certificates reported no SCT at all. This analysis also takes into account duplicate certificates since these may influence the overall count but do not reflect the issuance and are relevant for understanding the overall volume of certificates.

Furthermore, the adoption of version 2.0 of the Certificate Transparency (CT) protocol, defined in RFC 9162 [23], is currently absent. Most certificates continue to use Signed Certificate Timestamps (SCTs) compliant with version 1 of the RFC 6962 standard [22]. Although RFC 9162 introduced significant improvements, such as a new TLS extension for sending various CT artefacts, organisations have not yet fully transitioned to this new version.

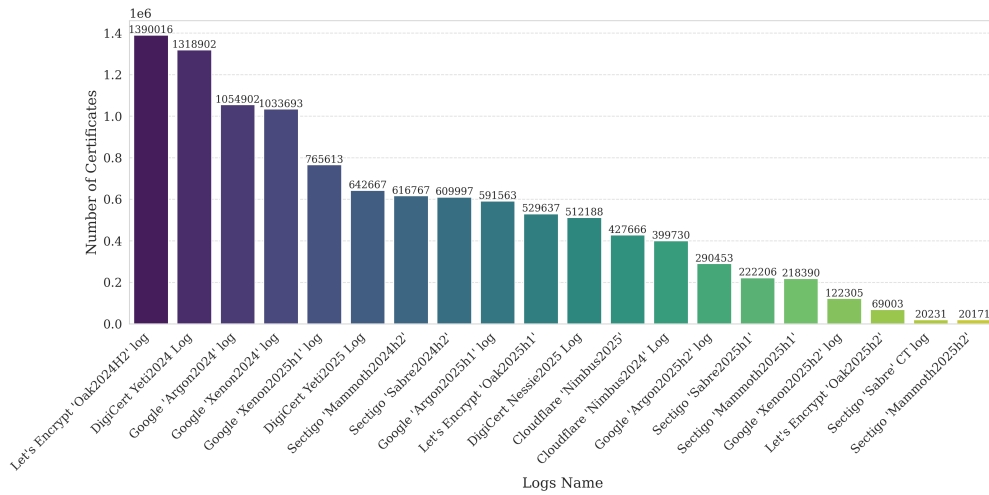


Figure 5.26. Top Signed Certificate Transparency (SCT) Logs.

Figure 5.26 illustrates the distribution of certificates in relation to Certificate Transparency logs. The data shows an almost even parity between the Let's Encrypt 'Oak2024H2' log (12.80%) and the DigiCert Yeti2024 Log (12.14%), followed by other logs. Interestingly, although many logs are in use, Google emerges as the predominant operator: in fact, as shown in Figure 5.27, 37.7% of the certificates have at least one SCT in a Google-managed log, followed by DigiCert (24.7%), Let's Encrypt (17%), Sectigo (13%) and Cloudflare (7.6%).

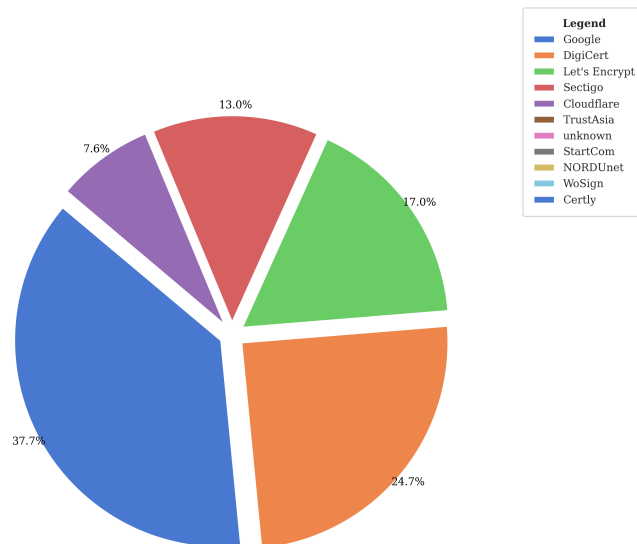


Figure 5.27. Top SCT Log Operators.

This distribution shows an effective diversification in SCT log management,

which helps minimise the risk of centralisation and increase the resilience of the Certificate Transparency system.

The CA/Browser Forum guidelines [4] and the RFC 6962 standard [22] highlight the importance of including SCTs in a certificate for transparency and security purposes in the chain of trust. The guidance states that public certificates must include SCTs from trusted and diverse logs, whereby there is transparency in the complete and timely view of certificate issuance.

## 5.6 Overview of Intermediate Certificates

This section focuses on intermediate certificates in the certificate chains associated with leaf certificates. The analysis revealed that a total of 9,428,980 intermediate certificate records were identified, but only 979 of these were distinct, almost 99.99% of the analysed intermediate certificates are duplicates. This result suggests that most leaf certificates are issued by the issuers themselves. Among the major issuers, as evidenced by Figure 5.28, stand out Let’s Encrypt, which holds 34.7% of the certificates, and Google Trust Services (together with Google Trust Services LLC), which together cover 33.14% of the issues.

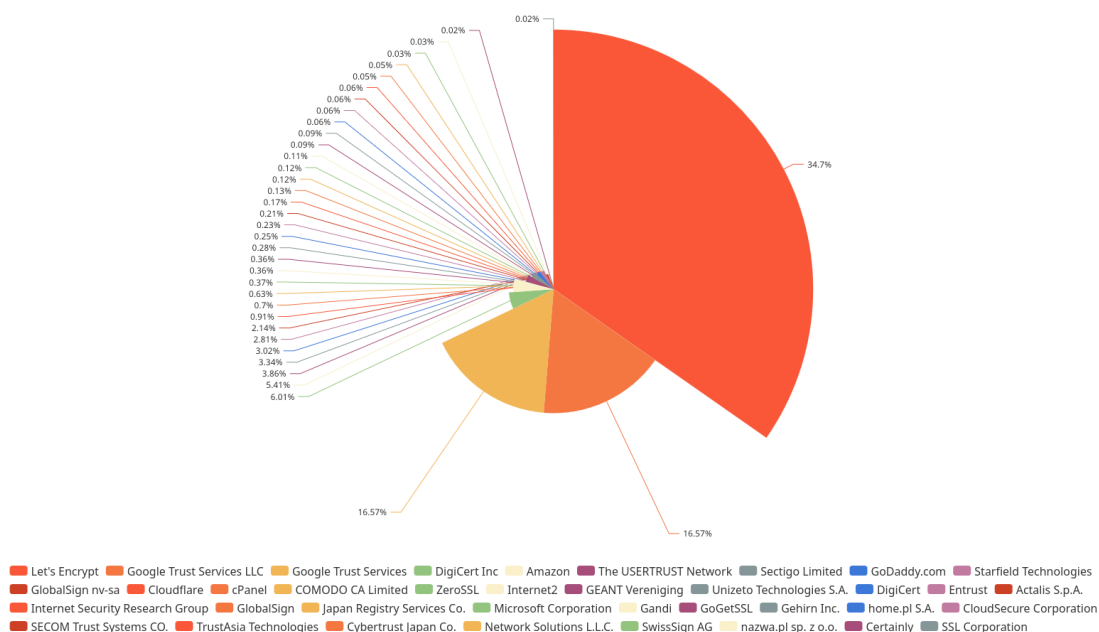


Figure 5.28. Distribution of various issuers for Leaf certificates across the analyzed dataset.

A further relevant detail concerns the structure of the chain: 64.08% of the intermediate certificates also serve as root certificates (i.e., represent the last certificate in the chain), while 32.43% appear as intermediate certificates with only

one certificate above it (the root) and only 3.48% of the intermediate certificates appear in longer chains with more than one intermediate certificate. The process of extracting intermediate certificates, performed via **DigitalCertiAnalytics**, took 13 hours and 27 minutes. In this process, each leaf certificate present in the output of ZGrab2 was analysed to trace the chain, uniquely extracting the intermediate certificates and removing any duplicates.

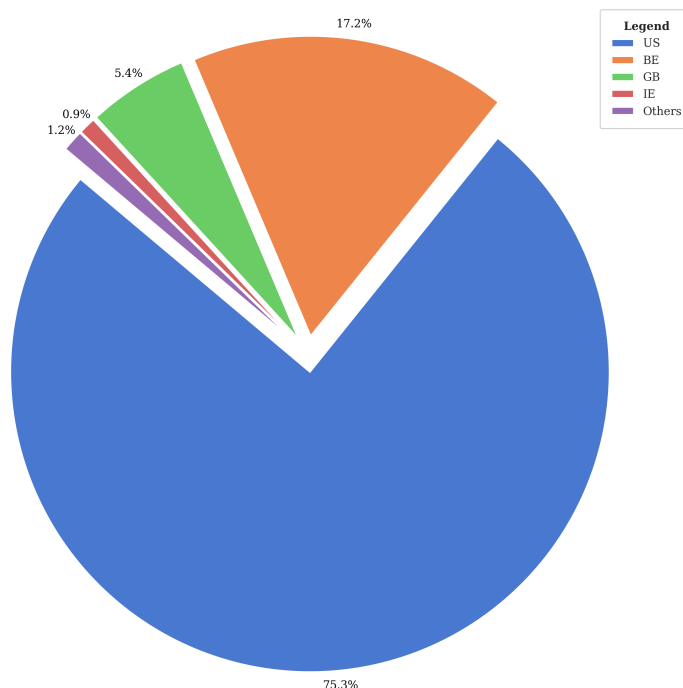


Figure 5.29. Distribution of intermediate certificates by country, with the majority issued by the United States and Belgium.

Figure 5.29 shows that 75.3% of the intermediate certificates were issued by US entities, followed by Belgium, which contributes 17.2%. It is also relevant that all intermediate certificates analysed adopted version 3 of the X.509 standard.

### 5.6.1 Validity Duration Distribution and Expiry Trends

In this analysis, the data show that 43% of intermediate certificates are valid for 10 years, while 7.76% are valid for 15 years and 6.84% for 5 years (Figure 5.30). These values indicate considerable variance in validity periods, which is further reflected in the distribution of expiry dates (Figure 5.31).

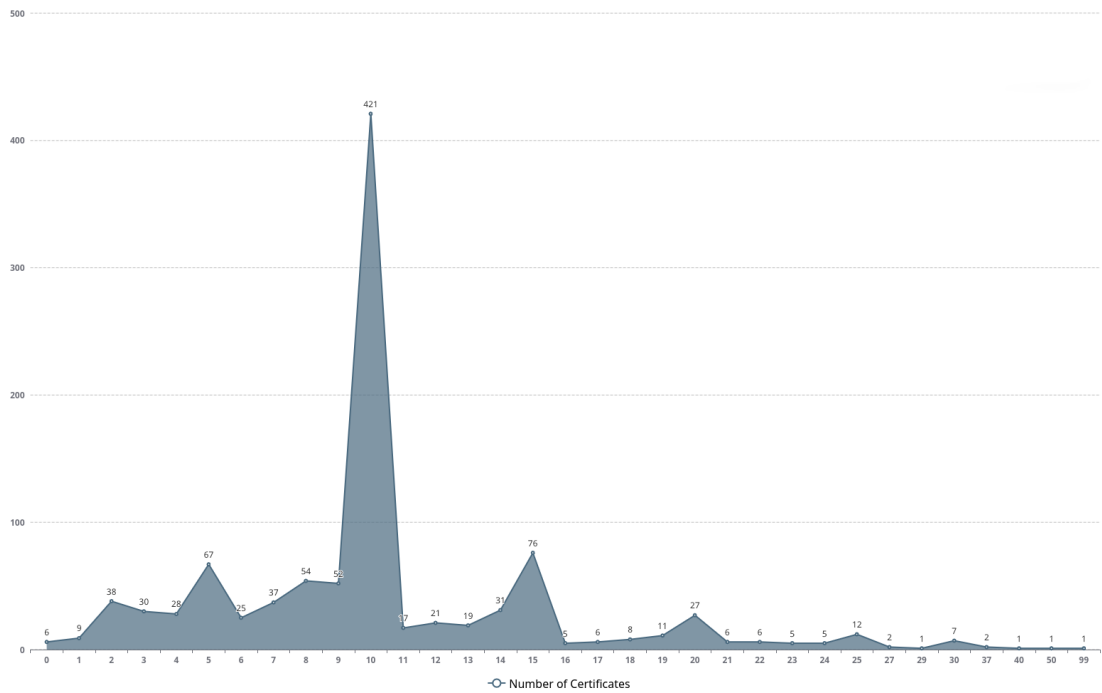


Figure 5.30. Distribution of the period of validity of intermediate certificates.

For instance, the highest concentration is found in certificates maturing in December 2030 (3.26%, or 32 certificates), followed by those maturing in May 2027 (2.45%, or 24 certificates), while slight variations are observed for maturities between 2025 and 2033. An additional notable fact is that 8.99% of the intermediate certificates are expired.

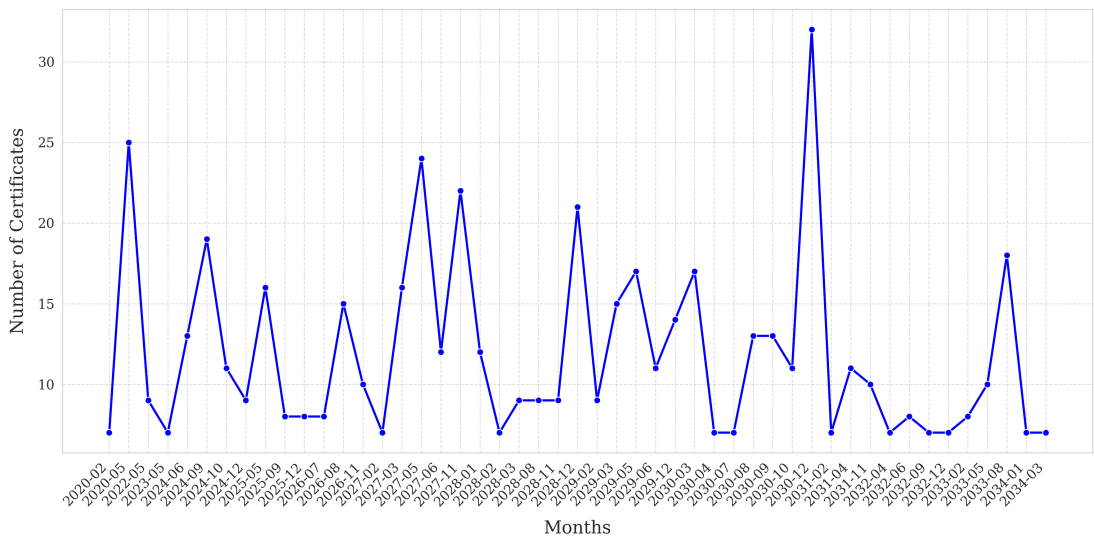


Figure 5.31. Deadlines for the distribution of intermediate certificates.



The CA/Browser Forum guidelines for intermediate certificates suggest that, although these certificates may have longer validity periods than leaf certificates, they must still comply with limits that balance the stability of issuance with the need to reduce exposure to risk in the event of compromise. In general, a validity of around 10 years is considered standard in the industry, as it offers an acceptable balance between operational management and safety. However, the presence of 15-year validity or significantly shorter durations (5 years) highlights a certain diversity in the issuing policies adopted by the various Certifying Authorities. In addition, the fact that a non-negligible percentage of intermediate certificates have expired (8.99%) underlines the need to improve the management and updating processes of the certification chains.

## 5.7 Analysis of X.509 Extensions in Intermediate Certificates

### 5.7.1 Analysis of Basic Constraints Extension

The analysis of the Basic Constraint extension showed excellent results (Figure 5.32), since all the intermediate certificates analysed had the `is_ca` field set to True. This requirement is fundamental, as it ensures that intermediate certificates can be used as certificate authorities, i.e. to sign other certificates within the chain of trust. The CA/Browser Forum guidelines emphasise the importance of certificates intended to act as CAs, whether root or intermediate, having the `is_ca` flag set to True, in order to ensure that their function of issuing and validating certificates is recognised and trusted [4].

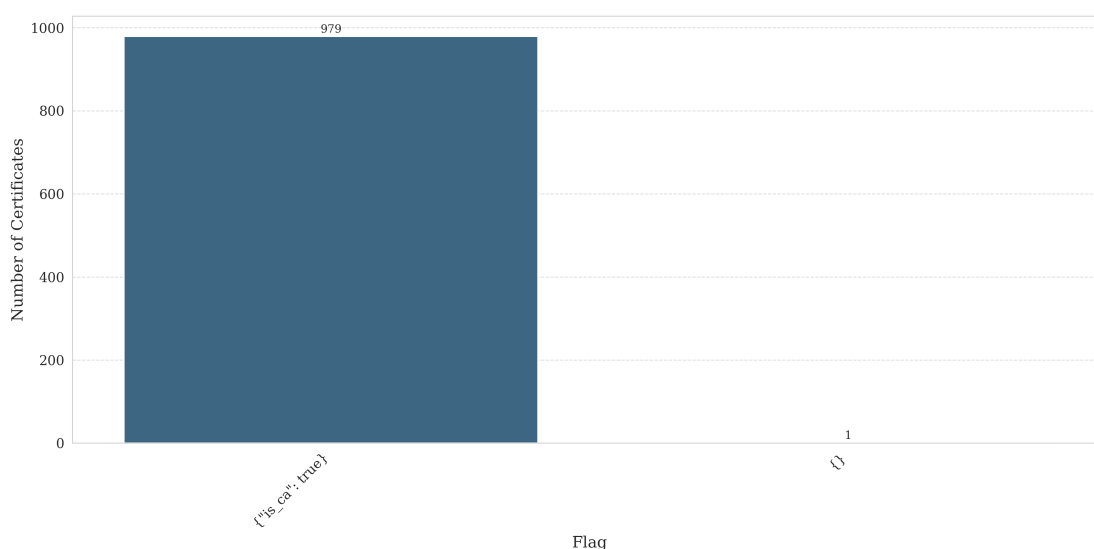


Figure 5.32. Distribution of Basic Constraint Extension.

## 5.7.2 Analysis of Key Usage Extension

The analysis of the Key Usage extension on intermediate certificates (Figure 5.33 and 5.34) reveals a distribution consistent with the best practices established by the CA/Browser Forum. In particular, 58.16% of the intermediate certificates present the combination of the `digital_signature`, `certificate_sign` and `crl_sign` flags, while 38.71% include only the `certificate_sign` and `crl_sign` flags, which represent the minimum requirements to enable CA functions, i.e. the ability to sign other certificates and manage revocation lists. Only 2.11% of the certificates report no value for this extension, a figure that denotes a negligible presence of incomplete configurations.

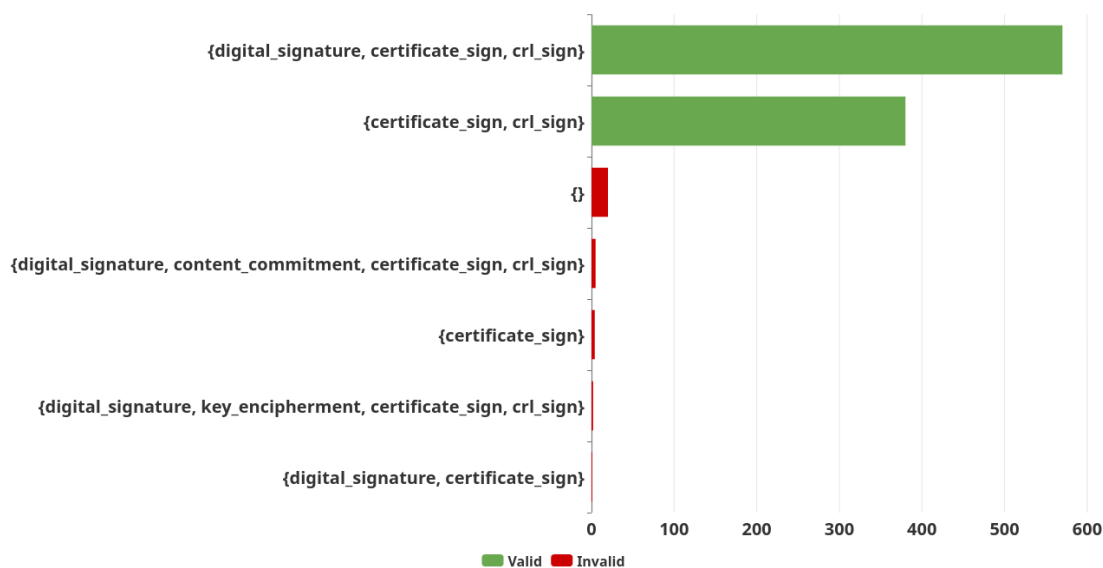


Figure 5.33. Key Usage in Extensions Field.

With regard to the criticality flag associated with the Key Usage extension, 89.42% of intermediate certificates set it as Critical, in conformity with the CA/Browser Forum guidelines, which recommend that the Key Usage extension be marked as critical in certificates intended to act as CA [4]. Through this marking, clients are obliged to strictly adhere to the limits of key usage, ensuring that the key is only used for the operations for which it was expressly intended such as signing certificates and management of CRLs.

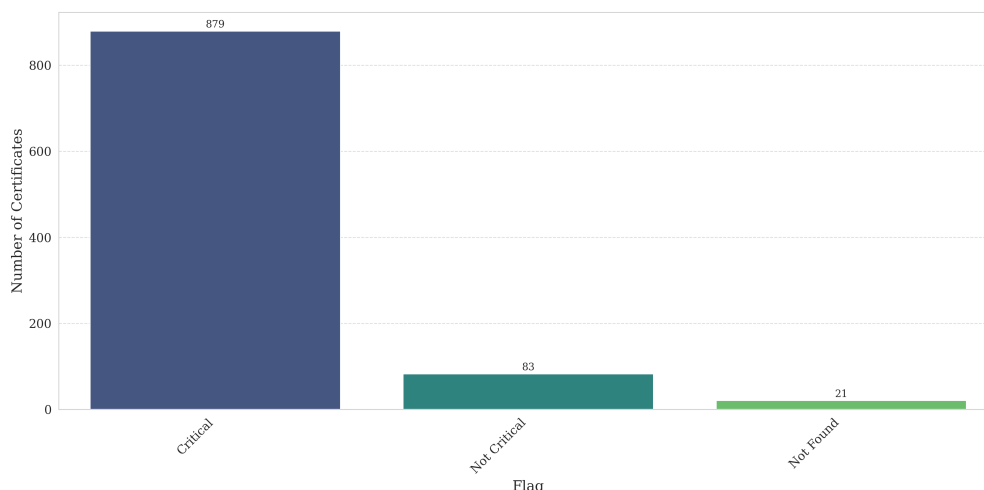


Figure 5.34. Critical vs Not Critical Key Usage in Extensions Field.

In contrast, 8.44% of the certificates define it as Not Critical, while in 2.14% the flag was not reported. These cases represent a minority, but indicate a slight deviation from recommended best practices, which could affect the overall operational security of the chain of trust.

### 5.7.3 Analysis of Authority Key Identifier Extension

According to the Baseline Requirements of the CA/Browser Forum, the inclusion of the AKI extension in intermediate certificates is mandatory. This requirement was introduced to ensure a clear and verifiable relationship between intermediate certificates and their issuing CA, reducing the risk of misconfiguration or man-in-the-middle attacks. The analysis showed that 99.92% of them include the AKI extension, while only 0.08% lack it. However, the presence of a small percentage of certificates without the AKI extension could pose a potential risk to the security and reliability of TLS connections.

### 5.7.4 Analysis of Certificate Policies Extension

The Baseline Requirements of the CA/Browser Forum recommend that the CP extension be included in intermediate certificates and, typically, be marked as non-critical [4]. This non-critical marking allows TLS clients to continue certificate validation even if they do not fully recognise or support this extension, without compromising the integrity of the verification process.

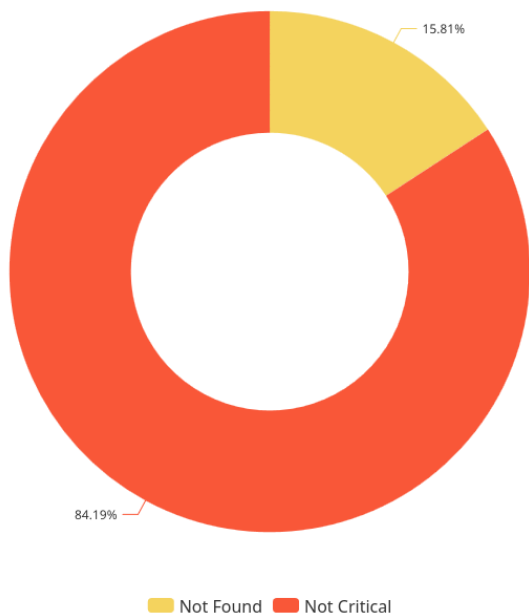


Figure 5.35. Comparison of Certificate Policies Extension: Critical vs Non-Critical in Intermediate Certificates.

The analysis of the intermediate certificates from the dataset indicated that in line with the CA/Browser Forum recommendations, 84.19% of the certificates with the CP extension were marked as Not Critical, while 15.81% of the intermediate certificates without the extension could indicate a slight non-compliance or a particular choice on the part of the issuer.

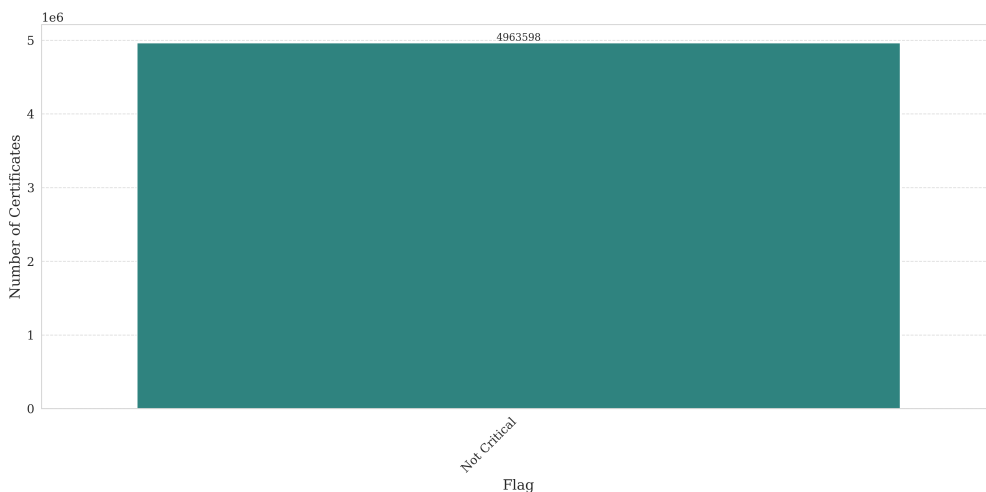


Figure 5.36. Critical vs Not Critical Certificate Policies Extension.

### 5.7.5 Analysis of Extended Key Usage Extension

The analysis of the Extended Key Usage (EKU) extension for intermediate certificates shows a modest variability in the configuration of the reported values. In particular, 52.79% of the intermediate certificates include in the EKU field the properties `server_auth` and `client_auth`, while only 1.79% include only the property `server_auth`. In contrast, 42.13% of the intermediate certificates do not report any value for the EKU extension.

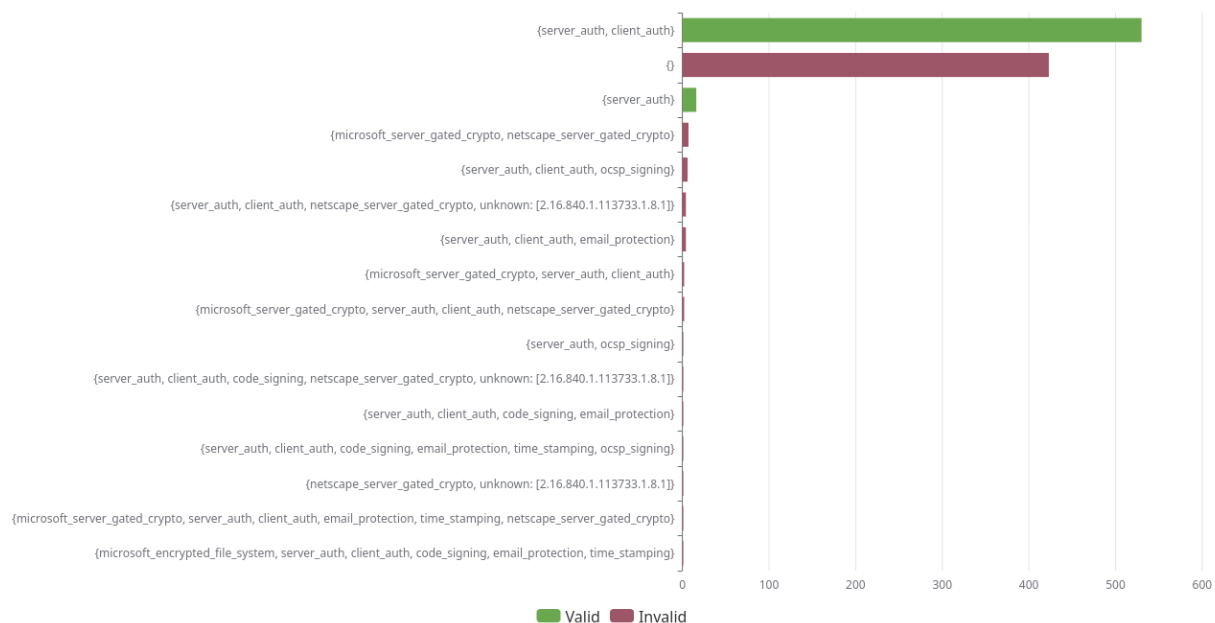


Figure 5.37. Extended Key Usage in Extensions Field.

The CA/Browser Forum states that the EKU field must be present in intermediate certificates, for its accurate setting is important to restrict the key usage to only the functionality for which it was designed [4]. Therefore, the combined joint use of `server_auth` and `client_auth` in over half of the cases stands for a common practice to provide flexibility and interoperability in authentication operations.

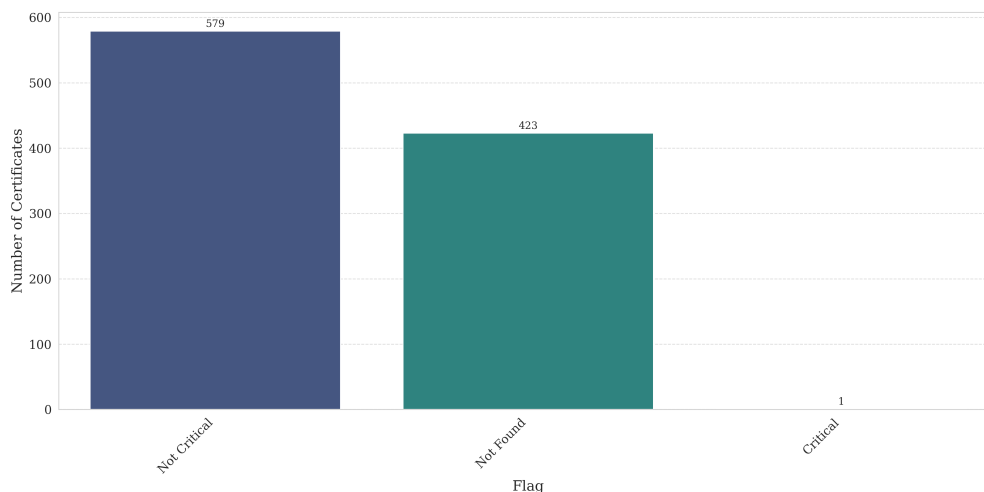


Figure 5.38. Critical vs Not Critical Extended Key Usage in Extensions Field.

The evaluation of the criticality flags of the EKU extension shows that 57.73% of intermediate certificates indicate Not Critical, while 42.17% show no indication of criticality. The CA/Browser Forum standards indicate that the Critical flag should remain inactive when applied to the EKU extension in intermediate certificates, since this indicator provides additional details on the use of keys that do not change the validation parameters [4].

### 5.7.6 Analysis of Subject Key Identifier Extension

According to the Baseline Requirements of the CA/Browser Forum, the inclusion of the SKI extension in certificates is mandatory [4]. This extension provides a unique identifier for the subject's public key, facilitating the management and verification of certificate chains. The analysis revealed that 99.98% of them include the SKI extension, while only 0.02% lack it. This figure shows a strong adherence to the recommendations of the CA/Browser Forum.

### 5.7.7 Analysis of Authority Information Access

The Authority Information Access (AIA) extension is crucial for intermediate certificates, as it provides the information needed to retrieve revocation lists (CRLs) and to access OCSP responses. According to the Baseline Requirements of the CA/Browser Forum, the inclusion of the AIA extension is strongly recommended in intermediate certificates, as it allows clients to check the revocation status of certificates in real time, helping to ensure the integrity of the chain of trust [4].

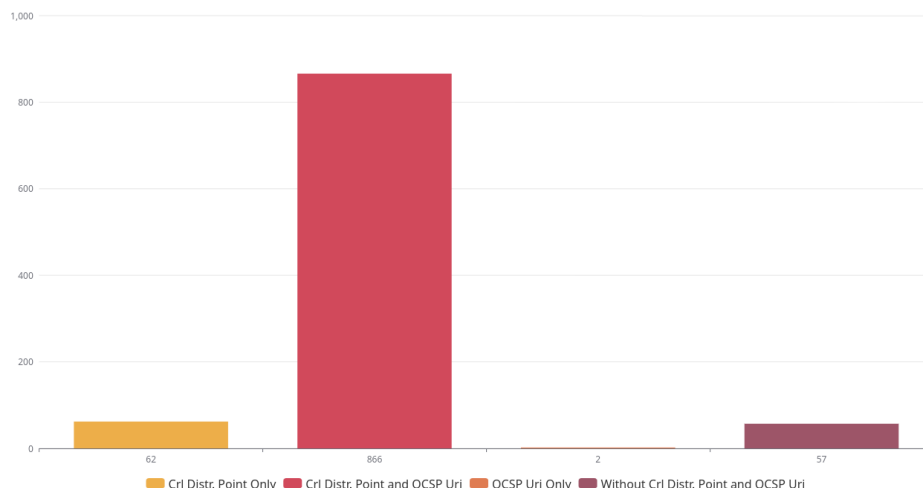


Figure 5.39. Distribution of Authority Information Access (AIA) in Intermediate Certificates.

An analysis of the dataset shows that the majority of intermediate certificates (89.77%) include both the CRL Distribution Point and the OCSP URI, thus offering complete coverage for revocation status checking. Only 0.2% of the certificates report only the OCSP URI, while 6.28% include only the CRL Distribution Point. A further 5.78% of the certificates lack the AIA extension, which may represent a potential weakness in the validation process, since the absence of access information may prevent clients from correctly verifying the revocation status.

Furthermore, the analysis on the criticality flag shows that 88.11% of the intermediate certificates have the AIA extension marked as Not Critical, while in 117 cases the extension was not detected. The marking as Not Critical is in line with the CA/Browser Forum guidelines, as it allows clients to proceed with validation even if the extension is not explicitly interpreted, thus guaranteeing a higher level of interoperability [4].

### 5.7.8 Analysis of Certificate Revocation List

Most of the intermediate certificates analyzed include the extension related to Certificate Revocation List (CRL) and, in particular, 94.09% of the certificates report this extension as Not Critical, while in 5.91% certificates the extension was not found. According to the Baseline Requirements of the CA/Browser Forum, the CRL Distribution Points extension must be configured as Not Critical and is essential to enable clients to retrieve the list of revoked certificates and check the revocation status of certificates in real time [4].

## 5.8 Distribution of Signature and Key Algorithms in Intermediate Certificates

The analysis of intermediate certificates reveals a strong adherence to the CA/Browser Forum guidelines, which recommends the use of RSA keys of at least 2048 bits and the use of signing algorithms employing secure hash functions, such as SHA256 [4]. In this context, the predominance of RSA (89% of intermediate certificates) not only confirms compliance with the minimum requirements, but also underscores a commitment to further strengthening security.

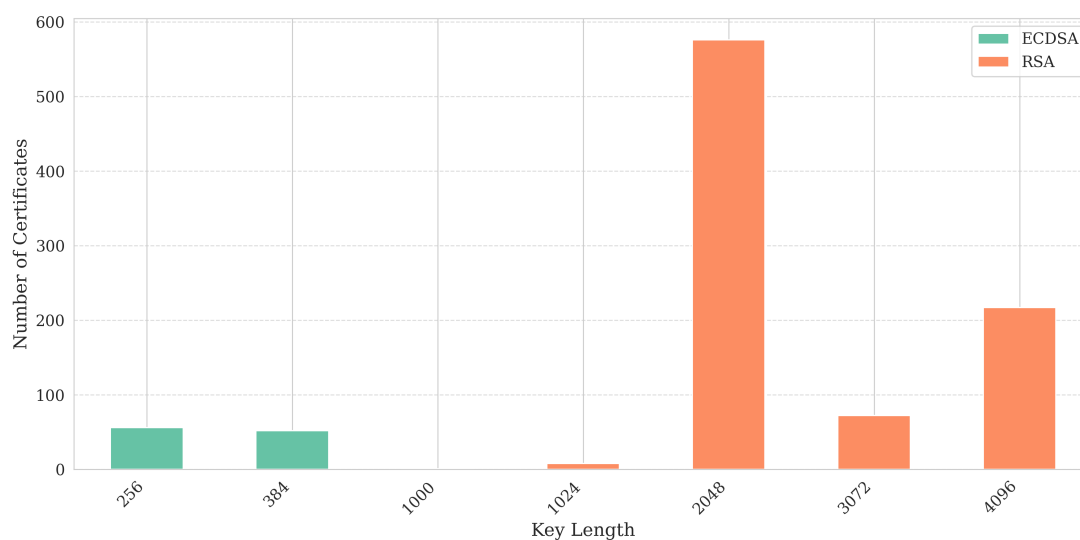


Figure 5.40. Distribution of Key and Length Algorithms.

Specifically, the analysis finds that the majority of RSA certificates use 2048-bit keys (65.90%), while a substantial proportion adopt 4096-bit keys (24.82%) and a smaller proportion opt for 3072-bit ones (8.24%). These data indicate a diversified strategy: while the minimum requirement is met, some CAs prefer to adopt longer keys to achieve an additional layer of protection, knowing that an increase in key length can reduce the risk of compromise, despite the fact that it entails a greater computational load.

The move toward the use of ECDSA, although still marginal compared to RSA, shows a well-considered balance between security and efficiency. Certificates employing ECDSA split almost equally between 256-bit (51.85%) and 384-bit (48.15%) curves. This distribution suggests that, in the absence of stringent performance constraints, CAs tend to choose more robust curves (384 bits) for scenarios where superior protection is required, while in contexts where computational efficiency is a critical factor they opt for 256-bit curves.



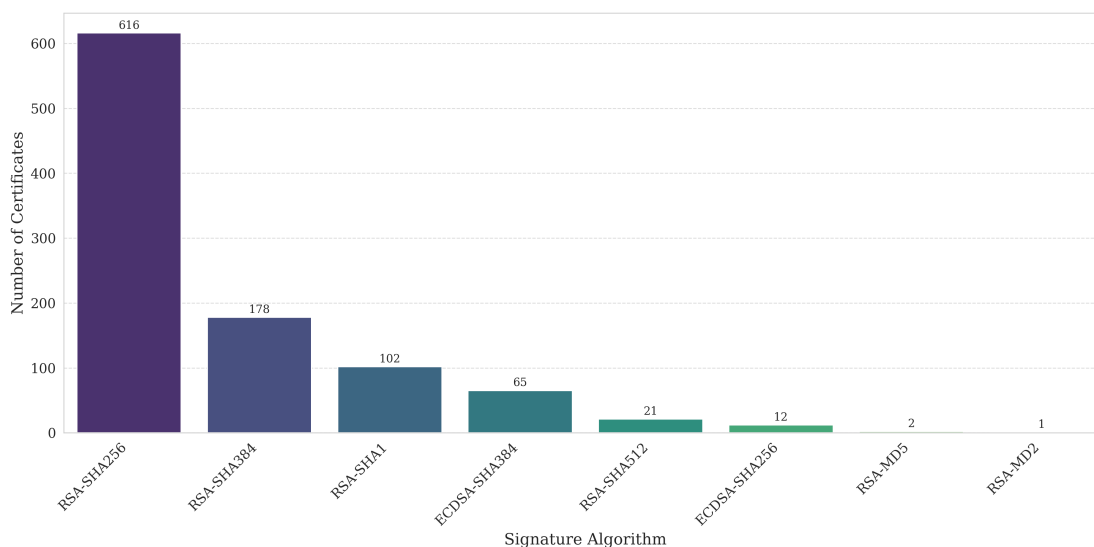


Figure 5.41. Signature Algorithm Used.

The analysis of signing algorithms, shown in Figure 5.41, alongside the other one clearly illustrates that RSA is the overwhelming favorite with 92.28% of intermediate certificates signed with it. Within that category, most certificates sign with RSA-SHA256 (66.96%) while close to RSA-SHA384 (19.35%). The conclusion drawn from this would certainly be: while safe hash algorithms are encouraged by CA/B Forum recommendations, RSA-SHA256 is the most popular choice perhaps due to its wide support and a pretty good trade-off between secure performance and efficiency. On the other hand, ECDSA certificates are duly used to sign ECDSA-SHA384 (84.42%) more than ECDSA-SHA256 (15.58%), which implies bias towards configurations where extra security is favored and performance is not an issue.

## 5.9 Overview of Root Certificates

In this section, the analysis focuses on root certificates included in certificate chains associated with leaf certificates, thus completing the overview of key components of the chain of trust. The data show that 370,592 leaf certificates include the root certificate in their chain. However, only 1,974 root certificates appear to be distinct, meaning that nearly 99.47% of root certificates are repeated across chains. It is important to point out that, in this context, `root certificate` means only those root certificates that are transmitted during the TLS process along with the leaf certificate, and not the trust anchors preinstalled in operating systems or browsers.

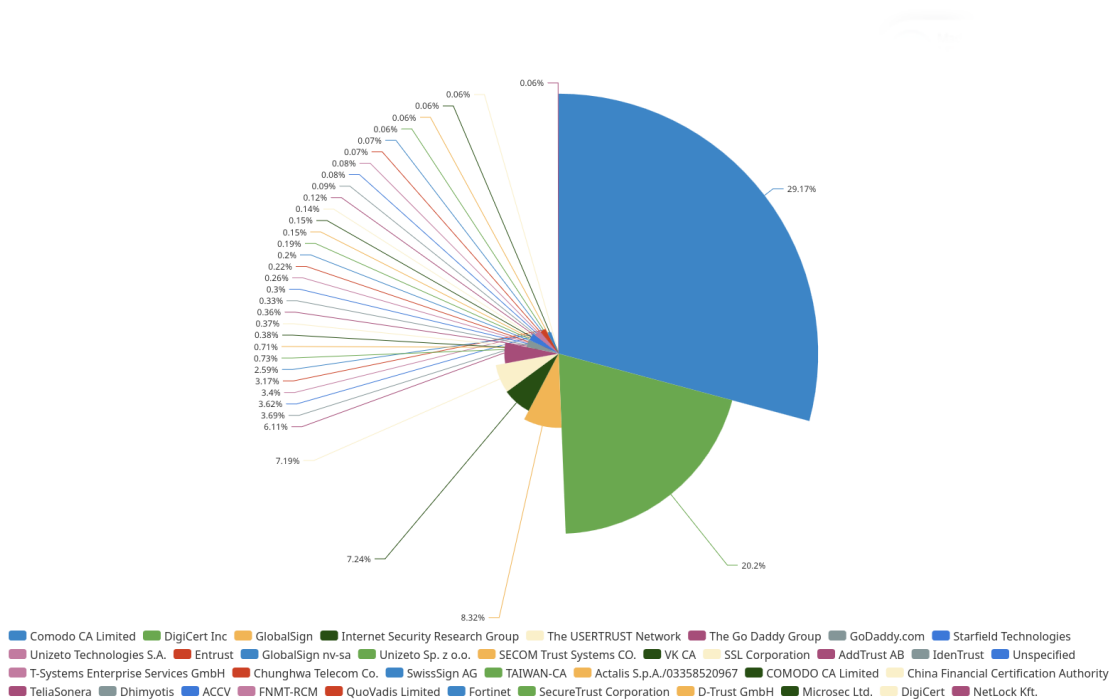


Figure 5.42. Overview of the distribution of root certificates in the dataset.

The distribution of root certificates, shown in Figure 5.42, shows that the majority of certificates are managed by Comodo CA Limited (29.17%), followed by DigiCert Inc (20.2%) and GlobalSign (8.32%). The root certificate analysis, also conducted with **DigitalCertiAnalytics**, took about 57 minutes. In this case, the process consisted of moving up the chain from each leaf certificate to the root certificate, which was then stored in the database.

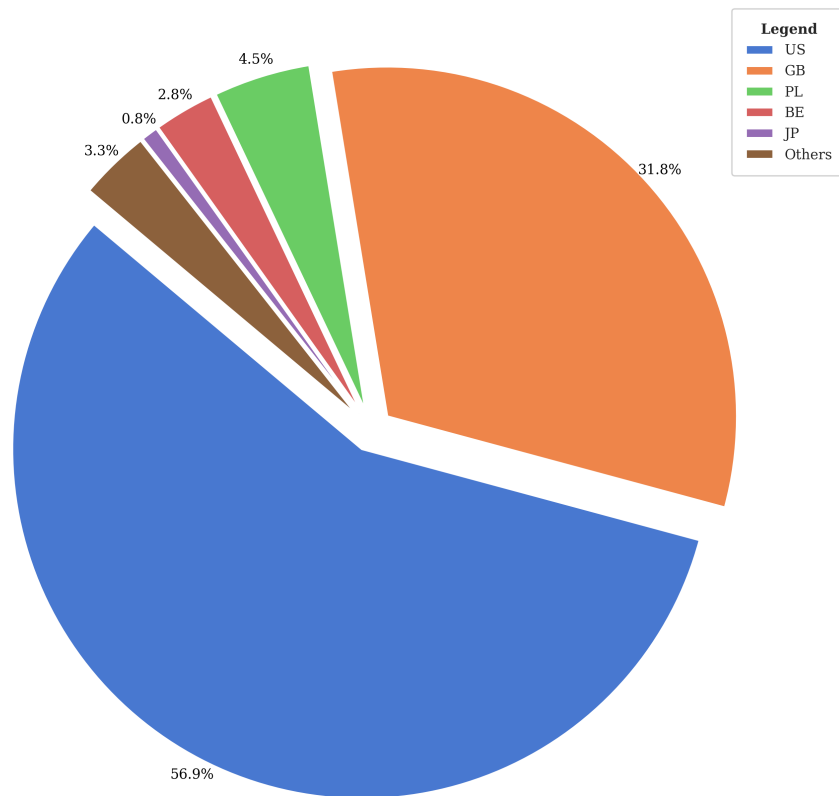


Figure 5.43. Distribution of root certificates by country, with the majority issued by the United States and Great Britain.

Figure 5.43 shows that 56.9% of root certificates were issued by U.S. entities, followed by Great Britain contributing 31.8%. In addition, it should be specified that during the analysis all root certificates adopt version 3 of the X.509 standard.

### 5.9.1 Distribution of Validity Duration and Expiry Trend

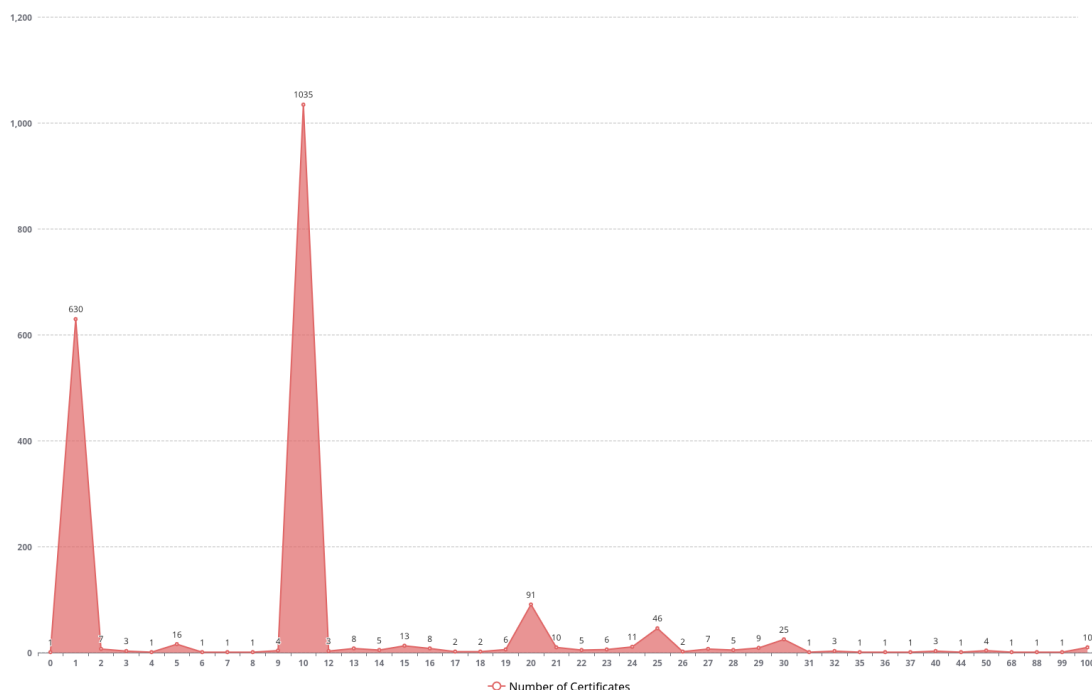


Figure 5.44. Distribution of the period of validity of root certificates.

Figure 5.44 clearly shows that root certificates are concentrated in two major groups in terms of duration: 51.96% of root certificates have a validity of 10 years, while 31.63% have a validity of 1 year. This result indicates that most root certificates follow a long-term policy, typical for anchor trusts, but there is also a sizeable minority with significantly shorter durations, which may reflect specific policies adopted by some issuers.

It is apparent from Figure 5.44 that root certificates are predominantly divided into two categories with respect to duration: 51.96% of root certificates are valid for 10 years and 31.63% for 1 year. This shows that most root certificates apply the long-term policy often followed by anchor or trust CAs, though there still stands a huge number of minority examples who have a far shorter duration, perhaps reflecting particular policies adopted by some issuers.

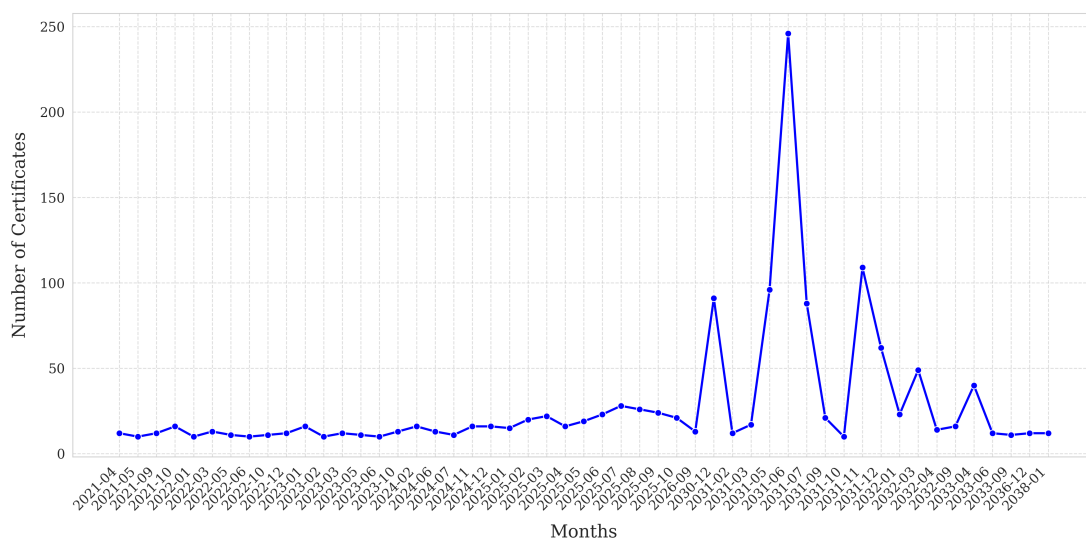


Figure 5.45. Deadlines for the distribution of root certificates.

Meanwhile, Figure 5.45 shows the trends of the root certificates maturity. The analysis we made shows the highest peak in certificates expiration to be in June 2031, with a proportion of 12.46%, while the following prominent peaks occurred in November 2031 (5.52%), May 2031 (4.86%), and, lastly, in December 2030 (4.61%). It is also worth noting that 25.03% of root certificates are expired, thus indicating necessity for regular updates or replacements.

According to CA/Browser Forum guidelines, root certificates should be issued with validity policies that ensure long-term stability and security [4]. Common practice calls for 10-year durations, as they provide a balance between reliability and the need to periodically renew keys to reduce risk in the event of compromise. The results obtained confirm this trend, although they show a minority of certificates with very low validity, which could be due to special operational needs or specific strategies of some issuers.

## 5.10 Analysis of X.509 Extensions in Root Certificates

### 5.10.1 Analysis of Basic Constraints Extension

Analysis of the Basic Constraint extension shows an extremely positive result, all the certificates analyzed have the `is_ca` field set to True. This indicates that each certificate, in this context, is configured to operate as a certificate authority, a basic requirement for a root certificate as specified by the CA/Browser Forum [4].

### 5.10.2 Analysis of Key Usage Extension

The CA/Browser Forum guidelines state that, for maximum security, root certificates should be configured to limit the use of keys to only necessary operations, particularly signing certificates and CRLs, by including the `certificate_sign` and `cr1_sign` flags, and where appropriate also another flag such as `digital_signature` for additional functions [4].

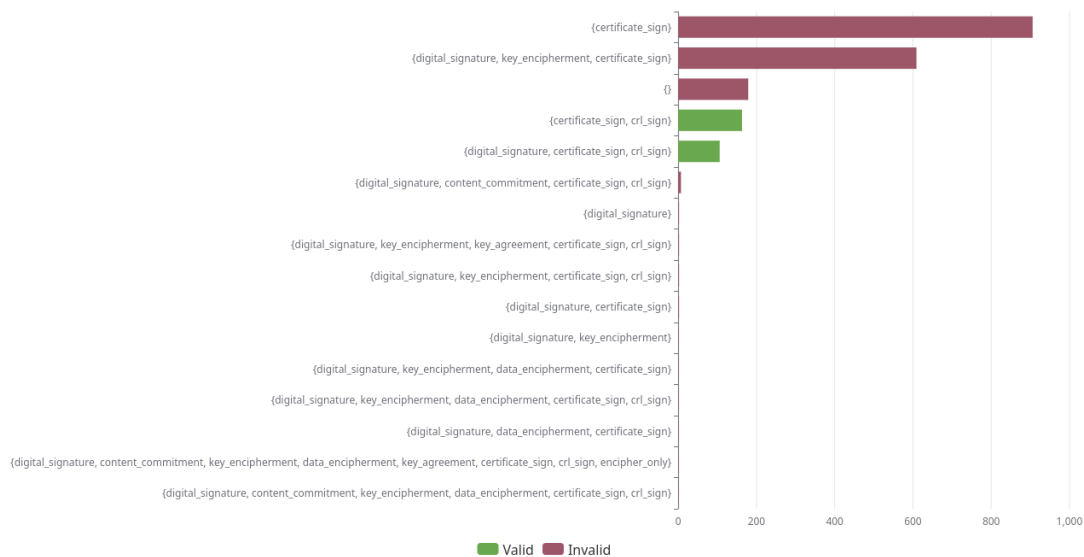


Figure 5.46. Key Usage in Extensions Field.

Analysis of the root certificates shows that only 13.56% of the certificates are in line with the CA/Browser Forum recommendations, where 8.21% report `certificate_sign` and `cr1_sign` as the value, while 5.34% also report `digital_signature`. 45.67% of the certificates report only `certificate_sign` as a value, while 30.70% show the combination of the `digital_signature`, `key_encipherment`, and `certificate_sign` flags, which are settings not allowed by the Baseline Requirements of the CA/Browser Forum. In contrast, 9.02% of root certificates report no value for the Key Usage field, a situation that could indicate anomalies or special configurations in the issuance process.

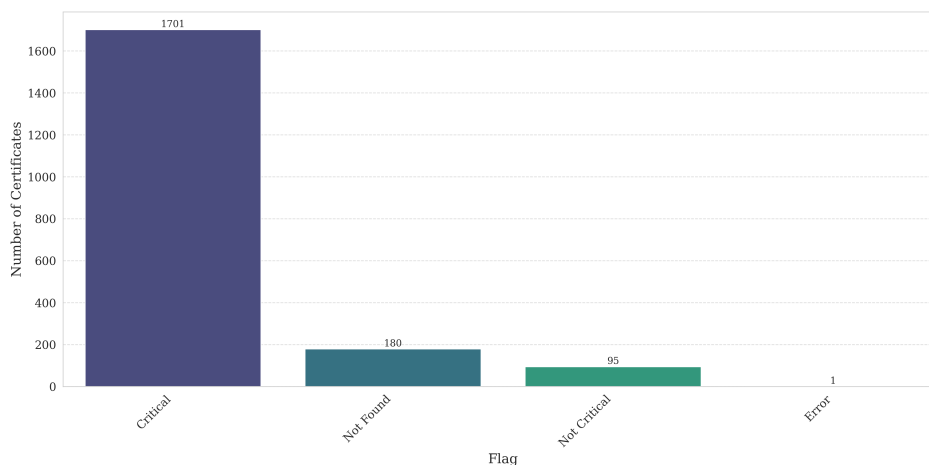


Figure 5.47. Critical vs Not Critical Key Usage in Extensions Field.

Concerning the criticality flag presented in Figure 5.47, 86.04% of root certificates set the Key Usage extension as Critical, in accord with CA/Browser Forum recommendations for critical marking for certificates serving as a trust anchor [4]. Only 4.86% of certificates declare the extension Not Critical, while in 9.10% of cases the flag was not declared. So those numbers are generally in favor of best practices, although the tiny amount of offending configurations could throw some doubt over operational consistency of certain cases of the certificate.

### 5.10.3 Analysis of Extended Key Usage

Figure 5.48 highlights that 99.19% of root certificates do not include the Extended Key Usage (EKU) extension, in line with CA/Browser Forum guidelines, which do not recommend the inclusion of this extension in root certificates [4]. These certificates, used exclusively as trust anchors, do not require additional restrictions on public key usage, which is why the absence of EKU is considered correct.

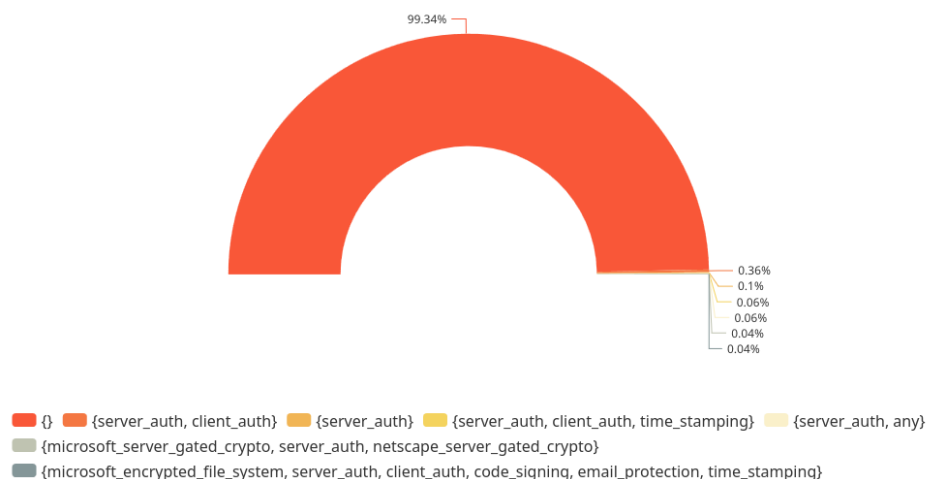


Figure 5.48. Distribution of Extended Key Usage (EKU) in Root Certificates.

However, 16 root certificates were identified that include information in the EKU extension: in 7 cases the combination of `server_auth` and `client_auth` is reported, and in 2 cases only `server_auth`. Among them, 15 certificates have the extension marked as Not Critical, while in one certificate it is set as Critical.

#### 5.10.4 Analysis of Subject Key Identifier

The CA/Browser Forum Baseline Requirements state that, even in root certificates, the Subject Key Identifier (SKI) extension must be present [4]. Analysis of the dataset showed that 98.95% of root certificates include the SKI extension, while only 1.05% are without it. The near-universal presence of the SKI in root certificates is essential to support efficient and secure validation, while the lack of it in a small percentage of cases could indicate configuration errors or non-compliance with best practices, elements that could compromise the proper construction of the chain of trust.

#### 5.10.5 Analysis of Authority Key Identifier

In the context of root certificates, the CA/Browser Forum guidelines specifically recommend the inclusion of the Authority Key Identifier (AKI) extension [4]. This extension is critical because it helps link the certificate to the correct trust anchor. Analysis of the dataset revealed that only 15.76% of root certificates include the AKI extension, while the remaining 84.24% lack it. This high rate of absence represents a significant deviation from CA/Browser Forum recommendations and could compromise the ability of validation systems to reliably trace back to the issuing authority.



### 5.10.6 Analysis of Certificate Policies

The CA/Browser Forum guidelines for root certificates do not require the inclusion of the Certificate Policies extension; indeed, if this extension is present, it should be marked as Not Critical [4]. This is because root certificates, being the trust anchors of the PKI, do not need further guidance on operational policies, and the possible presence of this extension could unnecessarily complicate the management of the chain of trust. Analysis of the dataset showed that 98.68% of root certificates do not have the Certificate Policies extension, which is fully compliant with the recommendations. Only 1.32% of root certificates include such an extension, and in all cases where it is present it is correctly marked as Not Critical. These results indicate that most issuers adopt practices that comply with CA/Browser Forum guidelines, keeping the structure of root certificates simple and focused on the trust anchor function. The presence of the extension in a minority of cases could represent isolated instances or specific operational choices.

## 5.11 Distribution of Signature and Key Algorithms in Root Certificates

The distribution of root certificates analyzed shows that all certificates bear a valid signature, in line with CA/Browser Forum recommendations for trust anchors. Specifically, 98.48% of root certificates use the RSA algorithm, while the adoption of ECDSA is marginal. Among RSA certificates, 84.25% use 4096-bit keys, while 13.90% adopt 2048-bit keys, thus meeting the guidelines that recommend robust keys to ensure a high level of security. For ECDSA-based certificates, 256- and 384-bit lengths are adopted almost equally, although their use is very limited.

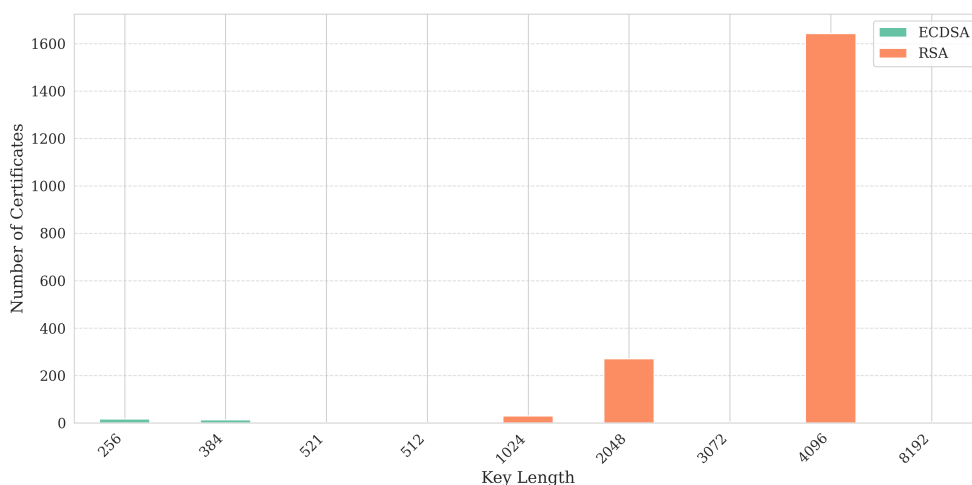


Figure 5.49. Distribution of Key and Length Algorithms.

In terms of signing algorithms, 97.11% of root certificates use RSA, with 86.42%

employing RSA-SHA256, an algorithm considered secure and compliant with current standards, while 10.14% use RSA-SHA1, which is less secure and being deprecated. Only 2.89% of root certificates use ECDSA, split almost equally between ECDSA-SHA256 and ECDSA-SHA384. Residual use of RSA-MD5 was also found in about 0.91% of cases, which is an obsolete configuration.

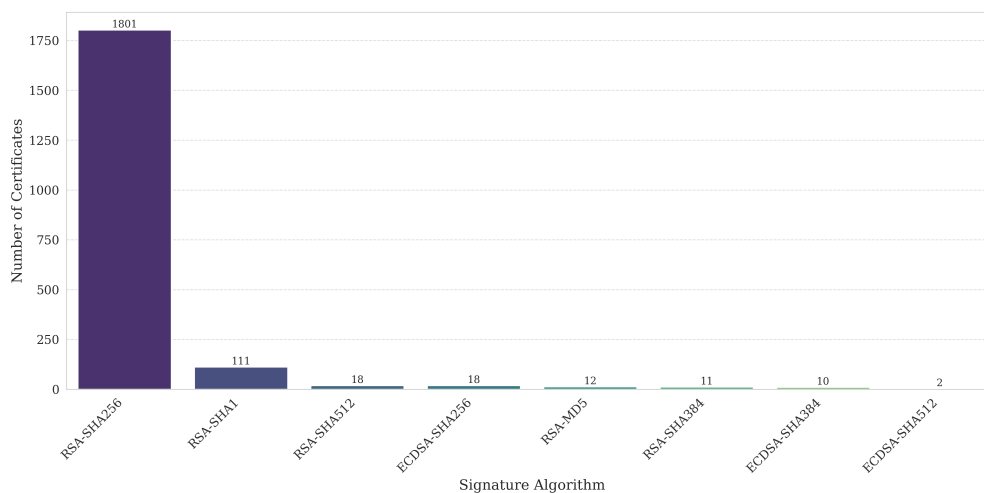


Figure 5.50. Signature Algorithm Used.

The CA/Browser Forum guidelines emphasize that root certificates, being the trust anchors of the PKI, should be issued with keys of appropriate length (typically a minimum of 2048 bits, with a preference for 4096 bits for greater security) and with signing algorithms based on SHA256 or higher. The data analyzed confirm a general adherence to these recommendations, although the presence of less secure configurations, such as RSA-SHA1 and RSA-MD5, highlights the need for further updates by some issuers.

## 5.12 Zlint Analysis for Leaf Certificates

The analysis of the leaf certificates was conducted using the Zlint tool, as described in the software chapter (4). In this phase, all leaf certificates stored in the non-relational database were extracted in order to rank the results returned by the various linters. The results are expressed with the following categories: **pass**, **info**, **warn**, **error**, **NA** (Not Applicable) and **NE** (Not Evaluated). The main objective is to identify which lint persists the most and, at the same time, to discover which issuers have the most errors, thus identifying the main perpetrators of the anomalies found.

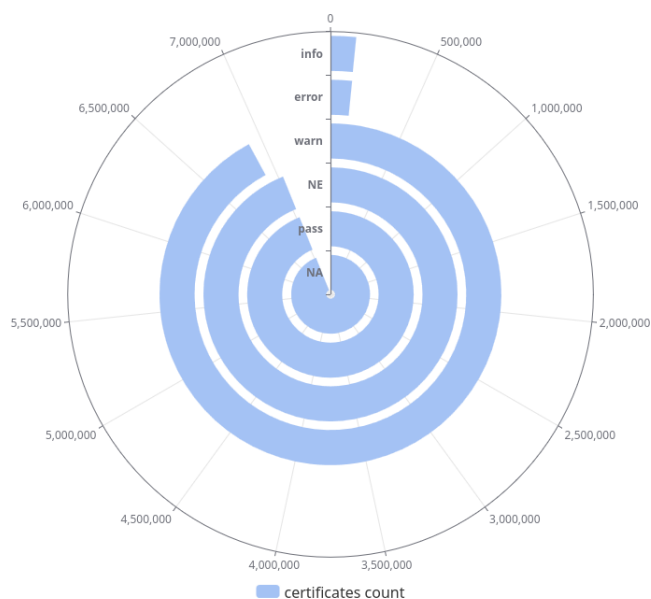


Figure 5.51. Distribution of Lints results.

The data show that only 1.71% of the leaf certificates had at least one lint with a **error** result, which is slightly higher than what was observed in the study conducted by [32], where on a dataset of 61 million certificates the error rate stood at 0.3%. It is important to consider, however, that the dataset used in this work has approximately 7 million certificates, making the comparison an indication of the level of severity of errors found in a significantly smaller data set.

The Figure 5.51 indicates the combinatorial distribution of lints per issuer: 99.99% of certificates that have at least one lint with a **pass**, **NA**, or **NE** result, while 98.02% of certificates contain at least one lint with a **warn** result. This indicates that, while the vast majority of checks are passing, there are consistently present warnings that could die further improvements to be made and very likely issues in the way that issuing is done.

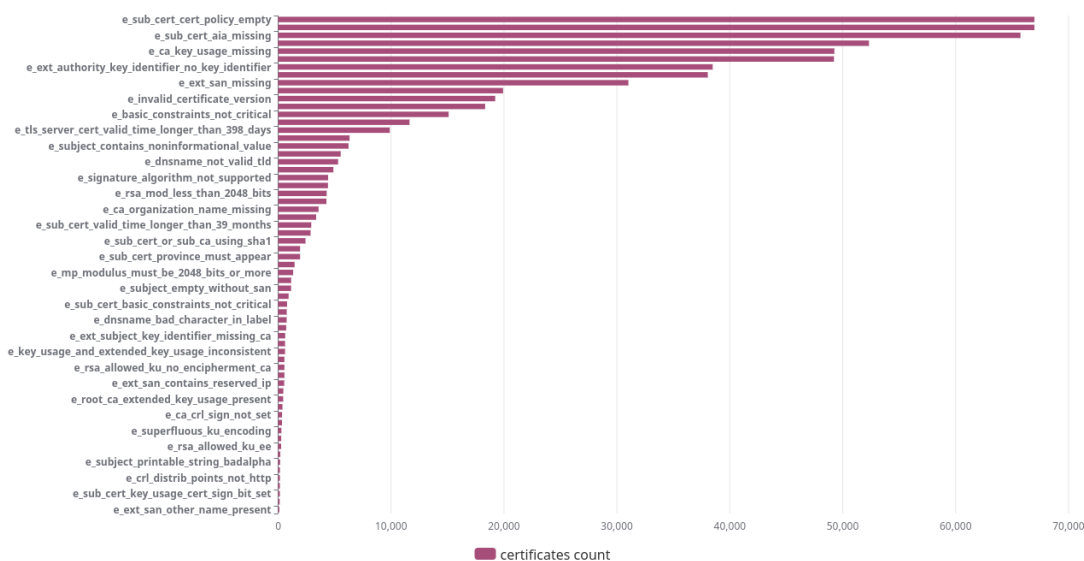


Figure 5.52. Most Common Lints Found in Digital Certificates.

Figure 5.52 presents, in descending order, the most frequently encountered lints in digital certificates. Of these, the most relevant are:

- **e\_sub\_cert\_cert\_policy\_empty**: Indicates that subscriber certificates must contain at least one policy identifier attesting adherence to CAB standards (BRs: 7.1.2.3).
- **e\_sub\_cert\_certificate\_policies\_missing**: Indicates the absence of the `certificatePolicies` extension, which, according to the Baseline Requirements, must be present and not marked as critical.
- **e\_sub\_cert\_aia\_missing**: The absence of the `authorityInformationAccess` extension, mandatory for subscriber certificates, is highlighted.
- **e\_sub\_cert\_aia\_does\_not\_contain\_ocsp\_url**: Indicates that, if present, the `authorityInformationAccess` extension must contain the HTTP URL of the CA's OCSP responder.
- **e\_ca\_key\_usage\_missing**: Indicates the absence of the `Key Usage` extension in CA certificates (both root and subordinate), as required by the BR and by the RFC 5280 standard.
- **e\_root\_ca\_key\_usage\_present**: Check that the root certificates add the `Key Usage` extension, as this is the minimum requirement for their trustworthy anchors.

These lints, being the most frequent, represent the main problems encountered during the analysis; other errors, present in smaller quantities, were considered less significant for the study. A complete description of each lint can be obtained by running the command:

```
zlint -list-lints-json
```

This command generates a JSON file that includes, for each lint, the name, description, reference standard citation and source.

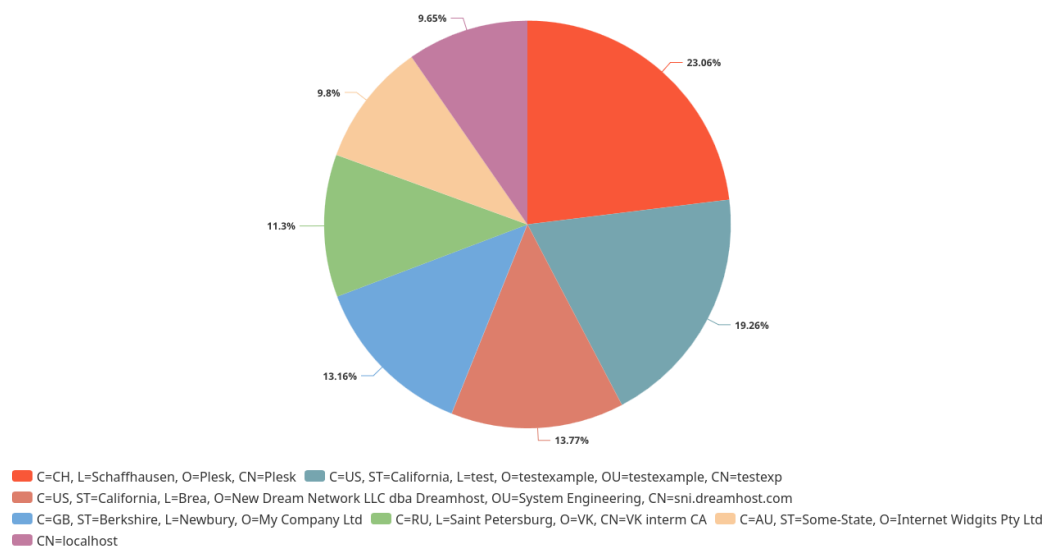


Figure 5.53. Distribution of Issuers with the Highest Number of Lint Errors.

Figure 5.53 shows, on the other hand, the distribution of issuers with the highest number of erroneous lints. The data reveals that the issuers with the highest number of errors are generally smaller organisations, such as Plesk, suggesting that the quality of certificates issued by these entities may be lower than that guaranteed by more established CAs, such as Let’s Encrypt or Google.

## 5.13 Compliance Analysis Against CA/Browser Forum Requirements

### 5.13.1 Comparison of Leaf TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.7.6)

- **Serial Number**

- **Dataset Results:**

- \* 100% present
- \* 99,98% valid serial number
- \* 0 certificates with serial number longer than 20 bytes
- \* 34 certificates with negative serial number
- \* 989 certificates with duplicate serial number
- \* 234 certificates with serial number equal to 0

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.7:** Must appear.
- \* Must be positive and greater than 0.
- \* Must not be longer than 20 bytes.
- **Validity Duration**
  - **Dataset Results:**
    - \* 99,88% have a valid validity
    - \* 0,12% do not meet the requirements
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 6.3.2:** For certificates issued between April 2015 and September 1, 2020, the validity must not exceed 39 months, while certificates issued after September 1, 2020, must not have a validity longer than 398 days.
- **Key Usage**
  - **Dataset Results:**
    - \* 99,36% extension present
    - \* 0,63% missing
    - \* 99,31% marked critical
    - \* 0,056% marked not critical
    - \* For RSA (71,24%):
      - 99,09% have `digitalSignature` and `keyEncipherment`
      - 0,88% empty
      - 0,02% others
    - \* For ECDSA (28,76%):
      - 97,21% have `digitalSignature`
      - 2,78% have `digitalSignature` and `keyAgreement`
      - 0,01% others
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.11:** Optional.
    - \* For RSA, if present, SHOULD include `digitalSignature`, MAY include `keyEncipherment` and NOT RECOMMENDED include `dataEncipherment`.
    - \* For ECC, if present, MUST include `digitalSignature` and NOT RECOMMENDED include `keyAgreement`.
    - \* Must be marked as critical.
- **Extended Key Usage (EKU)**
  - **Dataset Results:**
    - \* 99,65% present
    - \* 78,92% have `serverAuth` + `clientAuth`
    - \* 20,81% have `serverAuth` only

- \* 0,34% missing
- \* 99,65% marked not critical
- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.7.10:** Must include `serverAuth`.
  - \* `clientAuth` is optional.
  - \* Must not be marked as critical.
- **Subject Key Identifier (SKI)**
  - **Dataset Results:**
    - \* 99,81% present
    - \* 0,19% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.4:** Not Recommended.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 80% present
    - \* 19,73% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.1:** Must be present.
    - \* Must contain the `subjectKeyIdentifier` of the Issuing CA.
- **Subject Alternative Name (SAN)**
  - **Dataset Results:**
    - \* 98,20% correct match (DNS names)
    - \* 1,80% not compliant
    - \* 0,05% empty
    - \* With Non Empty Subject:
      - 99,84% marked not critical
      - 0,0001% marked critical
      - 0,15% missing
      - 0,0023% error
    - \* With Empty Subject:
      - 94,72% marked critical
      - 5,28% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.12:** Must contain all DNS names for the certificate.
    - \* Must be marked as critical if the SUBJECT field is an empty SEQUENCE.
    - \* Otherwise, must not be marked as critical.

- **Certificate Policies**

- **Dataset Results:**

- \* 99,27% present (4.963.598 certificates)
    - \* 87,45% of DV certificates (4.169.667 certificates) have only their own policy
    - \* 99,99% of DV certificates (4.767.624 certificates) include their own policy with other policies
    - \* 0,004% of DV certificates (198 certificates) are missing their own policy
    - \* 0,002% of DV certificates (98 certificates) have no policy
    - \* 0,16% of EV certificates (23 certificates) have only their own policy
    - \* 99,77% of EV certificates (14.203 certificates) include their own policy with other policies
    - \* 0,21% of EV certificates (30 certificates) are missing their own policy
    - \* 0,02% of EV certificates (3 certificates) have no policy
    - \* 52,90% of OV certificates (95.747 certificates) have only their own policy
    - \* 99,95% of OV certificates (180.921 certificates) include their own policy with other policies
    - \* 0,05% of OV certificates (89 certificates) are missing their own policy
    - \* 0,001% of OV certificates (2 certificates) have no CP extension
    - \* 100% marked not critical

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.7.9:** Must appear.
    - \* Must not be marked as critical.
    - \* Must contain at least one policy information (Reserved Certificate Policy Identifiers).

- **Basic Constraints**

- **Dataset Results:**

- \* 99,46% have cA = FALSE
    - \* 0,085% have cA = TRUE
    - \* 0,45% missing

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.7.8:** Optional.
    - \* If present, must have cA = FALSE.

- **Authority Information Access (AIA)**

- **Dataset Results:**

- \* 35,78% have CRL + OCSP



- \* 63,49% have only OCSP
- \* 0,003% have only CRL
- \* 0,72% missing
- \* 99,3% marked not critical
- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.7.7:** Must include the HTTP URL of the issuing CA's OCSP responder.
  - \* Must not be marked as critical.
- **CRL Distribution Points**
  - **Dataset Results:**
    - \* 35,9% present
    - \* 64,1% not present
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.2:** Optional.
    - \* If present, must not be marked as critical and must include an HTTP URL of the CA's CRL.
- **OCSP Checking**
  - **Dataset Results:**
    - \* 99,8% of certificates return an OCSP response status of **Good** (4.990.263 certificates).
    - \* 0,1% of certificates are reported as **Revoked**.
    - \* 0,1% of certificates generate errors in OCSP requests.
  - **CA/B Forum Baseline Requirements:**
    - \* Certificates must include up-to-date revocation status information via OCSP (or CRL).
    - \* The OCSP responses should reliably confirm the current validity of the certificate.
- **OCSP Stapling**
  - **Dataset Results:**
    - \* 47,93% enabled
    - \* 52,07% not enabled
  - **CA/B Forum Baseline Requirements:**
    - \* Optional for Baseline Requirements.
    - \* Recommended by some browsers to improve privacy and reduce latency, but not strictly required.
- **OCSP Must-Staple**
  - **Dataset Results:**
    - \* 0,08% enabled

- \* 99,92% not enabled
- **CA/B Forum Baseline Requirements:**
  - \* Optional for Baseline Requirements.
  - \* Recommended by some browsers to improve privacy and reduce latency, but not strictly required.
- **Signed Certificate Timestamps (SCTs)**
  - **Dataset Results:**
    - \* 70,02% include 2 SCTs
    - \* 27,85% include 3 SCTs
    - \* 1,77% with no SCT
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.3:** Optional.
    - \* Required by Chrome for most publicly trusted TLS certs issued after certain dates.
    - \* Strongly encouraged for Certificate Transparency.
- **Public Key Distribution**
  - **Dataset Results:**
    - \* RSA is used in 73,23% of leaf certificates, while ECDSA is used in 27,77%.
    - \* For ECDSA: 93,56% of certificates use 256-bit keys (curve P-256), 6,43% use 384-bit keys, and 0,01% use 521-bit keys.
    - \* For RSA: 88,15% of certificates employ 2048-bit keys, 11,07% use 4096-bit keys, and 0,72% use 3072-bit keys.
  - **CA/B Forum Baseline Requirements:**
    - \* For RSA, a minimum key length of 2048 bits is required.
    - \* For ECDSA, the recommended standard is the use of the P-256 curve.
- **Validity of Certificate Signatures**
  - **Dataset Results:**
    - \* 99,29% of leaf certificates have a valid signature.
    - \* 0,71% of certificates present an invalid signature, with 0,01% due to issuer configuration errors and 0,7% attributable to self-signed certificates.
  - **CA/B Forum Baseline Requirements:**
    - \* Every certificate must have a verifiable and valid signature that adheres to the CA/Browser Forum guidelines.
- **Self-Signed vs. CA-Signed Certificates**
  - **Dataset Results:**

- \* 99,3% of leaf certificates are signed by a recognized Certification Authority (CA).
- \* 0,7% of certificates are self-signed.
- **CA/B Forum Baseline Requirements:**
  - \* Certificates should be issued and signed by a trusted CA to ensure a verifiable chain of trust.

### 5.13.2 Comparison of Intermediate CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.6.1)

- **Basic Constraints**
  - **Dataset Results:**
    - \* 100% have `cA = TRUE`
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.4:** Must be present.
    - \* Must be set `cA=TRUE`.
- **Key Usage**
  - **Dataset Results:**
    - \* 58,16% have `digitalSignature`, `keyCertSign`, `cRLSign`
    - \* 38,71% have `keyCertSign`, `cRLSign` (no `digitalSignature`)
    - \* 2,11% missing
    - \* 89,42% marked critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.7:** Must appear.
    - \* Must be marked critical.
    - \* Must include `keyCertSign` and `cRLSign`.
    - \* Not required `digitalSignature`.
- **Extended Key Usage (EKU)**
  - **Dataset Results:**
    - \* 52,79% have `serverAuth` + `clientAuth`
    - \* 1,59% have `serverAuth`
    - \* 42,13% missing
    - \* 57,73% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.6:** Must appear.
    - \* Must include `serverAuth`.
    - \* `clientAuth` is optional.

- \* Must not be marked critical.
- **Subject Key Identifier (SKI)**
  - **Dataset Results:**
    - \* 99,98% present
    - \* 0,02 missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.4:** Must be present.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 99,92% present
    - \* 0,08 missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.1:** Must be present.
    - \* Must contains the `subjectKeyIdentifier` of the issuing CA.
- **Certificate Policies**
  - **Dataset Results:**
    - \* 84,19% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.5:** Must appear.
    - \* Must not be marked critical.
- **Authority Information Access (AIA)**
  - **Dataset Results:**
    - \* 89,77% have CRL + OCSP
    - \* 0,2% have only OCSP
    - \* 6,28% have only CRL
    - \* 5,78% missing
    - \* 88,11% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.3:** Should be present.
    - \* Must not be marked critical.
- **CRL Distribution Points**
  - **Dataset Results:**
    - \* 94,09% marked not critical
    - \* 5,91% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.2:** Must appear.

- \* Must not be marked critical.

### 5.13.3 Comparison of Root CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.1.2)

- **Basic Constraints**

- **Dataset Results:**

- \* 100% present with cA=TRUE

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.1.4:** Must appear.
    - \* Must be set cA=TRUE.

- **Key Usage**

- **Dataset Results:**

- \* 8,21% have certificate\_sign and crl\_sign
    - \* 5,34% have certificate\_sign, crl\_sign and digital\_signature
    - \* 45,67% have certificate\_sign
    - \* 30,70% have digital\_signature, key\_encipherment and certificate\_sign
    - \* 9,02 missing
    - \* 86,04% marked critical
    - \* 4,86% marked not critical

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.10.7:** Must appear.
    - \* Must be marked critical.
    - \* Must include keyCertSign and cRLSign.
    - \* Not required digitalSignature.

- **Extended Key Usage (EKU)**

- **Dataset Results:**

- \* 0,81% have data
    - \* 99,19% missing
    - \* 0,61% marked not critical
    - \* 0,05% marked critical

- **CA/B Forum Baseline Requirements:**

- \* Must Not be present.

- **Subject Key Identifier (SKI)**

- **Dataset Results:**

- \* 98,95% present
    - \* 1,05% missing

- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.11.4:** Must be present.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 15,76% present
    - \* 84,24% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.1.3:** Recommended.
- **Certificate Policies**
  - **Dataset Results:**
    - \* 98,68% missing
    - \* 1,32% present
    - \* 1,32% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.5:** Not Recommended.
    - \* Must not be marked critical.

## Chapter 6

# Digital Certificate Analysis Using the Google CrUX Dataset

This chapter aims to conduct a quick and focused analysis on the results obtained from a new dataset of domains extracted using a query on Google's big data system [29], querying the Chrome User Experience Report (CrUX) [35] available through Google Cloud (table `chrome-ux-report.all.202409`). Specifically, an SQL query was run on September domains, selecting the top 10 million most popular European domains, sorted by the `experimental.popularity.rank` field.

```

-- Query for Italian domains
SELECT DISTINCT(origin), experimental.popularity.rank
FROM `chrome-ux-report.country_it.202409`
WHERE origin LIKE '%it%'
ORDER BY experimental.popularity.rank;

-- Query for European domains
SELECT origin, experimental.popularity.rank
FROM `chrome-ux-report.all.202409`
WHERE origin LIKE '%it%' -- Italy
OR origin LIKE '%fr%' -- France
OR origin LIKE '%de%' -- Germany
OR origin LIKE '%es%' -- Spain
OR origin LIKE '%uk%' -- United Kingdom
OR origin LIKE '%nl%' -- Netherlands
OR origin LIKE '%se%' -- Sweden
OR origin LIKE '%pl%' -- Poland
OR origin LIKE '%be%' -- Belgium
OR origin LIKE '%gr%' -- Grece
OR origin LIKE '%cz%' -- Czech Republic
OR origin LIKE '%pt%' -- Portugal
OR origin LIKE '%fi%' -- Finland
OR origin LIKE '%ro%' -- Romania
OR origin LIKE '%dk%' -- Denmak
OR origin LIKE '%ie%' -- Ireland
OR origin LIKE '%at%' -- Austria
OR origin LIKE '%hu%' -- Hungary
OR origin LIKE '%sk%' -- Slovakia
OR origin LIKE '%si%' -- Slovenia
OR origin LIKE '%hr%' -- Croatia
OR origin LIKE '%ee%' -- Estonia
OR origin LIKE '%lv%' -- Latvia
OR origin LIKE '%lt%' -- Lithuania
OR origin LIKE '%mt%' -- Malta
OR origin LIKE '%cy%' -- Cipro
OR origin LIKE '%is%' -- Iceland
OR origin LIKE '%no%' -- Norway
OR origin LIKE '%ch%' -- Switzerland
OR origin LIKE '%al%' -- Albania
OR origin LIKE '%ba%' -- Bosnia and Herzegovina
OR origin LIKE '%mk%' -- North Macedonia
OR origin LIKE '%rs%' -- Serbia
OR origin LIKE '%xk%' -- Kosovo
ORDER BY experimental.popularity.rank ASC
LIMIT 10000000;

```

Figure 6.1. Google BigQuery query for extracting European domains based on popularity ranking.

The query focused primarily on the major European countries and the good dataset extracted from it, which represented a good representation of major domains of the continent (Figure 6.1). This separate query could focus on one country, such as Italy, and explore regional dynamics which can vary considerably.

The result of the query was downloaded from Google Cloud in a .txt file containing all selected domains. Subsequently, at the end of October 2024 that list was used as input for ZGrab2, the tool used to download the associated digital certificates, generating a JSON file of 132.1 GB in size, almost twice the size of the DomCop dataset.

```

C:\Users\franc\Desktop\zgrab2>zgrab2 tls --port 443 -f C:/Users/franc/Desktop/Domini_Elio/domini_europei_google.txt --ou
tput-file=certs_europei.json --timeout=30
time="2024-10-28T09:34:20+01:00" level=info msg="started grab at 2024-10-28T09:34:20+01:00"
time="2024-10-28T14:30:49+01:00" level=info msg="finished grab at 2024-10-28T14:30:49+01:00"
{"statuses":{"tls":{"successes":9440561,"failures":559440},"start":"2024-10-28T09:34:20+01:00","end":"2024-10-28T14:30:
49+01:00","duration":"4h56m28.9472486s"}

```

Figure 6.2. ZGrab2 scan results for domains in the Google dataset.



A timeout of 30 seconds was set for the ZGrab2 operation, as displayed in Figure 6.2, allowing the download of certificates to complete in 4 hours and 56 minutes. This significant reduction in time, compared to the previous DomCop dataset in which the majority of domains were U.S. based, is likely due to the lower latency and greater geographic proximity of the European servers.

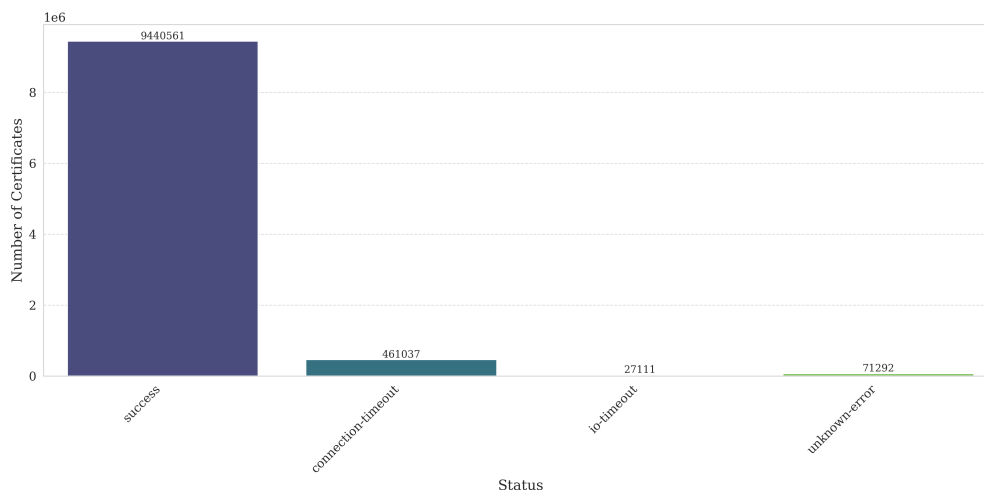


Figure 6.3. Status analysis of certificate extraction from European domains.

The process returned 9,440,561 valid X.509 leaf certificates, or 94.41% of the total, while 559,440 certificates (5.59%) produced connection errors or other anomalies. In this context, the number of unique leaf certificates, deduced from the presence of multiple Subject Alternative Names (SANs) per certificate, turns out to be **3,164,063**, showing a reduction of 36.72% compared to the total number of leaf certificates in the DomCop dataset. These results could point to substantial differences in the certificate issuance landscape between the two datasets and suggest possible variations due to the different geographic composition of the analyzed domains.

## 6.1 Overview of Leaf Certificates

Although having analyzed European domains they reported the same Issuer entities, with a slightly different percentage, as the DomCop dataset, so this strongly affects the analysis by anticipating that there will not be much difference from the analysis done in the DomCop dataset as the entities are still the same, as can be seen in Figure 6.4, where 85% of the entities turn out to be American followed by a 6.2% English and a 3.4% Belgian. Again, Let's Encrypt is seen in the vast majority with a 45.02%, followed by a 22.81% from Google Trust Services and a 7.16% from DigiCert Inc. There is not a big difference with the DomCop dataset in this respect.

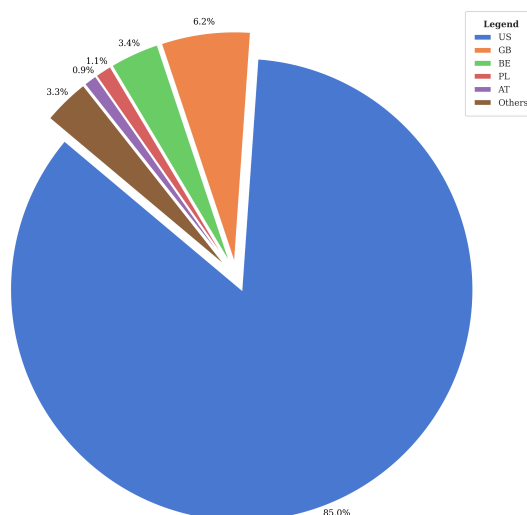


Figure 6.4. Distribution of certificates by issuer country in the dataset, showing a dominance of U.S. entities, with Let’s Encrypt leading the distribution.

Again, all the certificates analyzed were certificates belonging to X.509 version 3. Again, the distribution of validation level reflects what was expected seeing a large majority (94.11%) of certificates on DV followed by 0.61% from OV and the remainder from EV.

### 6.1.1 Validity Duration Distribution and Expiry Trends

In the analysis of the distribution of the duration of validity of certificates, it is observed that the dataset under consideration shows interesting differences according to the date of issuance. In particular, considering the start date, as many as 99.96% of the certificates are valid.

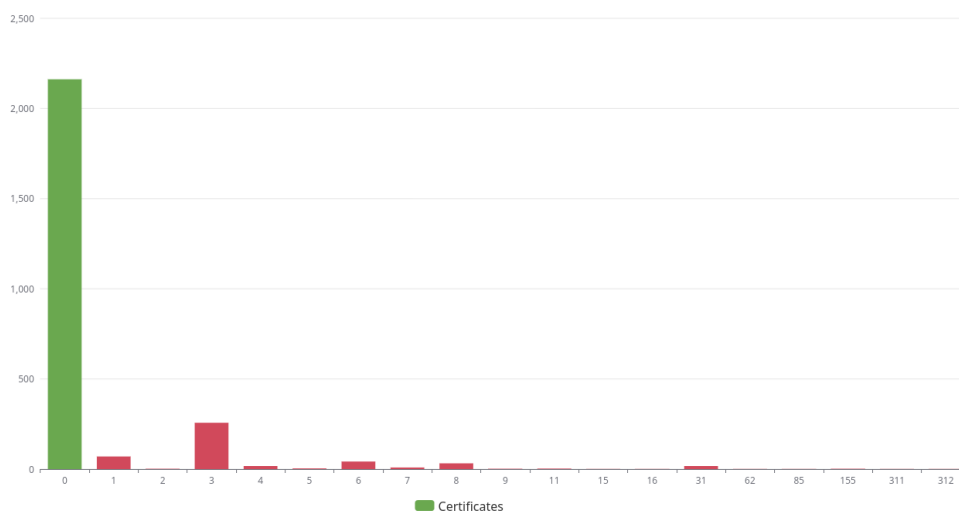


Figure 6.5. Validity duration distribution of certificates issued before September 2020, with a portion exceeding the 39-month validity period.

However, when analyzing the period between April 2015 and September 1, 2020, it shows that only 82.36% of the certificates meet the 39-month limit, while the remaining 17.64% exceed this threshold, as shown in Figure 6.5.

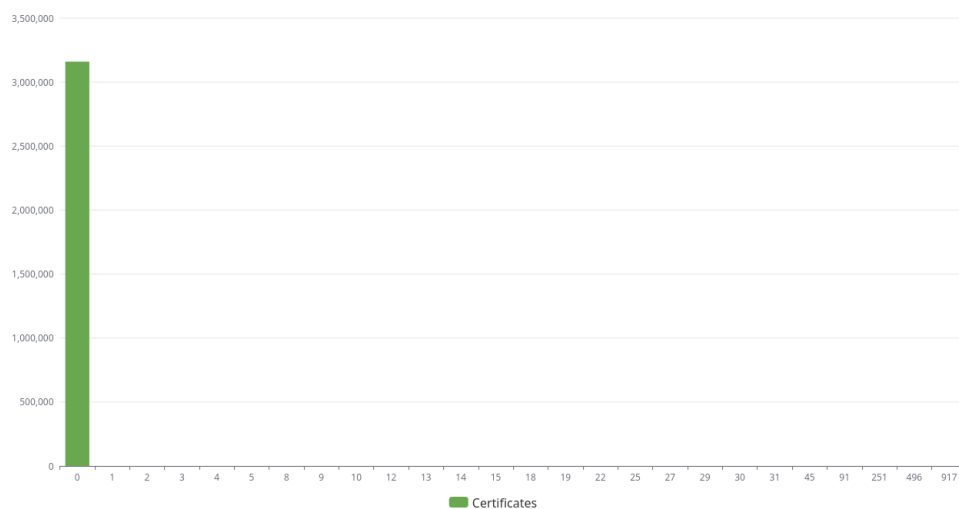


Figure 6.6. Validity duration distribution of certificates issued after September 2020, with almost all certificates adhering to the 398-day validity limit.

The scenario practically turns on its head for certificates issued on or after September 1, 2020, in Figure 6.6. In this regard, an almost totality of certificates, 99.97%, is issued with a validity of not more than 398 days without fault of current guidelines. A minuscule fraction, 0.03%, shows excessive validity over that limit. Another important point to be made is that certificates provided prior to April 2015 did not come under precisely specified restrictions, meaning that they are,

therefore, the remaining fraction of 0.02% in the dataset, thus falling out of context in the direct comparison with current regulations.

A comparison with the data provided by the DomCop dataset reveals some similarities and significant differences. For certificates issued after September 1, 2020, the DomCop dataset reports that 99.43% comply with the 398-day limit, while 0.07% of certificates have a higher validity. Similarly, for certificates issued between April 2015 and September 1, 2020, DomCop shows that 86.60% of the certificates comply with the 39-month limit, while 10.40% of the certificates exceed the allowed duration. These data, although somewhat discordant in terms of the percentages found in the Google dataset, underscore a high degree of compliance with regulatory guidelines, especially for more recent releases.

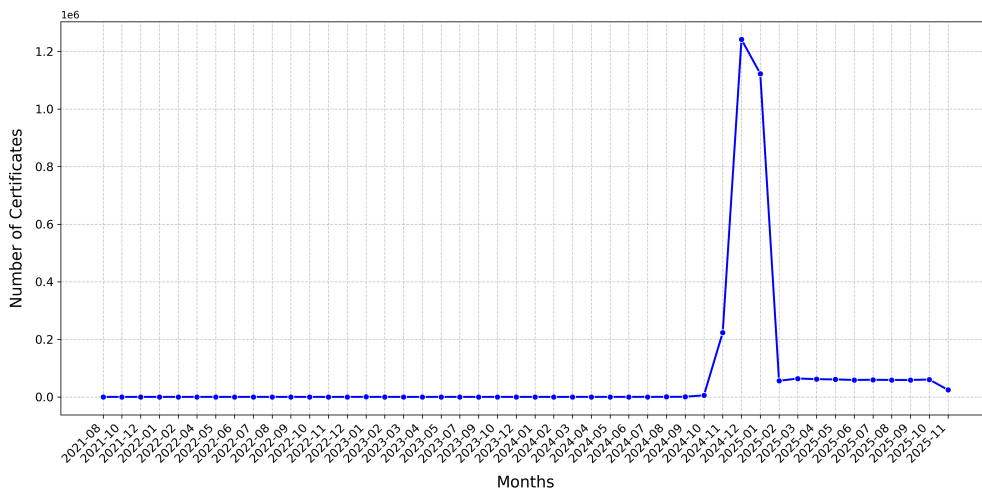


Figure 6.7. Expiration trend of certificates, with significant peaks observed in December 2024 and January 2025, suggesting a cascading issuance schedule.

The trend of maturity dates, illustrated in Figure 6.7, offers further food for thought: certificate maturities are more concentrated in the period between October 2024 and November 2025, with particularly pronounced peaks in December 2024 (39.21%) and January 2025 (35.47%). This temporal aggregation suggests an almost a **cascade** scheduling of issuances, probably related to the maturity of many certificates issued in earlier periods.

### 6.1.2 Leaf Certificate Serial Number Analysis

Overall, as illustrated in Figure 6.8, the analysis of the Serial Number field shows strong compliance with the technical specifications imposed by the RFC 5280 standard [6] and the CA/Browser Forum guidelines [4] in both the Google and DomCop datasets. These data show that although both datasets are largely compliant with the recommendations, some anomalies remain. In the Google dataset, almost 100% of the certificates have a valid serial number, with no cases where the length exceeds 20 bytes. However, 10 certificates with negative serial numbers and 133 duplicate cases were detected, of which 64 certificates have the value of 0.

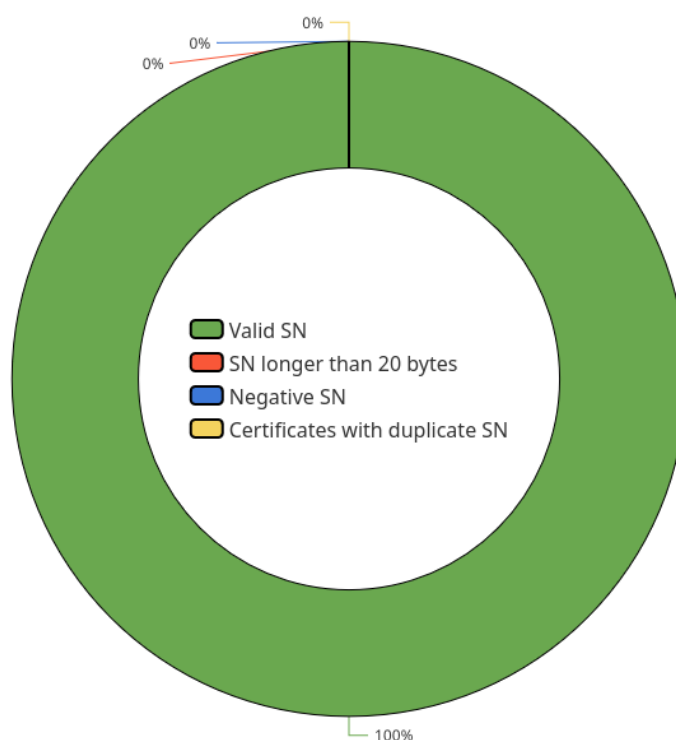


Figure 6.8. Analysis of Serial Number field, highlighting compliance and anomalies such as negative serial numbers and duplicates.

In comparison, analysis of the DomCop dataset shows slightly less compliance, with 99.98% of the certificates meeting the requirements, while 34 certificates have negative serial numbers and 989 certificates have duplicate serial numbers, with 23.66% of these having the value of 0.

## 6.2 Compliance Analysis of Google CrUX Certificates Against CA/Browser Forum Requirements

In this section we report, in a summarized manner and focusing on the main points, the results of the analysis performed on the dataset with the requirements set by the CA/Browser Forum.

### 6.2.1 Comparison of Leaf TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.7.6)

- **Serial Number**
  - **Dataset Results:**
    - \* 100% present

- \* 100% valid serial number
- \* 0 certificates with serial number longer than 20 bytes
- \* 10 certificates with negative serial number
- \* 133 certificates with duplicate serial number
- \* 64 certificates with serial number equal to 0
- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.7:** Must appear.
  - \* Must be positive and greater than 0.
  - \* Must not be longer than 20 bytes.
- **Validity Duration**
  - **Dataset Results:**
    - \* 99,96% have a valid validity
    - \* 0,04% do not meet the requirements
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 6.3.2:** For certificates issued between April 2015 and September 1, 2020, the validity must not exceed 39 months, while certificates issued after September 1, 2020, must not have a validity longer than 398 days.
- **Key Usage**
  - **Dataset Results:**
    - \* 100% extension present
    - \* 99,87% marked critical
    - \* 0,014% marked not critical
    - \* For RSA (68,49%):
      - 99,80% have `digitalSignature` and `keyEncipherment`
      - 0,17% empty
      - 0,02% others
    - \* For ECDSA (31,51%):
      - 97,11% have `digitalSignature`
      - 2,88% have `digitalSignature` and `keyAgreement`
      - 0,006% others
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.11:** Optional.
    - \* For RSA, if present, SHOULD include `digitalSignature`, MAY include `keyEncipherment` and NOT RECOMMENDED include `dataEncipherment`.
    - \* For ECC, if present, MUST include `digitalSignature` and NOT RECOMMENDED include `keyAgreement`.
    - \* Must be marked as critical.

- **Extended Key Usage (EKU)**
  - **Dataset Results:**
    - \* 99,92% present
    - \* 76,20% have `serverAuth` + `clientAuth`
    - \* 23,78% have `serverAuth` only
    - \* 0,08% missing
    - \* 99,92% marked not critical
    - \* 0,0006% marked critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.10:** Must include `serverAuth`.
    - \* `clientAuth` is optional.
    - \* Must not be marked as critical.
- **Subject Key Identifier (SKI)**
  - **Dataset Results:**
    - \* 99,70% present
    - \* 0,30% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.4:** Not Recommended.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 99,90% present
    - \* 0,10% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.1:** Must be present.
    - \* Must contain the `subjectKeyIdentifier` of the Issuing CA.
- **Subject Alternative Name (SAN)**
  - **Dataset Results:**
    - \* 99,57% correct match (DNS names)
    - \* 0,43% not compliant
    - \* 0,03% empty
    - \* With Non Empty Subject:
      - 99,94% marked not critical
      - 0,0002% marked critical
      - 0,06% missing
      - 0,0004% error
    - \* With Empty Subject:
      - 99,73% marked critical

- 0,27% missing
- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.7.12:** Must contain all DNS names for the certificate.
  - \* Must be marked as critical if the SUBJECT field is an empty SEQUENCE.
  - \* Otherwise, must not be marked as critical.
- **Certificate Policies**
  - **Dataset Results:**
    - \* 99,86% present (3.159.684 certificates)
    - \* 87,12% of DV certificates (2.594.326 certificates) have only their own policy
    - \* 99,99% of DV certificates (2.977.814 certificates) include their own policy with other policies
    - \* 0,001% of DV certificates (25 certificates) are missing their own policy
    - \* 0,0003% of DV certificates (10 certificates) have no policy
    - \* 0,09% of EV certificates (17 certificates) have only their own policy
    - \* 99,98% of EV certificates (19.604 certificates) include their own policy with other policies
    - \* 0,01% of EV certificates (2 certificates) are missing their own policy
    - \* 0,005% of EV certificates (1 certificates) have no policy
    - \* 55,11% of OV certificates (89.367 certificates) have only their own policy
    - \* 99,99% of OV certificates (162.156 certificates) include their own policy with other policies
    - \* 0,002% of OV certificates (4 certificates) are missing their own policy
    - \* 0% of OV certificates (0 certificates) have no CP extension
    - \* 100% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.9:** Must appear.
    - \* Must not be marked as critical.
    - \* Must contain at least one policy information (Reserved Certificate Policy Identifiers).
- **Basic Constraints**
  - **Dataset Results:**
    - \* 99,39% have `cA = FALSE`
    - \* 0,05% have `cA = TRUE`
    - \* 0,56% missing



- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.7.8:** Optional.
  - \* If present, must have `cA = FALSE`.
- **Authority Information Access (AIA)**
  - **Dataset Results:**
    - \* 40,78% have CRL + OCSP
    - \* 59,08% have only OCSP
    - \* 0,001% have only CRL
    - \* 0,13% missing
    - \* 99,86% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.7.7:** Must include the HTTP URL of the issuing CA's OCSP responder.
    - \* Must not be marked as critical.
- **CRL Distribution Points**
  - **Dataset Results:**
    - \* 40,78% present
    - \* 59,22% not present
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.2:** Optional.
    - \* If present, must not be marked as critical and must include an HTTP URL of the CA's CRL.
- **OCSP Checking**
  - **Dataset Results:**
    - \* 99,9% of certificates return an OCSP response status of **Good** (3.160.899 certificates).
    - \* 0,04% of certificates are reported as **Revoked**.
    - \* 0,06% of certificates generate errors in OCSP requests.
  - **CA/B Forum Baseline Requirements:**
    - \* Certificates must include up-to-date revocation status information via OCSP (or CRL).
    - \* The OCSP responses should reliably confirm the current validity of the certificate.
- **OCSP Stapling**
  - **Dataset Results:**
    - \* 50,01% enabled
    - \* 49,97% not enabled

- **CA/B Forum Baseline Requirements:**
  - \* Optional for Baseline Requirements.
  - \* Recommended by some browsers to improve privacy and reduce latency, but not strictly required.
- **OCSF Must-Staple**
  - **Dataset Results:**
    - \* 0,09% enabled
    - \* 99,91% not enabled
  - **CA/B Forum Baseline Requirements:**
    - \* Optional for Baseline Requirements.
    - \* Recommended by some browsers to improve privacy and reduce latency, but not strictly required.
- **Signed Certificate Timestamps (SCTs)**
  - **Dataset Results:**
    - \* 69,92% include 2 SCTs
    - \* 29,14% include 3 SCTs
    - \* 0,33% with no SCT
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.3:** Optional.
    - \* Required by Chrome for most publicly trusted TLS certs issued after certain dates.
    - \* Strongly encouraged for Certificate Transparency.
- **Public Key Distribution**
  - **Dataset Results:**
    - \* RSA is used in 68,62% of leaf certificates, while ECDSA is used in 31,38%.
    - \* For ECDSA: 93,13% of certificates use 256-bit keys (curve P-256) and 6,87% use 384-bit keys.
    - \* For RSA: 85,52% of certificates employ 2048-bit keys, 12,97% use 4096-bit keys, and 1,48% use 3072-bit keys.
  - **CA/B Forum Baseline Requirements:**
    - \* For RSA, a minimum key length of 2048 bits is required.
    - \* For ECDSA, the recommended standard is the use of the P-256 curve.
- **Validity of Certificate Signatures**
  - **Dataset Results:**
    - \* 99,86% of leaf certificates have a valid signature.
    - \* 0,14% of certificates present an invalid signature, with 0,02% due to issuer configuration errors and 0,12% attributable to self-signed

certificates.

– **CA/B Forum Baseline Requirements:**

- \* Every certificate must have a verifiable and valid signature that adheres to the CA/Browser Forum guidelines.

• **Self-Signed vs. CA-Signed Certificates**

– **Dataset Results:**

- \* 99,88% of leaf certificates are signed by a recognized Certification Authority (CA).
- \* 0,12% of certificates are self-signed.

– **CA/B Forum Baseline Requirements:**

- \* Certificates should be issued and signed by a trusted CA to ensure a verifiable chain of trust.

## 6.2.2 Comparison of Intermediate CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.6.1)

• **Basic Constraints**

– **Dataset Results:**

- \* 99,85% have `cA = TRUE`
- \* 0,15% have `cA = FALSE`

– **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.10.4:** Must be present.
- \* Must be set `cA=TRUE`.

• **Key Usage**

– **Dataset Results:**

- \* 62,17% have `digitalSignature`, `keyCertSign`, `cRLSign`
- \* 34,64% have `keyCertSign`, `cRLSign` (no `digitalSignature`)
- \* 2,17% missing
- \* 83,62% marked critical
- \* 4,06% marked not critical

– **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.10.7:** Must appear.
- \* Must be marked critical.
- \* Must include `keyCertSign` and `cRLSign`.
- \* Not required `digitalSignature`.

• **Extended Key Usage (EKU)**

– **Dataset Results:**

- \* 63,56% have `serverAuth` + `clientAuth`

- \* 1,43% have `serverAuth`
- \* 33,72% missing
- \* 66,28% marked not critical
- **CA/B Forum Baseline Requirements:**
  - \* **BR 7.1.2.10.6:** Must appear.
  - \* Must include `serverAuth`.
  - \* `clientAuth` is optional.
  - \* Must not be marked critical.
- **Subject Key Identifier (SKI)**
  - **Dataset Results:**
    - \* 99,59% present
    - \* 0,41 missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.4:** Must be present.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 97,97% present
    - \* 2,03 missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.1:** Must be present.
    - \* Must contains the `subjectKeyIdentifier` of the issuing CA.
- **Certificate Policies**
  - **Dataset Results:**
    - \* 90,14% marked not critical
    - \* 9,86 missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.5:** Must appear.
    - \* Must not be marked critical.
- **Authority Information Access (AIA)**
  - **Dataset Results:**
    - \* 91,59% have CRL + OCSP
    - \* 4,49% have only CRL
    - \* 3,91% missing
    - \* 91,59% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.3:** Should be present.

- \* Must not be marked critical.

- **CRL Distribution Points**

- **Dataset Results:**

- \* 99,99% marked not critical

- \* 0,001% missing

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.11.2:** Must appear.

- \* Must not be marked critical.

### 6.2.3 Comparison of Root CA TLS Certificates with CA/B Forum Baseline Requirements (BR 7.1.2.1.2)

- **Basic Constraints**

- **Dataset Results:**

- \* 99,71% present with `cA=TRUE`

- \* 0,28% present with `cA=FALSE`

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.1.4:** Must appear.

- \* Must be set `cA=TRUE`.

- **Key Usage**

- **Dataset Results:**

- \* 37,60% have `digital_signature`, `key_encipherment` and `certificate_sign`

- \* 26,46% have `certificate_sign` and `crl_sign`

- \* 12,53% have `certificate_sign`, `crl_sign` and `digital_signature`

- \* 0,27% have `certificate_sign`

- \* 21,45 missing

- \* 69,36% marked critical

- \* 9,19% marked not critical

- **CA/B Forum Baseline Requirements:**

- \* **BR 7.1.2.10.7:** Must appear.

- \* Must be marked critical.

- \* Must include `keyCertSign` and `cRLSign`.

- \* Not required `digitalSignature`.

- **Extended Key Usage (EKU)**

- **Dataset Results:**

- \* 1,41% have data

- \* 98,59% missing

- \* 0,85% marked not critical
- \* 0,28% marked critical
- **CA/B Forum Baseline Requirements:**
  - \* Must Not be present.
- **Subject Key Identifier (SKI)**
  - **Dataset Results:**
    - \* 81,84% present
    - \* 18,16% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.11.4:** Must be present.
- **Authority Key Identifier (AKI)**
  - **Dataset Results:**
    - \* 29,10% present
    - \* 70,90% missing
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.1.3:** Recommended.
- **Certificate Policies**
  - **Dataset Results:**
    - \* 95,73% missing
    - \* 4,27% present
    - \* 4,27% marked not critical
  - **CA/B Forum Baseline Requirements:**
    - \* **BR 7.1.2.10.5:** Not Recommended.
    - \* Must not be marked critical.

## 6.3 Comparison Analysis Between the DomCop and Google Datasets

Comparison between the DomCop dataset and the Google dataset highlights some differences in the configuration and quality of certificates, reflecting different operational practices and regulations in global and European markets. In the DomCop dataset, covering a global landscape, some anomalies emerge in critical fields, such as in authority information access. In contrast, the Google dataset focused on European domains, shows near-perfect compliance with the requirements of the CA/Browser Forum, at equal percentages. These differences raise important questions in terms of regulation and quality control. The greater uniformity and compliance of European certificates suggests that stringent regulations and strict controls can lead to a more secure and reliable PKI ecosystem. However, the

greater heterogeneity of the global landscape highlights opportunities for improvement in certificate issuance and management processes.

# Chapter 7

## Conclusions and Future Directions

In the course of the study, the functioning of certificate validation and revocation processes was analysed in detail, highlighting how transparency in management, strengthened by the innovations introduced by Certificate Transparency version 2.0, can significantly reduce the risk of fraudulent issuance and contribute to strengthening trust in digital environments.

The implementation of DigitalCertiAnalytics represents the central contribution of this work: a tool capable of collecting, verifying and analysing millions of certificates in a scalable manner, capable of operating on datasets from any type of context. The application of such a tool has made it possible to highlight significant differences in the configuration of certificates, as well as the challenges of managing large volumes of data and integrating heterogeneous modules. Although the adoption of open-source tools such as ZGrab2 and SSLyze made an in-depth analysis possible, it became apparent that further work was needed to ensure greater compatibility and upgradeability of components, as well as to optimise performance in real-world scenarios.

The analysis conducted also highlighted data quality and anomaly management issues, which require special attention to ensure the robustness of the entire certification infrastructure. The difficulties encountered in analysing the datasets, along with privacy and performance issues, suggest that further refinement of the tools and methods employed may be a viable avenue for future research. With this in mind, the adoption of parallelisation techniques and the exploration of microservices-based architectures appear as possible solutions to improve scalability and system efficiency.

Looking further, the development of machine learning algorithms for the automatic recognition of anomalous patterns and the detection of suspicious activities offers interesting prospects for an even deeper analysis of digital certificates. One relevant aspect concerns the current data storage management, which is based on the use of SQLite3. Although this solution is lightweight and easy to manage, it has obvious limitations in its ability to handle large volumes of data and in



the efficiency of write operations, especially in the presence of large JSON files. The adoption of an alternative database, capable of handling large amounts of information more efficiently, could significantly improve the organisation and scalability of the entire system. Furthermore, the analysis revealed the presence of duplicates, in particular between intermediate and root certificates, with multiple instances having the same subject and issuer, a situation that leads to unnecessary memory consumption and does not add value to the overall analysis. A more sophisticated deduplication system would be useful to reduce this redundancy and optimise overall efficiency.

Other concerns that came up are of operational duration, such as that of OCSP status checking and SSLyze results analysis, which can be time-consuming in both completions and storing into the database. Integration, in this case, of a much higher performance of the database can prove to value the validation process and speedy up the entire data storage process. At the same time, high-volume data indicated the OCSP request form for optimisation due to delays in updating the database, shifting the system temporarily to CSVs for results storage-an operation that needs to occur through much manual intervention. An enhancement in that sector could tendering in a very much reduced processing time and would ensure more trustworthy information regarding the certificates status.

Finally, a further potential development concerns the integration of a web user interface. Such an implementation would make certificate analysis more accessible, allowing users to query the system from any device and obtain a near real-time overview of the status of certificates on the web. In addition to improving the user experience, such a solution would offer significant advantages in terms of remote monitoring and management, although it would entail a thorough overhaul of the update processes.

# Bibliography

- [1] Casey Crane, “Pki architecture: Fundamentals of designing a private pki system.” <https://www.thesslstore.com/blog/pki-architecture-fundamentals-of-designing-a-private-pki-system/> Accessed: 2024/11/01
- [2] B. Laurie, “Certificate transparency”, *Commun. ACM*, vol. 57, September 2014, p. 40–46, DOI 10.1145/2659897
- [3] Alin Tomescu , “What is a merkle tree?.” <https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/> Accessed: 2025/02/23
- [4] CA/Browser Forum, “Ca/browser forum.” <https://cabforum.org/working-groups/server/baseline-requirements/documents/CA-Browser-Forum-TLS-BR-2.1.3.pdf> Accessed: 2025/03/05
- [5] Mozilla, “Ca.” <https://wiki.mozilla.org/CA>, n.d., Accessed: 2025/02/09.
- [6] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.” RFC 5280, May 2008, DOI 10.17487/RFC5280
- [7] The ZMap Project, “Zlint.” <https://github.com/zmap/zgrab2> Accessed: 2024/12/12
- [8] The ZMap Project, “Zlint.” <https://github.com/zmap/zlint> Accessed: 2024/12/12
- [9] Alban Diquet, “Sslyze.” <https://github.com/nabla-c0d3/sslyze> Accessed: 2024/12/15
- [10] SSL Support Team, “What is public key infrastructure (pki)?.” <https://www.ssl.com/article/what-is-public-key-infrastructure-pki/> Accessed: 2024/11/01
- [11] D. G. Berbecaru and A. Lioy, “An evaluation of x.509 certificate revocation and related privacy issues in the web pki ecosystem”, *IEEE Access*, vol. 11, 2023, pp. 79156–79175, DOI 10.1109/ACCESS.2023.3299357
- [12] B. Schneier, “Applied cryptography: Protocols, algorithms, and source code in c”, John Wiley & Sons, 2015, ISBN: 9781119183471
- [13] S. William, “Cryptography and network security - principles and practice, 7th edition”, Pearson Education India, 2018, ISBN: 9789353942564
- [14] C. Adams and S. Lloyd, “Understanding pki: Concepts, standards, and deployment considerations”, Addison-Wesley, 2003, ISBN: 9780672323911
- [15] R. Perlman, “An overview of pki trust models”, *IEEE Netw.*, vol. 13, 1999, pp. 38–43, DOI 10.1109/65.806987

- [16] H. Liping and S. Lei, “Research on trust model of pki”, 2011 Fourth International Conference on Intelligent Computation Technology and Automation, vol. 1, 2011, pp. 232–235, DOI 10.1109/ICICTA.2011.67
- [17] Q. hai Bai, Y. Zheng, L. Zhao, H. Chun, and C. Cheng, “Research on mechanism of pki trust model”, Applied Mechanics and Materials, vol. 536-537, 2014, pp. 694 – 697, DOI 10.4028/www.scientific.net/AMM.536-537.694
- [18] Casey Crane, “15 steps for setting up your own certificate authority.” <https://www.thesslstore.com/blog/setting-up-your-own-certificate-authority/> Accessed: 2024/11/01
- [19] N. D. Meulen, “Diginotar: Dissecting the first dutch digital disaster”, Journal of Strategic Security, vol. 6, 2013, pp. 46–58, DOI 10.5038/1944-0472.6.2.4
- [20] N. Leavitt, “Internet security under attack: The undermining of digital certificates”, Computer, vol. 44, no. 12, 2011, pp. 17–20, DOI 10.1109/MC.2011.367
- [21] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.” RFC 6960, June 2013, DOI 10.17487/RFC6960
- [22] B. Laurie, A. Langley, and E. Kasper, “Certificate Transparency.” RFC 6962, June 2013, DOI 10.17487/RFC6962
- [23] B. Laurie, E. Messeri, and R. Stradling, “Certificate Transparency Version 2.0.” RFC 9162, December 2021, DOI 10.17487/RFC9162
- [24] Google, “certificate-transparency-go.” <https://github.com/google/certificate-transparency-go> Accessed: 2025-02-25
- [25] H. Kwon, S. Lee, M. Kim, C. Hahn, and J. Hur, “Certificate transparency with enhanced privacy”, IEEE Transactions on Dependable and Secure Computing, vol. 20, no. 5, 2023, pp. 3860–3872, DOI 10.1109/TDSC.2022.3214235
- [26] C. Vecchio, “Certificate validation and domain impersonation”, Master’s thesis, Politecnico di Torino, 2021
- [27] DomCop, “Top 10 million websites.” <https://www.domcop.com/top-10-million-websites> Accessed: 2024/12/08
- [28] Common Crawl, “Common crawl.” <https://commoncrawl.org/>
- [29] Google, “From data warehouse to a unified, ai-ready data platform.” <https://cloud.google.com/bigquery/?hl=en#from-data-warehouse-to-a-unified-ai-ready-data-platform> Accessed: 2024/12/08
- [30] The ZMap Project, “The zmap project.” <https://zmap.io/> Accessed: 2024/12/10
- [31] National Science Foundation, “National science foundation.” <https://www.nsf.gov/> Accessed: 2024/12/10
- [32] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, “Tracking certificate misissuance in the wild”, 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 785–798, DOI 10.1109/SP.2018.00015
- [33] Alban Diquet, “In security.” <https://nabla-c0d3.github.io/> Accessed: 2024/12/20
- [34] Donald E. Eastlake 3rd, “Transport Layer Security (TLS) Extensions: Extension Definitions.” RFC 6066, January 2011, DOI 10.17487/RFC6066

- [35] Google, “Chrome user experience report (crux).” <https://developer.chrome.com/docs/crux> Accessed: 2024/12/08

# Appendix A

## User's Manual

### A.1 Installation Guide for DigitalCertiAnalytics

The following manual is intended to guide the reader through the installation process and how to run the DigitalCertiAnalytics software. If all installation steps have already been completed, you may proceed directly to the section on using the tool A.2. Otherwise, it is essential to check that the following components are installed and operational:

- Zgrab2: Ensure that Zgrab2 has been successfully installed and is working. If not, see A.3 for installation instructions.
- Zlint: Zlint has been successfully installed and is operational to ensure that Zlint is correctly installed and operational. If you encounter problems, refer to the A.4 section for further details.
- SSLyze: SSLyze has been successfully installed and is working that SSLyze has been successfully installed and is working. See the A.5 section for installation assistance.
- MongoDB: Ensure that MongoDB is installed and running.

The installation of MongoDB must be carried out following the specific instructions for the operating system in use. For example, on Linux systems it is possible to check the status of the service with:

```
sudo systemctl status mongod
```

and, if necessary, start it with:

```
sudo systemctl start mongod
```

In addition, graphical interface tools such as MongoDB Compass (<https://www.mongodb.com/products/compass>) can be used to facilitate data management and visualisation.

First of all, the tool is developed in Python, so it is essential to have an up-to-date version of Python 3. Therefore, it is necessary to download and install the latest version from the official site (<https://www.python.org/>) and, possibly, create a virtual environment to isolate the project dependencies.

After making sure Python is installed, open the Terminal and clone the DigitalCertiAnalytics repository from the repository on GitHub with the following command:

```
git clone git@github.com:MagliariElio/DigitalCertiAnalytics.git
cd DigitalCertiAnalytics
```

In addition, it is essential to install all the Python dependencies indicated in the `requirements.txt` file located in the base folder of the repository you have just downloaded. You can optionally create a virtual environment with, for example:

```
python -m venv env
```

and activating it with:

```
source env/bin/activate    % on Unix/macOS
```

or:

```
env\Scripts\activate      % on Windows
```

You can install the necessary libraries by running the command from the terminal:

```
pip install -r requirements.txt
```

or:

```
pip3 install -r requirements.txt
```

In addition, it is necessary to download the file containing the SCT Logger Operator log list from the following URL:

```
https://www.gstatic.com/ct/log\_list/v3/all\_logs\_list.json
```

After downloading, rename the file to `log_list.json` and place it in the `res` folder of the repository.

Finally, it is essential that the Zlint executable be located in the `zlint/v3/zlint` folder within the repository base directory, otherwise, the tool will not be able to locate the executable.

### A.1.1 Data Analysis Using ZGrab2

In order to run DigitalCertiAnalytics, it is necessary to provide, in addition to the previously downloaded log file, the output file generated by Zgrab2. The latter, named `certs_polito.json`, must be placed in the `res` folder of the repository.

The output file is created from a `.txt` file containing the list of domains to be analysed. To generate it, use the command:

```
zgrab2 tls --port 443 -f input_file.txt
      --output-file=certs_polito.json --timeout=30
```

Where:

- `--port 443`: specifies the port to contact (TLS/SSL).
- `-f input_file.txt`: indicates the file containing the domains to be analysed, which must only be domains without further details, as described in the official ZGrab2 documentation.
- `--output-file=certs_polito.json`: defines the output file containing the results of the analysis.
- `--timeout=30`: sets a timeout of 30 seconds for each request.

## A.2 Launching DigitalCertiAnalytics

In order to run the DigitalCertiAnalytics tool, it is necessary to go to the root directory of the repository (base folder) and then access the `src` folder using the following command:

```
cd DigitalCertiAnalytics/src
```

The tool can be executed with the following command:

```
python -m analysis.main --h
```

If, after execution, help information is displayed, this indicates that the tool has been installed correctly. The options available from the terminal are as follows:

- `--delete_all_db`: delete all existing databases before starting the analysis.
- `--delete_leaf_db`: delete the leaf certificate database before starting the analysis.
- `--delete_intermediate_db`: delete the database of intermediate certificates before starting the analysis.
- `--delete_root_db`: delete the root certificate database before starting the analysis.

- `--leaf_analysis`: performs the analysis of leaf certificates.
- `--leaf_ocsp_analysis`: performs OCSP analysis for leaf certificates.
- `--leaf_zlint_check`: performs Zlint analysis on leaf certificates to check for vulnerabilities and misconfigurations according to certain official requirements.
- `--leaf_chain_validation`: performs the validation of the leaf certificate chain to verify the conformity and reliability of the trust chain.
- `--intermediate_analysis`: performs the analysis of intermediate certificates.
- `--intermediate_ocsp_analysis`: performs OCSP analysis for intermediate certificates.
- `--intermediate_zlint_check`: performs the Zlint analysis on intermediate certificates to check for vulnerabilities and incorrect configurations according to certain official requirements.
- `--root_analysis`: performs the analysis of root certificates.
- `--root_ocsp_analysis`: performs OCSP analysis for root certificates.
- `--plot_all_results`: generates and displays graphs for all analysed certificate data.
- `--plot_leaf_results`: generates and displays graphs for the results of the leaf certificate analysis.
- `--plot_intermediate_results`: generates and displays graphs for intermediate certificate analysis results.
- `--plot_root_results`: generates and displays graphs for root certificate analysis results.
- `-v, --verbose`: activates verbose mode for detailed recording.

### A.2.1 Usage Examples

- Only perform analysis of leaf certificates:

```
python -m analysis.main --leaf_analysis
```

- Perform OCSP analysis for root certificates and generate graphs for root certificates only:

```
python -m analysis.main --root_ocsp_analysis --plot_root_results
```

- Activate verbose mode for detailed recording:

```
python -m analysis.main --delete_leaf_db --leaf_analysis -v
```

- Perform Zlint analysis on leaf certificates and validate the chain:



```
python -m analysis.main --leaf_zlint_check --leaf_chain_validation
```

## A.3 ZGrab2 Installation Guide

This section provides instructions for installing and compiling Zgrab2 on the main operating systems, namely macOS, Windows and Ubuntu.

### A.3.1 Installation on macOS

On macOS, the first step is to ensure that you have Homebrew installed, which makes it much easier to install software from the command line. If you do not yet have Homebrew, you can install it by following the instructions on the official site. Once you have verified the presence of Homebrew, proceed by installing Go, which is essential for compiling Zgrab2, by running the command:

```
brew install go
```

After installing Go, open the Terminal and clone the Zgrab2 repository from the official repository on GitHub with the following command:

```
git clone https://github.com/zmap/zgrab2.git
```

Once cloning is complete, enter the newly created folder using the command `cd zgrab2` and compile the project with the command:

```
go build
```

The compilation process will generate the Zgrab2 executable in the same directory. To check that everything is working properly, run the command:

```
./zgrab2 --help
```

If the output shows the available options, the installation was successful.

### A.3.2 Installation on Windows

For Windows users, it is first important to have both Git and Go installed. Go can be downloaded directly from the official site <https://golang.org/dl/>. Once the installation of these tools is complete, open Git Bash or the Command Prompt and clone the Zgrab2 repository by running:

```
git clone https://github.com/zmap/zgrab2.git
```

After cloning, move to the project directory with:

```
cd zgrab2
```

Then proceed to compile Zgrab2 using the command:

```
go build
```

This command will generate an executable file named `zgrab2.exe`. To verify that the installation is correct, run

```
zgrab2.exe --help
```

and check that the help message is displayed, which confirms successful installation.

### **A.3.3 Installation on Ubuntu**

On Ubuntu, the procedure starts with the installation of Go. Open the Terminal and run

```
sudo apt update  
sudo apt install golang
```

Once Go has been installed, clone the Zgrab2 repository with:

```
git clone https://github.com/zmap/zgrab2.git
```

Next, access the project folder by typing:

```
cd zgrab2
```

Finally, proceed with the compilation of the tool using the command:

```
go build
```

At the end of the compilation, to verify the installation, run:

```
./zgrab2 --help
```

and check that the output shows the usage options, confirming that everything has been installed correctly.

## **A.4 Zlint Installation Guide**

This section explains how to install and compile Zlint on the main operating systems, i.e. macOS, Windows and Ubuntu.

### A.4.1 Installation on macOS

On macOS, the installation of Zlint begins by checking that the necessary tools are present, in particular Homebrew and Go. If Homebrew is not already installed, it is advisable to proceed according to the official instructions, in order to simplify the installation of Go. Once Go has been installed (using the command `brew install go` in the Terminal), the official Zlint repository can be cloned using the command:

```
git clone https://github.com/zmap/zlint.git
```

Next, move to the newly created folder with the command `cd zlint`. To compile Zlint, simply run the command `make` (or alternatively `go build`, depending on the instructions provided in the repository). Once the build is complete, the executable will be generated in the current directory. To check that the tool is working properly, you can launch:

```
./zlint --help
```

and check that a guide to the available options is shown.

### A.4.2 Installation on Windows

On Windows, it is first recommended to install Git and Go, which are available from the official Git site and <https://golang.org/dl/> respectively. After setting up the environment, you can open Git Bash or the Command Prompt and proceed to clone the Zlint repository via:

```
git clone https://github.com/zmap/zlint.git
```

Once cloning is complete, move to the project folder with the command `cd zlint`. Compilation is performed by executing the command

```
make
```

or, if the repository requires it, the command `go build` may be used. The resulting executable, typically named `zlint.exe`, can be verified by running:

```
zlint.exe --help
```

in order to confirm that Zlint has been installed correctly and that the usage options are displayed.

### A.4.3 Installation on Ubuntu

On Ubuntu, the installation of Zlint follows a similar procedure. First, it is necessary to update the system and install Go by executing the following commands in the Terminal:

```
sudo apt update
sudo apt install golang
```

Once Go has been installed, the Zlint repository can be cloned by typing:

```
git clone https://github.com/zmap/zlint.git
```

After cloning, move to the project folder with `cd zlint` and proceed with the compilation of the tool, using the command:

```
make
```

or the command `go build`, as specified in the repository. At the end of the build, to verify that the installation was successful, we execute:

```
./zlint --help
```

and check that the user guide is shown, thus certifying the successful installation.

## A.5 SSLyze Installation Guide

This section provides instructions for installing and compiling SSLyze on the main operating systems, i.e. macOS, Windows and Ubuntu.

### A.5.1 General prerequisites

Before proceeding with the installation, you must ensure that your system has Python 3 installed, as `sslyze` is developed in Python. If Python 3 is not installed, you will have to install it following the standard procedures for your operating system. In addition, it is essential to have access to the command `pip` (or `pip3`), the Python package management tool.

### A.5.2 Installation on macOS

On macOS, you must first check that you have an up-to-date version of Python 3. If not, Python 3 can be installed easily, for example via Homebrew. Once Python 3 has been confirmed, proceed to install `sslyze` using `pip3` via the following command:

```
pip3 install sslyze
```

Once the installation is complete, it is advisable to check the correct functioning of the application by launching the command:

```
sslyze --help
```

This command displays the available options, confirming that `sslyze` has been correctly installed.

### **A.5.3 Installation on Windows**

On Windows, the process begins with verifying the installation of Python 3, which can be downloaded from the official site. After installing Python 3, open the Command Prompt (or PowerShell) and run the following command to install `sslyze`:

```
pip install sslyze
```

Once installation is complete, you can check operation by typing:

```
sslyze --help
```

This command will display the options and user guide, confirming the correct installation of `sslyze`.

### **A.5.4 Installation on Ubuntu**

On Ubuntu, the process starts with updating the system and installing `python3-pip` if it is not already present. You need to open a Terminal and run the following commands:

```
sudo apt update
sudo apt install python3-pip
```

Once `pip3` is installed, proceed with the installation of `sslyze` via:

```
pip3 install sslyze
```

Finally, to check that the installation was successful:

```
sslyze --help
```

The control will display the user manual, attesting to the correct operation of the instrument.

# Appendix B

## Developer's Reference Guide

This chapter provides a detailed overview of the structure of the tool, illustrating the composition of the folders, the location of the database schema and the internal workings of the system. In particular, it shows how the same schema is used for analysing *leaf*, *intermediate* and *root* certificates, with the tool being able to distinguish which databases to create or use based on input commands. In addition, the use of the Python file "src/scan/scan\_certificates.py" to perform quick scans on the Zgrab2 output file is described.

### B.1 Directory and File Structure

The repository is organised in a modular manner, with a clear distinction between resources, databases, analysis scripts and scanning tools. Below is an overview of the main components:

- **Folder res/:**
  - `certs_polito.json`: JSON file generated by Zgrab2 containing the certificates to be analysed.
  - `log_list.json`: JSON file containing the list of loggers (SCT Logger Operator) used by the programme for analysing SCTs.
  - `queries.sql`: SQL script used to perform analysis queries on relational databases.
  - `queries_big_query.sql`: script containing the query to obtain a list of European domains via Google's BigQuery service.
- **Folder src/:**
  - `analysis/`: contains databases and other analysis files, broken down by type:
    - \* `leaf/leaf_certificates.db`: SQLite database for analysing leaf certificates.
    - \* `intermediate/intermediate_certificates.db`: SQLite database for analysing intermediate certificates.

- \* `root/root_certificates.db`: SQLite database for analysing root certificates.
- \* Backup of OSCP analysis results and `plots` directory for each category (leaf, intermediate, root).
- \* `zlint_analysis/reading_zlint_results.py`: Python script designed to extract data from the non-relational MongoDB database and instrumental for performing detailed analyses on the stored certificate data.
- `db/schema_db.sql`: SQL script for the creation of the common schema used for the leaf, intermediate and root certificate databases.
- `scan/scan_collezione_certificati.py`: Python script that allows you to perform quick scans on the `certs_polito.json` (output of Zgrab2) to extract customised information.

## B.2 Log Management

The tool generates a log file called `app.log`, which documents the progress of the application and logs any errors. The log file is available in the base folder (`src`) of the programme's execution.

## B.3 MongoDB Database Dumping

To create a backup (dump) of MongoDB databases, the `mongodump` command can be used. You must ensure that `mongodump` is installed on your system. Below are the commands to perform the dump:

### 1. For the database *Leaf\_Zlint\_Checks*:

```
mongodump --db Leaf_Zlint_Checks --out Leaf_Zlint_Checks
```

The dumps generated may be used for data recovery or further analysis.

## B.4 Final Developer Observations

The tool is designed to be modular and flexible. The same database schema defined in `src/db/schema_db.sql` is uniformly applied to leaf, intermediate and root certificates. Based on the commands passed as input, the tool autonomously manages the creation or use of the respective databases, thus facilitating integrated analysis and data management. Moreover, no prior database setup is required,

everything is created autonomously, simply by following the installation procedure in the dedicated Section A.1. Additionally, a file named `/res/queries.sql` contains all the useful queries needed to interrogate the databases for retrieving results. This file is highly recommended as a direct reference and is ideally suited for modifying queries to generate customized charts.